SOFTWARE REFERENCE MANUAL

Turbo PMAC/PMAC2

Software Reference for Turbo Family

3Ax-01.937-xSxx

August 16, 2004

DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION

What is Turbo PMAC?

The Turbo PMAC is the newest addition to the renowned PMAC family of motion controllers. The Turbo refers to a new high-performance CPU section that can be used with existing PMAC(1) or PMAC2 interface circuitry to turbo-charge the application.

The Turbo PMAC is currently available in six versions:

- Turbo PMAC-PC: PMAC(1) servo interface circuitry, PC (ISA) bus interface
- Turbo PMAC-VME PMAC(1) servo interface circuitry, VME bus interface
- Turbo PMAC2-PC PMAC2 servo interface circuitry, PC (ISA) bus interface
- Turbo PMAC2-VME PMAC2 servo interface circuitry, VME bus interface
- Turbo PMAC2-PC Ultralite MACRO servo interface circuitry, PC(ISA) bus interface
- Turbo PMAC2-3U (UMAC Turbo and 3U Turbo Stack)
  PMAC2 servo interface circuitry, PC/104 bus interface

Each of these versions has its own Hardware Reference manual.

More versions will be available in the near future.

What is New about Turbo PMAC?

The Turbo PMAC uses the increased speed and memory of the newest generation of digital signal processing (DSP) ICs to enhance the capabilities of the PMAC family. The Turbo PMAC has the software capability to control 32 axes in 16 independent coordinate systems, up from 8 axes in 8 coordinate systems for the standard PMAC.

Many users will find the Turbo PMAC a very powerful and cost-effective solution when controlling large numbers of axes. Remember that a PMAC board itself has at most eight servo interface channels; the actual control of more than eight physical axes will require the use of either ACC-24 family axis expansion boards, or remote interface circuitry on the MACRO ring.

The extra software axis capability can be useful for virtual axes which do not require (full) physical hardware interface circuitry. Virtual axes have many important uses, including:

- Phantom coordinate systems in tool tip coordinates for inverse kinematics
- Virtual masters to replace mechanical line-shaft masters
- Redundant axes for error checking and recovery purposes
- Cascaded servo loops for hybrid control techniques (e.g. force and position)

Many other users will find the Turbo PMAC valuable even if less than 8 axes are used, just because of the additional computational speed. The DSP of the base version of the Turbo PMAC runs at 80 MHz, but because operations on internal registers (about half of all operations) run in one clock cycle instead of the two clock cycles required for the standard PMAC, performance is equivalent to that of a 120 MHz standard PMAC.

The additional memory addressing capability of the Turbo PMAC permits the use of more axes and coordinate systems, and more features for it. It also supports more variables, and (optionally) much larger user buffer spaces.

With the additional speed and memory, new features are possible on the Turbo PMAC. The most important of these are:

- Multi-block lookahead for acceleration control
- Built-in inverse-kinematic and forward-kinematic capability
- Three-dimensional cutter-radius compensation
- Altered destination of moves on the fly
- Simultaneous communications over multiple ports
- Individual custom commutation sine tables for each motor
- Individual selection by motor of PID or extended servo algorithm
- Significantly enlarged synchronous M-variable buffer
- 2 dedicated user servo-rate timers per coordinate system
- Trajectory reversal capability

How do I Convert a PMAC Application?

Converting a PMAC application to run on the Turbo PMAC will involve some change in the setup, but virtually no change in the applications programs, except as desired to take advantage of new Turbo features.

The key setup differences are the new I-variable numbering scheme and the new memory and I/O map, which affects the M-variable definitions. Most I-variables, particularly the motor I-variables, have not changed. Other I-variables have been moved in banks to new numbers, in what most users will consider a logical fashion.

The memory and I/O map is completely changed. This software reference manual contains a detailed memory and I/O map, plus an extensive list of suggested M-variables for both PMAC(1) and PMAC2 versions of the Turbo PMAC.

How do I use this Manual?

The Turbo PMAC Software Reference manual provides detailed information on all of the variables, commands, and registers of the Turbo PMAC family. Variables and registers are presented in numerical order; commands are presented in alphabetical order.

This manual is designed to be used in conjunction with the User's Manual for the entire PMAC/PMAC2 family of controllers, which explains the features and capabilities of the board in conceptual fashion. The User's Manual was written before the introduction of the Turbo PMAC boards, so it does not recognize some specifics of the Turbo PMACs. Chapter 2 of this manual presents the differences between Turbo and non-Turbo boards in tabular form for easy reference; Chapter 3 describes the significant new features of Turbo PMACs.

The hardware reference manuals for each particular version of the Turbo PMAC describe the hardware configuration, jumpers, and pinouts for the particular boards.

CHANGE SUMMARY: PMAC TO TURBO PMAC

Overview Feature Comparison

Feature	PMAC	Turbo PMAC
Maximum Number of Axes	8	32
Maximum Number of Servo Interface Channels	16 (8 off-board)	40 (32 off-board)
Maximum Number of MACRO Nodes	16	64 (Turbo Ultralite)
Maximum Number of Coordinate Systems	8	16
CPU Frequencies (MHz)	20, 40, 60, 80	80, 100
Instructions per CPU Clock Cycle	0.5	0.75
Effective CPU Frequencies (MHz)	20, 40, 60, 80	120, 150
Number of I-Variables	1024	8192
Number of M-Variables	1024	8192
Number of P-Variables	1024	8192
Number of Q-Variables	1024	8192
Commutation Sine Table Size	256	2048
Individual Custom Sine Table for each motor?	No	Yes
Individual Choice by Motor Between PID and Extended Servo Algorithm?	No	Yes
Firmware Memory Capacity	32k x 24	64k x 24
Compiled PLC Memory Capacity	15k x 24	48k x 24, 432k x 24
User Buffer Memory Capacity	32k x 48	26k x 48, 212k x 48
DPRAM Capacity	8k x 16	8k x 16, 32k x 16
Supplemental Battery-Backed Memory Capacity	7k x 48	16k x 48, 64k x 48
Maximum Number of Motion Programs	256	224
Built-In Kinematics Algorithm Capability?	No	Yes
Multi-Block Lookahead for Acceleration Control?	Optional (Opt 6L)	Standard
Number of Serial Ports	1	2 (2nd optional)
Simultaneous Use of Multiple Communications Ports?	No	Yes
Ability to Disable Automatic Serial-Port Command Parser?	No	Yes
Maximum Simultaneous Data Gathering Sources	24	48
Maximum Number of Lines in Conversion Table	32	192
Maximum Number of Synchronous M-Variable Assignments per Move	3	63 (1 Coordinate) System), user-set with lookahead
Direct Array Writing Capability (e.g. **P(P1)=2**)?	No	Yes

I-Variable Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Ix00 – Ix86 (x = 1 to 8) Motor I-Variables	Ixx00 – Ixx99 (xx = 1 to 32) Motor I-Variables
Ix30 – Ix58 (x = 1 to 8) Option 6 Extended Servo Algorithm Gains	Iyy10 – Iyy39 (xx = 2*[yy-32]-1) for odd-numbered motors; Iyy60 – Iyy89 (xx = 2*[yy-32]) for even-numbered motors: Extended Servo Algorithm Gains (Standard)
Ix87 – Ix99 (x = 1 to 8) Coordinate System I-Variables	I5x00 – I5x99 (x = 1 to 9) Coordinate System I-Variables I6y00 – I6y99 (y = x – 10; x = 10 to 16) C.S. I-Variables
I900 – I904, I905 – I909, … I975 – I979 PMAC(1) Encoder Setup I-Variables	I7mn0 – I7mn4 (m = 0 to 9, n = 1 to 4) PMAC(1) Servo IC m Channel n Encoder Setup I-Variables
I900 – I909 PMAC2 Multi-Channel Hardware Setup I-Variables	I7m00 – I7m05 (m = 0 to 9) Servo IC m Multi-Channel Hardware Setup I-Variables
I9n0 – I9n9 (n = 1 to 8) PMAC2 Channel n Hardware Setup I-Variables	I7mn0 – I7mn9 (m = 0 to 9, n = 1 to 4) PMAC2 Servo IC m Channel n Hardware Setup I-Variables
I990 – I999 MACRO & Supplemental Channel Hardware Setup I-Variables	I6800 – I6849 MACRO IC 0 Hardware Setup I-Variables
I1000 – I1005 MACRO Software Setup I-Variables	I70 – I82 MACRO Software Setup I-Variables
X/Y:$0700 - $0701 User Countdown Timers	Isx11, Isx12 Coordinate System Countdown Timers
Y:$0720 – Y:$073F Encoder Conversion Table Setup	I8000 – I8191 Encoder Conversion Table Setup
X:$0783 – X:$078C VME/DPRAM Setup	I90 – I99 VME/DPRAM Setup
I7 In-Position Number of Cycles (Global)	Ixx88 Motor xx In-Position Number of Cycles
I12 Jog Calculation Time (Global)	Ixx92 Motor xx Jog Calculation Time
I13 Move Segmentation Time (Global)	Isx13 Coord. Sys. ‘x’ Move Segmentation Time
I14 Auto Position Match on Run (Global)	No I-variable – function always active
I19 Data Gathering Period	I47 DPRAM Servo Data Reporting Period I5049 Data Gathering Period
I20 Data Gathering Mask	I5050, I5051 Data Gathering Masks
I21-I44 Data Gathering Source Addresses	I5001-I5048 Data Gathering Source Addresses
I45 Data Gathering Buffer Location and Mode	I5000 Data Gathering Buffer Location and Mode
I47 Pointer Address for <CTRL-W> Command	No I-variable – function not used
I50 Rapid Speed Select (Global)	Ixx90 Motor xx Rapid Speed Select
I52 ‘\’ Program Hold Slew Rate (Global)	No program hold: ‘\’ is “quick stop” within lookahead, using Ixx17; “feed hold” out of lookahead, using Ix95
I53 Single Step Mode (Global)	Isx53 Coord. Sys. ‘x’ Single Step Move
I57 DPRAM Binary Rotary Buffer Enable	On-line command OPEN BIN ROT
I59 DPRAM Buffer Max. Motor/C.S. Number	DPRAM Motor Enable Mask Word DPRAM Maximum C.S. Transfer Number
I60, I61 [PMAC(1)] or X:$0708 – X:$070F [PMAC2] A/D Processing Table Setup	I5060 – I5096 A/D Processing Table Setup
I8x Motor x 3rd-Resolver Gear Ratio	Ixx98 Motor xx 3rd-Resolver Gear Ratio
I89 Cutter Comp Outside Corner Break Angle (Global)	Isx99 Coordinate System ‘x’ Cutter Comp Outside Corner Break Angle
I90 Minimum Arc Length (Global)	Isx97 Coordinate System ‘x’ Minimum Arc Length
I9x Motor x 2nd-Resolver Gear Ratio / Yaskawa Absolute Encoder Ratio	Ixx99 Motor xx 2nd-Resolver Gear Ratio / Yaskawa Absolute Encoder Ratio
I99 Backlash Hysteresis (Global)	Ixx87 Motor xx Backlash Hysteresis

I-Variable Changes (continued)

PMAC/PMAC2	Turbo PMAC/PMAC2
Ix02 Motor x Command Output Address & Mode	Ixx02 Motor xx Command Output Address Ixx01 Motor xx Commutation Enable (& Address Type) Ixx96 Motor xx Command Output Mode
Ix03 Motor x Position-Loop Feedback Address & Mode	Ixx03 Motor xx Position-Loop Feedback Address Ixx97 Motor xx Position Capture & Trigger Mode
Ix05 Motor x Master Position Address & Mode Ix06 Motor x Master Position Following Enable	Ixx05 Motor xx Master Position Address Ixx06 Motor xx Position Following Enable & Mode
Ix10 Motor x Power-On Position Address & Format	Ixx10 Motor xx Power-On Position Address Ixx95 Motor xx Power-On Position Format
Ix25 Motor x Flag Address & Mode	Ixx25 Motor xx Flag Address Ixx24 Motor xx Flag Mode
Ix72 Motor x Commutation Phase Angle Units of 1/256 cycle	Ixx72 Motor xx Commutation Phase Angle Units of 1/2048 cycle
Ix81 Motor x Power-On Phase Position Address & Format	Ixx81 Motor xx Power-On Phase Position Address Ixx91 Motor xx Power-On Phase Position Format
Ix83 Motor x Commutation Feedback Address & Type	Ixx83 Motor xx Commutation Feedback Address Ixx01 Motor xx Commutation Enable (& Address Type)

DPRAM Function Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Servo Data Reporting Buffer contains only motor data that is updated at servo rates; Background Data Reporting Buffer contains lower frequency motor data, coordinate system data, and global data.	Motor Data Reporting Buffer contains all motor data that is reported; Background Data Reporting Buffer contains coordinate system data and global data
Servo Data Reporting Buffer must be reported as foreground task	Motor Data Reporting Buffer can be reported as foreground or background task
Servo Data Reporting Buffer reports raw (uncombined) motor position registers	Motor Data Reporting Buffer reports net (combined) motor position values
On-line ASCII commands prohibited while DPRAM binary rotary buffer(s) open.	On-line ASCII commands permitted while DPRAM binary rotary buffer(s) open.
Off-board Option 2 DPRAM board for PMAC(1) ISA bus boards can connect to all 50 pins of PMAC's JEXP connector	Off-board Option 2 DPRAM for Turbo PMAC(1) ISA bus boards cannot connect to pins 41 and 42 of PMAC's JEXP connector. These pins must be cut on the DPRAM board, or on strands of the cable to the DPRAM board

Compensation Table Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Table correction at motor position zero is 0 by definition; last entry in table must be 0 for seamless rollover.	Last entry in table sets correction at motor position zero as well as other end of table; seamless rollover is automatic.

Commutation Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Commutation lookup table has 256 entries; direct microstepping yields 64 microsteps per step	Commutation lookup table has 2048 entries; direct microstepping yields 512 microsteps per step
Phase reference error bit only set on unsuccessful phasing search or read. Loop is not closed then, but may be subsequently closed.	For any synchronous motor commutated by Turbo PMAC, phase reference error bit set automatically on power-up/reset. Cleared only on successful phasing search or read. Loop cannot be closed if bit is set.
Phasing search move on $ command is exclusive background task. No other background task can execute until phasing search move is complete. Command acknowledgment to host is not given until move is complete.	Phasing search move on $ command is non-exclusive background task. Other background tasks can execute while phasing search move is in progress. Command acknowledgment to host is given when move starts.

Overtravel Limit Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
If hardware or software overtravel limit is hit while in open-loop mode, motor loop is closed (zero-velocity)	If hardware or software overtravel limit is hit while in open-loop mode, motor is killed (amp disabled, zero command).

Cutter Radius Compensation Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Two-dimensional compensation only	Two-dimensional or three-dimensional compensation
2D compensation cannot “see through” out-of-plane move to maintain compensation properly in all cases.	Special CCBUF can be defined to store out-of-plane moves in 2D compensation to maintain proper compensation through these moves.

Communications Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Characters received on any communications port are loaded into a common command queue, so simultaneous commands from multiple ports are intermixed and garbled.	Characters received on each communications port are loaded into a separate command queue for that port, so that simultaneous commands from multiple ports can be accepted properly.
When a program buffer is open, it can accept commands from any port into the buffer, so another port cannot give on-line commands (in general).	A program buffer is opened for a particular port only; only commands from that port can be entered into the buffer; other ports can only give on-line commands.

Memory and I/O Map Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
$0000 - $001F Global Servo Registers	$000000 - $00007F Global Servo Registers
$0020 - $01FF Motor Servo Registers	$000080 - $00107F Motor Servo & Background Registers
$0400 - $04FF Commutation Sine Table	$003800 - $003FFF Commutation Sine Table
$0700 - $0701 User Countdown Timers	$002x13 Coordinate System User Countdown Timers (Addressable as Isx11 and Isx12)
$0708 - $070F Processed A/D Table	$003400 - $00341F Processed A/D Table
$0720 - $073F Conversion Table Registers	$003501 - $0035C0 Conversion Table Registers
$0770 - $077F Open registers set to 0 on power-up/reset	$0010F0 - $0010FF Open registers set to 0 on power-up/reset
$0800 - $0DFF Motor Background & Coordinate System Registers	$000080 - $00107F Motor Servo & Background Registers $008000 - $008FFF Coordinate System Registers
$0F70 - $0F7F MACRO Flag Registers	$003440 - $00347F MACRO Flag Registers
$1000 - $13FF P-Variable Registers	$006000 - $007FFF P-Variable Registers $050000 - $051FFF Alternate P-Variable Registers
$1400 - $17FF Q-Variable Registers	$008000 - $009FFF Q-Variable Registers $052000 - $053FFF Alternate Q-Variable Registers
$1800 - $9FFF User Program & Buffer Space	$00A000 - $0107FF Standard User Program & Buffer Space $00A000 - $03FFFF Extended User Program & Buffer Space
$A000 - $BBFF Opt. 16 Battery-Backed Memory	$050000 - $053FFF Opt. 16A Standard Battery-Backed Memory $050000 - $05FFFF Opt. 16B Extended Battery-Backed Memory
$BC00 - $BFFF M-Variable Definitions	$004000 - $005FFF M-Variable Definitions
$C000 - $C01F On-board PMAC(1) Servo IC Registers	$078000 - $07810F On-board PMAC(1) Servo IC Registers
$C020 - $C03F Off-board PMAC(1) Servo IC Registers	$078200 -$07B30F Off-board PMAC(1) Servo IC Registers
$C000 - $C03F On-board PMAC2 Servo IC Registers	$078000 - $07811F On-board PMAC2 Servo IC Registers
$C040 - $C07F Off-board PMAC2 Servo IC Registers	$078200 - $07B31F Off-board PMAC2 Servo IC Registers
$C080 - $C0BF On-board PMAC2 MACRO IC Registers	$078400 - $07843F On-board PMAC2 MACRO IC 0 Registers
$D000 - $DFFF DPRAM Registers	$060000 - $060FFF Standard DPRAM Registers $060000 - $063FFF Extended DPRAM Registers
$E000 - $E1FF VME Interface Registers	$070000 - $0701FF VME Interface Registers
$FFC0 - $FFC3 PMAC(1) On-board I/O Registers	$078800 - $078803 PMAC(1) On-board I/O Registers
$FFC0 - $FFC1 PMAC2 Opt. 12 ADC Registers	$078800 - $078801 PMAC2 Opt. 12 ADC Registers
$FFD0 - $FFFB Expansion Port I/O Registers	$078A00 - $078F03 Expansion Port I/O Registers

Jumper Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
PMAC(1) E40 – E43 Card Number/Clock Direction	Turbo PMAC(1) E40 – E43 Clock Direction I0 Serial Card Number
PMAC(1) E44 – E47 Serial Baud Rate	Turbo PMAC(1) E44 – E47 (not used) I54 Serial Baud Rate
PMAC(1) E48 Wait State Control	Turbo PMAC(1) E48 (not used)
PMAC1.5 E48 CPU Frequency Control	Turbo PMAC(1) E48 (not used) I52 CPU Frequency Multiplier
PMAC2 E2, E4 CPU Frequency Control	I52 CPU Frequency Multiplier
PMAC(1) E50 Save Enable Control	Turbo PMAC(1) E50 (not used)

On-line Command Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
<CTRL-Z>: Change Active Response Port to Serial Port	**{none}** {no Active Response Port – all ports are independent, responding to commands from that port}
<CTRL-B>, <CTRL-P>, <CTRL-V>, <CTRL-F> always report for all PMAC motors (1-8)	**<CTRL-B>, <CTRL-P>, <CTRL-V>, <CTRL-F>** report for the eight motors selected by **##** command
<CTRL-E> command causes single-shot binary reporting of data gathering registers	**{none}** {This function does not exist on Turbo PMAC}
<CTRL-T> command toggles half/full duplex serial communications	**<CTRL-T>** command ends MACRO ASCII pass-through mode
<CTRL-W> command causes PMAC to execute command at DPRAM address specified by I47	**{none}** {This function does not exist on Turbo PMAC}
V, <CTRL-V> report raw (unfiltered) velocity values	**V, <CTRL-V>** report filtered velocities as controlled by I60 and I61
P, <CTRL-P> report position values rounded to nearest 1/10 count	**P, <CTRL-P>** report position values to 1/32 count
H, <CTRL-O> feedhold commands do not permit jog moves while in feedhold mode	**H, <CTRL-O>** feedhold commands *do* permit jog moves while in feedhold mode
**** command is feedhold that permits jog; if not in segmentation mode, acts as **H** command	**** command is fastest legal stop in lookahead; if not in lookahead, acts as **H** command
?? returns 12-digit (48-bit) response	**??** returns 18-digit (72-bit) response
I{constant}={expression} **M{constant}={expression}** **P{constant}={expression}** **Q{constant}={expression}** Require constant to specify variable number	**I{data}={expression}** **M{data}={expression}** **P{data}={expression}** **Q{data}={expression}** Can use expression to specify variable number
<CTRL-U> and **<CTRL-L>** open and close rotary buffer, respectively.	**{none}** – Use **OPEN ROT** and **CLOSE** text commands.
<CTRL-Y> brings back last text command, ready to execute on **<CR>**	**{none}**{This function does not exist on Turbo PMAC}

Program Command Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
SEND: Send message to Active Response Port **SENDS**: Send message to Serial Port **SENDP**: Send message to Parallel Bus Port (not VME)	**SEND**: {not a legal command} **SENDS**: Send message to Main Serial Port **SENDP**: Send message to Parallel Bus Port (incl. VME) **SENDR**: Send message to DPRAM ASCII Port **SENDA**: Send message to Auxiliary Serial Port
CMD: Issue command; any response to Active Response Port	**CMD**: Issue command; no response possible **CMDS**: Issue command; any response to Main Serial Port **CMDP**: Issue command; any response to Parallel Bus Port **CMDR**: Issue command; any response to DPRAM ASCII Port **CMDA**: Issue command; any response to Auxiliary Serial Port
SPLINE, PVT mode move times specified by **TA**, range to 8,388,607 msec, resolution of 1 msec	**SPLINE, PVT** mode move times specified by **TM**, range to 4096 msec, resolution of 1/4096 msec
I{constant}={expression} **M{constant}={expression}** **P{constant}={expression}** **Q{constant}={expression}** Require constant to specify variable number	**I{data}={expression}** **M{data}={expression}** **P{data}={expression}** **Q{data}={expression}** Can use expression to specify variable number

Encoder Conversion Table Changes

PMAC/PMAC2	Turbo PMAC/PMAC2
Setup entries accessible by memory location only (Y:$0720 – Y:$073F)	Setup entries accessible by I-variable (I8000-I8191) or by memory location (Y:$003501 – Y:$0035C0)
Source addresses are 16-bit values	Source addresses are 19-bit values
Addition of two entries accomplished by setting bit 20 of second entry to 1	Addition (or subtraction) of two entries accomplished by using a special $E format third entry
Sources for $2, $3 parallel entries limited to single Y-word	Sources for $2, $3 parallel entries extensible to any 24 bits of consecutive Y-words
Sources for $6, $7 parallel entries limited to single X-word	Sources for $6, $7 parallel entries extensible to any 24 bits of X/Y double word
Width of parallel sources specified by bit mask in second setup line	Width of parallel sources specified by number of bits in second setup line
Offset of parallel sources specified by 2 bits in first setup line	Offset of parallel sources specified by number of bits in second setup line
Triggered time base entry estimates frequency to 1/32 count per servo cycle; time-base scale factor is 2^{17}/(real-time input frequency)	Triggered time base entry estimates frequency to 1/256 count per servo cycle; time-base scale factor is 2^{14}/(real-time input frequency)
$F entry is only high-resolution interpolator conversion	$F entry is extended entry; if first digit of second line is $0, it is high-resolution interpolator conversion

TURBO PMAC/PMAC2 SYSTEM CONFIGURATION AND AUTO-CONFIGURATION

Turbo PMAC, and especially Turbo PMAC2, boards have extensive capabilities for automatically identifying and self-configuring their systems. This is particularly important for UMAC Turbo systems, with their wide variety of configurations. These capabilities provide the user with ease of use and flexibility in getting started with a particular configuration.

Turbo PMAC2 Clock Source

In a Turbo PMAC2 system, the system phase and servo clocks, which interrupt the processor and latch key input and output data for the servos, come from one (and only one) of the Servo ICs or MACRO ICs in the system, or possibly from an external source. There must be a unique source of the phase and servo clocks for an entire Turbo PMAC2 system.

(A Turbo PMAC1 board uses fixed, discrete logic to generate its phase and servo clocks. If accessory boards with Servo ICs or MACRO ICs that can generate their own clock signals are added to a Turbo PMAC1, they must be set up to use the Turbo PMAC1's clock signals.)

Each PMAC2-style Servo IC (DSPGATE1 IC) and DSPGATE2-type MACRO IC has the capability for generating its own phase and servo clock signals, or for accepting external phase and servo clock signals. (Note: MACROGATE-type MACRO ICs can generate their own phase clock, but not servo clock. Therefore, they cannot be used to generate clocks for the entire system.) At most one of these ICs in a system may generate its own clock signals – none if the signals come from an external source.

Variables I7m07 and I7m57 control the clock direction for Servo ICs 'm' and 'm*', respectively. Variables I6807, I6857, I6907, and I6957 control the clock direction for MACRO ICs 0, 1, 2, and 3, respectively. If the variable value is 0, the IC generates its own clock signals and outputs them. If the variable value is 3, the IC accepts the clock signals from a source external to it. At most one of these ICs can have this variable at a value of 0; the rest must be set to 3.

Note:

If more than one of these ICs is set up to use its own clock signals and to output them, the processor will be interrupted by multiple sources and will not operate normally – it is possible that the watchdog timer will trip. (Because the outputs are open-collector types, there will be no hardware damage from signal contention, but system software operation will be compromised.)

Re-Initialization Actions

On re-initialization of a Turbo PMAC2 (**$$$*****) command, or power-on/reset with re-initialization jumper E3 ON), the CPU searches all possible locations of Servo ICs and MACRO ICs to see which are present. It then makes a decision as to which of these ICs it will use to generate the system's phase and servo clocks, using the first IC that it finds in the following list:

1.	Servo IC 0	(On-board or 3U Stack)	(I19=7007)
2.	MACRO IC 0	(On-board or ACC-5E)	(I19=6807)
3.	Servo IC 1	(On-board or 3U Stack)	(I19=7107)
4.	Servo IC 2	(ACC-24E2, 51E)	(I19=7207)
…			
11.	Servo IC 9	(ACC-24E2, 51E)	(I19=7907)
12.	Servo IC 2*	(ACC-24E2, 51E)	(I19=7257)
…			

19.	Servo IC 9*	(ACC-24E2, 51E)	(I19=7957)
20.	MACRO IC 1	(On-board or ACC-5E)	(I19=6857)
21.	MACRO IC 2	(On-board or ACC-5E)	(I19=6907)
22.	MACRO IC 3	(On-board or ACC-5E)	(I19=6957)

(MACRO ICs must be "DSPGATE2" ICs to be used as a clock source.)

Next, it automatically sets I19 to the number of the clock-direction I-variable for the first of these ICs it finds. For example, if it finds Servo IC 0, it will set I19 to 7007. Finally, it will set this clock-direction I-variable to 0, and the clock-direction I-variable for all of the other Servo and MACRO ICs that it finds to 3.

However, if it finds on re-initialization that the E1 jumper is ON, specifying that the Turbo PMAC2 system is to use external phase and servo clocks, I19 is set to 0 and all of the clock-direction I-variables are set to 3.

Note:

Once the system has been set up to take external phase and servo clock signals, these signals must always be present while the system is powered, or the watchdog timer will trip immediately.

To change a system set up for external clocks back to internal clocks, it is necessary to power it up with the E3 re-initialization jumper ON, the E1 external-clock jumper OFF, and with the external clock signals present. Once the new configuration for internal-clock source is established, either by the automatic re-initialization actions, or user-set configuration (see below), these settings must be stored to flash memory with the **SAVE** command. Then the system can be powered down, the E3 re-initialization jumper removed, and the external clock signals removed. Finally, the system can be powered up again normally.

Note:

If a Turbo PMAC2 is on a MACRO ring, but it is not the ring controller "synchronizing master", it must be set up to have its phase clock adjusted by receipt of the "sync packet" over the MACRO ring. This is done by setting bit 7 of I6840 to 1, and bits 16 – 19 of I6841 to the node number of the sync packet (usually $F [=15]). In this case, the phase and servo clocks are still generated internally, although they are locked to receipt of this sync packet. Systems of this type should have I6807 set to 0 (I19 set to 6807) to use MACRO IC 0 as the source of the phase and servo clocks. If re-initialization does not automatically cause this to happen, it must be done manually (see below).

User Customized Clock-Source Specification

The user does not have to accept Turbo PMAC2's default configurations for clock sources. The procedure to change the clock source is slightly different for 3U-format Turbo PMAC2 systems and other Turbo PMAC2 systems.

In a 3U-format Turbo PMAC2 system (UMAC Turbo or 3U Turbo Stack), if you wish to change the clock source, simply follow the following 3-step procedure:

1. Set I19 to the number of the clock-source I-variable of the IC you want to be the source.
2. Store this value to non-volatile flash memory with the **SAVE** command.
3. Reset the system normally (not re-initialization).

Do not try to set the clock-direction I-variables directly.

In other Turbo PMAC2 systems, change the clock-direction I-variables themselves in a single command (e.g. **I6807=0 I7007=3**). It is best to change I19 to the number of the I-variable that you have just set to 0 (**I19=6807** in this example), but this is not necessary. Store these new values to non-volatile flash memory with the **SAVE** command. They will then automatically be used on every subsequent power-up/reset.

Servo and phase clock lines are bi-directional on the UBUS backplane expansion port in UMAC Turbo systems, so these signals can go either from or to the CPU board. However, on the JEXP flat-cable expansion port, these clock lines are uni-directional, and can only be output from the main PMAC board or CPU board.

If you set I19 to an improper value, the watchdog timer will trip immediately on reset. To recover, you must power down, install the E3 re-initialization jumper, and power up again.

The most common reason to change from the default setting is tying a Turbo PMAC2 that has Servo IC 0 and/or 1 to a MACRO ring where it is not the ring controller. In this case, you want MACRO IC 0 to be the clock source, but the re-initialization procedure will decide on Servo IC 0. In this case, you would change I19 from 7007 to 6807, **SAVE**, and reset.

Normal Reset Actions

On a normal power-up or reset sequence, the Turbo PMAC2 CPU reads the value of I19 that was previously saved to flash memory and sets the I-variable whose number it finds there to 0, specifying that this IC uses its own phase and servo clocks and outputs them to the system. For example, if I19 were 6807, I6807 would be set to 0. It would then automatically set the clock-direction I-variables for all of the other Servo and MACRO ICs that it finds at power-up/reset to 3, so these ICs accept servo and phase clock signals as inputs.

MACRO IC Selection

Starting in Turbo PMAC firmware version 1.936, I-variables I20 – I23 must be set to specify the address(es) of the MACRO IC(s) used for automatic firmware functions. This is not compatible with older firmware versions. If updating an application from an older version, after loading the old I-variable file, the command **I20..23=*** should be issued, followed by a **SAVE** command, followed by a **$$$** reset command.

Some Turbo PMAC2 systems (presently UMAC Turbo) can address up to 16 MACRO ICs. However, there is automatic firmware support for only 4 of these ICs at any given time. These ICs are referred to as MACRO ICs 0, 1, 2, and 3. Variables I20 through I23 specify the base addresses of MACRO ICs 0 through 3, respectively. These variables must be set properly to use the desired ICs for any automatic firmware functions.

MACRO IC 0, specified by I20, has several functions which require automatic firmware support:

- Display port functions (can be changed dynamically)
- Multiplexer port functions (can be changed dynamically)
- I-variables I6800 – I6849 (values automatically assigned only at power-up/reset)
- MACRO nodes 0 – 15 (can be changed dynamically)
- MACRO Type 1 auxiliary communications if I84=0

MACRO IC 1, specified by I21, has several functions which require automatic firmware support:

- I-variables I6850 – I6899 (values automatically assigned only at power-up/reset)
- MACRO nodes 16 – 31 (can be changed dynamically)
- MACRO Type 1 auxiliary communications if I84=1

MACRO IC 2, specified by I22, has several functions which require automatic firmware support:

- I-variables I6900 – I6949 (values automatically assigned only at power-up/reset)
- MACRO nodes 32 – 47 (can be changed dynamically)
- MACRO Type 1 auxiliary communications if I84=2

MACRO IC 3, specified by I23, has several functions which require automatic firmware support:

- I-variables I6950 – I6999 (values automatically assigned only at power-up/reset)
- MACRO nodes 48 – 63 (can be changed dynamically)
- MACRO Type 1 auxiliary communications if I84=3

On re-initialization, Turbo PMAC2 searches for the MACRO ICs with the lowest base addresses. I20 is assigned the lowest base address (if one is found); I21 is assigned the next (if found), and so on. Also, the same action is taken when assigning the default value to one of these variables (e.g. **I20=***).

Dual-Ported RAM IC Selection

Starting in Turbo PMAC firmware version 1.936, it is possible to specify the base address of the dual-ported RAM IC used for automatic firmware communications functions. This permits support for new accessories such as the ACC-54E USB/Ethernet communications board. In firmware versions V1.935 and older, the base address was fixed at $060000 for the "on-board" DPRAM.

New variable I24 specifies the base address of the DPRAM IC used for the automatic firmware communications functions. For backward compatibility, if I24 is set to 0, the DPRAM will be assumed to have a base address of $060000.

I24 is used only at power-up/reset. To use other than the on-board DPRAM IC, follow the instructions of the accessory such as the ACC-54E to set the value of I24 to match the hardware settings on the accessory. Then issue the **SAVE** command and the **$$$** command so the accessory can be used for communications. On re-initialization, I24 is set to the lowest base address of any DPRAM IC found.

System Configuration Status Reporting

Turbo PMAC systems can automatically detect and report significant information about their configuration. They do this by having the processor query possible address locations for interface ICs – Servo ICs, MACRO ICs, DPRAM ICs, and I/O ICs. This information can be very useful in the initial setup of a Turbo PMAC system, and subsequently to verify that the configuration has not changed.

Servo IC Configuration

On power-up/reset, the Turbo PMAC CPU automatically tests for the presence and type of all possible Servo ICs and reports the results in I4900 and I4901. I4900 is a collection of 20 independent bits, in which bits 0 – 9 report the presence of Servo ICs 0 – 9, respectively, and bits 10 – 19 report the presence of Servo ICs 0* to 9*, respectively. A bit value of 1 indicates the IC is present; a bit value of 0 indicates the IC is absent.

I4901 is also a collection of 20 independent bits, in which bits 0 – 9 report the type of Servo ICs 0 – 9, respectively, and bits 10 – 19 report the type of Servo ICs 0* to 9*, respectively. A bit value of 1 indicates a PMAC2-style DSPGATE1 IC; a bit value of 0 indicates a PMAC1-style DSPGATE IC (or no IC present if the corresponding bit of I4900 is 0).

MACRO IC Configuration

On power-up/reset, the Turbo PMAC CPU automatically tests for the presence and type of all possible MACRO ICs and reports the results in I4902 and I4903. I4902 is a collection of 16 independent bits, each reporting the presence of a MACRO IC at one of the 16 possible locations. A bit value of 1 indicates the IC is present; a bit value of 0 indicates the IC is absent.

I4903 is also a collection of 16 independent bits, each reporting the type of MACRO IC at one of the 16 possible locations. A bit value of 1 indicates a "DSPGATE2" IC; a bit value of 0 indicates a MACROGATE IC (or no IC present if the corresponding bit of I4900 is 0).

While it is possible for up to 16 MACRO ICs to be installed in a Turbo PMAC system, only 4 of these can be supported at any time by automatic firmware functions. I20 – I23 contain the base addresses of these 4 ICs. When the system is re-initialized, these variables are set to values for the 4 ICs found with the lowest base addresses.

DPRAM IC Configuration

On power-up/reset, the Turbo PMAC CPU automatically tests for the presence of all possible dual-ported RAM ICs and reports the results in I4904. I4904 is a collection of 8 independent bits, each reporting the presence of a DPRAM IC at one of the 8 possible locations. Only one of these ICs can be supported at any time by automatic firmware functions. I24 contains the base address of this IC.

CPU Section Configuration

On power-up/reset, the Turbo PMAC automatically tests for the configuration of its own CPU section and reports the results in I4908. I4908 is a 36-bit value reporting the CPU type, active memory size, DPRAM size, battery-backed RAM size, flash memory size, presence of auxiliary serial port, part number, and vendor ID.

UBUS Accessory Board Identification

The Turbo PMAC can report detailed information about accessory boards installed on the UBUS expansion port in UMAC Turbo systems. This information is reported in variable I4910 – I4965. Each is a 36-bit variable with the following contents:

- Vendor ID: 8 bits
- Options present: 10 bits
- Revision number: 4 bits
- Card ID (part number): 14 bits

Each variable can report one part or all parts of this information, depending on the setting of I39. If I39 is set to 5, the variable reports the base address of the accessory board instead.

I4910 – I4925 report this information for the 16 possible accessory boards with Servo ICs, such as the ACC-24E2, 24E2A, 24E2S, and 51E.

I4926 – I4941 report this information for the 16 possible accessory boards with MACRO ICs, such as the ACC-5E.

I4942 – I4949 report this information for the 8 possible accessory boards with DPRAM ICs, such as the ACC-54E USB/Ethernet interface.

I4950 – I4965 report this information for the 16 possible accessory boards with I/O ICs, such as the ACC-14E, 28E, 36E, 53E, and 59E. (The ACC-9E, 10E, 11E, and 12E I/O boards currently cannot provide this information.)

Evaluating the Turbo PMAC's Computational Load

Starting with firmware version V1.936, Turbo PMAC controllers offer facilities that permit you to easily calculate the computational loads you are putting on the processor. There are several key timer registers to use in calculating these loads. These registers are scaled so that one increment of the timer is two clock cycles of the DSP. So if the DSP were running at a clock frequency of exactly 80 MHz – a clock period of 12.5 nsec – one increment of the timer would be 25 nsec.

The DSP's clock frequency is multiplied up from the crystal clock frequency of 19.66 MHz, using the saved value of I52, according to the formula

$$DSPfrequency = \frac{19.66\,MHz}{2} * (I52 + 1)$$

In terms of period, the timer increment – 2 DSP cycles – can be calculated as:

$$TimerIncrement(n\,sec) = \frac{203.4}{I52 + 1}$$

Phase Interrupt Tasks

There are two key timer registers for evaluating the computational load of the phase-interrupt tasks such as commutation, current-loop closure, and ADC de-multiplexing. The first is a hardware timer in the DSP, at address X:$FFFF8C. This register holds the number of timer increments between the last two phase interrupts, establishing the period of the phase interrupt. This can be used to verify the phase period you think you have, and with other registers, computational duty cycles.

The second register, located at X:$000037, holds the number of timer increments from the beginning to the end of the phase-interrupt tasks for the last interrupt. When divided by the time between phase interrupts, this will give the duty cycle of the phase-interrupt tasks.

Servo Interrupt Tasks

Another timer register can be used to evaluate the computation load of the servo-interrupt tasks such as the conversion table, interpolation, position/velocity-loop closure, and data gathering. This register, located at Y:$000037, holds the number of timer increments elapsed from the beginning to the end of the servo-interrupt tasks for the last interrupt.

If this time is less than the time between phase interrupts (Y:$37 < X:$FFFF8C), then this is the actual time the servo tasks took. However, if this time is greater than one phase cycle (Y:$37 > X:$FFFF8C), then the servo tasks were interrupted (at least once) by phase tasks, and the time for the interrupting phase tasks must be subtracted out (see example below).

When the net time for the servo tasks is divided by the product of the phase-interrupt period and the number of phase-interrupts per servo-interrupt, the result is the duty cycle of the servo-interrupt tasks. Note that certain servo tasks, such as data gathering, foreground motor data reporting, and even servo-loop closure if Ixx60 > 0, do not have to be executed every servo cycle, so the duty cycle can vary.

Real-Time Interrupt Tasks

Two timer registers provide information on the loading of real-time interrupt (RTI) tasks such as PLC 0, PLCC0, and motion-program calculations. The first register, at X:$00000B, holds the number of timer increments from the beginning to the end of the RTI tasks for the last interrupt. The second register, at Y:$00000B, holds the largest number of timer increments from the beginning to the end of a set of RTI tasks since the last power-up/reset.

If these times are less than the time between phase interrupts (X/Y:$0B < X:$FFFF8C), then these are the actual times the RTI tasks took. However, if these times are greater than one phase cycle (X/Y:$0B > X:$FFFF8C), then the RTI tasks were interrupted (at least once) by phase tasks, and the time for the interrupting phase tasks must be subtracted out. Also, if the these times are greater than one servo cycle, then the RTI tasks were also interrupted by servo tasks (see example below).

Dividing the latest net time for the RTI tasks by the product of the phase interrupt period, the number of phase interrupts per servo interrupt, and the number of servo interrupts per RTI yields the duty cycle of the RTI tasks. The duty cycle for real-time interrupt tasks can vary widely within an application, so it is advisable to compute a running average to compute general loading.

Note:

In Turbo PMAC firmware versions V1.936 and older, these RTI timer registers are in units of servo cycles, which does not give very fine time resolution. In firmware versions V1.937 and newer, these registers have the same units as the phase and servo timer registers.)

Total Interrupt Tasks

The total duty cycle for Turbo PMAC interrupt tasks can be calculated by summing the duty cycles for the three types of interrupt tasks.

Sample Monitoring Program

The following sample code can be used to monitor the total interrupt-task duty cycle

```
M70->X:$FFFF8C,0,24           ; Time between phase interrupts
M71->X:$000037,0,24           ; Time for phase tasks
M72->Y:$000037,0,24           ; Time for servo tasks
M73->X:$00000B,0,24           ; Time for RTI tasks
P70=4                         ; 4 phase interrupts per servo interrupt
P76=16                        ; Length of filter for averaging duty cycle
OPEN PLC 17 CLEAR
P71=M71/M70                   ; Phase task duty cycle
P69=INT(M72/M70)              ; # of times phase interrupted servo
P72=(M72-P69*M71)/(M70*P70)   ; Servo task duty cycle
P68=INT(M73/M70)              ; # of times phase interrupted RTI
P67=INT(M73/(M70*P70))        ; # of times servo interrupted RTI
P73=(M73-P68*M71-P67*(M72-P69*M71))/(M70*P70*(I8+1))
                              ; RTI task duty cycle
P74=P71+P72+P73               ; Latest total foreground duty cycle
P75=P75*(P74-P75)/P76         ; Averaged total foreground duty cycle
CLOSE
```

Note:

In V1.936, set P73 to 0 and just calculate servo and phase duty cycle.

Background Cycle Time

There are two timer registers important to evaluate the time required to execute a background cycle. These registers are involved in the operation of the watchdog timer. The register at address Y:$000025 is set to the value of I40 at the end of every background cycle. (If I40 is 0, the register is set to 4095.) Until the next background cycle is completed, this register is decremented every servo cycle. The data gathering function is useful to establish how long background cycles take.

The register at X:$000025 contains the lowest value reached by Y:$000025 since the last power-up/reset of the Turbo PMAC. If this is close to 0, the Turbo PMAC has come close to tripping its watchdog timer, and background tasks such as PLC program execution, communications response, and safety checks have been slow.

Turbo PMAC Lookahead Function

Turbo PMAC can perform highly sophisticated lookahead calculations on programmed trajectories to ensure that the trajectories do not violate specified maximum quantities for the axes involved in the moves. This permits the user to write the motion program simply to describe the commanded path. Vector feedrate becomes a constraint instead of a command; programmed acceleration times are used only to define corner sizes and minimum move block times. Turbo PMAC will automatically control the speed along the path (but without changing the path) to ensure that axis limits are not violated.

Lookahead calculations are appropriate for any execution of a programmed path where throughput has been limited by the need to keep execution slow throughout the path because of the inability to anticipate the few sections where slow execution is required. The lookahead function's ability to anticipate these problem areas permits much faster execution through most of the path, dramatically increasing throughput.

Because of the nature of the lookahead calculations – trajectory calculations are done well in advance of the actual move execution, and moves are kept within machine limits by the automatic adjustment of move speeds and times – they are *not* appropriate for some applications. Any application requiring quick reaction to external conditions should not use lookahead. Also, also any application requiring precise synchronization to external motion, such as those using PMAC's "external time base" feature, should not use lookahead.

When the lookahead function is enabled, Turbo PMAC will scan ahead in the programmed trajectories, looking for potential violations of its position, velocity, and acceleration limits. If it sees a violation, it will then work backward through the pre-computed buffered trajectories, slowing down the parts of these trajectories necessary to keep the moves within limits. These calculations are completed before these sections of the trajectory are actually executed.

Turbo PMAC can perform these lookahead calculations on LINEAR and CIRCLE mode moves. The coordinate system must be put in segmentation mode (Isx13 > 0) to enable lookahead calculations, even if only LINEAR mode moves are used. (The coordinate system must be in segmentation mode anyway to execute CIRCLE mode moves or cutter radius compensation.) In segmentation mode, Turbo PMAC automatically splits the moves into small segments, which are executed as a series of smooth splines to re-create the programmed moves.

Turbo PMAC stores data on these segments in a specially defined lookahead buffer for the coordinate system. Each segment takes Isx13 milliseconds when it is put into the buffer, but this time can be extended if it or some other segment in the buffer violates a velocity or acceleration limit.

This technique permits Turbo PMAC to create deceleration slopes in the middle of programmed moves, at the boundaries of programmed move, or over multiple programmed moves, whichever is required to create the fastest possible move that does not violate constraints. All of this is done automatically and invisibly inside the Turbo PMAC; the part programmer and operator do not need to understand the workings of the algorithm.

If Turbo PMAC's inverse kinematic calculations are used, the conversion from "tip" coordinates to "joint" coordinates takes place before lookahead calculations, segment by segment for LINEAR and CIRCLE mode moves. Therefore, Turbo PMAC can execute the lookahead calculations in "joint space", motor by motor, even if the system has been programmed in tip coordinates.

Once the lookahead function has been set up, the lookahead function operates transparently to the programmer and the operator. No changes need to be made to a motion program to use the lookahead function, although the programmer may choose to make some changes to take advantage of the increased performance capabilities that lookahead provides.

Quick Instructions: Setting up Lookahead

The following list quickly explains the steps required for setting up and using the lookahead function on the Turbo PMAC. Greater detail and context are given in the subsequent section.

1. Assign all desired motors to the coordinate system with axis definition statements.
2. Set Ixx13 and Ixx14 positive and negative position limits, plus Ixx41 desired position-limit band, in counts for each motor in coordinate system. Set bit 15 of Ixx24 to 1 to enable desired position limits.
3. Set Ixx16 maximum velocity in counts/msec for each motor in coordinate system.
4. Set Ixx17 maximum acceleration in counts/msec^2 for each motor in coordinate system.
5. Set Isx13 segmentation time in msec for the coordinate system to minimum programmed move block time or 10 msec, whichever is less.
6. Compute maximum stopping time for each motor as Ixx16/Ixx17.
7. Select motor with longest stopping time.
8. Compute number of segments needed to look ahead as this "stopping time" divided by (2 * Isx13).
9. Multiply the "segments needed" by 4/3 (round up if necessary) and set the Isx20 lookahead length parameter to this value.
10. If the application involves high block rates, set the Isx87 default acceleration time to the minimum block time in msec; the Isx88 default S-curve time to 0.
11. If the application does not involve high block rates, set the Isx87 default acceleration time and the Isx88 default S-curve time parameters to values that give the desired blending corner size and shape at the programmed speeds.
12. Store these parameters to non-volatile memory with the **SAVE** command if you want them to be an automatic part of the machine state.
13. After each power-up/reset, send the card a **DEFINE LOOKAHEAD {# of segments},{# of outputs}** command for the coordinate system, where **{# of segments}** is equal to Isx20 plus any segments for which backup capability is desired, and **{# of outputs}** is at least equal to the number of synchronous M-variable assignments that may need to be buffered over the lookahead length.
14. Load your motion program into the Turbo PMAC. Nothing special needs to be done to the motion program. The motion program defines the path to be followed; the lookahead algorithm may reduce the speed along the path, but it will not change the path.
15. Run the motion program, and let the lookahead algorithm do its work!

Detailed Instructions: Setting up to use Lookahead

A few steps are required to calculate and set up the lookahead function. Typically, the calculations only have to be done once in the initial configuration of the machine. Once configured, the lookahead function operates automatically and invisibly.

Defining the Coordinate System: The lookahead function checks the programmed moves against all motors in the coordinate system. The first step is therefore to define the coordinate system by assigning motors to axes in the coordinate system with axis definition statements. This action is covered in the User's Guide under *Setting Up the Coordinate System*.

Lookahead Constraints: Turbo PMAC's lookahead algorithm forces the coordinate system to observe four constraints for each motor. These constraints are defined in I-variables for each motor representing maximum position extents, velocities, and accelerations. These I-variables must be set up properly in order for the lookahead algorithm to work properly.

Position Limits*:* Variables Ixx13 and Ixx14 for each Motor xx define the maximum positive and negative position values, respectively, that are permitted for the motor ("software overtravel limits"). These variables are defined in counts, and are referenced to the motor zero, or home, position (often called "machine zero"). Even if the origin of the axis for programming purposes has been offset (often called "program zero"), the physical position of these position limits does not change; they maintain their reference to the machine zero point. Turbo PMAC checks the *actual* position for each motor as the trajectory is being executed against these limits; if a limit is exceeded, the program is aborted and the motors are decelerated at the rate set by Ixx15.

Variable Ixx41 for each Motor xx defines the distance between the *actual* position limits explained above, and the *desired* position limit that can be checked at move calculation time, even in lookahead. That is, if the calculated desired move position is greater than (Ixx13 – Ixx41), or less than (Ixx14 + Ixx41), this will constitute a desired position limit violation. Desired position limits are only checked if bit 15 of Ixx24 is set to 1.

In this mode, if the lookahead algorithm, while scanning ahead in the programmed trajectory, determines that any motor in the coordinate system would exceed one of its desired position limits, it will suspend the program and force a stop right at that limit. It will then work backwards through the buffered trajectory segments to bring the motors to a stop along the path at that point in the minimum time that does not violate any motor's Ixx17 acceleration constraint.

Note:

If bit 14 of Ixx24 is also set to 1, the program does not stop at the limit. Instead, it will continue, with the offending motor saturating at the limit value.

When stopped on a desired position limit within lookahead, the program is only suspended, not aborted. The action is effectively equivalent to issuing a \ quick-stop command. It is possible to "retrace" the path coming into the limit, or even to resume forward execution after changing the limit value. An "abort" command must be issued before another program can be started.

Note:

If an actual position limit is also tripped during the deceleration to a stop at the desired position limit, the program is aborted, so retracing and resuming are not possible. For this reason, if the possibility of retracing and resuming is important, Ixx41 should be set to a large enough value so that the actual position limit is never tripped during a desired position limit stop.

This technique permits these software position limits to be placed just within the hard stops of the machine. Without the desired position limits, the software position limits cannot be detected until the actual trajectory actually passes the limit. This requires that these limits be placed far enough within the hard stops so that the motors have enough distance to stop *after* they pass the limits. (When a motor hits a software position limit without lookahead, the deceleration of motors is controlled by Ixx15, not Ixx17, and deceleration is not necessarily along the programmed path.)

Velocity Limits*:* Variable Ixx16 for each Motor xx defines the magnitude of the maximum velocity permitted for the motor. These variables are defined in the raw PMAC units of counts per millisecond, so a quick conversion must be calculated from the user units (e.g. millimeters per minute).

If the algorithm, while looking ahead in the programmed trajectory, determines that any motor in the coordinate system is being asked to violate its velocity limit, it will slow down the trajectory at that point just enough so that no limit is violated. It will then work backwards through the buffered trajectory segments to create a controlled deceleration along the path to this limited speed in the minimum time that does not violate any motor's Ixx17 acceleration constraint.

Note:

During the initial move-block calculations, before move data is sent to the lookahead function, a couple of factors can result in commanded velocities lower than what is programmed. First, if the vector feedrate commanded in the motion program with the **F** command exceeds the maximum feedrate parameter Isx85, then Isx85 is used instead. Second, if the move-block time, either specified directly with the TM command, or calculated as vector-distance divided by vector-feedrate, is less than the programmed acceleration time (the larger of TA or 2 * TS), the programmed acceleration time is used instead. This results in a speed less than what was programmed. The lookahead function can further slow these moves, but it cannot speed them up.

Acceleration Limits*:* Variable Ixx17 for each Motor xx defines the magnitude of the maximum acceleration permitted for the motor. These variables are defined in the raw PMAC units of counts per (millisecond-squared), so a quick conversion must be calculated from the user units (e.g. in/sec^2, or g's).

If the algorithm, while looking ahead in the programmed trajectory, determines that any motor in the coordinate system is being asked to violate its acceleration limit, it will slow down the trajectory at that point just enough so that no limit is violated. It will then work backwards through the buffered trajectory segments to create a controlled deceleration along the path to this limited speed in the minimum time that does not violate any motor's Ixx17 acceleration constraint.

Calculating the Segmentation Time: Turbo PMAC's lookahead function operates on intermediate motion "segments" calculated from the programmed trajectory. An intermediate point for each motor is computed once per segment from the programmed path, and then a fine interpolation using a cubic spline to join these segments is executed at the servo update rate. The user settable "segmentation time" is therefore an important parameter for optimization of the lookahead function.

Variable Isx13 for each Coordinate System 'x' defines the time for each intermediate segment in the programmed trajectory, in milliseconds, before it is possibly extended by the lookahead function. Isx13 is an integer value; if a non-integer value is sent, Turbo PMAC will round to the next integer. If Isx13 is set to 0, the coordinate system is not in "segmentation mode"; no intermediate segments are calculated, and the lookahead function cannot be enabled.

Several issues must be addressed in setting the Isx13 segmentation time. These include its relationship to the maximum block rate, the small interpolation errors it introduces, and its effect on the calculation load of the Turbo PMAC. Each of these is addressed in turn, below.

Block rate relationship: In most applications, the Isx13 segmentation time will be set so that it is less than or equal to the minimum block (programmed move) time. Put another way, the segmentation *rate* defined by Isx13 is usually set greater than or equal to the maximum block *rate*. For example, if a maximum block rate of 500 blocks per second is desired, the minimum block time is 2 milliseconds, and Isx13 is set to a value no greater than 2.

This relationship holds because blocks of a smaller time than the segmentation time are skipped over as Turbo PMAC looks for the next segment point. While this does not cause any errors, there is no real point in putting these programmed points in the motion program if the controller is going to skip over them. However, some people "inherit" old motion programs with points closer together than is actually required; these users may have reason to set their segmentation time larger than their minimum block time.

Note:

The programmed acceleration time sets a limit on the maximum block rate. The move time for a programmed block, even before lookahead, is not permitted to be less than the programmed acceleration time. The programmed acceleration time is the larger of the TA time (TA = Isx87 by default) and twice the TS time (TS = Isx88 by default). In high-block-rate lookahead applications, the TA time is typically set equal to the minimum desired block time, and the TS time is typically set to (because it squares up corners).

Interpolation errors: The cubic-spline interpolation technique that Turbo PMAC uses to connect the intermediate segment points is very accurate, but it does create small errors. These errors can be calculated as:

$$Error = \frac{V^2 T^2}{6R}$$

where V is the vector velocity along the path, T is the segmentation time (watch the units!), and R is the local radius of curvature of the path. For example, if the speed is 100 mm/sec (~4 in/sec), the segmentation time is 0.01 sec (Isx13 = 10 msec), and the minimum radius at this speed is 50 mm (~2 in), then the worst-case interpolation error can be calculated as:

$$Error = \frac{100^2 \frac{mm^2}{sec^2} * 0.01^2 \ sec^2}{6 * 50mm} = 0.003mm = 3\mu m$$

If the programmed path itself introduces path error, such as the "chordal error" of linear interpolation, this must be added to the error budget as well. In addition, if the servo-loop execution adds servo errors, these must also be included.

Calculation Implications*:* While smaller Isx13 segmentation times permit higher real maximum block rates and permit more accurate interpolation, they increase the Turbo PMAC computational requirements, particularly when lookahead is active. The following table shows the result of benchmarking tests on the Turbo PMAC that shows the minimum segmentation times that can be used for a given number of axes executing lookahead calculations.

Number of Axes	Maximum Block Rate (blocks/sec)	Minimum Segmentation Time (msec)
2	2000	1 @ 200%
3	1000	1
4	500	2
5	500	2
6	500	2
8	333	3
12	250	4
16	200	5

Notes:
1. Tests performed on 80 MHz Turbo PMAC
2. Tests performed at default 2.25 kHz servo update rate
3. Tests performed with no PMAC motor commutation or current-loop closure
4. Higher block rates can be done, but segmentation will smooth out features

Note:

Subject to these constraints, the length of the lookahead is subject only to memory limitations in the Turbo PMAC.

In general, the Isx13 segmentation time is set to the largest value that meets user requirements in each of the above three concerns. However, it is seldom set larger than 10 msec.

Calculating the Required Lookahead Length: In order for the coordinate system to reach maximum performance, it must be looking ahead for the time and distance required for each motor to come to a full stop from maximum speed. Because the lookahead buffer stores motion segments, this lookahead length must be expressed in segments.

To calculate this value, first compute the worst-case time required to stop for each motor in the coordinate system. This value can be obtained by dividing the maximum motor velocity by the maximum motor acceleration. In terms of Turbo PMAC parameters:

$$StopTime(msec) = \frac{Ixx16}{Ixx17}$$

Now take the motor with the longest stop time, and divide this time by 2 (because the segments will come in at maximum speed, which takes half the time of ramping down to zero speed). Next, convert this value to a number of segments by dividing by the coordinate system segmentation time:

$$LookaheadLength(segs) = \frac{StopTime(msec)}{2 * Isx13(msecs/seg)} = \frac{Ixx16}{2 * Ixx17 * Isx13}$$

This is the number of segments in the lookahead buffer that must always be properly computed ahead of time. Because the Turbo PMAC does not fully recalculate the lookahead buffer every segment, it must actually look further ahead than this number of required segments

Lookahead Length Parameter: Variable Isx20 for the coordinate system tells the algorithm how many segments ahead in the program to look. This value is a function of the number of segments that must always be correct in the lookahead buffer (SegmentsNeeded). The formula is:

$$Isx20 = \frac{4}{3} * SegmentsNeeded$$

Setting Isx20 to a value larger than needed does not increase the computational load (although it does increase the time of heaviest computational load while the buffer is filling). However, it does require more memory storage, and it does increase the delay in having the program react to any external conditions.
Setting Isx20 to a value smaller than needed does not cause the limits to be violated. However, it may cause Turbo PMAC to limit speeds more severely than the Ixx16 limits require in order to ensure that acceleration limits are not violated. In addition, a saw-tooth velocity profile may be observed.

Note:

Preliminary versions of the Turbo PMAC firmware had three additional parameters controlling the dynamics of the lookahead operation: Isx21, Isx22, and Isx23. In the current versions of the firmware, these values are fixed at 3, 6, and 7, respectively, and the variables have been removed. Isx21 now permits direct control of the lookahead state of operation (see below).

Defining the lookahead buffer: In order to use the lookahead function in a Turbo PMAC coordinate system, a lookahead buffer must be defined for that coordinate system, reserving memory for the buffer. This is done with the on-line coordinate-system-specific **DEFINE LOOKAHEAD** command. Because lookahead buffers are not retained through a power down or reset, this command must be issued after every power-up or board reset.

There are two values associated with the **DEFINE LOOKAHEAD** command. The first determines the number of motion segments for each motor in the coordinate system that can be stored in the lookahead buffer. At a minimum, this must be set equal to Isx20.

If this value is set greater than Isx20, the lookahead buffer stores "historical" data. This data can be used to reverse through the already executed trajectory. If reversal is desired, the buffer should be sized to store enough "back segments" to cover the desired backup distance. There is no penalty for reserving more memory for these synchronous M-variable assignments than is needed, other than the loss of this memory for other uses.

The room reserved for the segment data in the lookahead buffer is dependent on the number of motors assigned to the coordinate system at the time of the **DEFINE LOOKAHEAD** command. If the number of motors assigned to the coordinate system then changes, the organization of the lookahead buffer will be wrong, and the program will abort with a run-time error on the next move after the coordinate system is changed.

If the coordinate system must be changed during an application that uses lookahead, the lookahead buffer must first be deleted, then defined again after the change. The following motion program code shows how this could be done:

```
DWELL 10                                ; Stop lookahead execution
CMD "&1 DELETE LOOKAHEAD"               ; Delete buffer
CMD "&1 #4->100C"                       ; Assign new motor to C. S. 1
CMD "&1 DEFINE LOOKAHEAD 1000,100"      ; Redefine buffer
DWELL 10                                ; Make sure commands execute
```

The second value associated with the **DEFINE LOOKAHEAD** command determines the number of "synchronous M-variable assignments" (e.g. **M1==1**) for the coordinate system that can be stored in the lookahead buffer. Synchronous M-variable assignments in the motion program delay the actual assignment of the value to the M-variable until the start of actual execution of the next move in the motion program. Therefore, these actions must be held in a buffer pending execution.

This size of the buffer for these assignments must be at least as great as the largest number of assignments expected during the time for lookahead. There is no penalty for reserving more memory for these synchronous M-variable assignments than is needed, other than the loss of this memory for other uses.

Note:

The buffer reserved in this manner for synchronous M-variables under lookahead is distinct from the fixed-size buffer used for synchronous M-variables without lookahead.

For example, the command **&1 DEFINE LOOKAHEAD 500,50** creates a lookahead buffer for Coordinate System 1 that can store 500 segments for each motor assigned to the coordinate system at that time, plus 50 synchronous M-variable assignments.

Running a Program with Lookahead

The lookahead function is automatically active when a motion program is run in a coordinate system provided the following conditions are true:

1. The coordinate system is in segmentation mode (Isx13 > 0).
2. The coordinate system is told to look ahead (Isx20 > 0).
3. A lookahead buffer has been defined for the coordinate system since the last board power-up/reset, or if the lookahead buffer structure has been saved with I14 = 1.
4. The motion program is executing LINEAR or CIRCLE-mode moves.

The lookahead function is active under these conditions even when Turbo PMAC is performing inverse-kinematic calculations every segment to convert tip positions to joint positions. This permits the user to write a motion program in convenient tip coordinates, yet still automatically observe all joint-motor limits. This is particularly important if the tip path passes near a singularity, requesting very high joint velocities and accelerations.

Other move modes – RAPID, SPLINE, and PVT – can be executed with the lookahead buffer defined, but the lookahead function is not active when these moves are being executed.

Absolutely no change is required to the motion program to utilize the lookahead function.

It is important to realize the implications of the lookahead function on several aspects of the motion program. Each of these areas is covered below.

Vector Feedrate: Without lookahead, the vector feedrate value (**Fxxx**) is a command for each programmed move block in the motion program. That is, each move is calculated so that it is traversed at the programmed vector feedrate (speed). With lookahead active, the feedrate value is only a constraint. The move will never be executed at a higher speed, but it may be executed at slower speeds during some or all of the move as necessary to meet the motor constraints.

If the move is programmed by move time instead of feedrate, the programmed move time becomes a (minimum) constraint; the move will never be executed in less time, but it may be executed in greater time.

Acceleration Time: The programmed acceleration times – Isx87 and Isx88 by default, or **TA** and **TS** in the motion program, are the times before lookahead. The lookahead function will control the actual acceleration times that are executed, but the programmed acceleration times are still important for two reasons.

First, the programmed acceleration time, which is the larger of TA or 2*TS, is the minimum move-block time. If PMAC initially computes a smaller move time, typically as (vector-distance divided by vector-feedrate), it will increase the time to be equal to the acceleration time, slowing the move. This check occurs even before lookahead (which can only slow the move further), and it is an important protection against computational overload. The acceleration time must be set low enough not to limit valid moves.

Note:

The acceleration time may be set to 0; in this case, Turbo PMAC sets a minimum move time of 0.5 milliseconds.

Second, as longer moves are blended together, the programmed acceleration time and feedrate control the corner size for the blending. The blended corner begins a distance of F*Ta/2 before the programmed corner point, where F is the programmed feedrate, and Ta is either the specified acceleration time (TA) or two times the specified S-curve time (2*TS), which ever is greater. The blended corner ends an equal distance past the programmed corner point.

If the lookahead algorithm determines that the blended corner violates the acceleration limit on one or more motors, it will automatically slow the speed of the path in the corner. This will make the time for the blended corner bigger than what was specified in the program. The lookahead will also automatically create a controlled deceleration ramp going into the blended corner, and a controlled acceleration ramp coming out of the corner. In this manner, the size of the rounding at a corner can be kept small without violating acceleration constraints and without limiting speeds far away from the corners.

In general, the acceleration time should be set as large as it can be without either making the minimum move time too large, or the corners too large. In high block-rate applications, the TA time is generally set to the minimum block time, and the TS time is set to 0. In low block-rate applications, the TA and TS times are generally set to get the desired corner size and shape.

Trajectory Filter: In high block-rate applications, rough motion can result from "quantization errors" in the programmed path. This can produce machine vibration, audible noise, and high surface roughness on cut parts. These errors can stem from the limited numerical resolution of the programmed points, from measuring errors if the programmed points were scanned, or both.

This behavior can be compensated with a simple filtering of the interpolated motor trajectory, using variable Ixx40 for each Motor xx. If Ixx40 is set to a value greater than zero, the desired trajectory is passed through a simple first-order digital low-pass filter for smoothing purposes. (If Ixx40 is set to 0.0, this filtering is disabled.) The higher the value of Ixx40, the greater the time constant of the filter.

The equation for the time constant T_f of the filter as a function of the servo update time T_s and Ixx40 is:

$$T_f = \frac{Ixx40 * T_S}{1 - Ixx40}$$

Generally, time constants of a few milliseconds are selected when the filter is used. Note that only the desired trajectory is filtered, so servo-loop stability is not affected. However, the filtering does introduce a very slight path error (only noticeable for very large time constants) that can be quantified according to the following equation:

$$Error = \frac{V^2 T_f^2}{2R}$$

where V is the velocity, T_f is the filter time constant, and R is the local radius of curvature of the path. For example, with a velocity of 5000 mm/min (~200 in/min), a filter time constant of 2 msec, and a local radius of 100 mm (~4 in), the path error would be:

$$Error = \frac{5000^2 \frac{mm^2}{min^2} * \frac{min^2}{3600\ sec^2} * 0.002^2\ sec^2}{2 * 100^2\ mm^2} = 1.39 \times 10^{-6}\ mm = 1.39nm$$

Feedrate Override: All lookahead calculations are performed assuming a feedrate override value of 100%. If the feedrate override value, from whatever source, changes from 100%, the velocity and acceleration calculations will be incorrect. True velocity values vary linearly with the override value; true acceleration values vary with the square of the override value.

For example, at 200% override, velocity values are twice the programmed values (and could exceed the limit values by a factor of 2), and acceleration values are 4 times the programmed values (and could exceed the limit values by a factor of 4).

Because the feedrate override can be changed at any time with immediate effects, the lookahead function cannot anticipate what the override will be when the move will actually be executed. Therefore, it cannot plan for any changes in the override, so it assumes operation at 100%.

The basic idea of lookahead is to remove the override function from the instantaneous judgment of the operator, and instead use the mathematical calculations of the controller, which effectively act as an override, to ensure proper and optimal execution of the path.

Computational Capabilities: The lookahead calculations can put significant real-time calculation loads on the Turbo PMAC processor. If the processor fails to keep up with these real-time requirements, execution the program will fail with a run-time error, and motion will be aborted. There is also a slight possibility of a watchdog timer trip if the processor is never released from the foreground lookahead calculations for background tasks.

It is important that the application be evaluated to ensure that the lookahead calculations can be properly performed under the worst-case conditions. The period when the most intensive calculations are being performed is at the beginning of a move sequence, when Turbo PMAC is dynamically filling the buffer to get ahead the specified distance.

A good worst-casetest is to run a motion program with programmed moves at the maximum move-block rate right at the beginning of a blended sequence. Make sure that the Turbo PMAC can get through this combination of high block-rate execution and dynamic filling of the lookahead buffer. To establish a margin of safety, increase the override value above 100% to see what extra capability exists. A 20% margin (proper execution at 120%) is strongly recommended.

Stopping while in Lookahead

If the user desires to stop axis motion while in lookahead mode, he must carefully consider how the stopping is to be done. It is important to realize what point in the chain of execution is being halted with the stopping command. Different stopping commands have different effects, and different uses.

Quick Stop: The \ quick-stop command causes Turbo PMAC to immediately calculate and execute the quickest stop within the lookahead buffer that does not exceed Ixx17 acceleration limits for motors in the coordinate system. Motion continues along the programmed path to a controlled stop, which is not necessarily at a programmed point (and probably will not be). This command is the effective equivalent of a feed hold within lookahead (even though the internal mechanism is quite different), and it should be the command issued when an operator presses a Hold button during lookahead. Outside of lookahead, this command causes an actual feed hold, as if the **H** command had been given.

The \ command is the best command to use to stop interactively within lookahead operation with the intention of resuming operation. Any synchronous M-variable assignments set to happen within the deceleration will execute.

Motors may be jogged away from this stop point, if desired. In addition, motion can be reversed along the path with the **<** command (see Reversal below).

Normal programmed motion can subsequently be resumed with the **>** resume-forward, **R** run, or **S** single-step command, provided all motors are commanded to be at the same position at which they originally stopped with the **/** command. If any motors have been jogged away from this point, they must first be returned with the **J=** command. Acceleration limits are observed during the ramp up from a stop here. The **>** resume command puts the coordinate system in either continuous run mode, or single-step mode, whichever mode it was in before the quick-stop.

End-Block Stop: The **/** end block command will stop motion at the end point of the move currently being added to the lookahead buffer, even if the next move has already been calculated. Motion segments up to the end of this move are still added to the lookahead buffer, and all segments and synchronous M-variable assignments in the lookahead buffer are completed.

Motion will come to a controlled stop at the end of the latest move block being added to the lookahead buffer without violating constraints. However, there can be a significant delay – over (Isx20*Isx13) msec if the lookahead buffer is full – from the time the **/** command is given and the time the axes stop.

Motors may be jogged away from this stop point, if desired. Motion can subsequently be resumed with the **R** or **S** command, provided all motors are commanded to be at the same position at which they originally stopped with the **/** command. If any motors have been jogged away from this stopped point, they must first be returned with the **J=** command.

Quit/Step: The **Q** quit command simply tells the motion program not to calculate any further motion program blocks. (The **S** single-step command will do the same thing if given while the program is running.) Motion segments up to the end of the latest calculated motion program move block are still added to the lookahead buffer, and all segments and synchronous M-variable assignments in the lookahead buffer are completed.

Motion will come to a controlled stop at the end of the latest calculated move block without violating constraints. However, there can be a significant delay – over (Isx20*Isx13) msec if the lookahead buffer is full – from the time the **Q** or **S** command is given and the time the axes stop.

Motors may be jogged away from this stop point, if desired. Motion can subsequently be resumed with the **R** or **S** command. Motors do not have to be at the same position at which they were originally stopped with the **Q** or **S** command. However, if it is desired to return them to this position, the **J=** command should be used.

Feed Hold: The **H** feed hold command brings the feedrate override value to zero, starting immediately, and ramping down at a rate controlled by coordinate system variable Isx95. Motion continues along the programmed path to a controlled stop, which is not necessarily at a programmed point (and probably will not be). Acceleration limits are not necessarily observed during the ramp down to a stop. Any synchronous M-variable assignments set to happen within the deceleration will execute.

Motors may be jogged away from this stop point, if desired. Programmed motion can subsequently be resumed with the **R** or **S** command, provided all motors are commanded to be at the same position at which they originally stopped with the **H** command. If any motors have been jogged away from this stopped point, they must first be returned with the **J=** command. Acceleration limits are not necessarily observed during the ramp up from a stop here.

Abort: The **A** abort command breaks into the executing trajectory immediately, and brings all motors in the coordinate system to a controlled stop, each at its own deceleration rate as set by Ixx15 for the motor. The stop is not necessarily at a programmed point (and probably will not be), and it is not necessarily even along the programmed path (and probably will not be).

Segments and synchronous M-variable assignments already in the lookahead buffer are discarded; they cannot be recovered. Although the program could be resumed with an **R** or **S** command, execution would miss all of the discarded segments from the lookahead buffer. Special recovery algorithms would be required to resume operation, so the abort command is not recommended except for stopping quickly under error conditions.

Kill All: The **<CTRL-K>** "kill-all" command breaks into the executing trajectory immediately, and disables all motors on the Turbo PMAC by opening the servo loops, forcing zero-value command outputs, and disabling the amplifiers. Motors will coast to a stop in the absence of brakes or regeneration circuits.

Segments and synchronous M-variable assignments already in the lookahead buffer are discarded; they cannot be recovered. Although the program could be resumed after re-enabling the servo loops with an **R** or **S** command, execution would miss all of the discarded segments from the lookahead buffer. Special recovery algorithms would be required to resume operation, so the "kill-all" command is not recommended except for emergency conditions.

Note:

The motor-specific **K** "kill" command is not permitted when the motor is in a coordinate system that is executing a motion program. The program must first be halted, usually with an **A** "abort" command.

Reversal while in Lookahead

If the lookahead buffer has been sized larger than what is required simply for the actual lookahead, it will contain historical data that can be used for reversal along the programmed path. This capability gives the system a retrace capability, allowing it easily to go backwards along the already executed path. The key command to be used in reversal is the **<** backup command, which causes the coordinate system to start execution in the reverse direction through the segments in the lookahead buffer.

Back-up Command: If the < command is given while the coordinate system is in normal forward execution in the lookahead buffer, Turbo PMAC will generate a \ quick-stop command internally to halt the forward execution, then start reverse execution.

The < command can also be given after execution of the program has been halted with a \ quick-stop command. It cannot be given after stopping with an / end-of-block, Q quit, S single-step, A abort or <CTRL-K> kill-all command, or an automatic error termination.

Reverse Execution: Execution in the reverse direction will observe the position, velocity, and acceleration limits, just as in the forward direction. Note, that if Isx20 is set to 1, limits are not observed in either the forward or reverse direction. In this mode, the lookahead buffer is simply used to buffer points to enable reversal, without the computational overhead of actual lookahead calculations. This mode is appropriate for EDM applications, which require quick reversal, but not careful acceleration limiting.

If not stopped by another command, reverse execution will continue until it reaches the beginning (oldest stored point) of the lookahead buffer. It will automatically stop at this point with a controlled deceleration within the acceleration constraints. The oldest stored point in the lookahead buffer will never be from before the first point in the current continuous blended motion sequence. This means that you cannot reverse into, past, or through any of the following:

- A DWELL point
- A RAPID, SPLINE, or PVT-mode move
- A homing-search move
- A point where the program was stopped with a **/**, **Q**, or **S** command.
- A point where blending was stopped for any other reason (e.g. Isx92=1, double jump-back)

Remember that a **DELAY** command in a motion program does not disable blending, so it is possible to reverse execution through a **DELAY** point. If a stop at a point is desired during execution of the program, but the ability to reverse through the point is required, **DELAY** should be used instead of **DWELL**.

Stopping Reverse Execution: The reverse execution can be halted before this point with the \ quick-stop command. Reverse execution can then be resumed with another < back-up command; forward execution can be re-started with a > resume, R run, or S single-step command. The > resume command puts the coordinate system in either continuous run mode, or single-step mode, whichever mode it was in before the back up.

No synchronous M-variable assignments are executed either during a reversal or during the forward execution over the reversed part of the path.

Forward execution over the reversed part of the path will blend seamlessly into previously unexecuted parts of the path. At this point, standard execution of the lookahead buffer will resume, with new points being added to the end of the lookahead buffer, and execution of buffered synchronous M-variable assignments starting again.

Quick Reversal from Within Turbo PMAC: If it is desired to reverse very quickly from within PMAC, as for quick retracts in EDM applications, it is best to bypass the command interpreter, which acts in background. This can be done by writing directly to a lookahead-control I-variable from the PMAC program.

Variable Isx21 for each coordinate system contains the control bits for the state of lookahead execution. By setting the value of this I-variable directly from a PLC program, the overhead and delay of the command interpreter can be avoided and slightly faster reaction obtained. There are three values of use:

- Setting Isx21 to 4 is the equivalent of issuing the **** quick-stop command
- Setting Isx21 to 7 is the equivalent of issuing the **<** back-up command
- Setting Isx21 to 6 is the equivalent of resuming forward motion with the **>** resume-forward command.

If you are monitoring Isx21 at other times, you will see that the 4's bit is cleared after the command has been processed. Therefore, you will see the following values:

- Isx21 = 0 when stopped with a quick-stop command
- Isx21 = 3 when running reversed in lookahead
- Isx21 = 2 when running forward in lookahead

Kinematic Calculations

Turbo PMAC provides structures to enable the user to easily implement and execute complex kinematic calculations. Kinematic calculations are required when there is a non-linear mathematical relationship between the tool-tip coordinates and the matching positions of the actuators (joints) of the mechanism, typical in non-Cartesian geometries. They are most commonly used in robotic applications, but can be used with other types of actuators that are not considered robotic. For example, in 4-axis or 5-axis machine tools with one or two rotary axes, it is desirable to program the cutter-tip path and let the controller compute the necessary motor positions.

This capability permits the motion for the machine to be programmed in the natural coordinates of the tool-tip, usually Cartesian coordinates, whatever the underlying geometry of the machine. The kinematic routines are embedded in the controller by the integrator, and operate invisibly to the people programming paths and the machine operators. These routines can be unchanging for the machines, but with parameterization and/or logic, they can adapt to normal changes such as tool lengths and different end-effectors.

In Turbo PMAC terminology, the tool-tip coordinates are for axes, which are specified by letter, and have user-specified engineering units. The joint coordinates are for motors, which are specified by numbers, and have the raw units of counts.

> ***Note:***
>
> PMAC's standard axis-definition statements handle linear mathematical relationships between joint motors and tool-tip axes. This section pertains to the more difficult case of the non-linear relationships.)

The forward-kinematic calculations use the joint positions as input, and convert them to tool-tip coordinates. These calculations are required at the beginning of a sequence of moves programmed in tool-tip coordinates to establish the starting coordinates for the first programmed move. The same type of calculations can also be used to report the actual position of the actuator in tool-tip coordinates, converting from the sensor positions on the joints. (The Turbo PMAC forward-kinematic program buffer does not support this position-reporting functionality, but functionally identical calculations can be used in a PLC program for this purpose.)

The inverse-kinematic calculations use the tool-tip positions as input, and convert them to joint coordinates. These calculations are required for the end-point of every move that is programmed in tool-tip coordinates, and if the path to the end-point is important, they must be done at periodic intervals during the move as well.

Note:

Formal robotic analysis makes a distinction between joint position, and the actuator positions required for that joint position. While the two positions are usually the same, there are cases, such as when two motors drive a joint differentially, where there is an important difference. If your system has a distinction between joint and actuator positions, your kinematic calculations must include this distinction, to go all the way between actuator positions and tool-tip positions, with joint positions as an intermediate step. This documentation will just refer to joint positions, although this could technically refer to actuator positions in some applications.

Creating the Kinematic Program Buffers

Turbo PMAC implements the execution of kinematic calculations through special forward-kinematic and inverse-kinematic program buffers. Each coordinate system can have one of each of these program buffers, and the algorithms in them can be executed automatically at the required times, called as subroutines from the motion program.

Creating the Forward-Kinematic Program

The on-line **OPEN FORWARD** command opens the forward-kinematic buffer for the addressed coordinate system for entry. The on-line **CLEAR** command erases any existing contents of that buffer. Subsequently, any program command sent to Turbo PMAC that is legal for a PLC program (except **ADDRESS**, **CMDx**, and **SENDx**) will be entered into the open buffer. The on-line **CLOSE** command stops entry into the buffer.

Before any execution of the forward-kinematic program, Turbo PMAC will place the present commanded motor positions for each Motor xx in the coordinate system into global variable Pxx. These are floating-point values, with units of counts. The program can then use these variables as the "inputs" to the calculations.

After any execution of the forward-kinematic program, Turbo PMAC will take the values in Q1 – Q9 for the coordinate system in the user's engineering units, and copy these into the 9 axis target position registers for the coordinate system. The following table shows the axis whose position each variable affects and the suggested M-variable number for each of these registers (listed for debugging purposes).

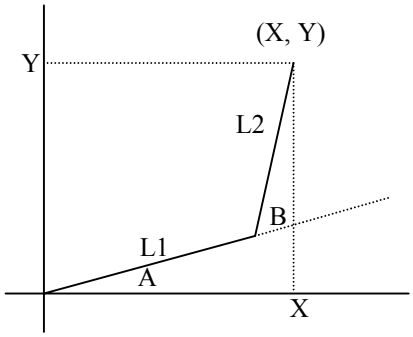
Axis-Position Q-Variable	Axis Letter	Target Register Suggested M-Variable	Axis-Position Q-Variable	Axis Letter	Target Register Suggested M-Variable	Axis-Position Q-Variable	Axis Letter	Target Register Suggested M-Variable
Q1	A	Msx41	Q4	U	Msx44	Q7	X	Msx47
Q2	B	Msx42	Q5	V	Msx45	Q8	Y	Msx48
Q3	C	Msx43	Q6	W	Msx46	Q9	Z	Msx49

The basic purpose of the forward-kinematic program, then, is to take the joint-position values found in P1 – P32 for the motors used in the coordinate system, compute the matching tip-coordinate values, and place them in variables in the Q1 – Q9 range.

Reserved Variables

If kinematic calculations are used in a system, the global variables P1 – P32 and the coordinate-system variables Q1 – Q10 should not be used for any other purposes, because Turbo PMAC will automatically write to these variables in executing the kinematic routines. (Q10 is used to distinguish between inverse-kinematic calculations that involve velocity calculations and those that do not, as explained below.) If inverse-kinematic calculations involving PVT-mode moves are used, additionally the global variables P101-P132 and the coordinate-system variables Q11 – Q19 should not be used for any other purposes, because Turbo PMAC will automatically write to these variables in executing the velocity portions of the inverse-kinematic routines.

Example:



Take the example of a 2-axis shoulder-elbow robot, with an upper-arm length (L_1) of 400mm, and a lower-arm length (L_2) of 300mm. Both the shoulder joint (A) and the elbow joint (B) have resolutions of 1000 counts per degree. When both joints are at their zero-degree positions, the two links are both extended along the X-axis. The forward-kinematic equations are:

$$X = L_1 \cos(A) + L_2 \cos(A+B)$$
$$Y = L_1 \sin(A) + L_2 \sin(A+B)$$

To implement these equations in a Turbo PMAC forward-kinematic program for Coordinate System 1 that converts the shoulder angle in Motor 1 and the elbow angle in Motor 2 to the X and Y tip coordinates in millimeters, the following setup and program could be used:

; Setup for program

```
I15=1                          ; Trig calculations in degrees
&1                             ; Address CS 1
Q91=400                        ; L1
Q92=300                        ; L2
Q93=1000                       ; Counts per degree for A and B
; Forward-kinematic program buffer for repeated execution
&1 OPEN FORWARD                ; Forward kinematics for CS 1
CLEAR                          ; Erase existing contents
Q7=Q91*COS(P1/Q93)+Q92*COS((P1+P2)/Q93)   ; X position
Q8=Q91*SIN(P1/Q93)+Q92*SIN((P1+P2)/Q93)   ; Y position
CLOSE
```

The forward-kinematic program must calculate the axis positions for all of the axes in the coordinate system, whether or not all of the motor positions are calculated in the inverse-kinematic program (see below). For instance, if this arm had a vertical axis at the tip with a normal axis definition statement in C.S. 1 of **#3->100Z** (100 counts per millimeter – a linear relationship between motor and axis), the above program would still need to perform the forward-kinematic calculation for this motor/axis with a line such as **Q9=P3/100**.

Iterative Solutions

Some systems, particularly parallel-link mechanisms such as Stewart platforms (hexapods), do not have reasonable closed-form solutions for the forward-kinematic equations, and require iterative numerical solutions. These cases are typically handled by a looping **WHILE … ENDWHILE** construct in the forward-kinematic program. The user should not permit indefinite looping – if the solution does not converge in the expected number of cycles, the program should be stopped (see the inverse-kinematic equations, below, for examples of how to stop the program).

In this case, it is best to leave the I11 program-calculation delay variable at its default value of 0, so the calculations can take as long as needed. If I11 is greater than 0, and the forward-kinematic calculations plus the first move calculations do not finish within I11 msec, Turbo PMAC will stop the program with a run-time error. In any case, if the forward-kinematic calculations take more than about 25 msec, it is possible to trip the watchdog timer.

Position Reporting

Another use of forward-kinematic calculations is for the position reporting function, reading actual joint positions at any time, and converting them to tip positions for reporting. The forward-kinematic program buffer on Turbo PMAC does not support this function. (Using the program for both initial-position calculations and position reporting could lead to potential overlapping use and register conflicts.)

If the application requires the Turbo PMAC to do forward-kinematic calculations for position reporting as well as for establishing initial tip position, the position-reporting calculations should be put into a PLC program. The following PLC program could be used for the position-reporting function of the example "shoulder-elbow" robot:

```
; M-variable definitions for actual position registers
M162->D:$8B                          ; Motor 1 actual position
M262->D:$10B                         ; Motor 2 actual position

; Forward-kinematic PLC program buffer for position reporting
OPEN PLC 10                          ; Forward kinematics for CS 1
CLEAR                                ; Erase existing contents
P51=M162/(I108*32*Q93)               ; Actual A position (deg)
P52=M262/(I208*32*Q93)               ; Actual B position (deg)
Q27=Q91*COS(P51)+Q92*COS(P51+P52)    ; Actual X position
Q28=Q91*SIN(P51)+Q92*SIN(P51+P52)    ; Actual Y position
CLOSE
```

Creating the Inverse-Kinematic Program

The on-line **OPEN INVERSE** command opens the inverse-kinematic buffer for the addressed coordinate system for entry. The on-line **CLEAR** command erases any existing contents of that buffer. Subsequently, any program command sent to Turbo PMAC that is legal for a PLC program (except **ADDRESS**, **CMDx**, and **SENDx**) will be entered into the open buffer. The on-line **CLOSE** command stops entry into the buffer.

Before any execution of the inverse-kinematic program, Turbo PMAC will place the present axis target positions for each axis in the coordinate system into variables in the range Q1 – Q9 for the coordinate system. These are floating-point values, in engineering units. The program can then use these variables as the "inputs" to the calculations.

The following table shows the variable for each axis, and the suggested M-variable for each source register (listed for debugging purposes).

Axis-Position Q-Variable	Axis Letter	Source Register Suggested M-Variable	Axis-Position Q-Variable	Axis Letter	Source Register Suggested M-Variable	Axis-Position Q-Variable	Axis Letter	Source Register Suggested M-Variable
Q1	A	Msx41	Q4	U	Msx44	Q7	X	Msx47
Q2	B	Msx42	Q5	V	Msx45	Q8	Y	Msx48
Q3	C	Msx43	Q6	W	Msx46	Q9	Z	Msx49

After any execution of the inverse-kinematic program, Turbo PMAC will read the values in those variables Pxx (P1 – P32) that correspond to Motors xx in the coordinate system with axis-definition statements of **#xx->I**. These are floating-point values, and Turbo PMAC expects to find them in the raw units of "counts". Turbo PMAC will automatically copy these values into the target position registers for these motors (suggested M-variable Mxx63), where they are used for the fine interpolation of these motors.

There can be other motors in the coordinate system that are not defined as inverse-kinematic axes; these motors get their position values directly from the axis-definition statement and are not affected by the inverse-kinematic program.

The basic purpose of the inverse-kinematic program, then, is to take the tip-position values found in Q1 – Q9 for the axes used in the coordinate system, compute the matching joint-coordinate values, and place them in variables in the P1 – P32 range.

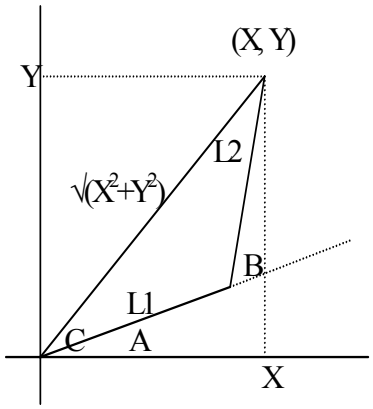
Example:
Continuing with our example of the two-axis shoulder-elbow robot, and for simplicity's sake limiting ourselves to positive values of B (the right-armed case), we can write our inverse-kinematic equations as follows:

$$B = +\cos^{-1}\left(\frac{X^2+Y^2-L_1^2-L_2^2}{2L_1L_2}\right)$$

$$A + C = a\tan 2(Y, X)$$

$$C = +\cos^{-1}\left(\frac{X^2+Y^2+L_1^2-L_2^2}{2L_1\sqrt{X^2+Y^2}}\right)$$

$$A = (A+C) - C$$



To implement these equations in a Turbo PMAC inverse-kinematic program for Coordinate System 1 that converts the X and Y tip coordinates in millimeters to the shoulder angle in Motor 1 and the elbow angle in Motor 2, the following program could be used. System constants Q91, Q92, and Q93 are the same as for the above forward kinematic program.

```
; Setup for program
&1
#1->I          ; Motor 1 assigned to inverse kinematic axis in CS 1
#2->I          ; Motor 2 assigned to inverse kinematic axis in CS 1
M5182->Y:$00203F,22,1          ; CS 1 run-time error bit
```

```
; Pre-compute additional system constants
Q94=Q91*Q91+Q92*Q92              ; L1^2 + L2^2
Q95=2*Q91*Q92                   ; 2*L1*L2
Q96=Q91*Q91-Q92*Q92             ; L1^2 – L2^2

; Inverse-kinematic algorithm to be executed repeatedly
&1 OPEN INVERSE                 ; Inverse kinematics for CS 1
CLEAR                           ; Erase existing contents
Q20=Q7*Q7+Q8*Q8                 ; X^2+Y^2
Q21=(Q20-Q94)/Q95         ; cos(B)
IF (ABS(Q21)<0.9998)            ; Valid solution w/ 1 deg margin?
  Q22=ACOS(Q21)                 ; B (deg)
  Q0=Q7                         ; X into cos argument for ATAN2
  Q23=ATAN2(Q8)                 ; A+C = ATAN2(Y,X)
  Q24=ACOS((Q20+Q96)/(2*Q91*SQRT(Q20)))  ; C (deg)
  Q25=Q23-Q24                   ; A (deg)
  P1=Q25*Q93                    ; Motor 1 = 1000A
  P2=Q22*Q93                    ; Motor 2 = 1000B
ELSE                            ; Not valid, halt operation
  M5182=1                       ; Set run-time error bit
ENDIF

CLOSE
```

Notes on the Example:

- By choosing the positive arc-cosine solutions, we are automatically selecting the right-armed case. In a more general solution, we would have to choose whether the positive or negative is used, based on some criterion.
- Increased computational efficiency could be obtained by combining more operations into single assignment statements. Calculations were split out here for clarity's sake.
- This example does not use the substitution macros permitted by the Executive program to substitute meaningful names for variables. Use of these substitution macros in complex applications is strongly encouraged.
- This example stops the program for cases where no inverse kinematic solution is possible. It does this by setting the "run-time error" status bit for the coordinate system, which causes Turbo PMAC to automatically halt motion program execution and issue the Abort command. Other strategies may be used to cope with this problem.

If this robot had a vertical axis at the tip, the relationship between motor and axis could be defined with a normal linear axis-definition statement (e.g. **#3->100Z** for 100 counts per millimeter), and the motor position would be calculated without the special inverse-kinematic program. Alternately, the motor could be defined as an inverse-kinematic axis (**#3->I**) and the motor position could be calculated in the inverse-kinematic program (e.g. **Q3=Q49*100** to set Motor 3 position from the Z-axis with 100 counts per unit).

Rotary Axis Rollover

If a rotary inverse-kinematic axis in the system has the capability to roll over, the inverse-kinematic program must handle the rollover calculations explicitly. The automatic rollover capability of the A, B, and C axes with Ixx27 is not available for inverse-kinematic axes. The key to handling rollover properly is to take the difference between the new and the old values and make sure that this difference is in the $\pm 180^{o}$ range. This can be done in Turbo PMAC with the % modulo (remainder) operator. This difference is then added to the old value. Mathematically, the equations are:

$$\Delta\theta = (\theta_{new-temp} - \theta_{old}) \%(-180)$$
$$\theta_{new} = \theta_{old} + \Delta\theta$$

When the modulo operation is done in Turbo PMAC with a negative operand '-n' (such as –180), the result is always in the $\pm n$ range.

For example, if the A-axis in the above example had the capability of rolling over, the line **Q25=Q24-Q23** could be replaced with:

```
Q25=P1/Q93+(Q24-Q23-P1/Q93)%-180      ; Handle rollover cases
```

The value (P1/Q93) is θ_{old}, from the previous cycle of the inverse kinematics or initially from the forward kinematics; and value (Q24-Q23) is $\theta_{new\text{-}temp}$, both in degrees.

Velocity Calculation Flag

In every move mode other than PVT mode, Turbo PMAC automatically sets the variable Q10 for the coordinate system to 0 as a flag to the inverse-kinematic program not to compute velocity values. If you plan to use both PVT mode and other modes, you must evaluate Q10 explicitly in your inverse-kinematic program (see below).

Iterative Solutions

Some robot geometries do not have closed-form inverse-kinematic solutions and require iterative numerical solutions. These cases are typically handled by a looping **WHILE … ENDWHILE** construct in the inverse-kinematic program. Multiple executions of the **WHILE** loop inside the inverse-kinematic program do not disable blending as they would inside the main motion program (due to the double jump-back rule), but excessive iterations can cause the calculations not to be done within the required time. This will cause a run-time error, aborting the program automatically.

Inverse-Kinematic Program for PVT Mode

The Turbo PMAC can also support the conversion of velocities from tip space to joint space in the inverse-kinematic program to enable the use of PVT mode with kinematic calculations. With PVT-mode moves, the position calculations are done just as for any other move mode. An additional set of velocity-conversion calculations must also be done.

When executing PVT-mode moves with kinematics active (Isx50 = 1), Turbo PMAC will automatically place the commanded axis velocity values from the PVT statements into variables Q11 – Q19 for the coordinate system before each execution of the inverse-kinematic program. These are signed floating-point values in the engineering velocity units defined by the engineering length/angle units and the coordinate system's Isx90 time units (e.g. mm/min or deg/sec). The following table shows the variable used for each axis:

Axis-Velocity Q-Variable	Axis Letter	Axis-Velocity Q-Variable	Axis Letter	Axis-Velocity Q-Variable	Axis Letter
Q11	A	Q14	U	Q17	X
Q12	B	Q15	V	Q18	Y
Q13	C	Q16	W	Q19	Z

Turbo PMAC will also set Q10 to 1 in this mode as a flag to the inverse-kinematic program that it should use these axis (tip) velocity values to compute motor (joint) velocity values.

In this mode, after any execution of the inverse-kinematic program, Turbo PMAC will read the values in those variables P1xx (P101 – P132) for each Motor xx in the coordinate system defined as inverse-kinematic axes (**#xx->I**). These are floating-point values, and Turbo PMAC expects to find them in units of counts per Isx90 milliseconds. Turbo PMAC will use them as motor (joint) velocity values along with the position values in Pxx to create a PVT move for the motor.

For PVT moves, then, the inverse-kinematic program must not only take the axis (tip) position values in Q1 – Q9 and convert them to motor (joint) position values in P1 – P32; it must also take the axis (tip) velocity values in Q11 – Q19 and convert them to motor (joint) velocity values in P101 – P132. Technically, the velocity conversion consists of the solution of the "inverse Jacobian matrix" for the mechanism.

Example:
Continuing with the shoulder-elbow robot of the above examples, the equations for joint velocities as a function of tip velocities are:

$$\dot{A} = \frac{L_2 \cos(A+B)\dot{X} + L_2 \sin(A+B)\dot{Y}}{L_1 L_2 \sin B}$$

$$\dot{B} = \frac{\left[-L_1 \cos A - L_2 \cos(A+B)\right]\dot{X} + \left[-L_1 \sin A - L_2 \sin(A+B)\right]\dot{Y}}{L_1 L_2 \sin B} = \frac{-X\dot{X} - Y\dot{Y}}{L_1 L_2 \sin B}$$

The angles A and B have been computed in the position portion of the inverse-kinematic program. Note that the velocities become infinite as the angle B approaches 0 degrees or 180 degrees. Since in our example we are limiting ourselves to positive values for B, we will trap any solution with a value of B less than 1^o or greater than 179^o (sin B < 0.0175) as an error.

```
&1
OPEN INVERSE
CLEAR
{Position calculations from above}
IF (Q10=1)                                  ; PVT mode?
  Q26=SIN(Q25)                              ; sin(B)
  IF (Q26>0.0175)                           ; Not near singularity?
    Q27=Q91*Q92*Q26                         ; L1*L2*sinB
    Q28=COS(Q25+Q22)                        ; cos(A+B)
    Q29=SIN(Q25+Q22)                        ; sin(A+B)
    Q30=(Q92*Q28*Q17+Q92*Q29*Q18)/Q27       ; dA/dt
    Q31=(-Q7*Q17-Q8*Q18)/Q27                ; dB/dt
    P101=Q30*Q93                            ; #1 speed in cts/(Isx90 msec)
    P102=Q31*Q93                            ; #2 speed in cts/(Isx90 msec)
  ELSE                                      ; Near singularity
    M5182=1                                 ; Set run-time error bit
  ENDIF
ENDIF
CLOSE
```

Note:

In this case the check to see if B is near 0^o or 180^o is redundant because we have already done this check in the position portion of the inverse-kinematic algorithm. This check is shown here to illustrate the principle of the method. In this example, a run-time error is created if too near a singularity; other strategies are possible.

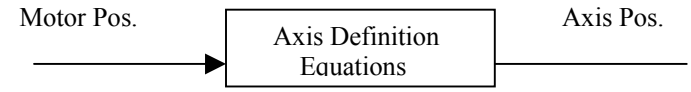
Executing the Kinematic Programs

Once the forward-kinematic and inverse-kinematic program buffers have been created for a coordinate system, Turbo PMAC will execute them automatically at the proper times once the kinematic calculations have been enabled by setting coordinate system I-variable Isx50 to 1. No modification to a motion program is required for access to the kinematic programs at the proper time.

The forward-kinematic program is executed automatically each time an **R** (run) or **S** (step) command is given to the coordinate system if Isx50 is 1. This is done to ensure that the starting tip (axis) position is correct for the calculation of the initial move, even if joint (motor) moves, such as jogs, have been done since the last programmed move. The forward-kinematic program is also executed automatically each time a **PMATCH** command is given to the coordinate system if Isx50 is 1.

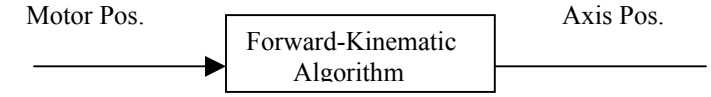
(With Isx50 = 0 and normal axis definition statements, Turbo PMAC executes this same function by mathematically inverting the equations of the axis-definition statements to derive the starting axis positions from present commanded motor positions. The axis-definition statements are technically inverse-kinematic equations, so their mathematical inverse forms the forward-kinematic equations. Because the standard axis-definition statements are limited to mathematically linear equations, their inverse can in general be derived automatically.)

Motor-to-Axis Conversion Without Forward-Kinematic Program



Motor 1: 5000	#1->1000X	X=5
Motor 2: 10000	#2->1000Y	Y=10

Motor-to-Axis Conversion With Forward-Kinematic Program



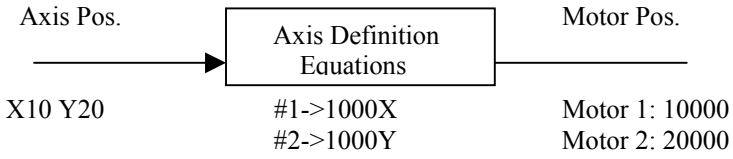
Motor 1: 5000	(P1=5000)	(Q7=f1[P1,P2])	X=(Q7)
Motor 2: 10000	(P2=10000)	(Q8=f2[P1,P2])	Y=(Q8)

The inverse-kinematic program is executed automatically each time Turbo PMAC computes new axis positions during the execution of a motion program. This occurs at the end-point of each programmed move block for non-segmented moves, such as those in **RAPID** mode. It occurs at the end of each intermediate segment – every Isx13 milliseconds – for segmented moves (**LINEAR** and **CIRCLE**-mode moves with Isx13 > 0).

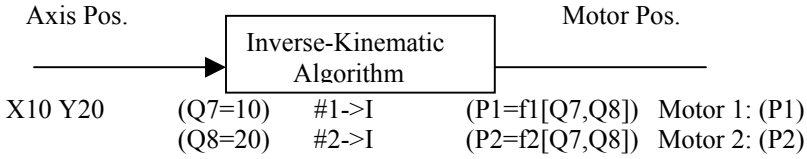
Note:

With normal axis definition statements, Turbo PMAC executes this same function by using the equations of the axis definition statements to derive motor positions from axis positions.

Axis-to-Motor Conversion Without Inverse Kinematics



Axis-to-Motor Conversion With Inverse Kinematics



When the inverse-kinematic program is executed only at programmed end-points, as in **RAPID** mode, all interpolation occurs in joint space. In this case, the path of the tip from point to point is not well defined if the programmed end-points are far apart, and in general it will not be a straight line.

When the inverse-kinematic program is executed at each intermediate segment boundary, the coarse interpolation (segmentation) is done in tip space, so the path is well defined. After the conversion of the segment coordinates to joint positions, the fine interpolation between segment boundaries is done in joint space as a cubic spline, but with the segments close together (typically 5 to 20 msec each), any deviations from the ideal tip path are negligible.

If the special lookahead buffer for the coordinate system is active (**LINEAR** or **CIRCLE**-mode moves with the lookahead buffer defined for the coordinate system, Isx13 > 0, and Isx20 > 0), the internal spline segments computed for the joints (motors) are automatically entered into the lookahead buffer. Here they are continually checked against position, velocity, and acceleration limits for each motor. This permits Turbo PMAC to check and correct automatically for the motion anomalies that occur near singularities, so the user does not need to do so.

Cutter Radius Compensation

Turbo PMAC provides the capability for performing cutter (tool) radius compensation on the moves it performs. This compensation can be performed among the X, Y, and Z axes, which should be physically perpendicular to each other. The compensation automatically offsets the described path of motion perpendicular to the path by a programmed amount, compensating for the size of the tool. This permits the user to program the path along the edge of the tool, letting Turbo PMAC calculate the tool-center path, based on a radius magnitude that can be specified independently of the program.

Cutter radius compensation is valid only in **LINEAR** and **CIRCLE** move modes. The moves must be specified by **F** (feedrate), not **TM** (move time). Turbo PMAC must be in move segmentation mode (Isx13 > 0) to do this compensation (Isx13 > 0 is required for **CIRCLE** mode anyway.)

Note:

In **CIRCLE** mode, a move specification without any center specification results in a linear move. This move is executed correctly without cutter radius compensation active, but if the compensation is active, it will not be applied properly in this case. A linear move must be executed in **LINEAR** mode for proper cutter-radius compensation.

Defining the Plane of Compensation

Several parameters must be specified for the compensation. First, the plane in which the compensation is to be performed must be set using the buffered motion-program **NORMAL** command. Any plane in XYZ-space may be specified. This is done by specifying a vector normal to that plane, with I, J, and K-components parallel to the X, Y, and Z-axes, respectively.

For example, **NORMAL K-1**, by describing a vector parallel to the Z-axis in the negative direction, specifies the XY-plane with the normal right/left sense of the compensation (**NORMAL K1** would also use the XY-plane, but invert the right/left sense). This same command also specifies the plane for circular interpolation. **NORMAL K-1** is the default. The compensation plane should not be changed while compensation is active.

Other common settings are **NORMAL J-1**, which specifies the ZX-plane for compensation, and **NORMAL I-1**, which specifies the YZ-plane. These three settings of the normal vector correspond to RS-274 "G-codes" G17, G18, and G19, respectively. If you are implementing G-codes in Turbo PMAC program 1000, you could incorporate in PROG 1000:

```
N17000 NORMAL K-1 RETURN
N18000 NORMAL J-1 RETURN
N19000 NORMAL I-1 RETURN
```

Defining the Magnitude of Compensation

The magnitude of the compensation – the cutter radius – must be set using the buffered motion program command **CCR{data}** (Cutter Compensation Radius). This command can take either a constant argument (e.g. **CCR0.125**) or an expression in parentheses (e.g. **CCR(P10+0.0625)**). The units of the argument are the user units of the X, Y, and Z-axes. In RS-274 style programs, these commands are often incorporated into "tool data" D-codes using Turbo PMAC motion program 1003.

Negative and zero values for cutter radius are possible. Note that the behavior in changing between a positive and negative magnitude is different from changing the direction of compensation. See Changes in Compensation, below. In addition, the behavior in changing between a non-zero magnitude and a zero magnitude is different from turning the compensation on and off. See the appropriate sections below.

Turning On Compensation

The compensation is turned on by buffered motion program command **CC1** (offset left) or **CC2** (offset right). These are equivalent to the RS-274 G-Codes **G41** and **G42**, respectively. If you are implementing G-Code subroutines in Turbo PMAC motion program 1000, you could simply incorporate in PROG 1000:

```
N41000 CC1 RETURN
N42000 CC2 RETURN
```

Turning Off Compensation

The compensation is turned off by buffered motion program command **CC0**, which is equivalent to the RS-274 G- Code **G40**. If your are implementing G-Code subroutines in Turbo PMAC motion program 1000, you could simply incorporate in PROG 1000:

```
N40000 CC0 RETURN
```

How Turbo PMAC Introduces Compensation

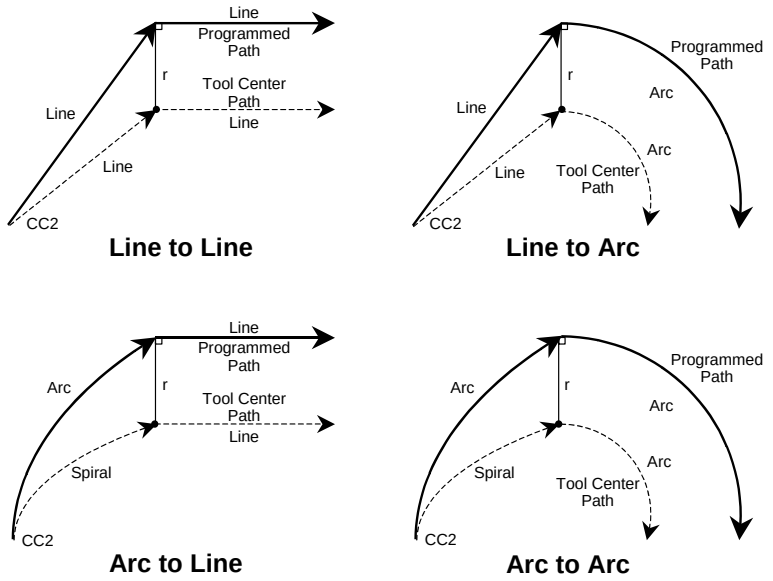
Turbo PMAC gradually introduces compensation over the next LINEAR or CIRCLE-mode move following the CC1 or CC2 command that turns on compensation. This lead-in move ends at a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the first fully compensated move at the intersection.

Note:

A few controllers can make their lead-in move a CIRCLE-mode move. This capability permits establishing contact with the cutting surface very gently, important for fine finishing cuts.

Inside Corner Introduction: If the lead-in move and the first fully compensated move form an inside corner, the lead-in move goes directly to this point. When the lead-in move is a LINEAR-mode move, the compensated tool path will be at a diagonal to the programmed move path. When the lead-in move is a CIRCLE-mode move, the compensated tool path will be a spiral.

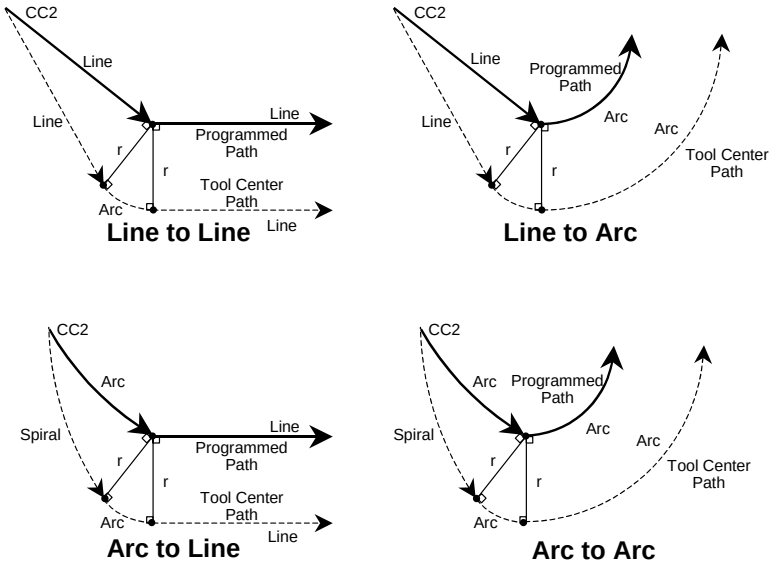
Introducing Compensation – Inside Corner



Outside Corner Introduction: If the lead-in move and the first fully compensated move form an outside corner, the lead-in move first moves to a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the lead-in move at the intersection. When the lead-in move is a LINEAR-mode move, this compensated tool path will be at a diagonal to the programmed move path.

When the lead-in move is a CIRCLE-mode move, this compensated tool path will be a spiral. Then a circular arc move with radius equal to the cutter radius is added, ending at a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the first fully compensated move at the intersection.

Introducing Compensation – Outside Corner



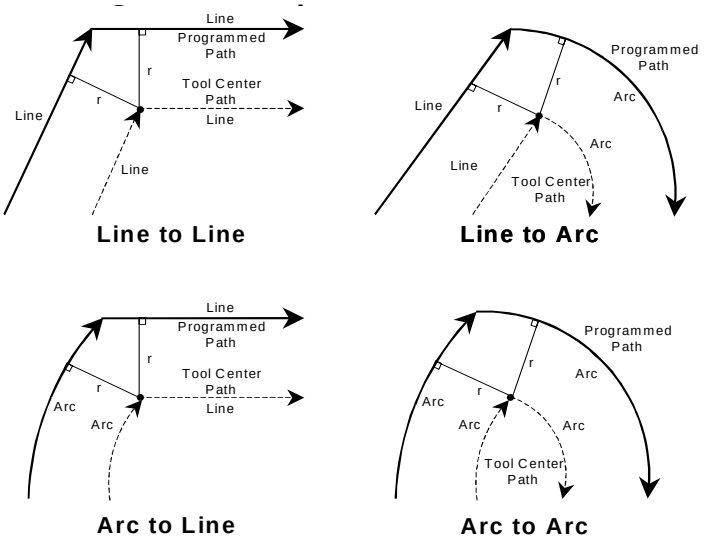
Note that the behavior for lead-in moves is different from changing the compensation radius from zero to a non-zero value while compensation is active. An arc move is always added at the corner, regardless of the setting of Isx99. This ensures that the lead-in move never cuts into the first fully compensated move.

Treatment of Inside Corners

Inside corners are still subject to the blending due to the **TA** and **TS** times in force (default values set by coordinate system I-variables Isx87 and Isx88, respectively). The longer the acceleration time, the larger the rounding of the corner. (The corner rounding starts and ends a distance **F*TA/2** from the compensated, but unblended corner.) The greater the portion of the blending is S-curve, the squarer the corner will be.

When coming to a full stop (e.g. Step, Quit, or **DWELL** at the corner) at an inside corner, Turbo PMAC will stop at the compensated, but unblended, corner point.

Inside Corner Cutter



Treatment of Outside Corners

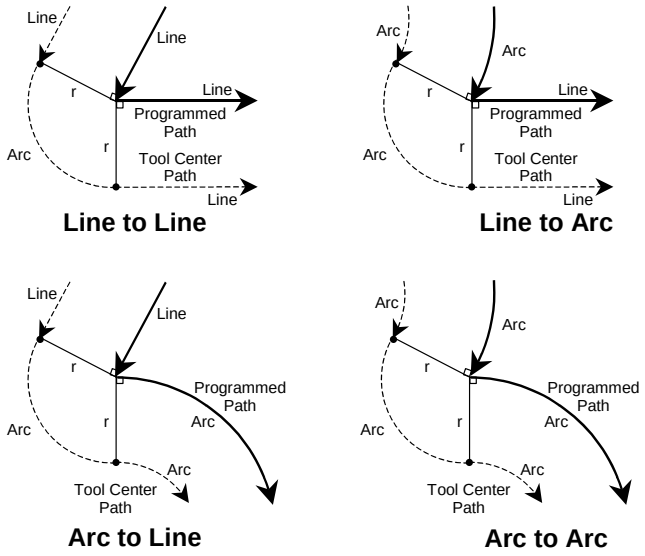
For outside corners, Turbo PMAC will either blend the incoming and outgoing moves directly together, or it will add an arc move to cover the additional distance around the corner. Which option it chooses is dependent on the relative angle of the two moves and the value of I-variable Isx99.

The relative angle between the two moves is expressed as the change in directed angle of the motion vector in the plane of compensation. If the two moves are in exactly the same direction, the change in directed angle is 0^o; if there is a right angle corner, the change is $+/-90^o$; if there is a complete reversal, the change in directed angle is 180^o.

Isx99 specifies the boundary angle between directly blended outside corners and added-arc outside corners. It is expressed as the cosine of the change in the directed angle of motion ($\cos 0^o=1.0$, $\cos 90^o=0.0$, $\cos 180^o=-1.0$) at the boundary of the programmed moves. The change in directed angle is equal to 180^o minus the "included angle" at the corner.

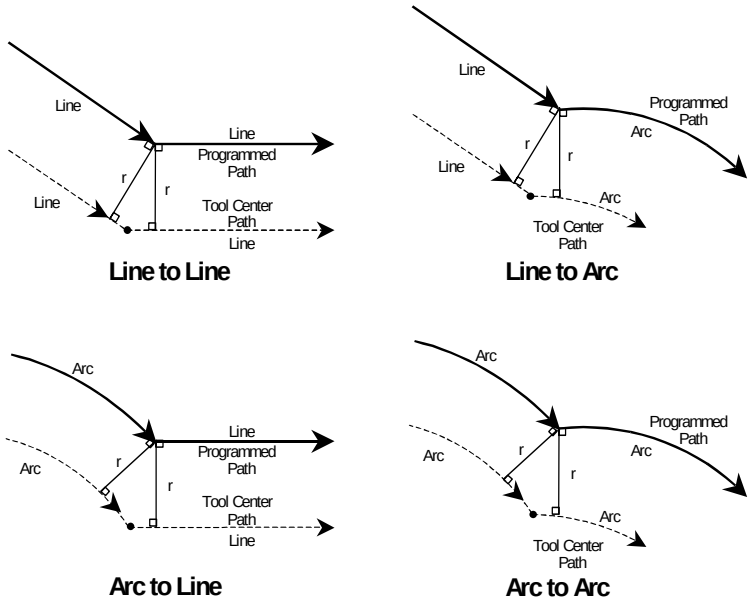
Sharp Outside Corner: If the cosine of the change in directed angle is less than Isx99, which means the corner is sharper than the specified angle, then an arc move will be added around the outside of the corner.

Outside Corner Cutter Compensation, Sharp Angle ($\cos\Delta\Theta$ < Isx99)



Shallow Outside Corner: However, if the cosine of the change in directed angle is greater than Isx99, which means that the corner is flatter than the specified angle, the moves will be directly blended together without an added arc.

Outside Corner Cutter Compensation, Shallow Angle (cos ΔΘ > Isx99)



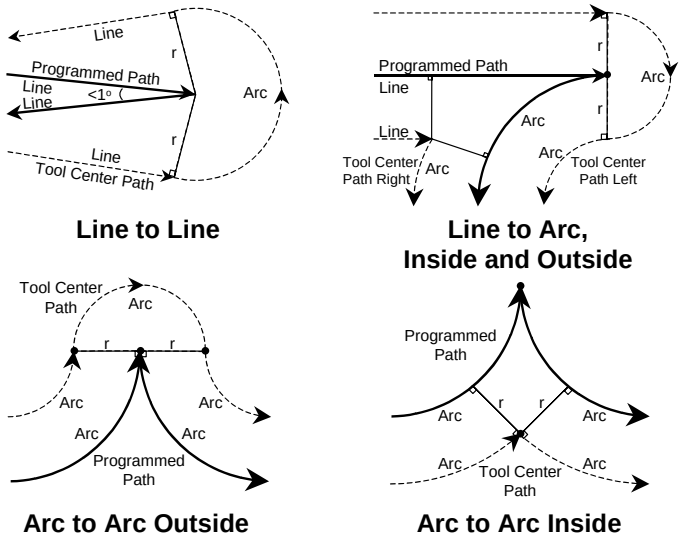
The added arc prevents the compensated corner from extending too far out on the outside of a sharp corner. However, as an added move, it has the minimum time of the acceleration time, which can cause a slowdown on a very shallow angle. While the default value for Isx99 of 0.9998 ($\cos 1^o$) causes an arc to be added on any change in angle greater than 1^o, many users will set Isx99 to 0.707 ($\cos 45^o$) or 0.0 ($\cos 90^o$) so arcs are only added on sharp corners.

When coming to a full stop (e.g. Step, Quit, /, or DWELL) at an outside corner with an added arc, Turbo PMAC will include the added arc move before stopping. When coming to a full stop at an outside corner without an added arc, Turbo PMAC will stop at the compensated, but unblended, corner point.

Treatment of Full Reversal

If the change in directed angle at the boundary between two successive compensated moves is $180^o \pm 1^o$ (the included angle is less than 1^o), this is considered a "full reversal" and special rules apply. If both the incoming and outgoing moves are lines, the corner is always considered an outside corner, and an arc move of approximately 180^o is added. If one or both of the moves is an arc, Turbo PMAC will check for possible inside intersection of the compensated moves. If such an intersection is found, the corner will be treated as an inside corner. Otherwise, it will be treated as an outside corner with an added 180^o arc move.

Reversal In Cutter Compensation



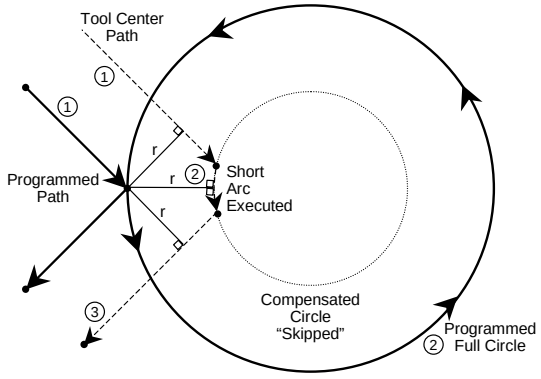
Note on Full Circles

If a full-circle move is executed while in cutter compensation, and one or both of the ends produces a shallow outside corner that is directly blended (no added arc – see Treatment of Outside Corners, above), the compensated arc move will be extended beyond 360°, and Turbo PMAC may produce just a very short arc, 360° shorter than what is desired (making it appear that the circle has been "skipped".

Typically, this is the result of sloppy programming – an outside corner with a full circle causes an overcut into the circle – many machine designers may want to permit slight cases of this. Coordinate system parameter Isx97 defines the shortest arc angle that may be executed; the longest arc angle is 360° plus this angle.

The default value of Isx97 sets a minimum arc angle of one-millionth of a semi-circle, enough to account for numerical round off, but sometimes not enough for compensated full circles. To handle these cases, Isx97 should be set to a somewhat larger value.

Failure When Compensation Extends Full Circle



Speed of Compensated Moves

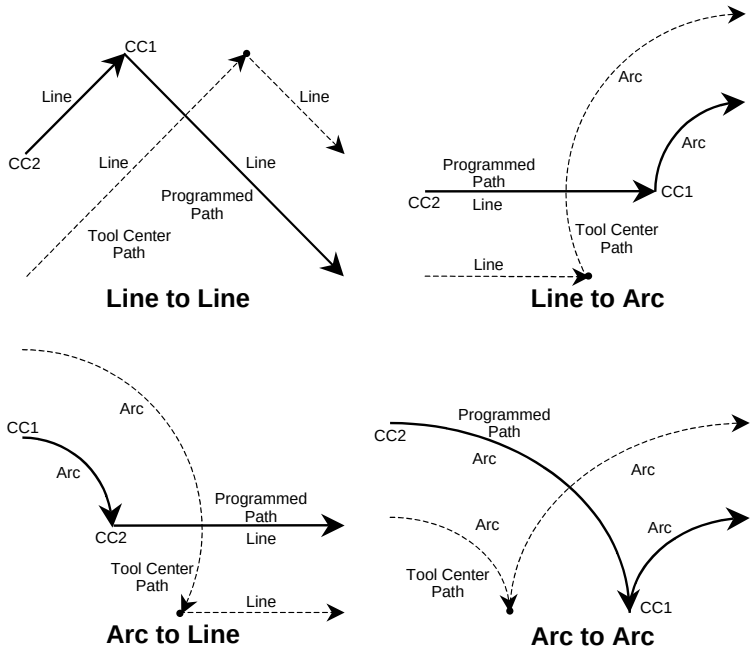
Tool center speed for the compensated path remains the same as that programmed by the F parameter. On an arc move, this means that the tool edge speed (the part of the tool in contact with the part) will be different from that programmed by the fraction R_{tool}/R_{arc}.

Changes in Compensation

Radius Magnitude Changes: Changes in the magnitude of compensation (new CCR values) made while compensation is active are introduced linearly over the next move. When this change is introduced over the course of a LINEAR-mode move, the compensated tool path will be at a diagonal to the programmed move path. When this change is introduced over the course of a CIRCLE-mode move, the compensated tool path will be a spiral.

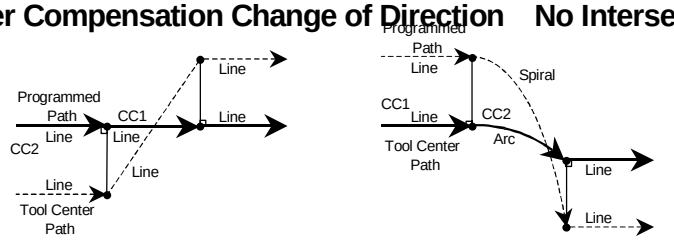
Compensation Direction Changes: Changes in the direction of compensation (between CC1 and CC2) made while compensation are generally introduced at the boundary between the two moves.

Cutter Compensation Change of Direction

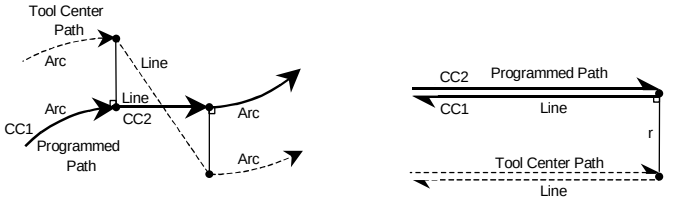


However, if there is no intersection between the two compensated move paths, the change is introduced linearly over the next move.

Cutter Compensation Change of Direction – No Intersection



Change Through a Line **Change Through an Arc**



Change Through a Line **Change in Reversal**

How Turbo PMAC Removes Compensation

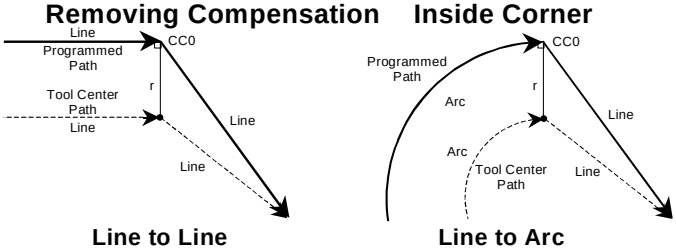
Turbo PMAC gradually removes compensation over the next LINEAR or CIRCLE-mode move following the CC0 command that turns off compensation. This lead-out move starts at a point one cutter radius away from the intersection of the lead-in move and the first fully compensated move, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the first fully compensated move at the intersection.

Note:

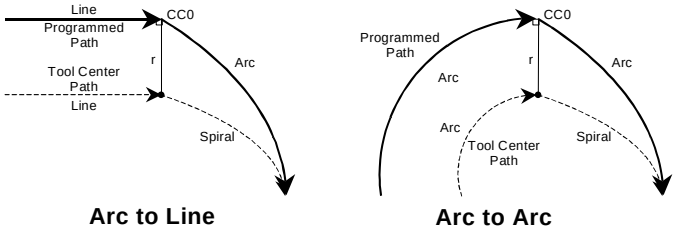
A few controllers can make their lead-out move a CIRCLE-mode move. This capability permits releasing contact with the cutting surface very gently, important for fine finishing cuts.

Inside Corner: If the last fully compensated move and the lead-out move form an inside corner, the lead-out move starts directly from this point to the programmed endpoint. When the lead-out move is a LINEAR-mode move, the compensated tool path will be at a diagonal to the programmed move path. When the lead-in move is a CIRCLE-mode move, the compensated tool path will be a spiral.

Removing Compensation **Inside Corner**



Line to Line **Line to Arc**



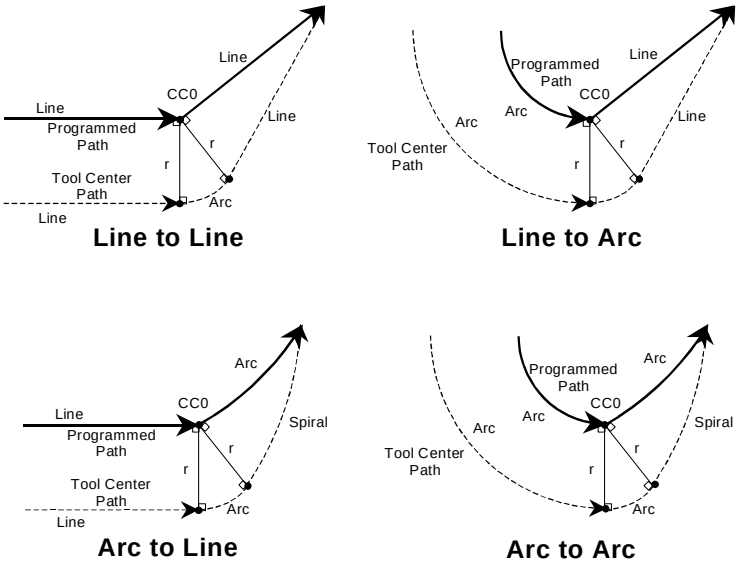
Arc to Line **Arc to Arc**

Outside Corner: If the last fully compensated move and the lead-out move form an outside corner, the last fully compensated move ends at a point one cutter radius away from the intersection of the last fully compensated move and the lead-out move, with the line from the programmed point to this compensated point being perpendicular to the path of the fully compensated move at the intersection.

Turbo PMAC then adds a circular arc move with radius equal to the cutter radius, ending at a point one cutter radius away from the same, with the line from the programmed point to this compensated endpoint being perpendicular to the path of the lead-out move at the intersection.

Finally, Turbo PMAC gradually removes compensation over the lead-out move itself, ending at the programmed endpoint of the lead-out move. . When the lead-out move is a LINEAR-mode move, this compensated tool path will be at a diagonal to the programmed move path. When the lead-in move is a CIRCLE-mode move, this compensated tool path will be a spiral.

Removing Compensation Outside Corner



Note:

This behavior is different from changing the magnitude of the compensation radius to zero while leaving compensation active. An arc move is always added at the corner, regardless of the setting of Isx99. This ensures that the lead-out move will never cut into the last fully compensated move.

Failures in Cutter Compensation

It is possible to give Turbo PMAC a program sequence in which the cutter compensation algorithm will fail, not producing desired results. There are three types of reasons the compensation can fail.

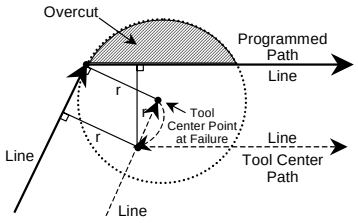
Inability to Calculate Through Corner: First, if Turbo PMAC cannot see ahead far enough in the program to find the next move with a component in the plane of compensation before the present move is calculated, then it will not be able to compute the intersection point between the two moves.

This can happen for several reasons:

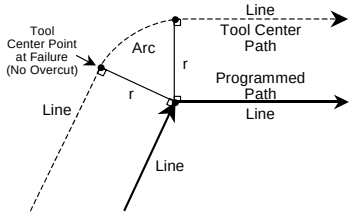
- There is a move with no component in the plane of compensation (i.e. perpendicular to the plane of compensation, as in a Z-axis-only move during XY compensation) before the next move in the plane of compensation, and no CCBUFFER compensation block buffer declared (see below).
- There are more moves with no component in the plane of compensation before the next move in the plane of compensation than the CCBUFFER compensation block buffer can hold (see below).
- There are more than 10 DWELLs before the next move in the plane of compensation.
- Program logic causes a break in blending moves (e.g. looping twice through a WHILE loop).

If Turbo PMAC cannot find the next move in time, it will end the current move as if the intersection with the next move would form an outside corner. If the next move, when found, does create an outside corner, or continues straight on, compensation will be correct. On an outside corner, an arc move is always added at the corner, regardless of the setting of Isx99. However, if the next move creates an inside corner, the path will have overcut into the corner. In this case, Turbo PMAC will then move to the correct intersection position and continue with the next move, leaving the overcutting localized to the corner.

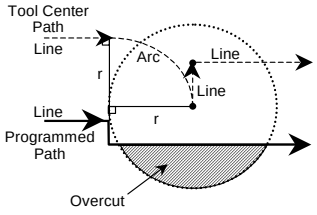
Failures in Cutter Compensation



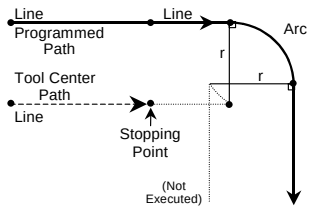
Failure to See Through Inside Corner



Failure to See Through Outside Corner



Inside Corner Smaller Than Cutter Radius



Arc Radius Smaller Than Cutter Radius

Inside Corner Smaller Than Radius: Second, if the compensated path produces an inside corner with one of the moves shorter than the cutter radius, the cutter compensation will not work properly. This situation results in a compensated move that is in the opposite direction from that of the uncompensated move, and there will be overcutting at the corner.

Inside Arc Radius Smaller Than Cutter Radius: Third, if the program requests an arc move with compensation to the inside, and the programmed arc radius is smaller than the cutter radius, then no proper path can be calculated. In this case, Turbo PMAC ends the program at the end of the previous move with a run-time error, setting the internal run-time error code in register Y:$002x14 to 7.

Block Buffering for Cutter Compensation

If your application requires the execution of moves perpendicular to the plane of compensation while cutter compensation is active, it will require that a special buffer be defined to hold these moves while Turbo PMAC scans ahead to find the next move in the plane of compensation so it can compute the proper intersection between the incoming move to this point in the plane and the outgoing move.

This buffer is created with the on-line coordinate-system-specific command **DEFINE CCBUF{constant}**, where **{constant}** is a positive integer representing the number of moves perpendicular to the compensation plane that can be stored in the buffer. This number should be at least as large as the largest number of consecutive perpendicular moves between any two moves in the plane.

With this buffer defined for the coordinate system, if Turbo PMAC encounters one or more moves perpendicular to the plane of compensation while compensation is active, these moves will be temporarily stored in the CCBUF while the next move in the plane is found, so the intersections can be correctly computed. However, if there is not enough room in the buffer to store all of the perpendicular moves found, Turbo PMAC will assume an outside-corner intersection; if the next move in the plane actually forms an inside corner, overcut will have occurred.

When programmed moves are actually stored in the CCBUF, commands that change the current position value – **HOME**, **HOMEZ**, and **PSET** – are not permitted. Turbo PMAC will report an ERR019 if I6 is set to 1 or 3.

The CCBUF, which stores motion program blocks for the purpose of computing proper cutter compensation intersection points, should not be confused with the LOOKAHEAD buffer, which stores small motion "segments" generated from these programmed blocks for the purpose of guaranteeing observance of position, velocity, and acceleration limits. Both of these buffers may be defined and active for a coordinate system at the same time.

The CCBUF is a temporary buffer. Its contents are never retained through a power-down or card reset; the buffer itself is only retained through a power-down or reset if it was defined, and I14 was set to 1, at the time of the last **SAVE** command.

Single-Stepping while in Compensation

It is possible to execute moves in single-step mode while cutter compensation is active, but the user should be aware of several special considerations for this mode of operation. Because of the need for the program to see ahead far enough to find the next move in the plane of compensation before the current move can be executed, the execution of an **S** single-step command may not produce the intuitively expected results. The single-step command on a move in compensation causes the preliminary calculations for that move to be done, not for the move actually to be executed. This has the following ramifications:

- A single-step command on the lead-in move for compensation will produce no motion, because the next move has not yet been found.
- Single-step commands on compensated moves in the plane of compensation will cause the previous move to execute.
- Single-step commands on compensated moves perpendicular to the plane of compensation will produce no motion, as these will just be held in the CCBUFFER. A single-step command on the next move in the plane of compensation will cause the previous move in the plane, plus all buffered moves perpendicular to the plane to execute.
- A single-step command on the lead-out move will cause both the last fully compensated move and the lead-out move to execute.

Unlike many controllers, Turbo PMAC can execute non-motion program blocks with single-step commands with cutter compensation active. However, the user should be aware that the execution of these blocks may appear out of sequence, because the motion from the previous programmed move block will not yet have been executed.

Synchronous M-variable assignments in this mode are still buffered and not executed until the actual start of motion execution of the next programmed move.

Three-Dimensional Cutter Radius Compensation

Turbo PMAC provides the capability for performing three-dimensional (3D) cutter (tool) radius compensation on the moves it performs. This compensation can be performed among the X, Y, and Z axes which should be physically perpendicular to each other (even if the motors assigned to the axes are not). Unlike the more common two-dimensional (2D) compensation, the user can independently specify the offset vector normal to the cutting surface, and the tool orientation vector.

The 3D compensation algorithm automatically uses this data to offset the described path of motion, compensating for the size and shape of the tool. This permits the user to program the path along the surface of the part, letting Turbo PMAC calculate the path of the center of the end of the tool.

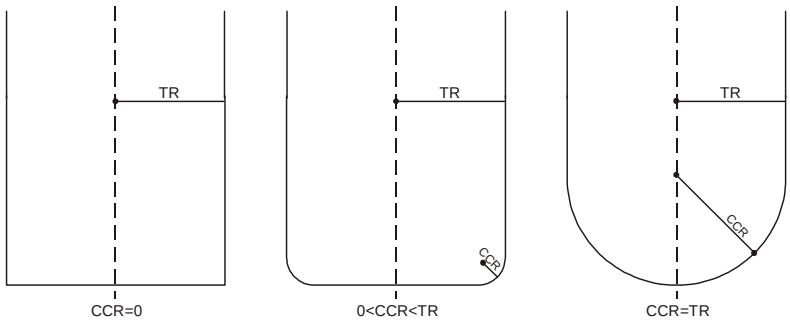
3D compensation is valid only in **LINEAR** and **CIRCLE** move modes, and is really intended only for **LINEAR** moves.

A note on terminology: Much of the documentation on the older two-dimensional cutter-radius compensation refers to just "cutter-radius compensation", since there was no 3D compensation at the time. Documentation specific to 3D compensation will always specify "3D" compensation.

Defining the Magnitude of 3D Compensation

The magnitude of 3D compensation is determined by two user-declared radius values. The first of these is the radius of the rounded end of the cutter, set by the buffered motion program command **CCR{data}** (Cutter Compensation Radius). This command can take either a constant argument (e.g. **CCR2.35**) or an expression in parentheses (e.g. **CCR(Q20-0.001)**). The units of the argument are the user units of the X, Y, and Z axes. In operation, the compensation first offsets the path by the cutter's end radius along the "surface-normal" vector (see below).

3D Compensation: Cutting Tool Cross Sections



The second value is the tool radius itself, the radius of the shaft of the tool. This is set by the buffered motion program command **TR{data}** (Tool Radius). This command can take either a constant argument (e.g. **TR7.50**) or an expression in parentheses (e.g. **TR(7.50-Q99)**). The units of the argument are the user units of the X, Y, and Z-axes. In operation, the compensation next offsets the path by an amount equal to the tool radius minus the cutter's end radius, perpendicular to the "tool-orientation" vector (see below).

A flat-end cutter will have a cutter-end radius of zero. A ball-end cutter (hemispherical tip) will have a cutter-end radius equal to the tool (shaft) radius. Other cutters will have a cutter-end radius in between zero and the tool radius.

Turning on 3D Compensation

3D cutter compensation is turned on by the buffered motion program command **CC3**. Since the offset vector is specified explicitly, there is no left or right compensation here. When 3D compensation is turned on, the surface-normal vector is automatically set to the null (zero-magnitude) vector, and also the tool-orientation vector is set automatically to the null vector. Until a surface-normal vector is explicitly declared with 3D compensation active, no actual compensation will occur. A tool-orientation vector must also be declared for compensation to work on anything other than a ball-nose cutter.

Turning off 3D Compensation

3D cutter compensation is turned off by the buffered motion program command **CC0**, just as for 2D compensation. Compensation will be removed over the next **LINEAR** or **CIRCLE** mode move after compensation has been turned off.

Declaring the Surface-Normal Vector

The direction of the surface-normal vector is determined by the **NX{data}**, **NY{data}**, and **NZ{data}** components declared in a motion program line. The absolute magnitude of these components does not matter, but the relative magnitudes define the direction. The direction must be from the surface into the tool.

Generally, all three components should be declared together. If only one or two components are declared on a program line, the remaining components are left at their old values, which could lead to unpredictable results. If it is desired that a component value be changed to zero, it should be explicitly declared as zero.

Note:

The coordinates of the surface-normal vector must be expressed in the machine coordinates. If the part is on a rotating table, these coordinates will not in general be the same as the original part coordinates from the part design – the vector must be rotated into machine coordinates before sending to Turbo PMAC.

The surface-normal vector affects the compensation for the move on the same line of the motion program, and all subsequent moves until another surface-normal vector is declared. In usual practice, a surface-normal vector is declared for each move, affecting that move alone.

Declaring the Tool-Orientation Vector

If the orientation of the cutting tool can change during the compensation, as in five-axis machining, the orientation for purposes of compensation is declared by means of a tool-orientation vector. (If the orientation is constant, as in three-axis machining, the orientation is usually declared by the normal vector to the plane of compensation, although the tool-orientation vector may be used.)

The direction of the tool-orientation vector is determined by the **TX{data}**, **TY{data}**, and **TZ{data}** components declared in a motion program line. The absolute magnitude of these components does not matter, but the relative magnitudes define the direction. The direction sense of the tool-orientation vector is not important; it can be from base to tip, or from tip to base

Generally, all three components should be declared together. If only one or two components are declared on a program line, the remaining components are left at their old values, which could lead to unpredictable results. If it is desired that a component value be changed to zero, it should be explicitly declared as zero.

Note:

The coordinates of the surface-normal vector must be expressed in the machine coordinates. If the part is on a rotating table, these coordinates will not in general be the same as the original part coordinates from the part design.

The tool-orientation vector affects the compensation for the move on the same line of the motion program, and all subsequent moves until another tool-orientation vector is declared. In usual practice, a tool-orientation vector is declared for each move, affecting that move alone.

Note:

The tool-orientation vector declared here does not command motion; it merely tells the compensation algorithm the angular orientation that has been commanded of the tool. Typically, the motion for the tool angle has been commanded with A, B, and/or C-axis commands, often processed through an inverse-kinematic subroutine on Turbo PMAC.

How 3D Compensation is Performed

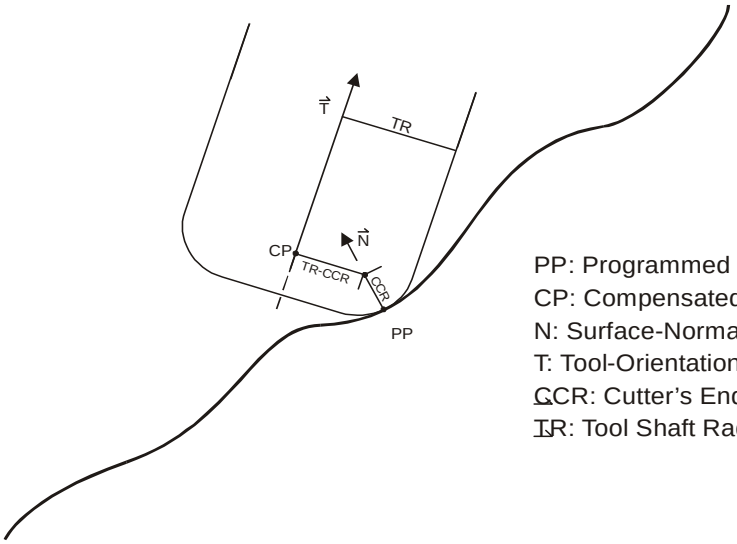
In operation, Turbo PMAC starts from the uncompensated X, Y, and Z-axis positions for each end-point programmed while 3D compensation is active. Then two offsets are applied to the X, Y, and Z-axis positions. The first offset is taken along the surface-normal vector, of a magnitude equal to the tip radius. The second offset is then taken toward the center of the tool, in the plane containing both the surface-normal vector and the tool-orientation vector, perpendicular to the tool-orientation vector, of a magnitude equal to the cutter radius minus the tip radius.

Once the modified end-point is calculated, the move to that end-point is calculated just as it would be without compensation. If the program is in **LINEAR** mode, it will be linearly interpolated. If the program is in **CIRCLE** mode (not advised), arc interpolation will be applied.

Because the offset to the end-point is directly specified for each move, there are no intersection points for Turbo PMAC to compute using the equations for the next move. This means there are no special lookahead or single-step execution considerations, as there are in 2D compensation.

All moves in 3D compensation are directly blended together. There are no special considerations for outside corners, as there are in 2D compensation. In addition, there are no special considerations for the lead-in and lead-out moves. The lead-in move is simply an interpolated move from the last uncompensated position to the first compensated position. The lead-out move is simply an interpolated move from the last compensated position to the first uncompensated position.

3D Cutter Radius Compensation



Altered-Destination Moves

Turbo PMAC gives the user the capability for altering the destination of certain moves in the middle of the execution of those moves by issuing an on-line command. This permits the user to start a move with a tentative destination and then change the destination during the move, with a smooth transition to the altered destination. If no move is currently executing, this feature also gives the capability of commanding a simple programmed move without using a program buffer.

This technique works with RAPID-mode moves only. The only motion mode whose destination can be altered on the fly is RAPID mode, and the only motion mode that can be used to approach the new destination is RAPID mode.

Altered-Destination Command

This feature is implemented by the on-line coordinate-system-specific command **!{axis}{constant}[{axis}{constant}…]** or its variant **!{axis}Q{constant}[{axis}Q{constant}…]** . The exclamation point identifies this command as the on-line altered-destination command. The axis letters and their associated values specify the new destination.

In the first case (e.g. **!X3.0Y2.7**), the constant value associated with each axis letter directly specifies the new destination of the axis. This first case is typically used when the command is issued from a host computer.

In the second case (e.g. **!XQ21YQ22**), the constant value associated with each axis letter after the Q character specifies the number of the Q-variable for the coordinate system whose value represents the new destination for the axis. For example, if Q21=3.0 and Q22=2.7, then **!XQ21YQ22** is equivalent to **!X3.0Y2.7**. This second case is usually used when the command is issued from a Turbo PMAC PLC program.

The values specified in this command are always positions of the new destinations (relative to program zero), not distances from previous commanded positions. That is, this command is always effectively in absolute mode, regardless of whether the axes are in absolute or incremental mode. If the axes are in incremental mode, they will stay in incremental mode for subsequent buffered program commands.

If there is no commanded move in progress when this command is issued, Turbo PMAC will just execute a RAPID-mode point-to-point move to the specified coordinates.

If a RAPID-mode move is in progress when the command is issued, Turbo PMAC will extend the current trajectory of each motor for Ixx92 milliseconds. At that point, it will break into the trajectory of each motor, compute a smooth blending for each motor to the RAPID-mode trajectory toward the new destination, and execute the modified trajectory. Because the altered-destination move is itself a RAPID-mode move, its destination can be modified with a subsequent altered-destination command.

If a move of some other mode is in progress when this command is issued, Turbo PMAC will reject the command with an error.

Use of Altered Destination

The altered-destination command is most often used to modify the destination of a RAPID-mode move executing from the coordinate system's rotary motion-program buffer as the last move in that buffer. In typical use, the RAPID move will be started with an approximate idea of the final destination, while some sensor, such as a vision system, determines the exact location. The altered-destination command is then sent to the coordinate system with the exact coordinates of the final destination.

If the altered-destination command is not received before the end of the move, there will be a momentary pause before the move to the final end position is started, but all axes end up in the same location as if the command were received before the end of the move. Note, however, that in this case, certain status bits such as "desired-velocity-zero", and "in-position" may get set at the end of the initial move, and so cannot be counted on by themselves to show that the modified end-point has been reached.

The altered-destination command can also be used to modify a RAPID-mode move that is not at the end of the rotary buffer, or one that is in a "fixed" motion-program buffer. In this case, there are a couple of things to watch. First, if axes are in incremental mode, the subsequent moves in the program are modified by the altered destination. Second, if the altered-destination command is received after the RAPID-mode move is finished, it may be rejected with an error, depending on what the program is executing subsequently.

Turbo PMAC Dual-Ported RAM Use

Dual-ported RAM (DPRAM) is an optional feature of the Turbo PMAC for high-speed communications with the host computer. Its purchase is recommended if more than 100 data items per second need to be transferred combined in both directions. Because this bank of memory has two ports, both the Turbo PMAC processor and the host processor have direct, random access to all of the registers of the DPRAM IC.

The Turbo PMAC family design supports both 8k x 16 and 32k x 16 banks of DPRAM. However, an individual member of the family may support only one of these sizes. As of this writing, the support is:

- Turbo PMAC(1)-PC: 8k x 16 only
- Turbo PMAC(1)-VME: 8k x 16 only
- Turbo PMAC2-PC: 8k x 16 only
- Turbo PMAC2-VME: 8k x 16 only
- Turbo PMAC2-PC Ultralite: 8k x 16 (Option 2A) or 32k x 16 (Option 2B)
- Turbo PMAC2-3U: 32k x 16 only

The Turbo PMAC family has preset structures for transferring data between the host computer and the controller; it also permits the user to define his own data structures. The pre-defined structures include:

- Control Panel Functions
- Motor Data Reporting Buffer
- Background Data Reporting Buffer
- ASCII Command and Response Buffers
- Data Gathering Buffer
- Background Variable Copying Buffers
- Binary Rotary Program Download

Physical Configuration and Connection

On the Turbo PMAC(1)-PC, the dual-ported RAM option is a separate ½-slot board that connects to the Turbo PMAC's CPU board with 2 short ribbon cables, and has its own ISA bus connector. On other Turbo PMAC boards, the dual-ported RAM is an on-board option in which the DPRAM IC is installed directly on the PMAC.

Host Address Setup

The dual-ported RAM has a fixed address space in the Turbo PMAC's address space. However, its address space in the host computer can vary depending on the setup of the card. The specification of the address of the card in the host computer is done entirely in software; there are no jumpers or DIP-switches to set.

ISA Bus Setup

There are two setup variables in the Turbo PMAC for the addressing of the DPRAM on the ISA bus in the PC's memory space: I93 and I94. (Note that the standard "host" bus communications port is mapped into the PC's I/O space, and has no relationship to the DPRAM memory address.) Because the PC uses byte addressing, a 16k x 8 slot of memory space must be found or created in the PC for the 8k x 16 DPRAM. For the 32k x 16 DPRAM, either a 64k x 8 slot of memory space must be found, or a 16k x 8 slot found and "bank" addressing used.

Note:

The PC/104 bus is completely software-compatible with the ISA bus, so these instructions apply to setting up DPRAM on the PC/104 interface of the 3U Turbo PMAC (Turbo Stack or UMAC Turbo).

Typically, in a PC, a slot of memory space between 640k ($0A0000) and 1M ($100000), where no standard memory resides, is used. Other devices may also occupy regions of this space. VGA displays often occupy the space from 640k to 704k ($0A0000 to $0B0000) and the BIOS often occupies from 960k to 1M ($0F0000 to $100000).

Locating the DPRAM between 1M ($100000) and 16M ($FFFFFF) is possible, but most operating systems cannot tolerate a break in their normal RAM addressing, so the DPRAM must be placed after the end of regular RAM. Since most PCs now have more than 16M of RAM, this is usually not feasible.

Therefore, in most PCs, the DPRAM is located somewhere between 704k ($0B0000) and 960k ($0F0000). The default settings locate it in the range from $0D4000 through $0D7FFF.

I93 is an 8-bit value that specifies ISA bus address bits A23 – A16 for the DPRAM. It is usually specified as a 2-digit hexadecimal value, and these two digits are the same as the first two digits of the six-digit ISA hexadecimal address, $0D in the default case.

I94 is an 8-bit value that controls the addressing of the DPRAM over the ISA bus. If only a 16k x 8 block is reserved for DPRAM, it also specifies ISA bus address bits A15 – A14. I94 is usually specified as a 2-digit hexadecimal number.

If a 16k x 8 block of memory on the ISA bus is to be used for DPRAM, the first digit should be set to equal the third digit of the six-digit base address. It can take a value of $0, $4, $8, or $C. For the default base address of $0D4000, it should be set to 4. If a 64k x 8 block of memory is to be used, the first digit should be set to 0.

The second digit represents the addressing mode. It should be set to 5 to use a 16k x 8 address space on the ISA bus. It should be set to 4 to use a 64k x 8 address space.

For example, to use a 16k x 8 block of memory from $0EC000 to $0EFFFF on the ISA bus, I93 should be set to $0E, and I94 should be set to $C5. To use a 64k x 8 block of memory from $0C0000 to $0CFFFF on the ISA bus, I93 should be set to $0C, and I94 should be set to $04.

To implement these settings and to hold them for future use, these I-variable values must be stored to non-volatile flash memory with the **SAVE** command, and the card must be reset (**$$$** command). Resetting the card copies the saved values of I93 and I94 back into the I-variable registers in RAM, and then into the active control registers at X:$070009 and X:$07000A, respectively.

If a 16k x 8 block of memory has been used for the larger (32k x 16) DPRAM, the PC can only view one-quarter of the DPRAM at a time. Following the instructions given above, this will be the first quarter (lowest addresses on the PMAC side). To get at other parts of the DPRAM, a "bank select" process must be used.

I94 can control the bank select with bits 1 and 3, but it is only used at power-on/reset, so it is not appropriate for dynamic bank selection. Therefore, it is better to use the active control register at X:$07000A directly. With the suggested M-variable definition of M94->X:$07000A,0,7, and I94 set as suggested above to select Bank 0 at power-on/reset, the following equations can be used to select each of the 4 banks (the vertical bar '|' is the logical bit-by-bit OR operator):

```
M94=I94 | $00        ; Bank 0 (PMAC addresses $060000 - $060FFF)
M94=I94 | $02        ; Bank 1 (PMAC addresses $061000 - $061FFF)
M94=I94 | $08        ; Bank 2 (PMAC addresses $062000 - $062FFF)
M94=I94 | $0A        ; Bank 3 (PMAC addresses $063000 - $063FFF)
```

VME Bus Setup

The address setup of the DPRAM on the VME bus is integrated with the general VME setup, including the "mailbox" registers, using variables I90 – I99.

I90 controls the VME address modifier. It should be set to $39 for 24-bit addressing, or $09 for 32-bit addressing.

I91 controls the don't care bits in the address modifier. Usually it should be set to $04.

I92 controls the VME address bus bits A31 – A24 when using 32-bit addressing for both the mailbox registers and the DPRAM. Usually it is specified as two hex digits and it should be the same as the first two hex digits of the 32-bit address. For example, if the base address of the DPRAM were $18C40000, I92 would be set to $18. When 24-bit addressing is set up, I92 is not used.

I93 controls the VME address bus bits A23 – A16 for the mailbox registers. Although it is possible for these address bits to be the same for both the mailbox registers and the small DPRAM, usually they are different.

I94 controls the VME address bus bits A15 – A08 for the mailbox registers. If bits A23 – A16 are the same for both the mailbox registers and the DPRAM, it is essential that I94 be set up so that there is no conflict between the 512 addresses required for the mailbox registers and the 16k registers required for the DPRAM.

I95 controls which interrupt line is used when PMAC interrupts the host computer over the bus. Values of $01 to $07 select IRQ1 to IRQ7, respectively. Turbo PMAC will use this interrupt line during DPRAM ASCII communications if I56 is set to 1 and I58 is set to 1.

I96 controls the interrupt vectors that are provided when Turbo PMAC interrupts the host computer. If the interrupt is asserted because PMAC has placed an ASCII response line in the DPRAM, the interrupt vector provided is equal to (I96 + 1).

I97 controls the VME address bus bits A23 – A20 for the DPRAM. It is usually specified as a 2-digit hexadecimal value. The first digit should always be set to 0. The second digit should be set to be equal to the 1st of 6 hex digits of the address if 24-bit addressing is used, or to the 3rd of 8 hex digits of the address if 32-bit addressing is used. For example, if the base address is $700000 in 24-bit addressing, I97 should be set to $70. If the base address is $18C40000 in 32-bit addressing, I97 should be set to $C0.

I98 controls whether the DPRAM is enabled. It should be set to $E0 to enable DPRAM access.

I99 controls the VME bus address width. It should be set to $90 for 24-bit addressing with DPRAM, or to $80 for 32-bit addressing with DPRAM.

To implement these settings and to hold them for future use, these I-variable values must be stored to non-volatile flash memory with the **SAVE** command, and the card must be reset (**$$$** command). Resetting the card copies the saved values of I90 – I99 back into the I-variable registers in RAM, and then into the active control registers at X:$070006 – X:$07000F.

One further step must be taken after every power-on/reset to select the VME address lines A19 – A14 for the DPRAM. These address lines are selected using a dynamic page-select technique, which must be used even if there is only a single "page" of DPRAM. One page consists of a 16k x 8 bank of memory addresses – for the small (8k x 16) DPRAM, this page selects the entire DPRAM. For the large (32k x 16) DPRAM (when available), this page selects one-quarter of the DPRAM.

These address lines are selected by writing a byte over the VME bus to (the mailbox base address + $121). The mailbox base address is defined by the settings of I92, I93, and I94 at the last power-on/reset. If the mailbox base address is at the default value of $7FA000, this byte must be written to VME bus address $7FA121.

Bits 0 to 5 of this byte must contain the values of A14 to A19, respectively, of the page of the DPRAM. One way to calculate this value is to take the 2nd and 3rd hex digits of the DPRAM page base address in 24-bit addressing, or the 4th and 5th hex digits in 32-bit addressing, and divide this value by 4 (shift right two bits). For example, if the base address is $780000 in 24-bit addressing, this byte should be set to $20 ($80/4 = $20). If the base address is $18C40000 in 32-bit addressing, this byte should be set to $10.

Note:

It is common that this byte value will be $00, and some Turbo PMAC-VME boards will power up with this byte already set at $00. However, this may not be true on some boards, so the user should not count on this default setting. For robust operation, this byte must be written after every power-on/reset.

PCI Bus, USB Setup

The address of the DPRAM to the host computer on a PCI-bus or Universal Serial Bus (USB) is established through an automatic "plug-and-play" software mechanism in the host computer, so there is no address setup on the Turbo PMAC for DPRAM interface on these buses.

Mapping of Memory Addresses

The mapping of memory addresses between the host computer on one side, and Turbo PMAC on the other side, is quite simple. Using this memory is a matter of matching the addresses on both sides. To Turbo PMAC, the DPRAM simply appears as extra memory in the fixed address range $060000 to $060FFF ($063FFF for the large DPRAM). Since Turbo PMAC has two (X and Y) registers per numerical address, the small DPRAM appears to the Turbo PMAC as a 4k x 32 block of memory; the large DPRAM appears as a 16k x 32 block of memory. When the PMAC hexadecimal addresses of the DPRAM are specified, the assembly-language convention of a '$' prefix is used to denote the use of hex numbers.

The host computer will almost certainly use byte addressing. Therefore, the small DPRAM appears to the host computer as a 16k x 8 block of memory. The large DPRAM appears as a 64k x 8 block of memory. Since the address range of the DPRAM in the host computer will vary from application to application, we can only talk of offsets from the base address when referring to individual registers. When the host hexadecimal address offsets of the DPRAM are specified, the C-language convention of a '0x' prefix is used to denote the use of hex numbers.

Because the Turbo PMAC effectively uses 32-bit addressing, and the host computer effectively uses 8-bit addressing, the host uses 4 numerical addresses for each 1 numerical address in PMAC. The following table shows how this address incrementing works for key addresses in the DPRAM.

Turbo PMAC Address	Host Address Offset	Example Host Address
Y:$060000	0x0000	0x0D0000
X:$060000	0x0002	0x0D0002
Y:$060001	0x0004	0x0D0004
X:$060001	0x0006	0x0D0006
…	…	…
Y:$060450	0x1140	0xD1140
…	…	…
Y:$060FFF	0x3FFC	0xD3FFC
X:$060FFF	0x3FFE	0xD3FFE
…	…	…
Y:$063FFF	0xFFFC	0xDFFFC
X:$063FFF	0xFFFE	0xDFFFE

The following two equations can be helpful for calculating matching DPRAM addresses:

PMAC_address = $060000 + 0.25*(Host_address – Host_base_address)

Host_address = Host_base_address + 4*(PMAC_address - $060000) + Offset

where:

Offset = 0 for accessing Y memory, or for X and Y together as 32 bits

Offset = 2 for accessing X memory alone

DPRAM Automatic Functions

Turbo PMAC provides many facilities for using the DPRAM to pass information back and forth between the host computer and the Turbo PMAC. Each of these functions has dedicated registers in the DPRAM. The following table shows each of these functions and the addresses used for it.

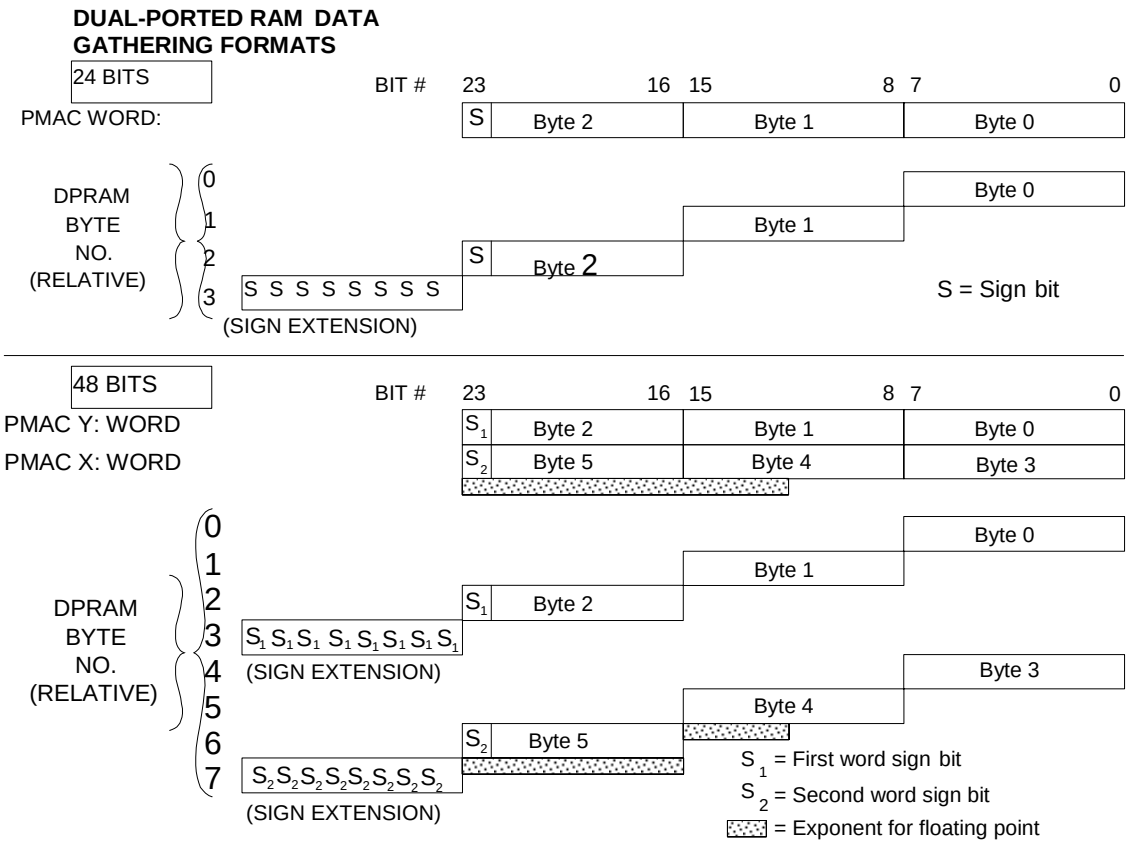
Host Address Offset	DPRAM Function	Turbo PMAC Address
0x0000	Control Panel Functions (pending)	$060000
0x006A	Motor Data Reporting Buffer	$06001A
0x0676	Background Data Reporting Buffer	$06019D
0x0E9E	DPRAM ASCII Command Buffer	$0603A7
0x0F42	DPRAM ASCII Response Buffer	$0603D0
0x1046	Background Variable Read Buffer Control	$060411
0x104C	Background Variable Write Buffer Control	$060413
0x1050	Binary Rotary Program Buffer Control	$060414
0x113E	DPRAM Data Gathering Buffer Control	$06044F
0x1140	Variable-Sized Buffers & Open-Use Space	$060450
0x3FFC	End of Small (8k x 16) DPRAM	$060FFF*
0xFFFC	End of Large (32k x 16) DPRAM	$063FFF*
*Turbo PMAC memory register Y:$3F contains the Turbo PMAC address of the last DPRAM address, plus one ($061000 or $064000).		

DPRAM Data Format

Data is stored in the DPRAM in 32-bit sign-extended form. That is, each short (24-bit) from PMAC is sign-extended and stored in 32 bits of DPRAM. The most significant byte is all ones or all zeros, matching bit 23. Each long (48-bit) word is treated as 2 24-bit words, with each short word sign-extended to 32 bits. The host computer must re-assemble these words into a single value. The data appears in the DPRAM in Intel format: the less significant bytes and words appear in the lower-numbered addresses.

To reassemble a long fixed-point word in the host, take the less significant 32-bit word, and mask out the sign extension (top eight bits). In C, this operation could be done with a bit-by- bit AND: (LSW & 16777215). Treat this result as an unsigned integer. Next, take the more significant word and multiply it by 16,777,216. Finally, add the two intermediate results together.

To reassemble a long floating-point word in the host, treat the less significant word the same as for the fixed-point case above. Take the bottom 12 bits of the more significant word (MSW & 4095), multiply by 16,777,216 and add to the masked less significant word. This forms the mantissa of the floating-point value. Now take the next 12 bits (MSW & 16773120) of the more significant word. This is the exponent to the power of two, which can be combined with the mantissa to form the complete value.



DPRAM Motor Data Reporting Buffer

Turbo PMAC can provide key motor data to the DPRAM, where it can be easily and quickly be accessed by the host computer. If this function is enabled, Turbo PMAC will copy key motor registers into fixed registers in the DPRAM.

Foreground vs. Background: This copying function can be done either as a foreground (interrupt) task in Turbo PMAC, or as a background task. Unless it is important to get the data at a guaranteed high frequency, it is strongly recommended that the copying be done in background, so as not to starve other important tasks on the Turbo PMAC for time. Even when the information is used for real-time operator display, background transfer is the recommended method.

Enabling Foreground Copying: Setting I48 to 1 enables foreground copying of motor data. With foreground copying, I47 sets the update period. If I47 is greater than 0, every I47 servo interrupts, Turbo PMAC will copy motor registers into the DPRAM. With I47 at 0, Turbo PMAC will check every servo cycle to see if the host computer has taken the previous data. If so, it will copy the current cycle's data, for an "on request" transfer.

Enabling Background Copying: Setting I57 to 1 enables background copying of motor data, and automatically sets I48 to 0 to disable the foreground copying. I49 must also be set to 1 to enable the background copying of coordinate-system and global data (see next section). If it is desired not to transfer any coordinate-system data, set the "maximum coordinate system number" register in 0x0674 (Y:$06019D) to 0 (see next section).

With background copying, I50 sets the update period. If I50 is greater than 0, each background cycle, Turbo PMAC will check to see if more than I50 servo cycles have elapsed since it last copied this data into DPRAM. If so, it will copy the present data. With I50 at 0, Turbo PMAC will check every background cycle to see if the host computer has taken the previous data. If so, it will copy the present data, for an "on request" transfer.

Motor Specification: A dedicated 32-bit "mask word" in DPRAM is used to specify which motors' data will be copied into DPRAM, whether for foreground or background transfers. This word can be set up from either the PMAC side or the host side. The format is as follows:

PMAC Address X:$06001C; Host Address Offset 0x0072

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Motor	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17

PMAC Address Y:$06001C; Host Address Offset 0x0070

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Motor	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

A value of '1' in the bit enables the transfer for the motor associated with the bit; a value of '0' disables the transfer. These bits may be changed at any time; the new value is effective for the next transfer. Setting this entire 32-bit word to 0 will stop all motor data copying.

Data Copied: For each motor enabled, the following values are transferred into DPRAM:

- Motor following error
- Motor servo command
- Motor servo status
- Motor general status
- Motor position bias
- Motor filtered actual velocity
- Motor master position
- Motor net actual position

Addresses of Data: For details as to the exact registers used for each of these values for each motor, consult the Turbo PMAC Memory Map section in the Software Reference Manual.

Foreground Handshaking: If foreground transfer is used (I48 = 1), Turbo PMAC will set Bit 15 of 0x006E (X:$06001B) to 0 while it is copying motor data into the DPRAM, and it will set this bit to 1 as soon as it is finished. The host computer should not try to read the data if this bit is 0. If I47 is set to 0 for "on request" transfers, the host computer should set this bit to 0 after reading the data to indicate to Turbo PMAC that it is time to provide the next set of data.

If foreground transfer is used, Turbo PMAC also copies the 24-bit servo-cycle counter value into 0x006C (Y:$06001B) and Bits 0 – 7 of 0x006E (X:$06001B) to "time-stamp" the data.

The host computer can set Bit 15 of 0x006A (X:$06001A) to 1 while it is reading the data. If Turbo PMAC sees that this bit is 1 when it is ready to transfer more data into the DPRAM, it will skip this cycle. The host must be sure to set this bit to 0 when it is done reading, to permit Turbo PMAC to transfer new data.

Background Handshaking: If background transfer of motor data is used (I57 = 1 and I49 = 1), the handshaking is the same as for the background coordinate system and global data buffers. Turbo PMAC will set Bit 15 of 0x067A (X:$06019E) to 0 while it is copying coordinate-system and global data into the DPRAM, and it will set this bit to 1 as soon as it is finished. The host computer should not try to read the data if this bit is 0. If I50 is set to 0 for "on request" transfers, the host computer should set this bit to 0 after reading the data to indicate to Turbo PMAC that it is time to provide the next set of data.

If background transfer is used, Turbo PMAC also copies the 24-bit servo-cycle counter value into 0x0678 (Y:$06019E) and Bits 0 – 7 of 0x067A (X:$06019E) to time-stamp the data.

The host computer can set Bit 15 of 0x0676 (X:$06019D) to 1 while it is reading the data. If Turbo PMAC sees that this bit is 1 when it is ready to transfer more data into the DPRAM, it will skip this cycle. The host must be sure to set this bit to 0 when it is done reading, to permit Turbo PMAC to transfer new data.

DPRAM Background Data Reporting Buffer

Turbo PMAC can provide key global and coordinate-system data as a background function to the DPRAM, where it can be easily and quickly be accessed by the host computer. If this function is enabled, Turbo PMAC will copy key global and coordinate-system registers into fixed registers in the DPRAM.

Enabling Copying: Setting I49 to 1 enables this copying of global and coordinate-system data into DPRAM as a background function. I50 sets the update period. If I50 is greater than 0, each background cycle, Turbo PMAC will check to see if more than I50 servo cycles have elapsed since it last copied this data into DPRAM. If so, it will copy the present data. With I50 at 0, Turbo PMAC will check every background cycle to see if the host computer has taken the previous data. If so, it will copy the present data, for an "on request" transfer.

Coordinate System Specification: Bits 0 – 4 of 0x0676 (Y:$06019D) specify the number of highest-numbered coordinate system in the Turbo PMAC whose data will be copied into DPRAM. The data for C.S. 1 through this coordinate system will be copied each time. If this value is set to 0, the transfer of coordinate-system data will be stopped.

Data Copied: For each coordinate system whose copying is enabled, the following data will be transferred:

- C.S. feedrate / move time
- C.S. time-base value (feedrate override)
- C.S. override source address
- C.S. status words
- C.S. axis target positions (ABCUVWXYZ)
- C.S. program status
- C.S. program lines remaining in rotary buffer
- C.S. time remaining in move segment
- C.S. time remaining in accel/decel
- C.S. program execution address offset

Addresses of Data: For details as to the exact registers used for each of these values for each motor, consult the Turbo PMAC Memory Map section in the Software Reference Manual.

Handshaking: Turbo PMAC will set Bit 15 of 0x067A (X:$06019E) to 0 while it is copying coordinate-system and global data into the DPRAM, and it will set this bit to 1 as soon as it is finished. The host computer should not try to read the data if this bit is 0. If I50 is set to 0 for "on request" transfers, the host computer should set this bit to 0 after reading the data to indicate to Turbo PMAC that it is time to provide the next set of data.

Turbo PMAC also copies the 24-bit servo-cycle counter value into 0x0678 (Y:$06019E) and Bits 0 – 7 of 0x067A (X:$06019E) to "time-stamp" the data.

The host computer can set Bit 15 of 0x0676 (X:$06019D) to 1 while it is reading the data. If Turbo PMAC sees that this bit is 1 when it is ready to transfer more data into the DPRAM, it will skip this cycle. The host must be sure to set this bit to 0 when it is done reading, to permit Turbo PMAC to transfer new data.

DPRAM ASCII Communications

Turbo PMAC can perform ASCII communications through the DPRAM, as well as through the normal bus communications port, the main serial port, and the auxiliary serial port. It can accept commands and provide responses simultaneously over multiple ports. The DPRAM provides the fastest path for ASCII communications.

Enabling: The DPRAM ASCII communications is enabled by setting I58 to 1. If I58 is set to 0, Turbo PMAC will not check the DPRAM for ASCII commands.

Sending a Command Line: To send an ASCII command line to Turbo PMAC:

1. Make sure that Bit 0 of the Host-Output Control Word at 0x0E9C (Y:$0603A7) – the "Host Data Ready" bit – is 0, to be sure that Turbo PMAC has read the previous command. For the first command after Turbo PMAC's power-on/reset, this bit may have to be set to 0 by the host.
2. Write the ASCII characters into the ASCII command buffer starting at 0x0EA0 (Y:$0603A8). Two 8-bit characters are packed into each 16-bit word; the first character is placed into the low byte. Subsequent characters are placed into consecutive higher addresses, two per 16-bit word. (In byte addressing, each character is written to an address one higher than the preceding character.) Up to 159 characters can be sent in a single command line.
3. Terminate the string with the NULL character (byte value 0). Do not use a carriage return to terminate the string, as you would on other ports.
4. Set Bit 0 of the Host-Output Control Word at 0x0E9C (Y:$0603A7) – the Host Data Ready bit – to 1 to tell Turbo PMAC that a command string is ready for it to read. Turbo PMAC will then read this command in the next background cycle, set this bit back to 0, and take the appropriate action for the command.

> ***Note:***
>
> The communications routines of the PCOMM32 library do all of these actions automatically. If you are writing your own low-level communications routines, this operation is fundamentally a "string copy" operation.

Sending a Control Character Command: Control-character commands can be sent through the DPRAM through a dedicated register, independent of the ASCII text commands. To send a control-character command to Turbo PMAC through the DPRAM:

1. Make sure that the control-character byte – Bits 0 – 7 of 0x0E9E (X:$0603A7) is set to 0. For the first control-character command after Turbo PMAC's power-on/reset, this byte may have to be set to 0 by the host.

2. Write the control character to Bits 0 – 7 of 0x0E9E (X:$0603A7).
3. Each background cycle, Turbo PMAC will read this byte. If the byte contains a non-zero value, Turbo PMAC will take the appropriate action for the command, and set the byte back to 0.

Reading a Response Line: To read an ASCII response line from the Turbo PMAC through the DPRAM:

1. Wait for the Host-Input Control Word at 0x0F40 (Y:$063D0) to become greater than 0, indicating that a response line is ready.
2. Interpret the value in this register to determine what type of response is present. If Bit 15 is 1, Turbo PMAC is reporting an error in the command, and there is no response other than this word. In this case, Bits 0 – 11 encode the error number for the command as 3 BCD digits.
3. If Bit 15 is 0, there is no error, and there is a response string. Bits 8 and 9 tell what caused the response. If they form a value of 0, a command from the host computer caused the response. If they form a value of 1, an internal CMDR statement caused the response. If they form a value of 2, an internal SENDR statement caused the response. Note the value in Bits 0 – 7. These will determine whether this is the last line in the response or not (see Step 5, below).
4. Read the response string starting at 0x0F44 (Y:$0603D1). Two 8-bit characters are packed into each 16-bit word; the first character is placed into the low byte. Subsequent characters are placed into consecutive higher addresses, two per 16-bit word. (In byte addressing, each character is read from an address one higher than the preceding character.) Up to 255 characters can be sent in a single response line. The string is terminated with the NULL character (byte value 0), convenient for C-style string handling. For Pascal-style string handling, the register at 0x0F42 (X:$0603D0) contains the number of characters in the string (plus one).
5. Clear the Host-Input Control Word at 0x0F40 (Y:$063D0) to 0. Turbo PMAC will not send another response line until it sees this register set to 0.
6. If Bits 0 – 7 of the Host-Input Control Word had contained the value $0D (13 decimal, "CR"), this was not the last line in the response, and steps 1 – 4 should be repeated. If they had contained the value $06 (6 decimal, "ACK"), this was the last line in the response.

Note:

The communications routines of the PCOMM32 library do all of these actions automatically. If you are writing your own low-level communications routines, this operation is fundamentally a "string copy" operation.

DPRAM Communications Interrupts

If I56 is set to 1, Turbo PMAC will interrupt the host computer whenever it has a response line ready for the host to read. This interrupt has the potential to make the host communications more efficient, because the computer does not need to poll the DPRAM to see when a response is ready.

VME Interrupt: On any of the VME-bus Turbo PMACs, this interrupt will appear on the VME-bus interrupt line specified by I95. It will have an interrupt vector equal to (I96 + 1).

ISA Interrupt: On the ISA-bus Turbo PMACs, this interrupt will appear on the ISA-bus interrupt line (IRQn) selected by an E-point jumper on the board. The interrupt controller IC on the board can pass interrupts from 8 different sources (IR0 – IR7) through this interrupt line.

On a Turbo PMAC(1)-PC, source IR7 is used to generate the interrupt. Jumper E85 must be ON, and jumpers E82 – E84 must be OFF for this feature to work. This brings the EQU4 line (position compare for encoder 4) into the interrupt controller – the position-compare function for this encoder may not be used for other purposes in this case.

On a Turbo PMAC2-PC, source IR5 is used to generate the interrupt from the EQU1 line (position compare for Encoder 1). The position-compare function for this encoder may not be used for other purposes in this case.

On a Turbo PMAC2-PC Ultralite, source IR5 is used to generate the interrupt from the CTRL0 line of the "DSPGATE2" IC, which does not have other functions.

Turbo PMAC will continue to assert this interrupt source until the host has cleared the Host-Interrupt Control Word. Because of this, the host may see the source still active when it gets an interrupt from another source.

DPRAM Background Variable Read Buffer

The Background Variable Data Read Buffer allows the user to have up to 128 user-specified Turbo PMAC registers copied into DPRAM during the background cycle. This function is controlled by I55. The buffer has two modes of operation, single-user and multi-user. The default mode is the single-user mode. It is active when bit 8 of the control word 0x1044 (Y:$060411) is set to zero. Multi-user mode is active when bit 8 of the control word is set to one.

General Description: The buffer has three parts. The first part is the header: 4 16-bit words (8 host addresses) containing handshake information and defining the location and size of the rest of the table. This is at a fixed location in DPRAM (see table below).

The second part contains the address specifications of the Turbo PMAC registers to be copied into DPRAM. It occupies 2 16-bit words (4 host addresses) for each Turbo PMAC location to be copied, starting at the location specified in the header.

The third part, starting immediately after the end of the second part, contains the copied information from the Turbo PMAC registers. It contains 2 16-bit words (4 host addresses) for each short (X or Y) Turbo PMAC location copied, and 4 16-bit words (8 host addresses) for each long Turbo PMAC location copied. The data format is the same as for data gathering to dual-ported RAM.

Register Map:
Background Variable Read Buffer Part 1
Definition and Basic Handshaking

Address	Description
0x1044 (Y:$060411)	PMAC to Host (Bit 0 = 1 for single user mode) Data Ready. PMAC done updating buffer - Host must clear for more data.
0x1046 (X:$060411)	Servo Timer (Updated at Data Ready Time)
0x1048 (Y:$060412)	Size of Data Buffer (measured in long integers of 32 bits each)
0x104A (X:$060412)	Starting Turbo PMAC Offset of Data Buffer from beginning of variable-buffer space $060450 (e.g.. $0100 for starting PMAC address $060550 – host address offset 0x1540)

Background Variable Read Buffer Part 2
Variable Address Buffer Format (2x16-bit words)

X:Mem Bits 15: Data Ready (multi-user mode)	X:Mem Bits 4 – 5: Variable type to read Bits 0 – 3: Bits 16 – 19 of address	Y:Mem Bits 0 – 15 of PMAC address of register to read	Dual Port Data Length
1 = PMAC data ready 0 = Host request data	Bits 4 – 5 = 0: PMAC Var. Y:Mem.	PMAC Address of Variable	32 bits
1 = PMAC data ready 0 = Host request data	Bits 4 – 5 = 1: PMAC Var. Long	PMAC Address of Variable	64 bits
1 = PMAC data ready 0 = Host request data	Bits 4 – 5 = 2: PMAC Var. X:Mem.	PMAC Address of Variable	32 bits

Enabling: To start operation of this buffer:

1. Write the starting location of the second part of the buffer into register 0x104A (X:$060412). This location is expressed as a Turbo PMAC address offset from the start of DPRAM's variable-buffer space at $060450, and it must be between $0000 and $0BAF for the 8k x 16 DPRAM, or between $0000 and $3BAF for the 32k x 16 DPRAM.
2. Starting at the DPRAM location specified in the above step, write the Turbo PMAC addresses of the registers to be copied, and the register types. The first 16-bit word holds the low 16 bits of the Turbo PMAC address of the first register to be copied; the second 16-bit word hold the high 4 bits of this address in bits 0 –3; bits 4 – 5 take a value of 0, 1, or 2 to specify Y, Long, or X, respectively, for the first register. The third and fourth words specify the address and type of the second register to be copied, and so on.
3. Write a number representing the size of the buffer into register 0x1048 (Y:$060412). This value must be between 1 and 128. When Turbo PMAC sees that this value is greater than zero and the individual data ready bit is zero, it is ready to start copying the registers you have specified into DPRAM.
4. To enable the single-user mode, write a zero into the control word at 0x1044 (Y:$060411). To enable the multi-user mode write a 256 (set bit 8 and clear bit 0) into this control and set bit 15 = 0 of each variable's data type register (X memory register). This will tell Turbo PMAC that the host is ready to receive data and what the mode is for the data.
5. Set I55 to 1. This enables both the background variable data reporting function and the background variable data writing function.

Single-User Mode Procedure: In operation, Turbo PMAC will try to copy data into the buffer each background cycle -- between each scan of each PLC program. If bit 0 of the control word 0x1044 is set to 1, it will assume that the host has not finished reading the data from the last cycle, so it will skip this cycle. If bit 0 is 0, it will copy all of the specified registers.

When Turbo PMAC is done copying the specified registers, it copies the low 16 bits of the servo timer register (X:$000000) into the DPRAM at 0x1046 (X:$060411). Then it sets Bit 0 of the control word 0x1044 (Y:$060411) to let the host know that it has completed a cycle.

When the host wants to read this data, it should check to see that Bit 0 of the control word at 0x1044 (the Data Ready bit) has been set. If it has, the host can begin reading and processing the data in the DPRAM. When it is done, it should clear the Data Ready bit to let Turbo PMAC know that it can perform another cycle.

Multi-User Mode Procedure: The operation of this mode is very similar to the Single-User Mode described above. The main difference is that the control word is no longer used as a global handshaking bit for updating the buffer. It only enables or disables the multi-user mode. In multi-user mode the control word is never modified by Turbo PMAC. Handshaking is now on an individual variable basis and is controlled by bit 15 of the variable's data type specifier.

Each background cycle, between each scan of each uncompiled PLC program, Turbo PMAC will try to copy data into each variable in the buffer. Bit 15 of each variable's data type specifier controls whether or not Turbo PMAC is allowed to update that particular variable's value. Turbo PMAC will skip updating any variable that has bit of its data type specifier set to 1. Any variable that has bit 15 set to 0 will be updated.

When Turbo PMAC is done servicing the buffer, it copies the low 16 bits of the servo timer register (X:$000000) into the DPRAM at 0x1046 (X:$060411). This is not dependent upon updating any variables in the buffer.

When the host wants to read a register, it should check to see that Bit 15 of the data type specifier (the Data Ready bit) has been set. If it has, the host can begin reading and processing the data from that register. When it is done, it should clear the Data Ready bit to let Turbo PMAC know that it can update that register the next cycle.

Data Format: Each 24-bit (X or Y) register is sign-extended to 32 bits. For a 48-bit (Long) register, each 24-bit half is sign-extended to 32 bits, for a total of 64 bits in the DPRAM. This data starts immediately after the last address specification register.

Disabling: To disable this function, you can set the size register 0x1048 (Y:$060412) to 0, or simply leave the individual Data Ready bit(s) set.

DPRAM Background Variable Data Write Buffer

The Background Variable Data Write Buffer is essentially the opposite of the Background Variable Data Read Buffer described above. It allows the user to write to up to 32 user-specified registers or particular bits in registers to Turbo PMAC without using a communications port (PCbus, serial, or DPRAM ASCII I/O). This allows the user to set any Turbo PMAC variable without using an ASCII command such as M1=1 and without worrying about an open Rotary Buffer. This function is controlled by I55.

General Description: The buffer has two parts. The first part is the header: 2 16-bit words (4 host addresses) containing handshake information and defining the location and size of the rest of the table. This is at a fixed location in DPRAM (Turbo PMAC address $060413 as shown in the table below).

The second part contains the address specifications of the Turbo PMAC registers to be copied into Turbo PMAC. It occupies 6 x 16-bit words (12 host addresses) for each Turbo PMAC location to be written to, starting at the location specified in the header.

Registers:
Background Variable Data Write Buffer Part 1
Definition and Basic Handshaking

Address	Description
0x104C (Y:$060413)	HOST to PMAC Data Transferred. PMAC is updated when cleared. Host must set for another update.
0x07E8 (X:$060413)	Starting Turbo PMAC Offset of Data Buffer from beginning of variable-buffer space $060450 (e.g.. $0100 for starting PMAC address $060550 – host address offset 0x1540)

Background Variable Write Buffer Part 2
Format for each Data Structure (6x16-bit)

Address	X-Register Contents	Y-Register Contents
n	Bits 11 – 15: Offset (= 0 – 23) – Starting bit number of target register into which value will be written Bits 6 – 10: Width (= 0, 1, 4, 8, 12, 16, or 20 – 0 represent 24 bits) – number of bits of target register into which value will be written Bits 3 – 5: Type of target register =0: Y-register = 1: Long (X/Y-register) =2: X-register Bits 0 – 2: Upper 3 bits (bits 16 – 18) of target register address	Bits 0 – 15 of target register address
n+1	Upper 16 bits of data word 1	Lower 16 bits of data word 1
n+2	Upper 16 bits of data word 2 (only used for writing into long register)	Lower 16 bits of data word 2 (only used for writing into long register)

Enabling: To start operation of this buffer:

1. Write the starting location of the second part of the buffer into register 0x104E (X:$060413). This location is expressed as a Turbo PMAC address offset from the start of DPRAM's variable-buffer space at $060450, and it must be between $0000 and $0BAF for the 8k x 16 DPRAM, or between $0000 and $3BAF for the 32k x 16 DPRAM.
2. Starting at the DPRAM location specified in the above step, write the Turbo PMAC addresses of the registers to be copied, and the register types. The first 16-bit word contains the low 16 bits of the Turbo PMAC address of the first register to be copied. The second 16-bit word takes a value of 0 to 65535 to specify the type, width, offset, and high 3 bits of address for this target Turbo PMAC register. The third, fourth, fifth, and sixth words specify the data to be written.

Note:

If you specify address 0, you will terminate the writing operation. No write operations further down in the buffer will be executed.

3. Write a number representing the size of the buffer into register 0x104C (Y:$060413). This value must be between 1 and 32. When Turbo PMAC sees that this value is greater than zero, it is ready to start copying the registers you have specified into Turbo PMAC. When it is finished it will change the value in this register to a 0.
4. Set I55 to 1. This enables both the background variable data read function and the background variable data write function.

Procedure: In operation, Turbo PMAC will copy the data from the buffer into Turbo PMAC during the background cycle whenever 0x104C (Y:$060413) is a not zero. If this register is 0 it will assume that the host has not finished placing the data in the buffer and will not write to Turbo PMAC. Once this register is set to a number from 1 to 32 it will copy that many registers, starting at the start of the header start address information, from the DPRAM to Turbo PMAC.

When Turbo PMAC is done copying the specified registers, it sets register 0x104C (Y:$060413) to zero to let the host know that it has completed a cycle.

When the host wants to update this buffer, it should check to see that 0x104C (Y:$060413) is zero. When it is done, it should set up the address/data structure. Then set 0x104C (Y:$060413) to the number of registers to copy to Turbo PMAC to let Turbo PMAC know that it can perform another cycle.

Data Format: Turbo PMAC X and Y registers will use the long 32-bit data 1word. The 32-bit data 2 word is not used in this case. The high 8 bits are sign-extension bits.

For a 48-bit Turbo PMAC integer or float point value, The L (Long) format should be used. L-format will have the lower 32 bits of the total 48 bits in the long 32-bit data 1 word and the upper 16 bits in the lower 32-bit data 2 word. This data starts immediately after the last address specification register.

Disabling: To disable this function, simply leave 0x104C (Y:$060413) set to zero.

DPRAM Binary Rotary Program Transfer Buffers

The binary rotary program transfer buffers in Turbo PMAC's DPRAM permit the host computer to send motion program commands to Turbo PMAC in its internal binary storage format for the fastest possible transmission of these commands. Each of the 16 possible coordinate systems in the Turbo PMAC can have its own binary rotary transfer program buffer in DPRAM.

Each coordinate system for which this feature is used must also have a rotary motion program buffer defined in Turbo PMAC's internal RAM. This is done with the **`&n DEFINE ROTARY {size}`** command. These internal rotary motion program buffers are not retained through a power-down or board reset, so they must be defined after every board power-up/reset. If multiple internal rotary program buffers are defined, they must defined from the highest-numbered coordinate system to the lowest.

The binary rotary program transfer buffers in DPRAM are simply pass-through buffers to the internal rotary program buffers. When Turbo PMAC receives a binary-format motion program command in the DPRAM buffer from the host computer, it simply copies this data into the rotary buffer in internal memory. The end result is the same as if an ASCII program command had been sent to Turbo PMAC through any of the ports, but the transmission is quicker for several reasons:

1. There is no handshaking of individual characters.
2. There is no parsing of an ASCII command into internal binary storage format.
3. Multiple command lines can be processed in a single communications cycle.

If I45 is set to the default value of 0, Turbo PMAC checks the binary rotary buffer(s) in DPRAM every background cycle, transferring any new contents to the internal rotary program buffer(s). If I45 is set to 1, it checks the binary buffers as a higher-priority foreground task, every real-time interrupt.

Routines in Delta Tau's PCOMM32 communications library provide automatic support for the binary rotary-program transfer buffer.

General Description: Each coordinate system's binary rotary transfer buffer has two parts. The first part is the header, at a fixed address in DPRAM. The header for each binary rotary transfer buffer occupies 6 16-bit words, and contains the key information on the size and status of the second part of the buffer.

The second part of the buffer is at a location in DPRAM specified by the user in the header. It contains the actual binary-format motion-program commands. The size of this part is also specified by the user in the header.

Registers: The following table shows the structure of the header. The addresses given are for the first coordinate system. Headers for the other coordinate systems follow immediately after. Those addresses can be found in the memory map in the Software Reference.

Binary Rotary Transfer Buffer Control

Address for first C.S. Rotary Buffer	Description
0x1050 (Y:$60414)	PMAC to HOST Binary Rotary Buffer Status Word
	Bit 15 = 1 :Error (Stops processing commands)
	Bit14 = 1 :Internal Rotary buffer full (Busy flag) PMAC Index stops updating.
	Bits 7-0 = Code Error
	------ ----------
	1 Internal Rotary Buffer size = 0
	or DPRAM Rotary Buffer Size = 0
	These flags are set and reset by the PMAC. The Busy flag is
	set when the PMAC internal rotary buffer is full.
	This however does not mean the DPRAM Binary
	Rotary buffer is full (See Rules). The Busy flag is
	reset when the PMAC internal rotary buffer is
	not full or the DPR binary rotary buffer is empty.
0x1052 (X:$60414)	Coordinate System Number and Enable Control Bits 0 – 4 represent C.S. #; buffer enabled if 0 < C.S.# < 17
0x1054 (Y:$60415)	Host Binary Rotary Buffer Index – PMAC address offset from start address of buffer as set in 0x105A
0x1056 (X:$60415)	PMAC Binary Rotary Buffer Index – PMAC address offset from start address of buffer as set in 0x105A
0x1058 (Y:$60416)	Size of Binary Rotary Buffer – in PMAC addresses (= host computer addresses / 4)
0x105A (X:$60416)	Starting Binary Rotary Buffer PMAC Address Offset – from start of DPRAM variable buffer space ($060450)

Using the Buffer: First, Turbo PMAC's internal binary rotary program buffer must be established with the **`&n DEFINE ROT`** command.

Next, the header information for the DPRAM transfer buffer must be set up. The starting address and size of the transfer buffer must be declared. The buffer should not overlap with any other use of DPRAM. The size parameter must be an even number, with an absolute minimum of 6 PMAC addresses. The size should be declared large enough so as not to limit throughput. Each basic PMAC command component (e.g. X10) occupies 2 PMAC addresses in the transfer buffer. The transfer occurs each background cycle, typically a few milliseconds.

Both indices should be set to zero to indicate that both sides are pointing to the start of the buffer. Next, the **`&n OPEN BIN ROT`** command should be given, so that Turbo PMAC checks for new data in the transfer buffer.

Now, binary format commands can be loaded into the transfer buffer, and the host index updated. As Turbo PMAC reads the commands from the buffer, it updates the PMAC index.

Binary Command Structure: Typically, PCOMM32 routines generate the binary command format automatically. Contact the factory if you require knowledge of this format.

DPRAM Data Gathering Buffer

Turbo PMAC's data gathering function can create a rotary buffer in DPRAM, so that the host computer can pick up the data as it is being gathered. This way, the size of the data gathering buffer is not limited by Turbo PMAC's own memory capacity. The data gathering buffer in DPRAM is selected if I5000 is set to 2 or 3; if I5000 is set to 3, it is used in a rotary fashion, which is how the buffer is typically used.

The DPRAM data gathering buffer always starts at address 0x1140 (Y:$060450). Its size is determined by the **DEFINE GATHER {size}** command, where **{size}** sets the number of PMAC addresses from the start. This size value is stored at 0x113C (Y:$06044F).

Variables I5001 through I5048 determine the potential registers to be gathered. I5050 and I5051 are 24-bit mask variables that determine which of the 48 possible sources will be gathered. I5049 determines the gathering period, in servo cycles.

The actual gathering is started by the on-line **GATHER** command, and stopped by the on-line **ENDGATHER** command. As Turbo PMAC gathers data into the DPRAM, it advances the pointer that shows the address offset where the next item to be gathered will be placed. This pointer is stored at 0x113E (X:$06044F). The host computer must watch for changes to this pointer to indicate that more data has been copied into DPRAM.

TURBO PMAC VARIABLE AND COMMAND SUMMARY

Notes

- PMAC syntax is not case sensitive.
- Spaces are not important in PMAC syntax, except where noted
- **{}** -- item in **{}** can be replaced by anything fitting definition
- **[]** -- item in **[]** is optional to syntax
- **[{item}...]** -- indicates previous item may be repeated in syntax
- **[..{item}]** -- the periods are to be included in the syntax to specify a range
- **()** -- parentheses are to be included in syntax as they appear

Definitions

- **constant** -- numerically specified non-changing value
- **variable** -- entity that holds a changeable value
- **I-variable** -- variable of fixed meaning for card setup and personality (1 of 8192)
- **P-variable** -- global variable for programming use (1 of 8192)
- **Q-variable** -- local variable (in coordinate system) for programming use (1 of 8192)
- **M-variable** -- variable assigned to memory location for user use (1 of 8192)
- **pre-defined variable** -- mnemonic that has fixed meaning in card
- **function** -- SIN, COS,TAN,ASIN,ACOS,ATAN,ATAN2,LN,EXP,SQRT,ABS,INT
- **operator** -- for arithmetic or bit-by-bit logical combination of two values: +, -, *, /, % (mod), & (and), | (or), ^ (xor)
- **expression** -- grouping of constants, variables, functions, and operators
- **data** -- constant without parentheses, or expression with parentheses
- **comparator** -- evaluates relationship between two values: =, !=, >, !>, <, !<, ~, !~
- **condition** -- evaluates as true or false based on comparator(s)
- **simple condition** -- {expression} {comparator} {expression}
- **compound condition** -- logical combination of simple conditions
- **motor** -- element of control for hardware setup; specified by number
- **coordinate system** -- collections of motors working synchronously
- **axis** -- element of a coordinate system; specified by letter chosen from X, Y, Z, A, B, C, U, V, W
- **buffer** -- space in user memory for program or list; contains up to 256 motion programs and 32 PLC blocks

On-Line Commands

(Executed immediately upon receipt by PMAC)

On-line Global Commands

Addressing Mode Commands

@n – Address card n (n is hex digit 0 to f); serial host only
@ – Report currently addressed card to host; serial host only
#n – Make motor n currently addressed motor
– Report currently addressed motor number to host
##n – Select motor group of 8 for multi-motor responses
- Report selected motor group of 8
&n – Make coordinate system n the currently addressed coordinate system
& – Report currently addressed coordinate system to host

Communications Control-Characters

<CTRL-H> – Erase last character from host (backspace)
<CTRL-I> – Repeat last command from host (tab)
<CTRL-M> – End of command line (carriage return)
<CTRL-N> – Report checksum of current command line
<CTRL-T> – End MACRO ASCII pass through mode
<CTRL-X> – Abort current PMAC command and response strings

General Global Commands

$$$ – Reset entire card, restoring saved values
$$$* – Reset and re-initialize entire card, using factory default values.
LOCK{constant},P{constant} – Check/set process locking bit
PASSWORD={string} – Set/confirm password for PROG1000-32767, PLC0-15
SAVE – Copy active memory into non-volatile flash memory
SETPHASE{constant}[,{constant}…] – Set commutation phase position for specified motor(s)
TIME={time} – Set time in active memory
TODAY={date} – Set date in active memory
UNLOCK{constant} – Clear process locking bit
UPDATE – Copy date and time into optional non-volatile clock/calendar
UNDEFINE ALL – Erase definition of all coordinate systems

Global Action Commands

<CTRL-A> – Abort all motion programs and moves
<CTRL-D> – Disable all PLC and PLCC programs
<CTRL-K> – Kill outputs for all motors
<CTRL-O> – Do feed hold on all coordinate systems
<CTRL-Q> – Quit all programs at end of calculated moves
<CTRL-R> – Run working programs in all coordinate systems
<CTRL-S> – Step working programs in all coordinate systems

Global Status Commands

<CTRL-B> – Report 8 motor status words to host
<CTRL-C> – Report all coordinate system status words to host
<CTRL-F> – Report 8 motor following errors (unscaled)
<CTRL-G> – Report global status words in binary form
<CTRL-P> – Report 8 motor positions (unscaled)
<CTRL-V> – Report 8 filtered motor velocities (unscaled)
??? – Report global status words in hex ASCII
CID – Report card ID (part) number
CPU – Report model of CPU used
DATE – Report release date of firmware version used
IDNUMBER – Report Option 18 electronic identification number
LIST – Report contents of open program buffer
LIST PROGRAM {constant} – Report contents of specified motion program
LIST PLC {constant} – Report contents of specified PLC program
SID – Report Option 18 electronic identification number
SIZE – Report size of open memory in words
STN – Report MACRO-ring station-order number
TIME – Report present time
TODAY – Report present date
TYPE – Report type of PMAC
VERSION – Report firmware revision level
VID – Report vendor ID number

Register Access Commands

R{address}[,{constant}] – Report contents of specified memory word address [or specified range of addresses] in decimal
RH{address}[,{constant}] – Report contents of specified memory word address [or specified range of addresses] in hex
W{address},{constant}[,{constant}..] – Write value to specified memory word address [or values to range]

PLC Control Commands

ENABLE PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s], starting at top of scan
DISABLE PLC{constant}[,{constant}...] – Disable operation of specified interpreted PLC program[s]
PAUSE PLC{constant}[,{constant}...] – Suspend operation of specified interpreted PLC program[s]
RESUME PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s], starting at paused point
ENABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]
DISABLE PLCC{constant}[,{constant}...] –- Disable operation of specified compiled PLC program[s]

Global Variable Commands

{constant} – Equivalent to **P0={constant}** if no unfilled table; otherwise value entered into table
I{data}={expression} – Assign expression value to specified I-variable
I{constant}..{constant}={constant} – Assign constant value to specified range of I-variable(s)
I{constant}[..{constant}]=* – Set specified I-variable[s] to default[s]
I{constant}=@I{constant}] – Set specified I-variable to address of another I-variable
I{constant}[..{constant}] – Report I-variable value(s) to host
P{data}={expression} – Assign expression value to specified P-variable
P{constant}..{constant}={constant} – Assign constant value to specified range of P-variable(s)
P{constant}[..{constant}] – Report P-variable value(s) to host
M{data}={expression} – Assign expression value to specified M-variable
M{constant}..{constant}={constant} – Assign constant value to specified range of M-variable(s)
M{constant}->{definition} – Define M-variable as specified
M{constant}->* – Erase M-variable definition; usable as non-pointer variable
M{constant}[..{constant}] – Report M-variable value(s) to host
M{constant}[..{constant}]-> – Report M-variable definition(s) to host

Buffer Control Commands

OPEN PROG{constant} – Open specified motion program buffer for entering/editing
OPEN ROT – Open all defined rotary program buffers for ASCII entry
OPEN BIN ROT – Open all defined rotary program buffers for binary entry
OPEN PLC{constant} – Open specified PLC program buffer for entry
CLOSE – Close buffer currently opened on this port
CLOSE ALL – Close buffer currently opened on any port
CLEAR – Erase contents of opened buffer
CLEAR ALL – Erase all motion and uncompiled PLC program buffers
CLEAR ALL PLCS – Erase all uncompiled PLC program buffers
DEFINE GATHER [{constant}] – Set up a data-gathering buffer using all open memory [or of specified size]
DELETE GATHER – Erase the data gathering buffer
GATHER [TRIGGER] – Start data gathering [on external trigger]
ENDGATHER – Stop data gathering
DELETE PLCC{constant} – Erase specified compiled PLC program
DEFINE TBUF{constant} – Set up specified number of axis transformation matrices
DELETE TBUF – Erase all axis transformation matrices
DEFINE UBUFFER{constant} – Set up a user buffer of specified number of words
DELETE ALL – Erase all DEFINEd buffers
DELETE ALL TEMP – Erase all DEFINEd buffers with temporary contents

MACRO Ring Commands

MACROASCII{master#} – Put this PMAC port in pass-through mode so communications are passed through MACRO to specified other master

MACROAUX{node#},{param#} – Report MACRO Type 0 auxiliary parameter value from slave node

MACROAUX{node#},{param#}={constant} – Set MACRO Type 0 auxiliary parameter value in slave node

MACROAUXREAD{node#},{param#},{variable} – Copy MACRO Type 0 auxiliary parameter value from slave node to PMAC variable

MACROAUXWRITE{node#},{param#},{variable} – Copy from PMAC variable to MACRO Type 0 auxiliary parameter value in slave node

MACROMST{master#},{master variable} – Report variable value from remote MACRO master through Type 1 MACRO protocol

MACROMST{master#},{master variable}={constant} – Set variable value on remote MACRO master through Type 1 MACRO protocol

MACROMSTASCII{master #} – Put this ring-controller Turbo PMAC in pass-through mode to other master on ring

MACROMSTREAD{master#},{master variable},{ring-master variable} – Copy variable value from remote MACRO master into own variable through Type 1 MACRO protocol

MACROMSTWRITE{master#},{master variable},{ring-master variable} – Copy variable value to remote MACRO master from own variable through Type 1 MACRO protocol

MACROSLAVE{command},{node#} – Send command to slave node with Type 1 protocol

MACROSLAVE{node#},{slave variable} – Report slave node variable value with Type 1 MACRO protocol

MACROSLAVE{node#},{slave variable}={constant} – Set slave node variable value with Type 1 MACRO protocol

MACROSLVREAD{node#},{slave variable},{PMAC variable} – Copy from slave node variable to PMAC variable with Type 1 MACRO protocol

MACROSLVWRITE{node#},{slave variable},{PMAC variable} – Copy PMAC variable to slave node variable with Type 1 MACRO protocol

MACROSTASCII{station #} – Put this ring-controller Turbo PMAC in pass-through mode to other station on ring

STN={constant} – Set MACRO-ring station-order number

On-line Coordinate System Commands

(These act immediately on currently addressed coordinate system)

Axis Definition Commands

#n->[{constant}]{axis}[+{constant}] – Define axis in terms of motor #, scale factor, and offset

Examples: **#1->X**
#4->2000A+500

#n->[{constant}]{axis}[+[{constant}]{axis}[+[{constant}]{axis}]][+{constant}] – Define 2 or 3 axes in terms of motor #, scale factors, and offset. Valid only within XYZ or UVW groupings.

Examples: **#1->8660X-5000Y**
#2->5000X+8660Y+5000

#n->I[+{constant}] – Assign motor as inverse kinematic axis
#n-> – Report axis definition of motor n in this C. S.
#n->0 – Erase axis definition of motor n in this C. S.
UNDEFINE – Erase definition of all axes in this C. S.

General Coordinate-System Commands

%{constant} – specify feedrate override value
$$ – Establish phase reference (if necessary) and close loop for all motors in C.S.
$$* – Read absolute position value for all motors in C.S.

Coordinate-System Reporting Commands

?? – Report coordinate system status in hex ASCII form
% – report current feedrate override value to host
LIST PC – Report next line to be calculated in motion program
LIST PE – Report executing motion line in motion program
LIST ROTARY – Report contents of coordinate system's rotary motion program buffer
MOVETIME – Report time left in presently executing move
PC – Report address of next line to be calculated in motion program
PE – Report address of executing motion line in motion program
PR – Report number of lines still to be calculated in rotary buffer

Program Control Commands

/ - Stop execution at end of currently executing move
**** - Execute quickest stop in lookahead that does not violate constraints
R – Run current program
S – Do one step of current program
B[{constant}] – Set program counter to specified location
H – Feed hold for coordinate system
A – Abort present program or move starting immediately
ABR[{constant}] – Abort present program and restart or start another program
Q – Halt program; stop moves at end of last calculated program command
MFLUSH – Erase contents of synchronous M-variable stack without executing

Coordinate-System Variable Commands

Q{data}={expression} – Assign expression value to specified Q-variable
Q{constant}..{constant}={constant} – Assign constant value to specified range of Q-variable(s)
Q{constant}[..{constant}] – Report Q-variable value(s) to host

Axis Attribute Commands

{axis}={expression} – Change value of commanded axis position
Z -- Make present commanded position of all axes in coordinate system equal to zero
INC [({axis}[,{axis}...])] – Make all [or specified] axes do their moves incrementally
ABS [({axis}[,{axis}...])] – Make all [or specified] axes do their moves absolute
FRAX ({axis}[,{axis}...]) – Make specified axes to be used in vector feedrate calculations
NOFRAX – Remove all axes from list of vector feedrate axes

PMATCH – Re-match coordinate system axis positions to motor commanded positions (used in case axis definition or motor position changed since last *axis* move)

Buffer Control Commands

DEFINE ROT {constant} – Establish rotary motion program buffer of specified word size for the addressed coordinate system
DELETE ROT – Erase rotary motion program buffer for addressed coordinate system
DEFINE LOOKAHEAD {constant},{constant} – Establish lookahead buffer for the addressed coordinate system with the specified number of motion segments and synchronous M-variable assignments
DELETE LOOKAHEAD – Erase lookahead buffer for addressed coordinate system
DEFINE CCBUFFER – Establish extended cutter-compensation block buffer
DELETE CCBUFFER – Erase extended cutter-compensation block buffer
LEARN – Read present commanded positions and add as axis commands to open program buffer
OPEN FORWARD – Open forward-kinematic program buffer for entry
OPEN INVERSE – Open inverse-kinematic program buffer for entry

On-line Motor Commands

(These act immediately on the currently addressed motor. Except for the reporting commands, these commands are rejected if the motor is in a coordinate system that is currently running a motion program.)

General Motor Commands

$ – Establish phase reference (if necessary) and close loop for motor
$* – Read absolute position for motor
HOME – Perform homing search move for motor
HOMEZ – Set present commanded position for motor to zero
K – Kill output for motor
O{constant} – Set open-loop servo output of specified magnitude

Jogging Commands

J+ – Jog motor indefinitely in positive direction
J- – Jog motor indefinitely in negative direction
J/ – Stop jogging motor; also restore to position control
J= – Jog motor to last pre-jog or pre-handwheel position
J={constant} – Jog motor to specified position
J=* – variable jog-to-position
J:{constant} – Jog motor specified distance from current commanded position
J:* – Variable incremental jog from current commanded position
J^{constant} – Jog motor specified distance from current actual position
J^* – Variable incremental jog from current actual position
{jog command}^{constant} – Jog until trigger, final value specifies distance from trigger position to stop

Motor Reporting Commands

P – Report position of motor
V – Report velocity of motor
F – Report following error of motor
? – Report status words for motor in hex ASCII form
LIST BLCOMP – Report contents of backlash compensation table for motor
LIST BLCOMP DEF – Report definition of backlash compensation table for motor
LIST COMP – Report contents of position compensation table for motor
LIST COMP DEF – Report definition of position compensation table for motor
LIST TCOMP – Report contents of torque compensation table for motor
LIST TCOMP DEF – Report definition of torque compensation table for motor

Buffer Control Commands

DEFINE BLCOMP {entries},{count length} – Establish backlash compensation table for motor; to be filled by specified number of values
DELETE BLCOMP – Erase backlash compensation table for motor
DEFINE COMP {entries},[#{source},[#{target},]],{count length} – Establish leadscrew compensation table for motor; to be filled by specified number of values
DEFINE COMP {rows}.{columns}, [#{source1}, [#{source2}, [#{target},]]],{count length1},{count length2} – Establish two-dimensional leadscrew compensation table for motor; to be filled by specified number of values
DELETE COMP – Erase leadscrew compensation table for motor
DEFINE TCOMP {entries},{count length} – Establish torque compensation table for motor; to be filled by specified number of values
DELETE TCOMP – Erase torque compensation table for motor

Motion Program Commands

Move Commands

{axis}{data}[{axis}{data}] – Simple position movement statement; can be used in LINEAR, RAPID, or SPLINE modes
Example: **X1000 Y(P1) Z(P2*P3)**
{axis}{data}:{data}[{axis}{data}:{data}...] – Position/velocity move statement; to be used only in PVT mode
Example: **X5000:750 Y3500:(P3) A(P5+P6):100**
{axis}{data}^{data}[{axis}{data}^{data}...] – Move-until-trigger statement, to be used only in RAPID mode
{axis}{data}[{axis}{data}...][{vector}{data}...] – Arc move statement; to be used only in CIRCLE mode; vector is to circle center
Example: **X2000 Y3000 Z1000 I500 J300 K500**
{axis}{data}[{axis}{data}...] R{data} -- Arc move statement; to be used only in CIRCLE mode; R-value is radius magnitude
Example: **X2000 Y3000 Z1000 R500**
DWELL{data} – Zero-distance statement; fixed time base
DELAY{data} – Zero-distance; variable time base
HOME{constant}[,{constant}...] – Homing search move statement for specified *motor(s)*

HOMEZ{constant}[,{constant}...] – Zero-move homing statement for specified *motor(s)*

Move Mode Commands

LINEAR – Set blended linear interpolation move mode
RAPID – Set minimum-time point-to-point move mode
CIRCLE1 – Set clockwise circular interpolation move mode
CIRCLE2 – Set counterclockwise circular interpolation move mode
PVT{data} – Set position/velocity/time move mode (parabolic velocity profiles)
SPLINE1 – Set uniform cubic spline move mode
SPLINE2 – Set non-uniform cubic spline move mode
CC0 – Set cutter radius compensation off
CC1 – Set 2D cutter radius compensation on left
CC2 – Set 2D cutter radius compensation right
CC3 – Turn on 3D cutter radius compensation

Axis Attribute Commands

ABS [({axis}[,{axis},...])] – Set absolute move mode for all [or specified] axes
INC [({axis}[,{axis},...])] – Set incremental move mode for all [or specified] axes
FRAX ({axis}[,{axis}...]) – Set specified axes as vector feedrate axes
NOFRAX – Remove all axes from list of vector feedrate axes
NORMAL{vector}{data}[{vector}{data}...] – Specify normal vector to plane for circular moves and cutter compensation
PSET{axis}{data}[{axis}{data}...] – Assign new values to present axis positions
CCR{data} – Specify 2D/3D cutter radius compensation value (modal)
TR{data} – Specify tool-shaft radius for 3D compensation
TSEL{data} – Select specified axis transformation matrix
TINIT – Initialize selected axis transformation matrix as identity matrix
ADIS{data} – Set displacement vector of selected matrix to values starting with specified Q-variable
IDIS{data} – Increment displacement vector of selected matrix to values starting with specified Q-variable
AROT{data} – Set rotation/scaling portion of selected matrix to values starting with specified Q-variable
IROT{data} – Incrementally change rotation/scaling portion of selected matrix by multiplying it with values starting with specified Q-variable
SETPHASE{constant}[,{constant}…] – Set commutation phase position value for specified motor(s)

Move Attribute Commands

TM{data} – Specify move time (modal)
F{data} – Specify move speed (modal)
TA{data} – Specify move acceleration time (modal)
TS{data} – Specify acceleration S-curve time (modal)
NX{data} – Specify surface-normal vector X-component for 3D comp
NY{data} – Specify surface-normal vector Y-component for 3D comp
NZ{data} – Specify surface-normal vector Z-component for 3D comp
TX{data} – Specify tool-orientation vector X-component for 3D comp
TY{data} – Specify tool-orientation vector Y-component for 3D comp

TZ{data} – Specify tool-orientation vector Z-component for 3D comp

Variable Assignment Commands

I{data}={expression} – Assign expression value to specified I-variable
P{data}={expression} – Assign expression value to specified P-variable
Q{data}={expression} – Assign expression value to specified Q-variable
M{data}={expression} – Assign expression value to specified M-variable
M{data}=={expression} – Assign expression synchronous with start of next move
M{data}&={expression} – 'AND' M-variable with expression synchronous with start of next move
M{data}|={expression} – 'OR' M-variables with expression synchronous with start of next move
M{data}^={expression} – 'XOR' M-variables with expression synchronous with start of next move

Program Logic Control

N{constant} – Line label
O{constant} – Line label, alternate entry form
GOTO{data} – Jump to specified line label; no return
GOSUB{data}[{letter}{axis}...] – Jump to specified line label [with arguments] and return
CALL{data}[.{data}][{letter}{axis}...] – Jump to specified program [and label] [with arguments] and return.
RETURN –- Return program operation to most recent **GOSUB** or **CALL**
READ ({letter} [,{letter}...]) – Read argument into subroutine/subprogram from calling line
G{data} – **Gnn[.mmm]** interpreted as **CALL 1000.nnmmm** (PROG 1000 provides subroutines for desired G-Code actions)
M{data} – **Mnn[.mmm]** interpreted as **CALL 1001.nnmmm** (PROG 1001 provides subroutines for desired M-Code actions)
T{data} – **Tnn[.mmm]** interpreted as **CALL 1002.nnmmm** (PROG 1002 provides subroutines for desired T-Code actions.)
D{data} – **Dnn[.mmm]** interpreted as **CALL 1003.nnmmm** (PROG 1003 provides subroutines for desired D-Code actions.)
S{data} – Set Q127 to value of **{data}** (spindle command)
PRELUDE1{call command} – Enable modal execution of call command before subsequent moves
PRELUDE0 – Disable modal **PRELUDE** calls
IF ({condition}){action} – Conditionally execute single-line action
IF ({condition}) – Conditionally execute following statements
ELSE {action} – Execute single-line action on previous false condition
ELSE –- Execute following statements on previous false IF condition
ENDIF – Mark end of conditionally executed branch statements
WHILE ({condition}){action} – Do single-line action as long as condition true
WHILE ({condition}) – Execute following statements as long as condition true
ENDWHILE – Mark end of conditionally executed loop statements
BLOCKSTART – So all commands until **BLOCKSTOP** to execute on Step
BLOCKSTOP – End of "single-step" statements starting on **BLOCKSTART**
STOP – Halt program execution, ready to resume

WAIT – Use with **WHILE** to halt execution while condition true
LOCK{constant},P{constant} – Check/set process-locking bit
UNLOCK{constant} – Clear process-locking bit

Miscellaneous Commands

COMMAND"{command}" – Issue text command, no response
COMMAND^{letter} – Issue control character command, no response
COMMANDS"{command}" – Issue text command, respond to main serial port
COMMANDS^{letter} – Issue control character command, respond to main serial port
COMMANDP"{command}" – Issue text command, respond to parallel bus port
COMMANDP^{letter} – Issue control character command, respond to parallel bus port
COMMANDR"{command}" – Issue text command, respond to DPRAM ASCII port
COMMANDR^{letter} – Issue control character command, respond to DPRAM ASCII port
COMMANDA"{command}" – Issue text command, respond to auxiliary serial port
COMMANDA^{letter} – Issue control character command, respond to auxiliary serial port
SENDS"{message}" – Transmit message over main serial interface
SENDP"{message}" – Transmit message over parallel bus interface
SENDR"{message}" – Transmit message over DPRAM ASCII interface
SENDA"{message}" – Transmit message over auxiliary serial interface
DISPLAY [{constant}] "{message}" – Send message to LCD display [starting at specified location]
DISPLAY {constant}, {constant}.{constant}, {variable} –- Send variable value to LCD using specified location and format
ENABLE PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s], starting at top of program
DISABLE PLC{constant}[,{constant}...] – Disable operation of specified interpreted PLC program[s]
PAUSE PLC{constant}[,{constant}...] – Suspend operation of specified interpreted PLC program[s]
RESUME PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s], starting at paused point
ENABLE PLCC{constant}[,{constant}...] – Enable operation of specified compiled PLC program[s]
DISABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]

PLC Program Commands

Conditions

IF ({condition}) – Conditionally execute following statements
WHILE ({condition}) – Execute following statements as long as condition true
AND ({condition}) – Forms compound condition with **IF** or **WHILE**
OR ({condition}) – Forms compound condition with **IF** or **WHILE**
ELSE – Execute following statements on previous false IF condition
ENDIF – Mark end of conditionally executed branch statements
ENDWHILE – Mark end of conditionally executed loop statements

Variable Value Assignment

I{data}={expression} – assigns expression value to specified I-variable
P{data}={expression} – assigns expression value to specified P-variable
Q{data}={expression} – assigns expression value to specified Q-variable
M{data}={expression} – assigns expression value to specified M-variable

Command Issuance

ADDRESS#n&n – Modally address specified motor and/or coordinate system
ADDRESS#Pn – Modally address motor specified in P-variable
ADDRESS&Pn – Modally address coordinate system specified in P-variable
COMMAND"{command}" – Issue text command, no response
COMMAND^{letter} – Issue control character command, no response
COMMANDS"{command}" – Issue text command, respond to main serial port
COMMANDS^{letter} – Issue control character command, respond to main serial port
COMMANDP"{command}" – Issue text command, respond to parallel bus port
COMMANDP^{letter} – Issue control character command, respond to parallel bus port
COMMANDR"{command}" – Issue text command, respond to DPRAM ASCII port
COMMANDR^{letter} – Issue control character command, respond to DPRAM ASCII port
COMMANDA"{command}" – Issue text command, respond to auxiliary serial port
COMMANDA^{letter} – Issue control character command, respond to auxiliary serial port

Message Transmission and Display

SENDS"{message}" – Transmit message over main serial interface
SENDP"{message}" – Transmit message over parallel bus interface
SENDR"{message}" – Transmit message over DPRAM ASCII interface
SENDA"{message}" – Transmit message over auxiliary serial interface
DISPLAY [{constant}] "{message}" – Send message to LCD display [starting at specified location]
DISPLAY {constant}, {constant}.{constant}, {variable} –- Send variable value to LCD using specified location and format

PLC Operational Control Commands

ENABLE PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s], starting at top of program
DISABLE PLC{constant}[,{constant}...] – Disable operation of specified interpreted PLC program[s]
PAUSE PLC{constant}[,{constant}...] – Suspend operation of specified interpreted PLC program[s]
RESUME PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s], starting at paused point
ENABLE PLCC{constant}[,{constant}...] – Enable operation of specified compiled PLC program[s]
DISABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]

MACRO Ring Commands

MACROAUXREAD{node#},{param#},{variable} – Copy MACRO Type 0 auxiliary parameter value from slave node to PMAC variable

MACROAUXWRITE{node#},{param#},{variable} – Copy from PMAC variable to MACRO Type 0 auxiliary parameter value in slave node

MACROMSTREAD{master#},{master variable},{ring-master variable} – Copy variable value from remote MACRO master into own variable through Type 1 MACRO protocol

MACROMSTWRITE{master#},{master variable},{ring-master variable} – Copy variable value to remote MACRO master from own variable through Type 1 MACRO protocol

MACROSLVREAD{node#},{slave variable},{PMAC variable} – Copy from slave node variable to PMAC variable with Type 1 MACRO protocol

MACROSLVWRITE{node#},{slave variable},{PMAC variable} – Copy PMAC variable to slave node variable with Type 1 MACRO protocol

TURBO PMAC GLOBAL I-VARIABLES

General Global Setup I-Variables

I0 Serial Card Number

Range: $0 to $F (0 to 15)

Units: None

Default: $0

I0 controls the Turbo PMAC card number for software addressing purposes on a multi-drop serial communications cable. If I1 is set to 2 or 3, the Turbo PMAC must be addressed with the **@n** command, where **n** matches the value of I0 on the board, before it will respond. If the Turbo PMAC receives the **@n** command, where n does not match I0 on the board, it will stop responding to commands on the serial port. No two boards on the same serial cable may have the same value of I0.

If the **@@** command is sent over the serial port, all boards on the cable will respond to action commands. However, only the board with I0 set to 0 will respond to the host with handshake characters (no data responses are permitted in this mode). All boards on the cable will respond to control-character action commands such as **<CTRL-R>**, regardless of the current addressing.

Note:

RS-422 serial interfaces must be used on all Turbo PMAC boards for multi-drop serial communications; this will not work with RS-232 interfaces. If the RS-422 interface is not present as a standard feature on the PMAC2 board, the Option 9L serial converter module must be purchased. It is possible to use an RS-232 interface on the host computer, connected to the RS-422 ports on the Turbo PMAC boards.

Typically, multiple Turbo PMAC boards on the same serial cable will share servo and phase clock signals over the serial port cable for tight synchronization. If the servo and phase clock lines are connected between multiple Turbo PMACs, only one of the Turbo PMAC boards can be set up to output these clocks (E40-E43 ON for Turbo PMAC(1); E1 jumper OFF for Turbo PMAC2). All of the other boards in the chain must be set up to input these clocks (any of E40-E43 OFF for Turbo PMAC(1); E1 jumper ON for Turbo PMAC2).

Note:

Any Turbo PMAC board set up to input these clocks is expecting its Servo and Phase clock signals externally from a Card 0. If it does not receive these clock signals, the watchdog timer will immediately shut down the board and the red LED will light.

If the Turbo PMAC is set to receive external Servo and Phase clock signals for synchronization purposes, but is not using multi-drop serial communications, I0 does not need to be changed from 0.

To set up a board to communicate as Card 1 to Card 15 on a multi-drop serial cable, first communicate with the board as Card 0. Set I0 to specify the card number (software address) that the board will have on the multi-drop cable. Also, set I1 to 2 to enable the serial software addressing. Store these values to the non-volatile flash memory with the **SAVE** command. Then turn off power; if the board is to input its clocks, remove any jumper E40-E43 (Turbo PMAC(1)) or put a jumper on E1 (Turbo PMAC2), connect the multi-drop cable, and restore power to the system.

I1 Serial Port Mode

Range: 0 to 3

Units: None

Default: 0

I1 controls two aspects of how Turbo PMAC uses its main serial port. The first aspect is whether PMAC uses the CS (CTS) handshake line to decide if it can send a character out the serial port. The second aspect is whether PMAC will require software card addressing, permitting multiple cards to be daisy-chained on a single serial line.

There are four possible values of I1, covering all the possible combinations:

Setting	Meaning
0	CS handshake used; no software card address required
1	CS handshake not used; no software card address required
2	CS handshake used; software card address required
3	CS handshake not used; software card address required

When CS handshaking is used (I1 is 0 or 2), Turbo PMAC waits for the CS line to go true before it will send a character. This is the normal setting for real serial communications to a host; it allows the host to hold off Turbo PMAC messages until it is ready.

When CS handshaking is not used (I1 is 1 or 3), Turbo PMAC disregards the state of the CS input and always sends the character immediately. This mode permits Turbo PMAC to "output" messages, values, and acknowledgments over the serial port even when there is nothing connected, which can be valuable in stand-alone and PLC-based applications where there are **SENDS** and **CMDS** statements in the program. If these strings cannot be sent out the serial port, they can back up, stopping program execution.

When software addressing is not used (I1 is 0 or 1), Turbo PMAC assumes that it is the only card on the serial line, so it always acts on received commands, sending responses back over the line as appropriate.

When software addressing is used (I1 is 2 or 3), Turbo PMAC assumes that there are other cards on the line, so it requires that it be addressed (with the **@{card}** command) before it responds to commands. The **{card}** number in the command must match the card number set up with variable I0.

I2 Control Panel Port Activation

Range: 0 to 3

Units: None

Default: 0

I2 allows the enabling and disabling of the control panel discrete inputs on the JPAN connector, should this connector exist. I2=0 enables these control panel functions; I2=1 disables them. When disabled, these inputs can be used as general purpose I/O. The reset, handwheel, and wiper inputs on the JPAN connector are not affected by I2.

On a Turbo PMAC(1) board, when I2=0, the IPOS, EROR and F1ER status lines to JPAN and the Programmable Interrupt Controller (PIC), and the BREQ status line to the PIC, reflect the hardware-selected coordinate system (by BCD-coded lines FPDn/ on JPAN); when I2=1, they reflect the software-addressed coordinate system (&n). (On a Turbo PMAC2, the lines to the PIC always reflect the software-addressed coordinate system.)

When I2=3, discrete inputs on a JPAN connector are disabled, and the dual-ported RAM control panel functions are enabled. Refer to the descriptions of DPRAM functions for more detail.

I3 I/O Handshake Control

Range: 0 to 3

Units: None

Default: 1

I3 controls what characters, if any, are used by Turbo PMAC to delimit a transmitted line, and whether PMAC issues an acknowledgment (handshake) of a command.

Note:

With communications checksum enabled (I4=1), checksum bytes are added after the handshake character bytes.

Valid values of I3 and the modes they represent are:

0: Turbo PMAC does not acknowledge receipt of a valid command. It returns a **<BELL>** character on receipt of an invalid command. Messages are sent without beginning or terminating **<LF>** (line feed); simply as **DATA <CR>** (carriage return).

1. Turbo PMAC acknowledges receipt of a valid **<CR>**-terminated command with a **<LF>**; of an invalid command with a **<BELL>** character. Messages are sent as **<LF> DATA <CR> [<LF> DATA <CR> ...] <LF>**. (The final **<LF>** is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [**SEND** or **CMD**]). This setting is good for communicating with dumb terminal display programs.
2. Turbo PMAC acknowledges receipt of a valid **<CR>**-terminated command with an **<ACK>**; of an invalid command with a **<BELL>** character. Messages are sent as **DATA <CR> [DATA <CR> ...] <ACK>**. (The final **<ACK>** is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [**SEND** or **CMD**]). This is probably the best setting for fast communications with a host program without terminal display.
3. Turbo PMAC acknowledges receipt of a valid **<CR>**-terminated command with an **<ACK>**; of an invalid command with a **<BELL>** character. Messages are sent as **<LF> DATA <CR> [<LF> DATA <CR> ...] <ACK>**. (The final **<ACK>** is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [**SEND** or **CMD**]).

Note:

I3 does not affect how DPRAM ASCII communications are performed.

Examples:

With I3=0:

```
#1J+<CR>.....          ; Valid command not requiring data response
.....                  ; No acknowledging character
UUU<CR>.....           ; Invalid command
<BELL>.....            ; PMAC reports error
P1..3<CR> .....        ; Valid command requiring data response
25<CR>50<CR>75<CR>     ; PMAC responds with requested data
```

With I3=1:

```
#1J+<CR>.....          ; Valid command not requiring data response
<LF> .....             ; Acknowledging character
UUU<CR>.....           ; Invalid command
<BELL>.....            ; PMAC reports error
P1..3<CR> .....        ; Valid command requiring data response
<LF>25<CR><LF>50<CR><LF>75<CR><LF>
```

```
                                ; PMAC responds with requested data
```

With I3=2:

```
#1J+<CR>.....                   ; Valid command not requiring data response
<ACK>.......                    ; Acknowledging character
UUU<CR> .....                   ; Invalid command
<BELL>.....                     ; PMAC reports error
P1..3<CR> .....                 ; Valid command requiring data response
25<CR>50<CR>75<CR><ACK>
                                ; PMAC responds with requested data
```

With I3=3:

```
#1J+<CR>.....                   ; Valid command not requiring data response
<ACK>.....                      ; Acknowledging character
UUU<CR> .....                   ; Invalid command
<BELL>.....                     ; PMAC reports error
P1..3<CR> .....                 ; Valid command requiring data response
<LF>25<CR><LF>50<CR><LF>75<CR><ACK>
.....                           ; PMAC responds with requested data
```

I4 Communications Integrity Mode

Range: 0 to 3

Units: None

Default: 1

This parameter permits Turbo PMAC to compute checksums of the communications bytes (characters) sent either way between the host and Turbo PMAC, and also controls how Turbo PMAC reacts to serial character errors (parity and framing), if found. Parity checking is only available on Turbo PMAC(1) boards; it is enabled only if jumper E49 is OFF.

The possible settings of I4 are:

Setting	Meaning
0	Checksum disabled, serial errors reported immediately
1	Checksum enabled, serial errors reported immediately
2	Checksum disabled, serial errors reported at end of line
3	Checksum enabled, serial errors reported at end of line

Communications Checksum: With I4=1 or 3, Turbo PMAC computes the checksum for communications in either direction and sends the checksum to the host. It is up to the host to do the comparison between PMAC's checksum and the checksum it computed itself. Turbo PMAC does not do this comparison. The host should never send a checksum byte to Turbo PMAC.

Host-to-Turbo-PMAC Checksum: Turbo PMAC will compute the checksum of a communications line sent from the host to Turbo PMAC. The checksum does not include any control characters sent (not even the final Carriage-Return). The checksum is sent to the host immediately following the acknowledging handshake character (**<LF>** or **<ACK>**), if any. Note that this acknowledging and handshake comes after any data response to the command (and its checksum!). If Turbo PMAC detects an error in the line through its normal syntax checking, it will respond with the **<BELL>** character, but will not follow this with a checksum byte.

Note:

The on-line command **<CTRL-N>** can be used to verify the checksum of a command line before the **<CR>** has been sent. The use of **<CTRL-N>** does not affect how I4 causes Turbo PMAC to report a checksum after the **<CR>** has been sent.

Turbo-PMAC-to-Host Checksum: Turbo PMAC will compute the checksum of any communications line it sends to the host. This checksum includes control characters sent with the line, including the final **<carriage-return>**. The checksum is sent immediately following this **<carriage-return>**. On a multiple-line response, one checksum is sent for each line. Note that this checksum is sent before the checksum of the command line that caused the response.

For more details on checksum, refer to the Writing a Host Communications Program section of the manual.

Serial character errors: If Turbo PMAC detects a serial character error, it will set a flag so that the entire command line will be rejected as having a syntax error after the **<CR>** is sent. With I4=0 or 1, it will also send a **<BELL>** character to the host immediately on detecting the character error. Note that this mode will catch a character error on the **<CR>** as well, whereas in the I4=2 or 3 mode, the host would have to catch an error on the **<CR>** character by the fact that Turbo PMAC would not respond (because it never saw a **<CR>**).

I5 PLC Program Control

Range: 0 to 3

Units: None

Default: 1

I5 controls which PLC programs may be enabled. There are two types of PLC programs: the foreground programs (PLC 0 and PLCC 0), which operate at the end of servo interrupt calculations, with a repetition rate determined by I8 (PLC 0 and PLCC 0 should be used only for time-critical tasks and should be short); and the background programs (PLC 1 to PLC 31, PLCC 1 to PLCC 31) which cycle repeatedly in background as time allows. I5 controls these as follows:

Setting	Meaning
0	Foreground PLCs off; background PLCs off
1	Foreground PLCs on; background PLCs off
2	Foreground PLCs off; background PLCs on
3	Foreground PLCs on; background PLCs on

Note that an individual PLC program still needs to be enabled to run -- a proper value of I5 merely permits it to be run. Any PLC program that exists at power-up or reset is automatically enabled (even if the saved value of I5 does not permit it to run immediately); also, the **ENABLE PLC n** or **ENABLE PLCC n** command enables the specified program(s). A PLC program is disabled either by the **DISABLE PLC n** or **DISABLE PLCC n** command, or by the **OPEN PLC n** command. A **CLOSE** command does not automatically re-enable the PLC program -- it must be done explicitly.

I6 Error Reporting Mode

Range: 0 to 3

Units: None

Default: 1

I6 controls how Turbo PMAC reports errors in command lines. When I6 is set to 0 or 2, PMAC reports any error only with a **<BELL>** character. When I6 is 0, the **<BELL>** character is given for invalid commands issued both from the host and from Turbo PMAC programs (using **CMD"{command}"**). When I6 is 2, the **<BELL>** character is given only for invalid commands from the host; there is no response to invalid commands issued from Turbo PMAC programs. (In no mode is there a response to valid commands issued from PMAC programs.)

When I6 is set to 1 or 3, an error number message can be reported along with the **<BELL>** character. The message comes in the form of **ERRnnn<CR>**, where **nnn** represents the three-digit error number. If I3 is set to 1 or 3, there is a **<LF>** character in front of the message.

When I6 is set to 1, the form of the error message is **<BELL>{error message}**. This setting is the best for interfacing with host-computer driver routines. When I6 is set to 3, the form of the error message is **<BELL><CR>{error message}**. This setting is appropriate for use with the PMAC Executive Program in terminal mode.

Currently, the following error messages can be reported:

Error	Problem	Solution
ERR001	Command not allowed during program execution	(should halt program execution before issuing command)
ERR002	Password error	(should enter the proper password)
ERR003	Data error or unrecognized command	(should correct syntax of command)
ERR004	Illegal character: bad value (>127 ASCII) or serial parity/framing error	(should correct the character and or check for noise on the serial cable)
ERR005	Command not allowed unless buffer is open	(should open a buffer first)
ERR006	No room in buffer for command	(should allow more room for buffer -- DELETE or CLEAR other buffers)
ERR007	Buffer already in use	(should CLOSE currently open buffer first)
ERR008	MACRO auxiliary communications error	(should check MACRO ring hardware and software setup)
ERR009	Program structural error (e.g. ENDIF without IF)	(should correct structure of program)
ERR010	Both overtravel limits set for a motor in the C. S.	(should correct or disable limits)
ERR011	Previous move not completed	(should Abort it or allow it to complete)
ERR012	A motor in the coordinate system is open-loop	(should close the loop on the motor)
ERR013	A motor in the coordinate system is not activated	(should set Ix00 to 1 or remove motor from C.S.)
ERR014	No motors in the coordinate system	(should define at least one motor in C.S.)
ERR015	Not pointing to valid program buffer	(should use B command first, or clear out scrambled buffers)
ERR016	Running improperly structured program (e.g. missing ENDWHILE)	(should correct structure of program)
ERR017	Trying to resume after H or Q with motors out of stopped position	(should use J= to return motor[s] to stopped position)
ERR018	Attempt to perform phase reference during move, move during phase reference., or enabling with phase clock error.	(should finish move before phase reference, finish phase reference before move, or fix phase clock source problem)
ERR019	Illegal position-change command while moves stored in CCBUFFER	(should pass through section of Program requiring storage of moves in CCBUFFER, or abort)

I7 Phase Cycle Extension

Range: 0 to 15

Units: Phase Clock Cycles

Default: 0

I7 permits the extension of the software phase update period to multiple Phase clock interrupt periods. The software phase update algorithms, which do the commutation and current loop calculations for motors, are executed every (I7+1) Phase clock cycles. In other words, the phase update cycle is extended by I7 phase clock cycles.

The hardware Phase clock period (frequency) is controlled by jumpers E98 and E29-E33 on a Turbo PMAC(1), variables I7000 and I7001 on a Turbo PMAC2 that is not Ultralite, or variables I6800 and I6801 on a Turbo PMAC2 Ultralite.

Most Turbo PMAC users will leave I7 at the default value of 0, so that phase update algorithms are executed every phase clock cycle. There are two reasons to extend the phase update cycle by setting I7 greater than 0.

First, if the Turbo PMAC is doing direct PWM control of motors over the MACRO ring, it is advisable to set I7 to 1 so that the MACRO ring, which operates on the hardware phase clock, cycles twice per software phase cycle. This will eliminate one phase cycle delay in the closing of the current loops, which permits higher gains and higher performance. For example, the hardware phase clock could be set to 18 kHz, but with I7=1, the current loop would be closed at a reasonable 9 kHz.

Second, if many multiplexed A/D converters from the on-board Option 12, or ACC-36 boards, are used for servo feedback, I7 can be set greater than zero to ensure that each A/D converter is processed once per servo cycle. One pair of multiplexed ADCs is processed each hardware phase clock cycle.

For example, if 8 pairs of multiplexed ADCs needed to be processed each 440 µsec (2.25 kHz) servo cycle, and the software phase update were desired to be at 220 µsec (4.5 kHz), the phase clock update would be set to 18 kHz (18/8 = 2.25) to get through all 8 ADC pairs each servo cycle, I7 would be set to 3 (18/[3+1] = 4.5) to get the software phase update at 4.5 kHz, and the servo cycle clock divider would be set to divide-by-8 (E3-E6 on Turbo PMAC(1), I7002=7 on non-Ultralite Turbo PMAC2, I6802=7 on Turbo PMAC2 Ultralite).

There must be an integer number of software phase updates in a Servo clock period. For example if the Servo clock frequency is ¼ the Phase clock frequency (I7002 or I6802 = 3), the legitimate values of I7 are 0, which provides 4 software phase updates per servo clock period; 1, which provides 2 updates per period; and 3, which provides 1 update per period. Note that this rule means that the software phase update period must never be longer than the servo clock period.

I8 Real-Time Interrupt Period

Range: 0 to 255

Units: Servo Clock Cycles

Default: 2

I8 controls how often certain time-critical tasks, such as PLC 0, PLCC 0, and checking for motion program move planning, are performed. These tasks are performed every (I8+1) servo cycles, at a priority level called the "real-time interrupt" (RTI). A value of 2 means that these tasks are performed after every third servo interrupt, 3 means every fourth interrupt, and so on. The vast majority of users can leave this at the default value. In some advanced applications that push PMAC's speed capabilities, tradeoffs between performance of these tasks and the calculation time they take may have to be evaluated in setting this parameter.

Turbo PMAC cannot compute more than one programmed move block, or more than one internal move segment if the coordinate system is in segmentation mode (Isx13 > 0), per real-time interrupt. If very high programmed move block rates (small move times), or very high segmentation rates (small segmentation times) are desired, it is best to make I8 as small as possible (preferably 0). This will ensure that the calculations are done every move or segment, and that they are started as early as possible in the move or segment to maximize the likelihood of completing the calculations in time.

If move or segment calculations are not completed in time, Turbo PMAC will abort the program automatically with a run-time error.

Note:

A large PLC 0 with a small value of I8 can cause severe problems, because Turbo PMAC will attempt to execute the PLC program every I8 cycle. This can starve background tasks, including communications, background PLCs, and even updating of the watchdog timer, for time, leading to erratic performance or possibly even shutdown.

In multiple-card Turbo PMAC applications where it is very important that motion programs on the two cards start as closely together as possible, I8 should be set to 0. In this case, no PLC 0 should be running when the cards are awaiting a Run command. At other times, I8 may be set greater than 0 and PLC 0 re-enabled.

I9 Full/Abbreviated Listing Control

Range: 0 to 3

Units: None

Default: 2

I9 controls how Turbo PMAC reports program listings and variable values. I9 is a 2-bit value. Bit 0 whether short-form or long-form reporting is used; bit 1 controls whether address I-variable values are reported in decimal or hexadecimal form. The following table summarizes:

Setting	Meaning
0	Short form, decimal address I-variable return
1	Long form, decimal address I-variable return
2	Short form, hex address I-variable return
3	Long form, hex address I-variable return

When this parameter is 0 or 2 (bit 0 = 0), programs are sent back in abbreviated form for maximum compactness, and when I-variable values or M-variable definitions are requested, only the values or definitions are returned, not the full statements. When this parameter is 1 or 3 (bit 0 = 1), programs are sent back in full form for maximum readability. Also, I-variable values and M-variable definitions are returned as full command statements, which is useful for archiving and later downloading.

When this parameter is 0 or 1 (bit 1 = 0), I-variable values that specify PMAC addresses are returned in decimal form. When it is 2 or 3 (bit 1 = 1), these values are returned in hexadecimal form (with the '$' prefix). You are always free to send any I-variable values to PMAC either in hex or decimal, regardless of the I9 setting. This does not affect how I-variable assignment statements inside Turbo PMAC motion and PLC programs are reported when the program is listed.

I10 Servo Interrupt Time

Range: 0 to 8,388,607

Units: 1 / 8,388,608 msec

Default: 3,713,707 [Turbo PMAC(1): 442.71 µsec]
3,713,991 [Turbo PMAC2: 442.74 µsec]

This parameter tells Turbo PMAC how much time there is between servo interrupts (which is controlled by hardware circuitry), so that the interpolation software knows how much time to increment each servo interrupt.

The fundamental equation for I10 is:

$$I10 = \frac{8{,}388{,}608}{ServoFrequency(\,kHz\,)} = 8{,}388{,}608 * ServoTime(\,m\,sec)$$

On Turbo PMAC(1), the servo interrupt time is determined by the settings of hardware jumpers E98, E29-E33, and E3-E6. The proper value of I10 can be determined from the settings of these jumpers by the formula:

$$I10 = 232{,}107 * E98JumperFactor * PhaseJumperFactor * ServoJumperFactor$$

where the factors can be taken from the following:

E98 Setting	1-2	2-3
E98JumperFactor	1	2

Phase Jumper ON	E29	E30	E31	E32	E33
Phase Jumper Factor	16	8	4	2	1

$$ServoJumperFactor = 1 + E3 + (2 * E4) + (4 * E5) + (8 * E6)$$

in which *En* = 0 if the jumper is ON, and En = 1 if the jumper is OFF.

On Turbo PMAC2, the servo interrupt time is determined on PMAC2 Ultralite boards by MACRO IC 0 I-variables I6800, I6801, and I6802; on non-Ultralite boards by Servo IC 0 I-variables I7000, I7001, and I7002; on UMAC Turbo systems by Servo IC m I-variables I7m00, I7m01, and I7m02, or MACRO IC 0 I-variables I6800, I6801, or I6802. The proper setting of I10 can be determined from Servo IC variables by the formula:

$$I10 = \frac{640}{9}(2 * I7m00 + 3)(I7m01 + 1)(I7m02 + 1)$$

The proper setting of I10 can be determined from MACRO IC 0 variables by the formula:

$$I10 = \frac{640}{9}(2 * I6800 + 3)(I6801 + 1)(I6802 + 1)$$

When changing I10, a **%100** command must be issued, or the value saved and the controller reset, before the new value of I10 will take effect.

I10 is used to provide the delta-time value in the position update calculations, scaled such that 2^{23} – 8,388,608 – means one millisecond. Delta-time in these equations is I10*(%value/100). The % (feedrate override) value can be controlled in any of several ways: with the on-line '%' command, with a direct write to the command '%' register, with an analog voltage input, or with a digital input frequency. The default % value is 100, and many applications can always leave it at 100.

I11 Programmed Move Calculation Time

Range: 0 to 8,388,607

Units: msec

Default: 0

I11 controls the delay from when the run signal is taken (or the move sent if executing immediately) and when the first programmed move starts. If several Turbo PMACs need to be run synchronously, I11 should be set the same on all of the cards. If I11 is set to zero, the first programmed move starts as soon as the calculation is complete.

This calculation time delay is also used after any break in the continuous motion of a motion program: a **DWELL**, a **PSET**, a **WAIT**, or each move if Ix92=1 (a **DELAY** is technically a zero-distance move, and so does not constitute a break).

The actual delay time varies with the time base (e.g. at a value of 50, the actual delay time will be twice the number defined here), which keeps it as a fixed *distance* of the master in an external time base application. If it is desired to have the slave coordinate system start up immediately with the master, I11 should be set to zero, and the program commanded to run *before* the master starts to move.

Note:

If I11 is greater than zero, defining a definite time for calculations, and Turbo PMAC cannot complete the calculations for the first move of a sequence by the end of the I11 time, Turbo PMAC will terminate the running of the program with a run-time error.

I12 Lookahead Time Spline Enable

Range: 0 - 1

Units: none

Default: 0

I12 permits the enabling of a new lookahead technique called "time splining". If I12 is set to 1, all coordinate systems that are executing lookahead will use this technique. If I12 is set to 0, none of them will.

"Time splining" permits smoother transitions from one vector velocity to another during lookahead when there is little or no change in direction. As long as the commanded vector velocity going into lookahead does not change by more than a factor of two in a single Isx13 segment, the velocity change will be made without any velocity "undershoot".

Without this technique, large changes in vector velocity that have to be extended by lookahead can cause significant velocity undershoot.

Setting I12 to 1 adds a small but potentially significant computational load to the lookahead calculations.

I13 Foreground In-Position Check Enable

Range: 0 - 1

Units: none

Default: 0

I13 controls whether the activated motors on Turbo PMAC check for "in-position" as a foreground servo-interrupt task or not. If I13 is set to the default value of 0, in-position checking is done as a lower-priority background task only. If I13 is set to 1, a basic in-position check operation is done for all active motors every servo interrupt as well.

The foreground in-position check function is intended for very rapid move-and-settle applications for which the background check is too slow. Enabling this function permits the fastest possible assessment of whether a motor is "in position"

For the foreground check to consider a motor to be "in position", the following four conditions must all be met:

1. The motor must be in closed-loop control;
2. The desired velocity must be zero;
3. The magnitude of the following error must be less than the motor's Ixx28 parameter;
4. The move timer for the motor must not be active.

Note:

Unlike the background in-position check, there is no capability in the foreground check to require these conditions be true for Ixx88+1 consecutive scans.

If the foreground check decides that the motor is "in position", it sets bit 13 of the motor status word (Y:$0000C0 for Motor 1) to 1; if it decides that the motor is "not in position", it sets this bit to 0. This foreground status bit is distinct from the background motor status bit at bit 0 of the same word. The coordinate system's in-position status bit, which is the logical OR of the

background motor in-position bits for all of the motors in the coordinate system, is not affected by the foreground in-position check.

Setting I13 to 1 to enable the foreground in-position check adds about 5% to the required time of the servo-interrupt tasks for each active motor.

I14 Temporary Buffer Save Enable

Range: 0 – 1

Units: none

Default: 0

I14 controls whether the structure of the "temporary" buffers on Turbo PMAC can be retained through a board power-down or reset. The temporary buffers are those where the information in the buffer is never retained through a power-down or reset. These buffers are:

- The rotary motion program buffer (**ROTARY**) for each coordinate system
- The segment lookahead buffer (**LOOKAHEAD**) for each coordinate system
- The extended cutter radius compensation block buffer (**CRCOMP**) for each coordinate system

If I14 is set to 0 when a **SAVE** command is issued, the structure for these buffers is *not* stored to non-volatile flash memory, and so will not be present after the next power-down or board reset. In this case, any of these buffers to be used must be re-defined after each power-down or reset (e.g. **DEFINE ROTARY**, **DEFINE LOOKAHEAD**).

If I14 is set to 1 when a **SAVE** command is issued, the structure for these buffers is stored to non-volatile flash memory, although the contents of these buffers are not stored. In this case, any of these buffers that existed at the time of the **SAVE** command will be present after the next power-down or reset, and so do not need to be re-defined. However, these buffers will always be empty after a board power-down or reset.

The structure for the temporary data-gathering buffer is not retained through a power down or reset, regardless of the setting of I14.

I15 Degree/Radian Control for User Trig Functions

Range: 0 to 1

Units: None

Default: 0 (degrees)

I15 controls whether the angle values for trigonometric functions in user programs (motion and PLC) and on-line commands are expressed in degrees (I15=0) or radians (I15=1).

I16 Rotary Buffer Request On Point

Range: 0 to 8,388,607

Units: Program lines

Default: 5

I16 controls the point at which an executing rotary program will signal that it is ready to take more command lines (BREQ line taken high, coordinate system *Rotary Buffer Full* status bit taken low). This occurs when the executing point in the program has caught up to within fewer lines behind the last line sent to Turbo PMAC than the value in this parameter. This can be detected as an interrupt to the host or be checked by the host on a polled basis.

Note:

On Turbo PMAC(1), the BREQ line to the interrupt controller reflects the status of the hardware-selected coordinate system (by JPAN pins FPDn/) if the control-panel inputs are enabled (I2=0); it represents the status of the software-host-addressed coordinate system if the control-panel inputs are disabled (I2=1). In virtually all applications using this feature, the user will want to set I2 to 1 so the BREQ line reflects the status of the coordinate system to which he is currently talking. On Turbo PMAC2, the BREQ line always reflects the status of the software-host-addressed coordinate system.

I17 Rotary Buffer Request Off Point

Range: 0 to 8,388,607

Units: Program lines

Default: 10

This parameter controls how many lines ahead of the executing line the host can provide a PMAC rotary motion program buffer before it signals that it is not ready for more lines (BREQ line held low, coordinate system status bit Rotary Buffer Full becomes 1). This status information can be detected either by polling **??** or **PR**, by using the interrupt line to the host, or by polling the status register of the interrupt controller.

If you send a program line to the rotary buffer, the BREQ line will be taken low (at least momentarily). If there are still fewer than I17 number of lines in the buffer ahead of the executing line, the BREQ line will be taken high again (giving the ability to generate an interrupt), and the Rotary Buffer Full status bit will stay 0. If there are greater than or equal to I17 lines in the buffer ahead of the executing line, the BREQ line will be left low, and the Rotary Buffer Full status bit will become 1. Normally at this point, the host will stop sending program lines (although this is not required) and wait for program execution to catch up to within I16 lines and take BREQ high again.

Note:

On Turbo PMAC(1), the BREQ line to the interrupt controller reflects the status of the hardware-selected coordinate system (by JPAN pins FPDn/) if the control-panel inputs are enabled (I2=0); it represents the status of the software-host-addressed coordinate system if the control-panel inputs are disabled (I2=1). In virtually all applications using this feature, the user will want to set I2 to 1 so the BREQ line reflects the status of the coordinate system to which he is currently talking. On Turbo PMAC2, the BREQ line always reflects the status of the software-host-addressed coordinate system.

I18 Fixed Buffer Full Warning Point

Range: 0 to 8,388,607

Units: Long memory words

Default: 10

I18 sets the level of open memory below which BREQ (Buffer Request) will not go true (global status bit Fixed Buffer Full will become 0) during the entry of a fixed (non-rotary) buffer.

Every time a command line is downloaded to an open fixed buffer (PROG or PLC), the BREQ line will be taken low (at least momentarily). If there are more than I18 words of open memory left, the BREQ line will be taken high again (giving the ability to generate an interrupt), and Fixed Buffer Full will stay at 0. If there are I18 words or less, the BREQ line will be left low, and Fixed Buffer Full will become 1.

The number of available words of memory can be found using the **SIZE** command.

I19 Clock Source I-Variable Number (Turbo PMAC2 only)

Range: 6807, 6857 … 7907, 7957

Units: I-variable number

Default: 7007 (non-Ultralite Turbo PMAC2)
6807 (Turbo PMAC2 Ultralite)
Configuration-dependent (Turbo PMAC2-3U)

I19 contains the number of the servo/phase clock-direction I-variable whose value is set by default to 0, indicating that the matching Servo IC or MACRO IC is the source of the servo and phase clock signals for the Turbo PMAC2 system. This I-variable for all other Servo ICs and MACRO ICs in the system is set to 3, indicating that these ICs will use servo and phase clock signals from a source external to them.

The clock-direction I-variables for MACRO ICs 0, 1, 2, and 3 are I6807, I6857, I6907, and I6957, respectively. The clock direction I-variables for Servo ICs *m* and *m** (*m* = 0 to 9) are I7*m*07 and I7*m*57, respectively.

Note:

Only in 3U-format Turbo PMAC2 systems (UMAC Turbo and 3U Turbo Stack) can the clock signals come from ICs on accessory boards. In other Turbo PMAC2 systems, the clock signals must come from an IC on the base PMAC board, or be brought in through the serial port.

During system re-initialization (reset with E3 jumper ON, or **$$$***** command), then Turbo PMAC2 first determines the "default" value of I19 by searching for the presence of all possible Servo and MACRO ICs, and assigning the clock source to the first IC it finds in the following list:

1.	Servo IC 0	(On-board or 3U Stack)	(I19=7007)
2.	MACRO IC 0	(On-board or ACC-5E)	(I19=6807)
3.	Servo IC 1	(On-board or 3U Stack)	(I19=7107)
4.	Servo IC 2	(ACC-24E2, 51E)	(I19=7207)
…			
11.	Servo IC 9	(ACC-24E2, 51E)	(I19=7907)
12.	Servo IC 2*	(ACC-24E2, 51E)	(I19=7257)
…			
19.	Servo IC 9*	(ACC-24E2, 51E)	(I19=7957)
20.	MACRO IC 1	(On-board or ACC-5E)	(I19=6857)
21.	MACRO IC 2	(On-board or ACC-5E)	(I19=6907)
21.	MACRO IC 3	(On-board or ACC-5E)	(I19=6957)

(MACRO ICs must be "DSPGATE2" ICs to be used as a clock source.)

If the E1 external-clock-source jumper is ON during re-initialization, I19 is set to 0, indicating that no Servo IC or MACRO IC will be the source of the system clocks.

If one of the clock-direction I-variables is commanded to be set to its default value (e.g. **I7007=***), Turbo PMAC2 looks to I19 to decide whether this variable will be set to 0 or not.

In 3U-format Turbo PMAC2 systems, I19 also operates at the system's power-up/reset. At this time, the saved value of I19 determines which single one of the Servo-IC or MACRO-IC clock-

direction I-variables is set to 0 at reset to provide the system with that ICs servo and phase clock signals.

The clock-direction I-variables for all of the other Servo ICs and MACRO ICs are set to 3 at reset to tell them to input the servo and phase clock signals, regardless of the saved values for these I-variables. (On other Turbo PMAC2 boards, the saved values of the clock-direction I-variables are used.) If the Servo IC or MACRO IC thus selected is not present, the watchdog timer will trip immediately.

In 3U-format Turbo PMAC2 systems, if the saved value of I19 is 0, the clock-direction I-variable for all Servo ICs and MACRO ICs is set to 3. In this case, jumper E1 must be ON to admit externally generated servo and phase clocks on the serial port, and these signals must be present immediately; otherwise the watchdog timer will trip immediately.

On Turbo PMAC(1) boards, the Servo and Phase clock signals are generated in the same discrete logic (or come in from an external source), so I19 is not needed to control which ASIC provides the clock signals.

I20 MACRO IC 0 Base Address (Turbo PMAC2 only)

Range: $0, $078400 - $07B700

Units: Turbo PMAC2 Addresses

Default: Auto-detected

I20 sets the base address of the first MACRO IC (called "MACRO IC 0") in the Turbo PMAC2 system, normally the one with the lowest base address. A setting of 0 for I20 tells the Turbo PMAC2 CPU that no MACRO IC 0 is present, and none of the firmware's automatic functions for that IC will be active.

On re-initialization – either on resetting with the E3 re-initialization jumper ON or on issuing the **$$$*** ** command, Turbo PMAC2 will auto-detect which MACRO ICs are present, and set I20 to the base address of the MACRO IC with the lowest base address. Turbo PMAC2 will also do this when commanded to set I20 to its default value (**I20=***). If no MACRO ICs are found, I20 will be set to 0 instead.

If automatic use of the multiplexer port or the display port is desired, I20 must be set to the base address of the DSPGATE2 IC serving as MACRO IC that is connected to this port. In UMAC Turbo systems it is possible to have multiple multiplexer and display ports, but only those ports connected to the single IC selected by I20 support the automatic firmware functions for those ports. In other Turbo PMAC2 systems, the on-board multiplexer and display ports using the MACRO IC at $078400 are always used, regardless of the setting of I20.

I-variables I6800 – I6849 reference registers in MACRO IC 0, whose addresses are relative to the address contained in I20. These addresses are established at power-up/reset. If the value of I20 is incorrect at power-up/reset, these I-variables will not work. It is possible to set the value of I20 directly, saving the value and resetting the card, but users are strongly encouraged just to let Turbo PMAC2 set I20 itself by re-initialization or default setting, and to treat I20 as a status variable. If I20 is set to 0, these variables will always return a value of 0.

A Turbo PMAC2 will look to find MACRO nodes 0 – 15 in MACRO IC 0, referenced to the address contained in I20. These addresses are established at power-up/reset. If the value of I20 is incorrect at power-up/reset, these MACRO nodes will not be accessed.

UMAC versions of the Turbo PMAC2 have the addressing capability for up to 16 MACRO ICs, but only the 4 MACRO ICs referenced by I20 – I23 can have I-variable support. Master-to-master MACRO communications can only be done on MACRO IC 0, referenced by I20, when I84=0.

For a Turbo PMAC2 that is not "Ultralite" or "UMAC", the only valid MACRO IC 0 base address is $078400. For a Turbo PMAC2 Ultralite, the valid base addresses are $078400, $079400, $07A400, and $07B400. For a UMAC Turbo system, the valid base addresses can be expressed as $07xy00, where 'x' can be 8, 9, A, or B, and 'y' can be '4', '5', '6', or '7'.

If the configuration of the MACRO ICs in a modular Turbo PMAC system, such as a UMAC Turbo rack, is changed, the values of I20 – I23 will need to be changed.

See Also:
I-Variables I21, I22, I23, I24, I4902 – I4903, I4926 – I4941, I6800 – I6999.

I21 MACRO IC 1 Base Address (Turbo PMAC2 only)

Range: $0, $078400 - $07B700

Units: Turbo PMAC Addresses

Default: Auto-detected

I21 sets the base address of the second MACRO IC (called "MACRO IC 1") in the Turbo PMAC2 system, normally the one with the second-lowest base address. A setting of 0 for I21 tells the Turbo PMAC2 CPU that no MACRO IC 1 is present, and none of the firmware's automatic functions for that IC will be active.

On re-initialization – either on resetting with the E3 re-initialization jumper ON or on issuing the **$$$***** command, Turbo PMAC2 will auto-detect which MACRO ICs are present, and set I21 to the base address of the MACRO IC with the second-lowest base address. Turbo PMAC2 will also do this when commanded to set I21 to its default value (**I21=***). If less than two MACRO ICs are found, I21 will be set to 0 instead.

I-variables I6850 – I6899 reference registers in MACRO IC 1, whose addresses are relative to the address contained in I21. These addresses are established at power-up/reset. If the value of I21 is incorrect at power-up/reset, these I-variables will not work. It is possible to set the value of I21 directly, saving the value and resetting the card, but users are strongly encouraged just to let Turbo PMAC2 set I21 itself by re-initialization or default setting, and to treat I21 as a status variable. If I21 is set to 0, these variables will always return a value of 0.

A Turbo PMAC2 will look to find MACRO nodes 16 – 23 in MACRO IC 1, referenced to the address contained in I21. These addresses are established at power-up/reset. If the value of I21 is incorrect at power-up/reset, these MACRO nodes will not be accessed.

UMAC versions of the Turbo PMAC2 have the addressing capability for up to 16 MACRO ICs, but only the 4 MACRO ICs referenced by I20 – I23 can have I-variable support. Master-to-master MACRO communications can only be done on MACRO IC 1, referenced by I21, when I84=1.

For a Turbo PMAC2 that is not "Ultralite" or "UMAC", the only valid MACRO IC base address is $78400. For a Turbo PMAC2 Ultralite, the valid base addresses are $78400, $79400, $7A400, and $7B400. For a UMAC Turbo system, the valid base addresses can be expressed as $7xy00, where 'x' can be 8, 9, A, or B, and 'y' can be '4', '5', '6', or '7'.

If the configuration of the MACRO ICs in a modular Turbo PMAC system, such as a UMAC Turbo rack, is changed, the values of I20 – I23 will need to be changed.

See Also:
I-Variables I20, I22, I23, I24, I4902 – I4903, I4926 – I4941, I6800 – I6999.

I22 MACRO IC 2 Base Address (Turbo PMAC2 only)

Range: $0, $078400 - $07B700

Units: Turbo PMAC Addresses

Default: Auto-detected

I22 sets the base address of the third MACRO IC (called "MACRO IC 2") in the Turbo PMAC2 system, normally the one with the third-lowest base address. On re-initialization – either on resetting with the E3 re-initialization jumper ON or on issuing the **$$$***** command, Turbo PMAC2 will auto-detect which MACRO ICs are present, and set I22 to the base address of the MACRO IC with the third-lowest base address. Turbo PMAC2 will also do this when commanded to set I22 to its default value (**I22=***). If less than three MACRO ICs are found, I22 will be set to 0 instead.

I-variables I6900 – I6949 reference registers in MACRO IC 2, whose addresses are relative to the address contained in I22. These addresses are established at power-up/reset. If the value of I22 is incorrect at power-up/reset, these I-variables will not work. It is possible to set the value of I22 directly, saving the value and resetting the card, but users are strongly encouraged just to let Turbo PMAC2 set I22 itself by re-initialization or default setting, and to treat I22 as a status variable. If I22 is set to 0, these variables will always return a value of 0.

A Turbo PMAC2 will look to find MACRO nodes 32 – 47 in MACRO IC 2, referenced to the address contained in I22. These addresses are established at power-up/reset. If the value of I22 is incorrect at power-up/reset, these MACRO nodes will not be accessed.

UMAC versions of the Turbo PMAC2 have the addressing capability for up to 16 MACRO ICs, but only the 4 MACRO ICs referenced by I20 – I23 can have I-variable support. Master-to-master MACRO communications can only be done on MACRO IC 2, referenced by I22, when I84=2.

For a Turbo PMAC2 that is not "Ultralite" or "UMAC", the only valid MACRO IC base address is $78400. For a Turbo PMAC2 Ultralite, the valid base addresses are $78400, $79400, $7A400, and $7B400. For a UMAC Turbo system, the valid base addresses can be expressed as $7xy00, where 'x' can be 8, 9, A, or B, and 'y' can be '4', '5', '6', or '7'.

If the configuration of the MACRO ICs in a modular Turbo PMAC system, such as a UMAC Turbo rack, is changed, the values of I20 – I23 will need to be changed.

See Also:
I-Variables I20, I21, I23, I24, I4902 – I4903, I4926 – I4941, I6800 – I6999.

I23 MACRO IC 3 Base Address (Turbo PMAC2 only)

Range: $0, $078400 - $07B700

Units: Turbo PMAC Addresses

Default: Auto-detected

I23 sets the base address of the fourth MACRO IC (called "MACRO IC 3") in the Turbo PMAC2 system, normally the one with the fourth-lowest base address. On re-initialization – either on resetting with the E3 re-initialization jumper ON or on issuing the **$$$***** command, Turbo PMAC2 will auto-detect which MACRO ICs are present, and set I23 to the base address of the MACRO IC with the fourth-lowest base address. Turbo PMAC2 will also do this when commanded to set I23 to its default value (**I23=***). If less than four MACRO ICs are found, I23 will be set to 0 instead.

I-variables I6950 – I6999 reference registers in MACRO IC 3, whose addresses are relative to the address contained in I23. These addresses are established at power-up/reset. If the value of I23 is incorrect at power-up/reset, these I-variables will not work. It is possible to set the value of I23 directly, saving the value and resetting the card, but users are strongly encouraged just to let Turbo PMAC2 set I23 itself by re-initialization or default setting, and to treat I23 as a status variable. If I23 is set to 0, these variables will always return a value of 0.

A Turbo PMAC2 will look to find MACRO nodes 48 – 63 in MACRO IC 3, referenced to the address contained in I23. These addresses are established at power-up/reset. If the value of I23 is incorrect at power-up/reset, these MACRO nodes will not be accessed.

UMAC versions of the Turbo PMAC2 have the addressing capability for up to 16 MACRO ICs, but only the 4 MACRO ICs referenced by I20 – I23 can have I-variable support. Master-to-master MACRO communications can only be done on MACRO IC 3, referenced by I23, when I84=3.

For a Turbo PMAC2 that is not "Ultralite" or "UMAC", the only valid MACRO IC base address is $78400. For a Turbo PMAC2 Ultralite, the valid base addresses are $78400, $79400, $7A400, and $7B400. For a UMAC Turbo system, the valid base addresses can be expressed as $7xy00, where 'x' can be 8, 9, A, or B, and 'y' can be '4', '5', '6', or '7'.

If the configuration of the MACRO ICs in a modular Turbo PMAC system, such as a UMAC Turbo rack, is changed, the values of I20 – I23 will need to be changed.

See Also:
I-Variables I20, I21, I22, I24, I4902 – I4903, I4926 – I4941, I6800 – I6999.

I24 Main DPRAM Base Address

Range: $0, $060000 - $077000

Units: Turbo PMAC Addresses

Default: Auto-detected

I24 sets the base address of the dual-ported RAM IC in the Turbo PMAC system that is used for the automatic DPRAM communications functions. On re-initialization – either on resetting with the E3 re-initialization jumper ON or on issuing the **$$$***** command, Turbo PMAC will auto-detect which DPRAM ICs are present, and set I24 to the base address of the DPRAM IC with the lowest base address. If no DPRAM ICs are found, I24 will be set to 0 instead.

The automatic DPRAM communications functions reference registers in a DPRAM IC, whose addresses are relative to the address contained in I24. These addresses are established at power-up/reset. If the value of I24 is incorrect at power-up/reset, these functions will not work. To select a new DPRAM IC that the CPU will use for the automatic DPRAM IC functions, it is necessary to change I24, issue the **SAVE** command, and reset the Turbo PMAC.

If the saved value of I24 is 0 at power-up/reset, the DPRAM addresses will be set up for a DPRAM base address of $060000.

The following are Turbo PMAC addresses where DPRAM ICs can be found:

Address	Location	Address	Location
$060000	Main board or CPU board	$06F000	UBUS DPRAM board w/ SW=1100
$064000	UMAC-CPCI bridge board*	$074000	UBUS DPRAM board w/ SW=0010
$06C000	UBUS DPRAM board w/ SW=0000	$075000	UBUS DPRAM board w/ SW=0110
$06D000	UBUS DPRAM board w/ SW=0100	$076000	UBUS DPRAM board w/ SW=1010
$06E000	UBUS DPRAM board w/ SW=1000	$077000	UBUS DPRAM board w/ SW=1110
*Not auto-detected on re-initialization.			

See Also:
Dual-Ported RAM Communications
I-Variables I47 – I50, I55 – I58, I4904, I4942 – I4949

I30 Compensation Table Wrap Enable

Range: 0 - 1

Units: none

Default: 0

I30 controls whether the compensation tables entered into Turbo PMAC will automatically "wrap" or not. This affects position ("leadscrew"), backlash, and torque compensation tables. If I30 is set to 0, when a table is downloaded to PMAC, the compensation correction at motor position 0 is always set to 0. In this case, if smooth rollover of the table is desired, the last entry of the table must explicitly be set to 0.

If I30 is set to 1, the last entry of the table also becomes the correction at motor position 0, automatically yielding a smooth rollover of the table, and permitting non-zero corrections at the rollover point.

I30 affects table values only as they are being downloaded to Turbo PMAC; it does not affect the values of tables already in Turbo PMAC's memory.

I37 Additional Wait States

Range: $000000 - $032403

Units: Instruction cycle wait states (by bit)

Default: $000000

I37 controls the number of "wait states" added to the factory default values when the Turbo PMAC processor accesses external memory or memory-mapped I/O devices. Wait states are the number of instruction cycles the processor idles when reading from or writing to a register of memory or I/O. On power-up/reset, Turbo PMAC automatically sets the number of wait states based on the programmed CPU frequency as set by I54. Under certain circumstances, particularly in accessing third-party devices, more robust operation may be obtained by increasing the number of wait states from the factory default values (at the cost of slightly slower operation.

I37 is divided into four parts, each controlling the wait states for a different area of the memory and I/O map. Bits 0 and 1 control the number of added wait states for I/O devices (such as ASICs and A/D converters; dual-ported RAM also counts as an I/O device) mapped into Y-registers. Bits 16 and 17 control the number of added wait states for I/O devices mapped into X-registers. With 2 bits each, up to 3 wait states can be added to these accesses; generally, these are both set to the same value.

Bit 10 of I37 controls the number of added wait states for "P" (program, or machine-code) memory register access. Bit 13 controls the number of added wait states for "X" and "Y" (data) memory register access. As single-bit values, they can add only one wait state to these memory accesses. Generally, these are both set to the same value.

I37 is used at power-up/reset only, so to change the number of I/O wait states, change the value of I37, issue a **SAVE** command, and reset the Turbo PMAC. At power-up/reset, Turbo PMAC automatically adds the value of I37 to the value from its internal look-up table to set the number of I/O wait states. The resulting number of wait states for different areas of the memory and I/O map is in internal CPU register X:$FFFFFB.

Examples:

```
I37=$020002     ; Add 2 wait states to X and Y I/O access.
I37=$002400     ; Add 1 wait state to X/Y and P memory access
I37=$032403     ; Add 3 wait states for I/O, 1 for memory
```

I39 UBUS Accessory ID Variable Display Control

Range: 0 – 5

Units: none

Default: 0

I39 controls which portions of the identification variables I4909 – I4999, which provide information about accessory boards on UMAC's "UBUS" backplane expansion port, are reported. These variables are 36-bit variables in total, with 4 parts:

1. Vendor ID (Bits 0 – 7)
2. Options Installed (Bits 8 – 17)
3. Revision Number (Bits 18 – 21)
4. Card ID [Part #] (Bits 22 – 35)

The following list shows the possible values of I39, and which parts of these ID variables are reported for each value:

- I39 = 0: Vendor ID, Options Installed, Revision Number, Card ID (36 bits)
- I39 = 1: Vendor ID only (8 bits)
- I39 = 2: Options Installed only (10 bits)
- I39 = 3: Revision number only (4 bits)
- I39 = 4: Card ID only (14 bits)
- I39 = 5: Base Address of Card (19 bits)

Note:

The base address of the card reported with I39 = 5 is not part of the card identification variable, but it is still very useful in determining the configuration of the system.

The value of I39 is not saved, and I39 is set to 0 automatically on power-up/reset.

Example:

```
I39=1                ; Report Vendor ID only
I4910                ; Query first axis card vendor ID
1                    ; (Delta Tau is Vendor ID #1)
I39=2                ; Report Options Installed only
I4910                ; Query first axis card options installed
3                    ; First 2 options installed (bits 0 and 1 set)
I39=3                ; Report revision number only
I4910                ; Query first axis card revision number
2                    ; Revision 2 (-102 board)
I39=4                ; Report Card ID (part number) only
I4910                ; Query first axis card part number
3397                 ; Card ID 3397 (Delta Tau part # 603397: ACC-24E2)
I39=5                ; Report base address only
I4910                ; Query first axis card base address
$78200               ; Base address $78200
I39=0                ; Report all of ID variable
I4910                ; Query first axis card full ID variable
14248575745          ; Full ID variable for card
```

See Also:

I-Variables I4909 – I4999

I40 Watchdog Timer Reset Value

Range: 0 – 65,535

Units: servo cycles

Default: 0 (sets 4095)

I40 controls the value to which the watchdog timer's counter is reset each background cycle. Each servo interrupt cycle, Turbo PMAC automatically decrements this counter by 1, and if the counter becomes less than 0, the real-time interrupt task will no longer strobe the watchdog circuit, permitting it to trip and shut down the card. Therefore, one background cycle must execute every I40 servo cycles, or the board will shut down.

I40 permits the user to optimize the sensitivity of the watchdog timer for a particular application. Register X:$25 contains the lowest value that the counter has reached before being reset since the last power-on/reset.

For purposes of backward compatibility, if I40 is set to 0, Turbo PMAC will reset the watchdog timer counter to 4095 each background cycle.

I41 I-Variable Lockout Control

Range: $0 – $F (0 – 15)

Units: none

Default: 0

I41 permits the user to lock out changes to any of several sets of I-variables in Turbo PMAC. I41 is a 4-bit value, and each bit independently controls access to a set of I-variables. If the bit of I41 is set to 1, the corresponding I-variables cannot be changed with an **{I-variable}={value}** command. The purpose of I41 is to prevent inadvertent changes to certain I-variable values.

The following table shows the I-variable set that each bit of I41 controls.

I41 Bit #	Bit value	I-Variable Range	I-Variable Function
0	1	I100 – I4899	Motor Setup
1	2	I5100 – I6699	Coordinate System Setup
2	4	I6800 – I7999	MACRO/Servo IC Setup
3	8	I8000 – I8191	Conversion Table Setup

If there is an attempt to execute a command to set an I-variable value, either an on-line command or a buffered program command, while the controlling bit is set to 1, the command is ignored (no error is generated).

I41 does not prevent changes to an I-variable by means of an M-variable assignment or a direct memory write command.

Care must be taken in downloading a complete set of I-variables with I41 at a non-zero value. Because I41 is typically set before any of the variables it controls, if it has a non-zero value in this list, some of the subsequent variables will not get set.

The restore function of the PEWIN32 Executive Program for 32-bit Windows operating systems Versions 2.30(?) and newer (September 1999 and later) automatically handle this situation, setting I21 to 0 at the beginning of a download, then setting the file's I41 value at the end of the download. Older versions of the Executive will not perform a proper "restore" function with a non-zero value of I41.

If a Servo or MACRO IC is not present in the Turbo PMAC system, Turbo PMAC cannot set a value for any of the setup I-variables for that IC, regardless of the setting of I41.

I42 Spline/PVT Time Control Mode

Range: 0 – 1

Units: none

Default: 0

I42 controls whether **TM** or **TA** is used to define the time for SPLINE and PVT-mode moves. For PVT-mode moves, the **PVT{data}** command can be used to set the move time regardless of the setting of I42.

If I42 is set to 0, the **TM{data}** command must be used to define the time for SPLINE-mode moves, and can be used to define the time for PVT-mode moves, once a **PVT{data}** command has been used to establish that move mode.

If I42 is set to 1, the **TA{data}** command must be used to define the time for SPLINE-mode moves, and can be used to define the time for PVT-mode moves, once a **PVT{data}** command has been used to establish that move mode.

In both modes, the time has units of milliseconds, with a range of 0 – 4095.9998 milliseconds, and a resolution of ¼-microsecond.

See Also:
Spline Moves, PVT Moves
Program commands **PVT{data}**, **TA{data}**, **TM{data}**

I43 Auxiliary Serial Port Parser Disable

Range: 0 - 1

Units: none

Default: 0

I43 controls whether Turbo PMAC firmware automatically parses the data received on the Option 9T serial port as commands or not. If I43 is set to the default value of 0, Turbo PMAC automatically tries to interpret the data received on this port as Turbo PMAC commands. However, if I43 is set to 1, it will not try to interpret this data as commands, permitting the user's application to interpret this data as required for the application. The queue of characters sent in over the auxiliary serial port is located at X:$001C00 – X:$001CFF (low byte of each register).

This ability to disable the automatic command parser permits the auxiliary port to be interfaced to other devices such as vision systems, which send out serial data.

I44 PMAC Ladder Program Enable {Special Firmware Only}

Range: 0 - 1

Units: none

Default: 0

I44 controls whether the "PMAC Ladder" graphical PLC programs that can be used with optional firmware are running or not. If I44 is set to 1, any PMAC ladder programs that have been downloaded into Turbo PMAC program memory are active. If I44 is set to 0, these programs will not execute, even if they are present.

If the firmware does not support these "PMAC Ladder" PLC programs, I44 cannot be changed from 0.

I45 Foreground Binary Rotary Buffer Transfer Enable

Range: 0 - 1

Units: none

Default: 0

I45 controls whether the transfer of binary rotary buffer commands from dual-ported RAM to internal memory is done as a background task or as a foreground task. If I45 is set to the default value of 0 when the **OPEN BIN ROT** command is given, Turbo PMAC checks the DPRAM binary rotary buffer once per background cycle (if the binary buffer is open) and copies commands received in the last cycle to the buffer in internal memory. If I45 is set to 1 when the **OPEN BIN ROT** command is given, Turbo PMAC checks the DPRAM buffer every real-time interrupt (every I8+1 servo cycles) instead.

Setting I45 to 1 permits a quicker and more predictable reaction to the receipt of binary rotary buffer commands from the host computer.

I46 P & Q-Variable Storage Location

Range: 0 to 3

Units: None

Default: 0

I46 controls the memory locations that Turbo PMAC uses to store the P and Q-Variables. For each type of variable, there is a choice between the main flash-backed memory and the optional supplemental battery-backed memory. Option 16 must be purchased in order to be able to select the battery-backed memory storage.

I46 can take 4 values: 0, 1, 2, and 3. The meaning of each is:

- I46=0: P-Variables in flash-backed RAM; Q-Variables in flash-backed RAM
- I46=1: P-Variables in battery-backed RAM; Q-Variables in flash-backed RAM
- I46=2: P-Variables in flash-backed RAM; Q-Variables in battery-backed RAM
- I46=3: P-Variables in battery-backed RAM; Q-Variables in battery-backed RAM

For variables stored in flash-backed RAM, values must be copied to flash memory with the **SAVE** command in order to be retained through a power-down or reset. The **SAVE** command operation can take up to 10 seconds. On power-up/reset, Turbo PMAC automatically copies the last saved values for the P and Q-variables from flash memory to the flash-backed locations in main RAM memory.

For variables stored in battery-backed RAM, values are automatically retained in the RAM by the battery voltage. No **SAVE** operation is required. These values are not affected by a **SAVE** command or a power-up/reset.

Access to battery-backed RAM is significantly slower than access to flash-backed RAM, because either read or write access to the battery-backed RAM requires 2 wait cycles of 9 instruction cycles each, but read or write access to the flash-backed RAM requires 2 wait cycles of only 1 instruction cycle each.

Storing P and/or Q-variables in battery-backed RAM frees up flash-backed memory for user program and buffer storage. Storing either P or Q-variables alone in battery-backed RAM allots 8K additional words for user storage, on top of the standard 26K words (212K with the optional expanded user memory), for a total of 34K words (optionally 220K); storing both P and Q-variables in battery backed RAM allots 16K additional words, for a total of 42K words (optionally 228K).

A change in the value of I46 takes effect only at power-up/reset. Therefore, to change the location where P and/or Q-variables are stored, the value of I46 must be changed, the **SAVE** command must be issued, and then the board must be reset. If the new value of I46 would move the P and/or Q-variables from battery-backed to flash-backed RAM, the **SAVE** operation copies the variable values from battery-backed RAM into flash memory so that present values are not lost. At the reset, these values are copied from flash memory to flash-backed RAM.

I47 DPRAM Motor Data Foreground Reporting Period

Range: 0 to 255

Units: Servo Cycles

Default: 0

I47 specifies the period, in servo cycles, that Turbo PMAC will copy data from servo control registers into fixed registers in DPRAM for easy access by the host computer, if this function has been enabled by setting I48 to 1. The data is reported for those motors specified by a mask word in DPRAM.

If I47 is set to 0, the reporting is "on demand". In this mode, Turbo PMAC will check every servo cycle to see if the host computer has set the request bit in DPRAM, signaling that it has read the previous cycle's data. Turbo PMAC will copy the latest data into DPRAM only if this bit is set, and it will clear the bit.

I48 DPRAM Motor Data Foreground Reporting Enable

Range: 0 to 1

Units: None

Default: 0

I48 enables or disables the dual-ported RAM (DPRAM) motor data reporting function as a "foreground" task at the servo interrupt priority level. When I48=1, Turbo PMAC copies key data from the motor control registers to fixed registers in the DPRAM every I47 servo cycles (or on demand if I47=0) for easy access by the host computer. The data is reported for those motors specified by a mask word in DPRAM.

Reporting this data as a high-priority foreground task permits a reliable high-frequency transfer of motor data to the host, but it can have a significant impact on the capabilities of lower priority tasks, such as motion program trajectory calculations, and PLCs.

When I48=0, the DPRAM motor data reporting function in foreground is disabled.

If I57 is set to 1 to enable DPRAM reporting of the motor registers as a background task, Turbo PMAC automatically sets I48 to 0 to disable the foreground reporting.

Refer to the description of DPRAM functions for more information.

I49 DPRAM Background Data Reporting Enable

Range: 0 to 1

Units: None

Default: 0

I49 enables or disables the dual-ported RAM (DPRAM) background data reporting function. When I49=1, PMAC copies key data from coordinate-system and global data registers to fixed registers in the DPRAM approximately every I50 servo cycles (or on demand if I50=0) for easy access by the host computer. The data for coordinate systems up to the number specified by a designated register in DPRAM are reported.

When I49=0, the DPRAM background data reporting function is disabled.

Refer to the description of DPRAM functions for more information.

I50 DPRAM Background Data Reporting Period

Range: 0 to 255

Units: Servo Cycles

Default: 0

I50 specifies the minimum period, in servo cycles, that Turbo PMAC will copy data from coordinate-system and global data registers into fixed registers in DPRAM for easy access by the host computer, if this function has been enabled by setting I49 to 1. In addition, if I57 is set to 1, I50 specifies the minimum period that Turbo PMAC will copy motor data registers into DPRAM. If I49 and/or I57, and I50 are greater than 0, then each background cycle, Turbo PMAC will check to see if at least I50 servo cycles have elapsed since the last reporting; if this is so, it will copy the current data into DPRAM. The data for coordinate systems up to the number specified by a designate register in DPRAM are reported.

If I50 is set to 0, the reporting is "on demand". In this mode, Turbo PMAC will check every background cycle to see if the host computer has set the request bit in DPRAM, signaling that it has read the previous cycle's data. Turbo PMAC will copy the latest data into DPRAM only if this bit is set, and it will clear the bit.

I51 Compensation Table Enable

Range: 0 to 1

Units: None

Default: 0 (disabled)

I51 the enabling and disabling of all of the compensation tables for all motors on Turbo PMAC: leadscrew compensation tables, backlash compensation tables, and torque compensation tables. When I51 is 0, all tables are disabled and there is no correction performed. When I51 is 1, all tables are enabled and corrections are performed as specified in the tables.

The constant backlash as controlled by Ixx85 and Ixx86 is not affected by the setting of I51.

I52 CPU Frequency Control

Range: 0 to 31

Units: Multiplication factor

Default: 7 (80 MHz)

I52 controls the operational clock frequency of the Turbo PMAC's CPU by controlling the multiplication factor of the phase-locked loop (PLL) inside the CPU. The PLL circuit multiplies the input 10 MHz (actually 9.83 MHz) clock frequency by a factor of (I52 + 1) to create the clock frequency for the CPU. Formally, this is expressed in the equation:

*CPU Frequency (MHz) = 10 * (I52 + 1)*

I52 should usually be set to create the highest CPU frequency for which the CPU is rated. For the standard 80 MHz CPU, it should be set to 7.

Note:

It may be possible to operate a CPU at a frequency higher than its rated frequency, particularly at low ambient temperatures. However, safe operation cannot be guaranteed under these conditions, and any such operation is done entirely at the user's own risk.

I52 is actually used at power-on/reset only, so to make a change in the CPU frequency with I52, change the value of I52, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command.

If too high a value of I52 has been set, the watchdog timer on the Turbo PMAC will likely trip immediately after reset due to CPU operational failure. If this happens, the Turbo PMAC must be re-initialized, using E51 on a Turbo PMAC(1), or E3 on a Turbo PMAC2.

I53 Auxiliary Serial Port Baud Rate Control

Range: 0 to 15

Units: None

Default: 0 (disabled)

I53 controls the baud rate for communications on the Option 9T auxiliary serial port. Turbo PMAC uses I53 only at power-up/reset to set up the frequency of the clocking circuit for the auxiliary serial port. To change the baud rate, it is necessary to change the value of I53, store this value to non-volatile flash memory with the **SAVE** command, and reset the card. At this time, Turbo PMAC will establish the new baud rate.

The possible settings of I53 and the baud rates they define are:

I53	Baud Rate	I53	Baud Rate
0	Disabled	8	9600
1	600	9	14,400
2	1200	10	19,200
3	1800	11	28,800
4	2400	12	38,400
5	3600	13	57,600
6	4800	14	76,800
7	7200	15	115,200

If the optional auxiliary serial port is not present on your Turbo PMAC, or if it is not being used, it is best to set I53 to 0 to disable the port, so you will not have the computational overhead of continually checking the port.

Baud rates set by odd values of I53 are not exact unless the CPU is running at an exact multiple of 30 MHz (I52 = 2, 5, 8, 11, 14, 17, 20, 23). For most of these baud rates, the errors are small enough not to matter. However, for 115,200 baud, the CPU must be running at an exact multiple of 30 MHz to establish serial communications.

If your host computer baud rate cannot be made to match the Turbo PMAC's baud rate, the Turbo PMAC's baud rate must be changed through another communications port.

I54 Serial Port Baud Rate Control

Range: 0 to 15

Units: None

Default: 12 (38400 baud)

I54 controls the baud rate for communications on the main serial port. Turbo PMAC uses I54 only at power-up/reset to set up the frequency of the clocking circuit for the serial port. To change the baud rate, it is necessary to change the value of I54, store this value to non-volatile flash memory with the **SAVE** command, and reset the card. At this time, Turbo PMAC will establish the new baud rate.

The possible settings of I54 and the baud rates they define are:

I54	Baud Rate	I54	Baud Rate
0	600	8	9600
1	900	9	14,400
2	1200	10	19,200
3	1800	11	28,800
4	2400	12	38,400
5	3600	13	57,600
6	4800	14	76,800
7	7200	15	115,200

Baud rates set by odd values of I54 are not exact unless the CPU is running at an exact multiple of 30 MHz (I52 = 2, 5, 8, 11, 14, 17, 20, 23). For most of these baud rates, the errors are small enough not to matter. However, for 115,200 baud, the CPU must be running at an exact multiple of 30 MHz to establish serial communications.

If the host computer baud rate cannot be made to match the Turbo PMAC's baud rate, either the Turbo PMAC's baud rate must be changed through the bus communications port, or the Turbo PMAC must be re-initialized by resetting or powering up with the E51 jumper ON for Turbo PMAC(1), or the E3 jumper ON for Turbo PMAC2. This forces the Turbo PMAC to the default baud rate of 38,400.

I55 DPRAM Background Variable Buffers Enable

Range: 0 to 1

Units: None

Default: 0 (disabled)

I55 enables or disables the dual-ported RAM (DPRAM) background variable read and write buffer function. When I55 is 0, this function is disabled. When I55 is 1, this function is enabled. When enabled, the user can specify up to 128 Turbo PMAC registers to be copied into DPRAM each background cycle to be read by the host (background variable read) and up to 128 Turbo PMAC registers to be copied each background cycle from values written into the DPRAM by the host (background variable write).

I56 DPRAM ASCII Communications Interrupt Enable

Range: 0 to 1

Units: None

Default: 0 (disabled)

This parameter controls the interrupt feature for the dual-ported RAM (DPRAM) ASCII communications function enabled by I58=1. When I56=1, PMAC will generate an interrupt to the host computer each time it loads a line into the DPRAM ASCII buffer for the host to read. When I56=0, it will not generate this interrupt.

For the Turbo PMAC(1)-PC, the interrupt line used is the EQU4 interrupt. For this to reach the host, jumper E55 must be ON, and jumpers, E54, E56, and E57 must be OFF. When using this feature, do not use the EQU4 line for any other purpose, including position compare.

For the Turbo PMAC2-PC the interrupt line used is the EQU1 interrupt. When using this feature, do not use the EQU1 line for any other purpose, including position compare.

For the VME-bus versions of Turbo PMAC (Turbo PMAC(1)-VME, Turbo PMAC2-VME and Turbo PMAC2-VME Ultralite), the interrupt line used is the normal communications interrupt (the only interrupt available). This line -- IRQn on the VME bus, is determined by the VME setup variable I95. The interrupt vector provided to the host is one greater than the value in VME setup variable I96. For example, if I96 is set to the default value of $A1, this interrupt will provide an interrupt vector of $A2.

For the Turbo PMAC2-PC Ultralite, this feature is not presently supported with the standard hardware.

I57 DPRAM Motor Data Background Reporting Enable

Range: 0 to 1

Units: None

Default: 0

I57 enables or disables the dual-ported RAM (DPRAM) motor data reporting as a background function. When I57=1, Turbo PMAC copies key data from internal motor system and global data registers to fixed registers in the DPRAM as a background task approximately every I50 servo cycles (or on demand if I50=0) for easy access by the host computer. The data is reported for those motors specified by a mask word in DPRAM at Turbo PMAC address $06001C.

If I57 is set to 1, then Turbo PMAC automatically sets I48 to 0, disabling the foreground reporting of the same data.

When I57=0, the DPRAM background motor data reporting function is disabled. In this setting, I48 can be set to 1 to enable foreground reporting of the motor data.

For most purposes, background reporting of the motor data will provide the data at a high enough rate, and it will not degrade the performance of motion programs. Only if the data is required at a guaranteed high frequency should the foreground reporting be used.

Refer to the description of DPRAM functions for more information.

I58 DPRAM ASCII Communications Enable

Range: 0 to 1

Units: None

Default: 0 (disabled) if no DPRAM present
1 (enabled) if DPRAM present

I58 enables or disables the dual-ported RAM (DPRAM) ASCII communications function. When I58=1, this function is enabled and the host computer can send ASCII command lines to the Turbo PMAC through the DPRAM and receive ASCII responses from Turbo PMAC through the DPRAM. When I58=0, this function is disabled.

At power-up/reset, if Turbo PMAC finds a DPRAM IC present in the system, I58 is automatically set to 1, immediately enabling this communications. If no DPRAM IC is found in the system at this time, I58 is automatically set to 0.

I3 does not affect the handshaking characters used in DPRAM ASCII communications.

If I56 is also equal to 1, PMAC will provide an interrupt to the host computer when it provides a response string.

I59 Motor/C.S. Group Select

Range: 0 – 3

Units: none

Default: 0

I59 controls which group of 8 motors and 8 coordinate systems can be selected by the FPDn inputs on the Turbo PMAC(1) control panel port. The possible values of I59 and the motors and coordinate systems they select are:

- I59 = 0: Motors 1 – 8; C.S. 1 – 8
- I59 = 1: Motors 9 – 16; C.S. 9 – 16
- I59 = 2: Motors 17 – 24; C.S. 1 – 8
- I59 = 3: Motors 25 – 32; C.S. 9 – 16

The value of I59 can be set from the control panel of a Turbo PMAC(1). If none of the FPDn lines are pulled low ("selecting" Motor/C.S. 0), then pulling any of 4 input lines low will cause the value of I59 to be set:

- HOME/: I59 = 0
- PREJOG/: I59 = 1
- START/: I59 = 2
- STEP/: I59 = 3

Note:

In Turbo PMAC firmware versions 1.934 and older, I59 also controlled which group of 8 motors' data was supplied in response to the on-line commands **<CTRL-B>**, **<CTRL-P>**, **<CTRL-V>**, and **<CTRL-F>**, when issued from any port. Starting in firmware version V1.935, each port can select a different group of 8 motors for these commands, as set by the most recent **##** command sent over that port.

See Also:
On-line commands **<CTRL-B>**, **<CTRL-F>**, **<CTRL-P>**, **<CTRL-V>**, **##{constant}**

I60 Filtered Velocity Sample Time

Range: 0 to 15

Units: Servo Cycles - 1

Default: 15

I60 controls the frequency at which actual positions for each motor are placed into the 16-slot rotary velocity calculation buffer for the motor. Every (I60+1) servo cycles, PMAC compares the actual position for each active motor to the actual position from 16 * (I60+1) servo cycles before to compute a filtered velocity for reporting purposes (with the **V** and **<CTRL-V>** commands), then overwrites that old value in the 16-slot buffer.

I60 must be set equal to a value 2^n-1 (0, 1, 3, 7, or 15) for proper operation. At the default value of 15, Turbo PMAC stores a position value every 16 servo cycles and computes the velocity by comparing to the position stored 256 servo cycles before. I61 must be set in the appropriate relationship to I60 in order for the filtered velocity value to be scaled properly.

See Also:
I-variables I61, Ix09
On-line commands **<CTRL-V>**, **V**
Suggested M-variables Mxx74
Memory registers D:$0000EF, etc.

I61 Filtered Velocity Shift

Range: 0 to 255

Units: Bits

Default: 8

I61 controls the scaling of reported filtered velocity values for all motors in a Turbo PMAC. It does this by telling the filtered velocity calculation routines how many bits to shift the difference between the latest position stored in the buffer, and the position stored 16*(I60+1) servo cycles before.

To make the filtered velocity report as counts per servo cycle with the **V** and **<CTRL-V>** commands, and store as 1 / (Ix09*32) counts per servo cycle, I61 should be set according to the following formula:

$$I61 = \log_2(I60+1)+4$$

The following table shows the typical relationship between I60 and I61:

I60	I60+1	$\log_2$(I60+1)	I61
0	1	0	4
1	2	1	5
3	4	2	6
7	8	3	7
15	16	4	8

See Also:
I-variables I60, Ix09
On-line commands **<CTRL-V>**, **V**
Suggested M-variables Mxx74
Memory registers D:$0000EF, etc.

I62 Internal Message Carriage Return Control

Range: 0 to 1

Units: None

Default: 1

I62 permits the user to control whether internally generated messages sent from Turbo PMAC to the host computer are terminated with the carriage return (**<CR>**) character or not. It affects only those messages generated by a **CMDx** and **SENDx** statements (where **x** represents the port) in a PMAC motion or PLC program. The ability to suppress the **<CR>** provides more flexibility in controlling the format display of a terminal window or printer.

If I62 is set to the default value of 0, these messages are terminated with a **<CR>** . If I62 is set to 1, the **<CR>** is suppressed. With I62 set to 1, if it desired for a Turbo PMAC program to cause a **<CR>** to be sent, the **SEND^M** command must be used (the carriage return character is **<CTRL-M>**).

Note:

Do not set I62 to 1 if using dual-ported RAM ASCII communications (I58=1).

Example:
With program code:

```
I62=1                                ; Suppress <CR> on SEND
SENDS "THE VALUE OF P1 IS "          ; String sent with no <CR>
CMDS "P1"                            ; Response string follows on same line, no <CR>
SENDS^M.....                         ; Send a <CR>
```

PMAC responds with:

```
THE VALUE OF P1 IS 42<CR>
```

I63 Control-X Echo Enable

Range: 0 – 1

Units: None

Default: 1

I63 permits the PMAC to echo the **<CONTROL-X>** character back to the host computer when it is received. If I63 is set to 1, PMAC will send a **<CONTROL-X>** character (ASCII value 24 decimal) back to the host computer when it receives a **<CONTROL-X>** character.

If I63 is set to 0, PMAC will send nothing back to the host computer when it receives a **<CONTROL-X>** character. This is equivalent to the action of older versions of PMAC firmware without an I63 variable.

The host computer can use the **<CONTROL-X>** character to clear out PMAC's communications buffers and make sure that no unintended responses are received for the next command. However, without an acknowledgement that the buffers have been cleared, the host computer has to add a safe delay to ensure that the operation has been done before the next command can be issued.

Setting I63 to 1 permits a more efficient clearing of the buffer, because the response character lets the host computer know when the next command can safely be sent.

Versions of the PCOMM32 communications library 2.21 and higher (March 1999 and newer) can take advantage of this feature for more efficient communications. I63 should be set to 0 when using older versions of PCOMM32.

I64 Internal Response Tag Enable

Range: 0 – 1

Units: None

Default: 0

I64 permits PMAC to "tag" ASCII text lines that it sends to the host computer as a result of internal commands, so these can easily be distinguished from responses to host commands.

If I64 is set to 1, a line of text sent to the host computer as a result of an internal **SEND** or **CMD** statement is preceded by a **<CONTROL-B>** ("start-transmission") character. In the case of an error report, the **<CONTROL-B>** character replaces the leading **<CONTROL-G>** ("bell") character. The text line is always terminated by a **<CR>** ("carriage return") character, regardless of the setting of I62.

If I64 is set to 0, a text line sent in response to an internal PMAC command is not preceded by any special character. Reported errors are preceded by the **<CONTROL-G>** ("bell") character. This is equivalent to the action of older versions of PMAC firmware, before I64 was implemented.

Regardless of the setting of I64, if I6 = 2, errors on internal commands are not reported to the host computer.

Example:

With I64=0, lines sent from PMAC are:

```
Motion Stopped on Limit<CR>
<BELL>ERR003<CR>
```

With I64=1, the same lines from PMAC are:

```
<CTRL-B>Motion Stopped on Limit<CR>
<CTRL-B>ERR003<CR>
```

I68 Coordinate System Activation Control

Range: 0 - 15

Units: None

Default: 15

I68 controls which coordinate systems are activated on a Turbo PMAC. A coordinate system must be activated in order for it to be addressed and accept commands, to have its automatic user countdown timers (Isx11 and Isx12) enabled (even if used by some other function), and for it to have some of the Synchronous M-variable Assignment stack assigned to it.

I68 can take values from 0 to 15. The highest numbered coordinate system that is activated is Coordinate System (I68 + 1). In other words, a given value of I68 activates Coordinate System 1 through Coordinate System (I68 + 1).

The Synchronous M-Variable Stack allocation is binary; it can only be split by powers of 2. The stack allocation per coordinate system is detailed in the following table:

I68 Value	Highest Numbered Coordinate System Activated	Sync. M-Var. Stack per C.S.	Max. Sync M-Var. Assignments per move, no cutter comp	Max. Sync M-Var. Assignments per move, cutter comp on
0	C.S. 1	256 words	63	42
1	C.S. 2	128 words	31	20
2 - 3	C.S. 3 - 4	64 words	15	10
4 - 7	C.S. 5 - 8	32 words	7	4
8 - 15	C.S. 9 - 16	16 words	3	2

The default I68 value of 15 (all coordinate systems activated) will always work, even if fewer coordinate systems are actually being used. Lowering I68 from this default if fewer coordinate systems will be used brings two advantages. First, there is a slight improvement in computational efficiency because de-activated coordinate systems do not have to be checked periodically.

Second, each remaining active coordinate system has a bigger piece of the synchronous M-variable assignment stack, so more synchronous M-variable assignments can be executed per move. Each synchronous M-variable assignment requires two words of the stack; one additional word is required per move. The above table lists how many synchronous M-variables assignments can be performed per move in each active coordinate system.

If the special lookahead function is enabled, synchronous M-variable assignments made during lookahead are stored in the area reserved in the lookahead buffer, and the number of assignments that can be buffered is limited by the space reserved with the **DEFINE LOOKAHEAD** command.

I68 is actually used at power-on/reset only, so to make a change in the number of activated coordinate systems, change the value of I68, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command.

MACRO Ring Configuration I-Variables

I70 MACRO IC 0 Node Auxiliary Register Enable

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I70 controls which nodes of MACRO IC 0 for which Turbo PMAC performs automatic copying into and out of the auxiliary registers. Enabling this function for a node is required to use the auxiliary register as the flag register for a motor.

I70 is a 16-bit variable. Bits 0 to 15 control the enabling of this copying function for MACRO nodes 0 to 15, respectively. A bit value of 1 means the copying function is enabled; a bit value of 0 means the copying function is disabled.

If the copying function is enabled for Node n (where n = 0 to F hex or 0 to 15 decimal), during each background "housekeeping" software cycle, PMAC copies the contents of Y:$000344n to the Node n auxiliary write register, and copies the contents of the Node n auxiliary read register into X:$00344n.

The copying function enabled by I70 permits the use of the auxiliary registers for command and status flags plus Type 0 auxiliary read and write functions in PLC programs and on-line commands.

For each node whose auxiliary functions are enabled by I70, I71 must correctly specify for the node whether the Type 0 or Type 1 MACRO protocol is used.

If a value of I78 greater than 0 has been saved into PMAC's non-volatile memory to enable Type 1 MACRO master/slave auxiliary communications with Node 15, then at subsequent power-up/resets, bit 15 of I70 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I70. This reserves Node 15 for the Type 1 master/slave auxiliary communications alone.

If a value of I79 greater than 0 has been saved into PMAC's non-volatile memory to enable Type 1 MACRO master/master auxiliary communications with Node 14, then at subsequent power-up/resets, bit 14 of I70 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I70. This reserves Node 14 for the Type 1 master/master auxiliary communications alone.

I71 MACRO IC 0 Node Protocol Type Control

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I71 controls for each node (0 - 15) on MACRO IC 0 whether the Type 0 or Type 1 MACRO protocol is used on that node. I71 is a 16-bit value; each bit 0 -15 controls the protocol type for the MACRO node of the same number. A value of 0 in the bit selects the Type 0 protocol for the matching MACRO node; a value of 1 in the bit selects the Type 1 protocol for the node.

The key difference between Type 0 and Type 1 protocols is in which node register is used for control and status flags. In the Type 0 protocol, the 1st register (24 bits) is used for the flags; in the Type 1 protocol, the 4th register (16 bits) is used for the flags. The bits of I71 must be set properly for any node whose auxiliary flag function is enabled by I70.

The Type 0 protocol is generally used for single-node MACRO devices, such as the Performance Controls FLX Drive. The Type 1 protocol is generally used for multi-node MACRO devices, such as Delta Tau's Compact MACRO Station.

I72 MACRO IC 1 Node Auxiliary Register Enable

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I72 controls which nodes of MACRO IC 1 for which Turbo PMAC performs automatic copying into and out of the auxiliary registers. Enabling this function for a node is required to use the auxiliary register as the flag register for a motor.

Note:

MACRO IC 1 can be present only on Turbo PMAC2 Ultralite boards with Option 1U1 ordered, or on a 3U Turbo PMAC2 with some configurations of its ACC-5E.

I72 is a 16-bit variable. Bits 0 to 15 control the enabling of this copying function for MACRO nodes 0 to 15, respectively. A bit value of 1 means the copying function is enabled; a bit value of 0 means the copying function is disabled.

If the copying function is enabled for Node n (where n = 0 to F hex or 0 to 15 decimal), during each background "housekeeping" software cycle, PMAC copies the contents of Y:$000345n to the Node n auxiliary write register, and copies the contents of the Node n auxiliary read register into X:$00345n.

The copying function enabled by I72 permits the use of the auxiliary registers for command and status flags plus Type 0 auxiliary read and write functions in PLC programs and on-line commands.

For each node whose auxiliary functions are enabled by I72, I73 must correctly specify for the node whether the Type 0 or Type 1 MACRO protocol is used.

If a value of I78 greater than 0 has been saved into PMAC's non-volatile memory to enable Type 1 MACRO auxiliary communications with Node 15, then at subsequent power-up/resets, bit 15 of I72 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I72. This reserves Node 15 for the Type 1 auxiliary communications alone.

I73 MACRO IC 1 Node Protocol Type Control

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I73 controls for each node (0 - 15) on MACRO IC 1 whether the Type 0 or Type 1 MACRO protocol is used on that node. I73 is a 16-bit value; each bit 0 -15 controls the protocol type for the MACRO node of the same number. A value of 0 in the bit selects the Type 0 protocol for the matching MACRO node; a value of 1 in the bit selects the Type 1 protocol for the node.

Note:

MACRO IC 1 can only be present on Turbo PMAC2 Ultralite boards with Option 1U1 ordered, or on a 3U Turbo PMAC2 with some configurations of its ACC-5E.

The key difference between Type 0 and Type 1 protocols is in which node register is used for control and status flags. In the Type 0 protocol, the 1st register (24 bits) is used for the flags; in the Type 1 protocol, the 4th register (16 bits) is used for the flags. The bits of I73 must be set properly for any node whose auxiliary flag function is enabled by I72.

The Type 0 protocol is generally used for single-node MACRO devices, such as the Performance Controls FLX Drive. The Type 1 protocol is generally used for multi-node MACRO devices, such as Delta Tau's Compact MACRO Station.

I74 MACRO IC 2 Node Auxiliary Register Enable

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I74 controls which nodes of MACRO IC 2 for which Turbo PMAC performs automatic copying into and out of the auxiliary registers. Enabling this function for a node is required to use the auxiliary register as the flag register for a motor.

Note:

MACRO IC 2 can only be present on Turbo PMAC2 Ultralite boards with Option 1U2 ordered, or on a 3U Turbo PMAC2 with some configurations of its ACC-5E.

I74 is a 16-bit variable. Bits 0 to 15 control the enabling of this copying function for MACRO nodes 0 to 15, respectively. A bit value of 1 means the copying function is enabled; a bit value of 0 means the copying function is disabled.

If the copying function is enabled for Node n (where n = 0 to F hex or 0 to 15 decimal), during each background "housekeeping" software cycle, PMAC copies the contents of Y:$000346n to the Node n auxiliary write register, and copies the contents of the Node n auxiliary read register into X:$00346n.

The copying function enabled by I74 permits the use of the auxiliary registers for command and status flags plus Type 0 auxiliary read and write functions in PLC programs and on-line commands.

For each node whose auxiliary functions are enabled by I74, I75 must correctly specify for the node whether the Type 0 or Type 1 MACRO protocol is used.

If a value of I78 greater than 0 has been saved into PMAC's non-volatile memory to enable Type 1 MACRO auxiliary communications with Node 15, then at subsequent power-up/resets, bit 15 of I74 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I74. This reserves Node 15 for the Type 1 auxiliary communications alone.

I75 MACRO IC 2 Node Protocol Type Control

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I75 controls for each node (0 - 15) on MACRO IC 2 whether the Type 0 or Type 1 MACRO protocol is used on that node. I75 is a 16-bit value; each bit 0 -15 controls the protocol type for the MACRO node of the same number. A value of 0 in the bit selects the Type 0 protocol for the matching MACRO node; a value of 1 in the bit selects the Type 1 protocol for the node.

Note:

MACRO IC 2 can only be present on Turbo PMAC2 Ultralite boards with Option 1U2 ordered, or on a 3U Turbo PMAC2 with some configurations of its ACC-5E.

The key difference between Type 0 and Type 1 protocols is in which node register is used for control and status flags. In the Type 0 protocol, the 1st register (24 bits) is used for the flags; in the Type 1 protocol, the 4th register (16 bits) is used for the flags. The bits of I75 must be set properly for any node whose auxiliary flag function is enabled by I74.

The Type 0 protocol is generally used for single-node MACRO devices, such as the Performance Controls FLX Drive. The Type 1 protocol is generally used for multi-node MACRO devices, such as Delta Tau's Compact MACRO Station.

I76 MACRO IC 3 Node Auxiliary Register Enable

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I76 controls which nodes of MACRO IC 3 for which Turbo PMAC performs automatic copying into and out of the auxiliary registers. Enabling this function for a node is required to use the auxiliary register as the flag register for a motor.

Note:

MACRO IC 3 can only be present on Turbo PMAC2 Ultralite boards with Option 1U3 ordered, or on a 3U Turbo PMAC2 with some configurations of its ACC-5E.

I76 is a 16-bit variable. Bits 0 to 15 control the enabling of this copying function for MACRO nodes 0 to 15, respectively. A bit value of 1 means the copying function is enabled; a bit value of 0 means the copying function is disabled.

If the copying function is enabled for Node n (where n = 0 to F hex or 0 to 15 decimal), during each background "housekeeping" software cycle, PMAC copies the contents of Y:$000347n to the Node n auxiliary write register, and copies the contents of the Node n auxiliary read register into X:$00347n.

The copying function enabled by I76 permits the use of the auxiliary registers for command and status flags plus Type 0 auxiliary read and write functions in PLC programs and on-line commands.

For each node whose auxiliary functions are enabled by I76, I77 must correctly specify for the node whether the Type 0 or Type 1 MACRO protocol is used.

If a value of I78 greater than 0 has been saved into PMAC's non-volatile memory to enable Type 1 MACRO auxiliary communications with Node 15, then at subsequent power-up/resets, bit 15 of I76 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I76. This reserves Node 15 for the Type 1 auxiliary communications alone.

I77 MACRO IC 3 Node Protocol Type Control

Range: 0 .. $FFFF (0 .. 65,535)

Units: none

Default: 0

I77 controls for each node (0 - 15) on MACRO IC 3 whether the Type 0 or Type 1 MACRO protocol is used on that node. I77 is a 16-bit value; each bit 0 -15 controls the protocol type for the MACRO node of the same number. A value of 0 in the bit selects the Type 0 protocol for the matching MACRO node; a value of 1 in the bit selects the Type 1 protocol for the node.

> ***Note:***
>
> MACRO IC 3 can only be present on Turbo PMAC2 Ultralite boards with Option 1U3 ordered, or on a 3U Turbo PMAC2 with some configurations of its ACC-5E.

The key difference between Type 0 and Type 1 protocols is in which node register is used for control and status flags. In the Type 0 protocol, the 1st register (24 bits) is used for the flags; in the Type 1 protocol, the 4th register (16 bits) is used for the flags. The bits of I77 must be set properly for any node whose auxiliary flag function is enabled by I76.

The Type 0 protocol is generally used for single-node MACRO devices, such as the Performance Controls FLX Drive. The Type 1 protocol is generally used for multi-node MACRO devices, such as Delta Tau's Compact MACRO Station.

I78 MACRO Type 1 Master/Slave Communications Timeout

Range: 0 .. 255

Units: Servo Cycles

Default: 0

I78 permits the enabling of MACRO Type 1 master-slave auxiliary communications using Node 15, which are executed with the **MS**, **MSR**, and **MSW** commands. If I78 is set to 0, these communications are disabled. If I78 is set to a value greater than 0, these communications are enabled, and the value of I78 sets the "timeout" value for the auxiliary response, in Turbo PMAC servo cycles.

If Turbo PMAC has not received a response to the MACRO auxiliary communications command within I78 servo cycles, it will stop waiting and register a "MACRO Auxiliary Communications Error", setting Bit 5 of global status register X:$000006. A value of 32 for I78 is suggested.

Bit 15 of I70, I72, I74, and I76 must be set to 0 to disable Node 15's Type 0 (node-specific) auxiliary communications for each MACRO IC if I78 is greater than 0. If a value of I78 greater than 0 has been saved into PMAC's non-volatile memory, then at subsequent power-up/resets, bit 15 of I70, I72, I74, and I76 are automatically forced to 0 by PMAC firmware, regardless of the value saved for I70.

This function is controlled by I1003 on non-Turbo PMACs.

I79 MACRO Type 1 Master/Master Communications Timeout

Range: 0 .. 255

Units: Servo Cycles

Default: 0

I79 permits the enabling of MACRO Type 1 master-to-master auxiliary communications using Node 14, which are executed with the **MM**, **MMR**, and **MMW** commands. If I79 is set to 0, these communications are disabled. If I79 is set to a value greater than 0, these communications are enabled, and the value of I79 sets the "timeout" value for the auxiliary response, in Turbo PMAC servo cycles.

If Turbo PMAC has not received a response to the MACRO auxiliary communications command within I79 servo cycles, it will stop waiting and register a "MACRO Auxiliary Communications Error", setting Bit 5 of global status register X:$000006. A value of 32 for I79 is suggested.

Bit 14 of I70 must be set to 0 to disable Node 14's Type 0 (node-specific) auxiliary communications if I79 is greater than 0. If a value of I79 greater than 0 has been saved into PMAC's non-volatile memory, then at subsequent power-up/resets, bit 14 of I70 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I70.

Certain master-to-master communications registers are only set up at the Turbo PMAC power-up/reset, so before master-to-master communications can be performed, a non-zero value of I79 must be stored in flash memory with the **SAVE** command, and the board must be reset.

I80 MACRO Ring Check Period

Range: 0 .. 255

Units: servo cycles

Default: 0

I80 determines the period for Turbo PMAC to evaluate whether there has been a MACRO ring failure. If I80 is greater than 0, Turbo PMAC must receive the "sync node" packet (as specified by I6841) at least I82 times within I80 servo cycles. It also must detect less than I81 ring errors (byte violation error, packet parity error, packet overflow error, or packet underflow error) in this same period, and find no errors for at least one of its checks during the period. If either of these conditions is not met, Turbo PMAC will assume it is a ring fault, and will disable all motors.

If I80 is 0, Turbo PMAC does not perform these checks, even if MACRO is active.

A ring check period of about 20 milliseconds is recommended in a typical MACRO application. I80 can be set as function of the desired period according to the formula:

I80 = Desired ring check period (msec) * Servo update frequency (kHz)

If I80 is greater than 0, activating this check function, bits 16 to 19 of I6841 (Sync Packet Number) must specify the number of a packet that is regularly being received by this card. Otherwise, Turbo PMAC will immediately detect a ring fault. Typically, Packet 15 ($F) is used as the sync packet, and it is always sent because bit 15 of I6841 is set to 1 to activate the node to send the packet around the ring every cycle.

When a ring fault is detected, Turbo PMAC sets bit 4 of global status word X:$000006 to 1. It disables all motors using the MACRO ring, and attempts to notify all of its MACRO slave stations that a ring fault has occurred.

Turbo PMAC performs this check each real-time interrupt (every I8+1 servo cycles), so it will perform I80 / (I8 + 1) checks during the check period. This value must be greater than I82, or ring failures will be detected because not enough checks were done to detect the required number of sync packets received.

This function is controlled by I1001 on non-Turbo PMACs.

I81 MACRO Maximum Ring Error Count

Range: 0 .. 255

Units: Detected ring errors

Default: 2

I81 sets the maximum number of MACRO ring communications errors that can be detected in one "ring check period" before a "MACRO communications fault" is declared. The ring check period is set at I80 servo cycles; if I80 is 0, this checking is not performed.

There are four types of ring communications errors that can be detected: byte violation errors, packet parity errors, packet overflow errors, and packet underflow errors. If any one of these is detected during a check, this counts as a "ring error" towards the I81 counts.

Turbo PMAC performs the check every real-time interrupt (every I8+1 servo cycles), so it will perform I80 / (I8 + 1) checks during the check period. If I81 or more ring errors are detected during this period, a "ring fault" is declared, and the ring is shut down. Regardless of the setting of I81, if a ring error is detected on every check during the period, a "ring fault" is declared.

This function is controlled by I1004 on non-Turbo PMACs.

I82 MACRO Minimum Sync Packet Count

Range: 0 .. 255

Units: Detected sync packets

Default: 2

I82 sets the minimum number of MACRO "sync packets" that must be received in one "ring check period" for Turbo PMAC to conclude that the ring is operating properly. The ring check period is set at I80 servo cycles; if I80 is 0, this checking is not performed.

The number of the sync packet is determined by bits 16 – 19 of I6841. Usually Packet 15 is used as the sync packet, and its transmission around the ring is enabled by setting bit 15 of I6841 to 1, activating Node 15. If the sync packet is defined as a packet that is not regularly transmitted around the ring, this check will shut down the ring immediately.

If fewer than I82 sync packets are detected during any ring check period of I80 servo cycles, Turbo PMAC will shut down operation of the ring, declaring a "ring fault". Turbo PMAC performs the check during the real-time interrupt (every I8+1 servo cycles), so it will perform I80 / (I8 + 1) checks during the check period. If I82 is set to a value greater than I80 / (I8 +1), Turbo PMAC will find a ring fault immediately.

This function is controlled by I1005 on non-Turbo PMACs.

I83 MACRO Parallel Ring Enable Mask

Range: 0 – 15

Units: none

Default: 0

I83 specifies which MACRO ICs on Turbo PMAC2 control their own independent rings so independent checking of ring communications using variables I80 to I82 is done using registers in that MACRO IC.

I83 is a 4-bit value. Bit *n* of I83 corresponds to MACRO IC *n*. If bit *n* is set to 1, ring checking is performed using registers in MACRO IC *n*. If bit *n* is set to 0, no ring checking is performed using registers in MACRO IC *n*. (However, if all bits are 0, checking can still be done on MACRO IC 0; see below.)

I80 must be set greater than 0 to specify a ring-check period and activate any ring checking. If I80 is set greater than 0, ring checking is done automatically on MACRO IC 0, so bit 0 if I83 is not used. However, if multiple rings are used, it is recommended that Bit 0 be set to 1 for clarity's sake.

Presently, only the UMAC configuration of the Turbo PMAC2 supports multiple rings (through multiple ACC-5E boards). All other versions of Turbo PMAC2 can only support a single ring and do ring checking on MACRO IC 0. For these boards, I83 can be left at the default value of 0.

If multiple MACRO ICs share a common ring, the lowest-numbered MACRO IC on the ring should be used for ring checking. For example, if MACRO ICs 0 and 1 share one ring, and MACRO ICs 2 and 3 share another, bits 0 and 2 of I83 should be set to 1, yielding a value of 5.

I-variables I20 – I23 specify the base addresses of MACRO ICs 0 – 3, respectively. These must be set correctly in order for the ring-checking function on these ICs to work properly.

The following table shows which MACRO rings are enabled by the I83 bits.

I83 Bit #, MACRO IC #	Bit Value	I-Variable for IC Address
0	1	I20
1	2	I21
2	4	I22
3	8	I23

See Also:
I-Variables I20 – I23, I80 – I82

I84 MACRO IC # for Master Communications

Range: 0 – 3

Units: MACRO IC #

Default: 0

I84 specifies which MACRO IC on the Turbo PMAC2 is used for "MACRO Master" communications with the **MACROMSTASCII**, **MACROSTASCII**, **MACROMSTREAD**, and **MACROMSTWRITE** commands.

I84 can take a value from 0 to 3. The value of I84 specifies that the MACRO IC of that number will be used. Variables I20 – I23 specify the base addresses of MACRO ICs 0 – 3, respectively.

Note:

The UMAC Turbo firmware will support up to four parallel MACRO Rings and, if desired up to sixteen by changing I20 – I23 before initiating communication over the MACRO Ring. Each parallel MACRO Ring will be a Ring Controller with the MACRO IC tied to I20 being the source of the Phase and Servo clock.

See Also:
I-variables I20 – I23
Commands **MACROMASTASCII**, **MACROSTASCII**, **MACROMSTREAD**, **MACROMSTWRITE**

I85 MACRO Ring Order Number

Range: 0 – 254

Units: none

Default: 0

I85 is used to store the order of the Turbo PMAC2 in the MACRO ring. The first device (Turbo PMAC2, MACRO Station, or other device) "downstream" in the ring from the ring controller is 1, the next is 2, and so on. If I85 is 0, the Turbo PMAC2 has not been assigned an order in the ring yet.

If I85 has a value from 1 to 254, the Turbo PMAC2 will respond when the **{constant}** in the **MACROSTASCII{constant}** command matches the value of I85. The first device in the ring with I85 = 0 will respond to the **MACROSTASCII255** command.

Note:

For the ring controller, I85 should remain at 0, even though it has no effect on the ordered ring communications.

The **STN** command will return the value of I85.

See Also:
Commands **MACROSTASCII**, **STN**

VME/DPRAM Setup I-Variables

I90 VME Address Modifier

Range: $00 - $FF

Units: None

Default: $39

I90 controls which "address modifier" value Turbo PMAC will respond to when sent by the VME bus host. I90 takes one of three valid values in normal use, depending on the address bus width used:

- I90 = $29: 16-bit addressing
- I90 = $39: 24-bit addressing
- I90 = $09: 32-bit addressing

I90 is actually used at power-on/reset only, so to set or change the VME address modifier, change the value of I90, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I90 is copied at power-on/reset is X:$070006 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M90) to change the active setup without a **SAVE** and reset.

I91 VME Address Modifier Don't Care Bits

Range: $00 - $FF

Units: None

Default: $04

I91 controls which bits of the I90 VME address modifier are "don't care" bits. I91 is set to $04 in all normal use, which permits both "non-privileged" and "supervisory" data access by the VME host.

I91 is actually used at power-on/reset only, so to set or change the VME address modifier "don't care" bits, change the value of I91, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I91 is copied at power-on/reset is X:$070007 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M91) to change the active setup without a **SAVE** and reset.

I92 VME Base Address Bits A31-A24

Range: $00 - $FF

Units: None

Default: $FF

I92 controls bits A31 through A24 of the VME bus base address of Turbo PMAC, both for the mailbox registers, and the dual-ported RAM. It is only used if 32-bit addressing has been selected with I90 and I99.

I92 is actually used at power-on/reset only, so to set or change bits 16-23 of the VME bus base address, change the value of I92, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I92 is copied at power-on/reset is X:$070008 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M92) to change the active setup without a **SAVE** and reset.

I93 VME Mailbox Base Address Bits A23-A16 ISA DPRAM Base Address Bits A23-A16

Range: $00 - $FF

Units: None

Default: $7F (VME); $0D (ISA)

On VME bus systems, I93 controls bits A23 through A16 of the VME bus base address of the mailbox registers for Turbo PMAC. Bit 7 of I93 corresponds to A23 of the base address, and bit 0 of I93 corresponds to A16. I93 is only used on VME systems if 24-bit or 32-bit addressing has been selected with I90 and I99.

On ISA bus systems (PC, PC Ultralite, 3U Turbo with PC/104), I93 controls bits A23 through A16 of the ISA bus base address of the DPRAM. Bit 7 of I93 corresponds to A23 of the base address, and bit 0 of I93 corresponds to A16. A23 through A20 are only used on ISA bus systems if bit 2 of I94 is set to 1, enabling 24-bit addressing.

Note:

When DPRAM is used on the PCI bus, Universal Serial Bus (USB), or Ethernet, the host address is set by a "plug-and-play" process, and I93 is not used.

I93 is actually used at power-on/reset only, so to set or change the base address, change the value of I93, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I93 is copied at power-on/reset is X:$070009 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M93) to change the active setup without a **SAVE** and reset.

I94 VME Mailbox Base Address Bits A15-A08 ISA DPRAM Base Address Bits A15-A14 & Control

Range: $00 - $FF

Units: None

Default: $A0 (VME); $45 (ISA)

On VME bus systems, I94 controls bits A15 through A08 of the VME bus base address of the mailbox registers of Turbo PMAC. Bit 7 of I93 corresponds to A23 of the base address, and bit 0 of I93 corresponds to A16. I94 is used whether 16-bit, 24-bit, or 32-bit addressing has been selected with I90 and I99.

On ISA bus systems (PC, PC Ultralite, 3U Turbo with PC/104), I94 controls the enable state and addressing mode of the DPRAM. If the DPRAM is to appear as a 16k block of memory on the ISA bus, it also sets bits A15 and A14 of the ISA bus base address.

The first hex digit of I94 contains bits 4 – 7. When the DPRAM is addressed as a 16k x 8 block of memory on the ISA bus, bit 7 of I94 corresponds to A15, and bit 6 of I94 corresponds to A14. Bits 5 and 4 must be set to 0. When the extended 32k x 8 DPRAM is addressed as a 64k x 8 block of memory on the ISA bus, bits 7 through 4 of I94 must all be set to 0.

The second hex digit of I94 contains bits 0 – 3. These are individual control bits. Bits 0 and 2 control the addressing mode and block size. Bits 1 and 3 control the bank selection if the large DPRAM is addressed as a small block of memory. These should almost always be set to 0 in the I-variable. The commonly used settings of the second hex digit of I94 are:

- 0: DPRAM not enabled
- 1: 20-bit addressing (below 1M), 16k x 8 address block
- 4: 24-bit addressing (above or below 1M), 64k x 8 address block
- 5: 24-bit addressing (above or below 1M), 16k x 8 address block

> ***Note:***
>
> When DPRAM is used on the PCI bus, Universal Serial Bus (USB), or Ethernet, the host address is set by a "plug-and-play" process, and I94 is not used.

I94 is actually used at power-on/reset only, so to set or change, and keep, these settings, change the value of I94, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I94 is copied at power-on/reset is X:$07000A bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M94) to change the active setup without a **SAVE** and reset.

If the large (32k x 16) DPRAM is addressed through a small (16k x 8) address block, it is necessary to change the bank select bits (bits 1 and 3) of the active register to access all of the DPRAM from the PC. This is best done through the active control register at X:$07000A using suggested M-variable M94. The bit settings are:

- Bit 1 = 0, Bit 3 = 0: Bank 0 (PMAC addresses $060000 - $060FFF)
- Bit 1 = 1, Bit 3 = 0: Bank 1 (PMAC addresses $061000 - $061FFF)
- Bit 1 = 0, Bit 3 = 1: Bank 2 (PMAC addresses $062000 - $062FFF)
- Bit 1 = 1, Bit 3 = 1: Bank 3 (PMAC addresses $063000 - $063FFF)

I95 VME Interrupt Level

Range: $01 - $07

Units: None

Default: $02

I95 controls which interrupt level (1 to 7) Turbo PMAC will assert on the VME bus. Multiple boards on the same VME bus may assert the same interrupt level if each one has a unique set of interrupt vectors as set by I96.

I95 is actually used at power-on/reset only, so to set or change the VME interrupt level, change the value of I95, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I95 is copied at power-on/reset is X:$07000B bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M95) to change the active setup without a **SAVE** and reset.

I96 VME Interrupt Vector

Range: $00 - $FF

Units: None

Default: $A1

I96 controls which interrupt vectors will be provided when Turbo PMAC asserts a VME bus interrupt. If Turbo PMAC asserts the interrupt to signify that it has read a set of mailbox registers and is ready to accept another set, the interrupt vector value will be equal to (I96-1). If Turbo PMAC asserts the interrupt to signify that it has written to a set of mailbox registers and is ready for the host computer to read these, the interrupt vector value will be equal to I96. If Turbo PMAC asserts the interrupt to signify that it has put a line of text in the DPRAM ASCII response buffer and is ready for the host computer to read this, the interrupt vector value will be equal to (I96+1).

If there are multiple Turbo PMAC boards asserting the same interrupt level in the VME bus as set by I95, they each must assert a unique, non-overlapping set of interrupt vectors.

I96 is actually used at power-on/reset only, so to set or change the VME interrupt vector, change the value of I96, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I96 is copied at power-on/reset is X:$07000C bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M96) to change the active setup without a **SAVE** and reset.

I97 VME DPRAM Base Address Bits A23-A20

Range: $00 - $FF

Units: None

Default: $00

I97 controls bits A23 through A20 of the VME bus base address of the dual-ported RAM of Turbo PMAC. Bit 3 of I93 corresponds to A20 of the base address, and bit 0 of I93 corresponds to A16. I97 is only used if 24-bit or 32-bit addressing has been selected with I90 and I99.

Bits A19 through A14 of the DPRAM VME base address must be set by the host computer after every power-on/reset by writing a byte over the bus to the "page select" register in the Turbo PMAC's VME mailbox IC at the mailbox base address + $0121. This must be done even with the single-page 8k x 16 standard DPRAM option. With the extended DPRAM option, the host computer must write to the page select register every time a new page is accessed.

I97 is actually used at power-on/reset only, so to set or change bits 8 to 15 of the VME bus DPRAM base address, change the value of I97, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I97 is copied at power-on/reset is X:$07000D bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M97) to change the active setup without a **SAVE** and reset.

I98 VME DPRAM Enable

Range: $00 - $FF

Units: None

Default: $60

I98 controls whether VME access to the DPRAM IC on the Turbo PMAC is enabled or not. It should be set to $60 if DPRAM is not present to disable access; it should be set to $E0 if DPRAM is present to enable access.

I98 is actually used at power-on/reset only, so to set or change the DPRAM enabling, change the value of I98, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I98 is copied at power-on/reset is X:$07000E bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M98) to change the active setup without a **SAVE** and reset.

I99 VME Address Width Control

Range: $00 - $FF

Units: None

Default: $10

I99 controls the VME bus address width, with or without DPRAM. It should take one of six values in normal use:

- I99 = $00: 32-bit addressing, no DPRAM
- I99 = $10: 24-bit addressing, no DPRAM
- I99 = $30: 16-bit addressing, no DPRAM
- I99 = $80: 32-bit addressing, with DPRAM

- I99 = $90: 24-bit addressing, with DPRAM
- I99 = $B0: 16-bit addressing, with DPRAM

I99 is actually used at power-on/reset only, so to set or change the VME bus address width, change the value of I99, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of I99 is copied at power-on/reset is X:$07000F bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M99) to change the active setup without a **SAVE** and reset.

Motor Setup I-Variables

Motor Definition I-Variables

Ixx00 Motor xx Activation Control

Range: 0 - 1

Units: none

Default: I100 = 1, I200 .. I3200 = 0

Ixx00 determines whether Motor xx is de-activated (Ixx00 = 0) or activated (Ixx00 = 1). If activated, position monitoring, servo, and trajectory calculations are performed for the motor. An "activated" motor may be "enabled" – either in open or closed loop – or "disabled" (killed), depending on commands or events.

If Ixx00 is 0, no calculations are performed for Motor xx, not even position monitoring, so a position query command would not reflect position changes. Any Turbo PMAC motor not used in an application should be de-activated, so Turbo PMAC does not waste time doing calculations for that motor. The fewer motors that are activated, the faster the servo-update time will be.

Do not try to de-activate an active and enabled motor by setting Ixx00 to 0. The motor outputs would be left enabled with the last command level on them.

Ixx01 Motor xx Commutation Enable

Range: 0 - 3

Units: none

Default: 0

Ixx01 determines whether Turbo PMAC will perform the commutation calculations for Motor xx and controls whether X or Y registers are accessed for the motor. If Ixx01 is set to 0 or 2, Turbo PMAC performs no commutation calculations for this motor, and the single command output from the position/velocity-loop servo is output to the register specified by Ixx02. If Ixx01 is 0, this register is a Y-register; if Ixx01 is 2, this register is an X-register.

If Ixx01 is set to 1 or 3, Turbo PMAC performs the commutation calculations for the motor, and the output from the position/velocity-loop servo is an input to the commutation algorithm. Commutation position feedback is read from the register specified by Ixx83. If Ixx01 is 1, this register is an X-register; if Ixx01 is 3, this register is a Y-register. X-registers are typically used for commutation feedback directly on the Turbo PMAC; Y-registers are typically used for commutation feedback through the MACRO ring.

If Ixx01 is set to 1 or 3, Ixx70 through Ixx84 must be set to perform the commutation as desired. If Ixx82 is set to 0, Turbo PMAC will not perform current-loop calculations, and it outputs 2 phase-current commands. If Ixx82 is set greater than zero, then the Turbo PMAC performs current-loop calculations as well as commutation, and it outputs 3 phase-voltage commands.

Summarizing the values of Ixx01, and their effect:

- Ixx01 = 0: No Turbo PMAC commutation, command output to Y-register
- Ixx01 = 1: Turbo PMAC commutation, commutation feedback from X-register

(used for commutating with PMAC encoder register feedback)

- Ixx01 = 2: No Turbo PMAC commutation, command output to X-register
- Ixx01 = 3: Turbo PMAC commutation, commutation feedback from Y-register
  (used for commutating with feedback from MACRO ring)

Ixx02 Motor xx Command Output Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC Addresses

Turbo PMAC(1) Ixx02 Defaults

Ixx02	Value	Register	Ixx02	Value	Register
I102	$078003	PMAC DAC1	I1702	$079203	2nd ACC-24P/V DAC1
I202	$078002	PMAC DAC2	I1802	$079202	2nd ACC-24P/V DAC2
I302	$07800B	PMAC DAC3	I1902	$07920B	2nd ACC-24P/V DAC3
I402	$07800A	PMAC DAC4	I2002	$07920A	2nd ACC-24P/V DAC4
I502	$078103	PMAC DAC5	I2102	$079303	2nd ACC-24P/V DAC5
I602	$078102	PMAC DAC6	I2202	$079302	2nd ACC-24P/V DAC6
I702	$07810B	PMAC DAC7	I2302	$07930B	2nd ACC-24P/V DAC7
I802	$07810A	PMAC DAC8	I2402	$07930A	2nd ACC-24P/V DAC8
I902	$078203	1st ACC-24P/V DAC1	I2502	$07A203	3rd ACC-24P/V DAC1
I1002	$078202	1st ACC-24P/V DAC2	I2602	$07A202	3rd ACC-24P/V DAC2
I1102	$07820B	1st ACC-24P/V DAC3	I2702	$07A20B	3rd ACC-24P/V DAC3
I1202	$07820A	1st ACC-24P/V DAC4	I2802	$07A20A	3rd ACC-24P/V DAC4
I1302	$078303	1st ACC-24P/V DAC5	I2902	$07A303	3rd ACC-24P/V DAC5
I1402	$078302	1st ACC-24P/V DAC6	I3002	$07A302	3rd ACC-24P/V DAC6
I1502	$07830B	1st ACC-24P/V DAC7	I3102	$07A30B	3rd ACC-24P/V DAC7
I1602	$07830A	1st ACC-24P/V DAC8	I3202	$07A30A	3rd ACC-24P/V DAC8

Turbo PMAC2 Ixx02 Defaults

Ixx02	Value	Register	Ixx02	Value	Register
I102	$078002	PMAC2 DAC/PWM1A	I1702	$079202	2nd ACC-24P/V2 DAC/PWM1A
I202	$07800A	PMAC2 DAC/PWM2A	I1802	$07920A	2nd ACC-24P/V2 DAC/PWM2A
I302	$078012	PMAC2 DAC/PWM3A	I1902	$079212	2nd ACC-24P/V2 DAC/PWM3A
I402	$07801A	PMAC2 DAC/PWM4A	I2002	$07921A	2nd ACC-24P/V2 DAC/PWM4A
I502	$078102	PMAC2 DAC/PWM5A	I2102	$079302	2nd ACC-24P/V2 DAC/PWM5A
I602	$07810A	PMAC2 DAC/PWM6A	I2202	$07930A	2nd ACC-24P/V2 DAC/PWM6A
I702	$078112	PMAC2 DAC/PWM7A	I2302	$079312	2nd ACC-24P/V2 DAC/PWM7A
I802	$07811A	PMAC2 DAC/PWM8A	I2402	$07931A	2nd ACC-24P/V2 DAC/PWM8A
I902	$078202	1st ACC-24P/V2 DAC/PWM1A	I2502	$07A202	3rd ACC-24P/V2 DAC/PWM1A
I1002	$07820A	1st ACC-24P/V2 DAC/PWM2A	I2602	$07A20A	3rd ACC-24P/V2 DAC/PWM2A
I1102	$078212	1st ACC-24P/V2 DAC/PWM3A	I2702	$07A212	3rd ACC-24P/V2 DAC/PWM3A
I1202	$07821A	1st ACC-24P/V2 DAC/PWM4A	I2802	$07A21A	3rd ACC-24P/V2 DAC/PWM4A
I1302	$078302	1st ACC-24P/V2 DAC/PWM5A	I2902	$07A302	3rd ACC-24P/V2 DAC/PWM5A
I1402	$07830A	1st ACC-24P/V2 DAC/PWM6A	I3002	$07A30A	3rd ACC-24P/V2 DAC/PWM6A
I1502	$078312	1st ACC-24P/V2 DAC/PWM7A	I3102	$07A312	3rd ACC-24P/V2 DAC/PWM7A
I1602	$07831A	1st ACC-24P/V2 DAC/PWM8A	I3202	$07A31A	3rd ACC-24P/V2 DAC/PWM8A

Turbo PMAC2 Ultralite Ixx02 Defaults

Ixx02	Value	Register	Ixx02	Value	Register
I102	$078420	MACRO IC 0 Node 0 Reg. 0	I1702	$07A420	MACRO IC 2 Node 0 Reg. 0
I202	$078424	MACRO IC 0 Node 1 Reg. 0	I1802	$07A424	MACRO IC 2 Node 1 Reg. 0
I302	$078428	MACRO IC 0 Node 4 Reg. 0	I1902	$07A428	MACRO IC 2 Node 4 Reg. 0
I402	$07842C	MACRO IC 0 Node 5 Reg. 0	I2002	$07A42C	MACRO IC 2 Node 5 Reg. 0
I502	$078430	MACRO IC 0 Node 8 Reg. 0	I2102	$07A430	MACRO IC 2 Node 8 Reg. 0
I602	$078434	MACRO IC 0 Node 9 Reg. 0	I2202	$07A434	MACRO IC 2 Node 9 Reg. 0
I702	$078438	MACRO IC 0 Node 12 Reg. 0	I2302	$07A438	MACRO IC 2 Node 12 Reg. 0
I802	$07843C	MACRO IC 0 Node 13 Reg. 0	I2402	$07A43C	MACRO IC 2 Node 13 Reg. 0
I902	$079420	MACRO IC 1 Node 0 Reg. 0	I2502	$07B420	MACRO IC 3 Node 0 Reg. 0
I1002	$079424	MACRO IC 1 Node 1 Reg. 0	I2602	$07B424	MACRO IC 3 Node 1 Reg. 0
I1102	$079428	MACRO IC 1 Node 4 Reg. 0	I2702	$07B428	MACRO IC 3 Node 4 Reg. 0
I1202	$07942C	MACRO IC 1 Node 5 Reg. 0	I2802	$07B42C	MACRO IC 3 Node 5 Reg. 0
I1302	$079430	MACRO IC 1 Node 8 Reg. 0	I2902	$07B430	MACRO IC 3 Node 8 Reg. 0
I1402	$079434	MACRO IC 1 Node 9 Reg. 0	I3002	$07B434	MACRO IC 3 Node 9 Reg. 0
I1502	$079438	MACRO IC 1 Node 12 Reg. 0	I3102	$07B438	MACRO IC 3 Node 12 Reg. 0
I1602	$07943C	MACRO IC 1 Node 13 Reg. 0	I3202	$07B43C	MACRO IC 3 Node 13 Reg. 0

UMAC Turbo Ixx02 Defaults

Ixx02	Value	Register	Ixx02	Value	Register
I102	$078202	1st ACC-24E2x DAC/PWM1A	I1702	$07A202	5th ACC-24E2x DAC/PWM1A
I202	$07820A	1st ACC-24E2x DAC/PWM2A	I1802	$07A20A	5th ACC-24E2x DAC/PWM2A
I302	$078212	1st ACC-24E2x DAC/PWM3A	I1902	$07A212	5th ACC-24E2x DAC/PWM3A
I402	$07821A	1st ACC-24E2x DAC/PWM4A	I2002	$07A21A	5th ACC-24E2x DAC/PWM4A
I502	$078302	2nd ACC-24E2x DAC/PWM1A	I2102	$07A302	6th ACC-24E2x DAC/PWM1A
I602	$07830A	2nd ACC-24E2x DAC/PWM2A	I2202	$07A30A	6th ACC-24E2x DAC/PWM2A
I702	$078312	2nd ACC-24E2x DAC/PWM3A	I2302	$07A312	6th ACC-24E2x DAC/PWM3A
I802	$07831A	2nd ACC-24E2x DAC/PWM4A	I2402	$07A31A	6th ACC-24E2x DAC/PWM4A
I902	$079202	3rd ACC-24E2x DAC/PWM1A	I2502	$07B202	7th ACC-24E2x DAC/PWM1A
I1002	$07920A	3rd ACC-24E2x DAC/PWM2A	I2602	$07B20A	7th ACC-24E2x DAC/PWM2A
I1102	$079212	3rd ACC-24E2x DAC/PWM3A	I2702	$07B212	7th ACC-24E2x DAC/PWM3A
I1202	$07921A	3rd ACC-24E2x DAC/PWM4A	I2802	$07B21A	7th ACC-24E2x DAC/PWM4A
I1302	$079302	4th ACC-24E2x DAC/PWM1A	I2902	$07B302	8th ACC-24E2x DAC/PWM1A
I1402	$07930A	4th ACC-24E2x DAC/PWM2A	I3002	$07B30A	8th ACC-24E2x DAC/PWM2A
I1502	$079312	4th ACC-24E2x DAC/PWM3A	I3102	$07B312	8th ACC-24E2x DAC/PWM3A
I1602	$07931A	4th ACC-24E2x DAC/PWM4A	I3202	$07B31A	8th ACC-24E2x DAC/PWM4A

Ixx02 tells Motor xx which register or registers to which it writes its command output values. It contains the address of this register or the first (lowest addresses) of these multiple registers. This determines which output lines transmit the command output signals.

No Commutation: If Turbo PMAC is not commutating Motor xx (Ixx01=0 or 2), only one command output value is calculated, which is written to the register at the address specified in Ixx02. If Ixx01 is set to 0, this register is a Y-register; if Ixx01 is set to 2, this register is an X-register. Almost all output registers on PMAC are Y-registers; the only common use of X-register outputs is in the Type 0 MACRO protocol.

On Turbo PMAC(1) boards, if Ixx01 is set to 0 or 2 and Ixx96 is set to 1, then only the magnitude of the command is written to the register specified by Ixx02; the sign of the command is written to bit 14 of the flag register specified by Ixx25, which is usually the AENA/DIR output. If this sign-and-magnitude mode is used, bit 16 of Ixx24 should be set to 1 so this bit is not used for the amplifier-enable function. Sign-and-magnitude mode does not work with PMAC2-style Servo ICs.

The default values listed above are usually suitable for commanding single analog outputs (velocity or torque mode) when the Turbo PMAC is not commutating the motor.

Commutation, No Current Loop: If Turbo PMAC is commutating Motor xx (Ixx01=1 or 3), but not closing its current loop (Ixx82=0), two command output values are calculated, which are written to the Y-register at the address specified in Ixx02, plus the Y-register at the next higher address.

The default values listed above are usually suitable for commanding analog output pairs when the Turbo PMAC is commutating the motor, but not closing the current loop.

Commutation and Current Loop: If Turbo PMAC is commutating Motor xx (Ixx01=1 or 3) and closing its current loop (Ixx82>0), three command output values are calculated, which are written to the Y-register at the address specified in Ixx02, plus the Y-registers at the next two higher addresses.

The default values listed above are usually suitable for commanding three-phase PWM sets when the Turbo PMAC is commutating the motor, and closing the current loop.

Pulse Frequency Output: One common application type for which the default value of Ixx02 cannot be used is the direct pulse-and-direction output for stepper motor drives (Turbo PMAC2 only). This mode uses the 'C' output register alone for each channel, and I7mn6 for Servo IC **m** Channel **n** must be set to 2 or 3 to get pulse frequency output. In this case, the following values should be used:

Turbo PMAC2 Ixx02 Pulse Frequency Output Settings

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078004	$07800C	$078014	$07801C	1st IC on board PMAC2, 3U stack
1	$078104	$07810C	$078014	$07801C	2nd IC on board PMAC2, 3U stack
2	$078204	$07820C	$078214	$07821C	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078304	$07830C	$078314	$07831C	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079204	$07920C	$079214	$07921C	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079304	$07930C	$079314	$07931C	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A204	$07A20C	$07A214	$07A21C	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A304	$07A30C	$07A314	$07A31C	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B204	$07B20C	$07B214	$07B21C	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B304	$07B30C	$07B314	$07B31C	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

MACRO Type 1 Command Outputs: To write command outputs to MACRO registers for Type 1 MACRO devices such as the Delta Tau MACRO Station, the values of Ixx02 shown above as defaults for the Turbo PMAC2 Ultralite can be used.

MACRO Type 0 Command Outputs: To write single velocity or torque command outputs to MACRO registers for Type 0 MACRO drives such as the Performance Controls FLX Drive and the Kollmorgen FAST Drive, the values of Ixx02 in the following table should be used. Each value can select two registers (e.g. for Node 0 and Node 2). To select the lower-numbered node's register, which is a Y-register in Turbo PMAC, Ixx01 should be set to 0; to select the higher-numbered node's register, which is a Y-register, Ixx01 should be set to 2.

Ixx02 for Type 0 MACRO Commands

Ixx02	Value	Register	Ixx02	Value	Register
I102	$078423	MACRO IC 0 Node 0/2 Reg. 3	I1702	$07A423	MACRO IC 2 Node 0/2 Reg. 3
I202	$078427	MACRO IC 0 Node 1/3 Reg. 3	I1802	$07A427	MACRO IC 2 Node 1/3 Reg. 3
I302	$07842B	MACRO IC 0 Node 4/6 Reg. 3	I1902	$07A42B	MACRO IC 2 Node 4/6 Reg. 3
I402	$07842F	MACRO IC 0 Node 5/7 Reg. 3	I2002	$07A42F	MACRO IC 2 Node 5/7 Reg. 3
I502	$078433	MACRO IC 0 Node 8/10 Reg. 3	I2102	$07A433	MACRO IC 2 Node 8/10 Reg. 3
I602	$078437	MACRO IC 0 Node 9/11 Reg. 3	I2202	$07A437	MACRO IC 2 Node 9/11 Reg. 3
I702	$07843B	MACRO IC 0 Node 12/14 Reg. 3	I2302	$07A43B	MACRO IC 2 Node 12/14 Reg. 3
I802	$07843F	MACRO IC 0 Node 13/15 Reg. 3	I2402	$07A43F	MACRO IC 2 Node 13/15 Reg. 3
I902	$079423	MACRO IC 1 Node 0/2 Reg. 3	I2502	$07B423	MACRO IC 3 Node 0/2 Reg. 3
I1002	$079427	MACRO IC 1 Node 1/3 Reg. 3	I2602	$07B427	MACRO IC 3 Node 1/3 Reg. 3
I1102	$07942B	MACRO IC 1 Node 4/6 Reg. 3	I2702	$07B42B	MACRO IC 3 Node 4/6 Reg. 3
I1202	$07942F	MACRO IC 1 Node 5/7 Reg. 3	I2802	$07B42F	MACRO IC 3 Node 5/7 Reg. 3
I1302	$079433	MACRO IC 1 Node 8/10 Reg. 3	I2902	$07B433	MACRO IC 3 Node 8/10 Reg. 3
I1402	$079437	MACRO IC 1 Node 9/11 Reg. 3	I3002	$07B437	MACRO IC 3 Node 9/11 Reg. 3
I1502	$07943B	MACRO IC 1 Node 12/14 Reg. 3	I3102	$07B43B	MACRO IC 3 Node 12/14 Reg. 3
I1602	$07943F	MACRO IC 1 Node 13/15 Reg. 3	I3202	$07B43F	MACRO IC 3 Node 13/15 Reg. 3

Ixx03 Motor xx Position Loop Feedback Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC Addresses

Turbo PMAC/PMAC2 Ixx03 Defaults

Ixx03	Value	Register	Ixx03	Value	Register
I103	$003501	Conversion Table Line 0	I1703	$003511	Conversion Table Line 16
I203	$003502	Conversion Table Line 1	I1803	$003512	Conversion Table Line 17
I303	$003503	Conversion Table Line 2	I1903	$003513	Conversion Table Line 18
I403	$003504	Conversion Table Line 3	I2003	$003514	Conversion Table Line 19
I503	$003505	Conversion Table Line 4	I2103	$003515	Conversion Table Line 20
I603	$003506	Conversion Table Line 5	I2203	$003516	Conversion Table Line 21
I703	$003507	Conversion Table Line 6	I2303	$003517	Conversion Table Line 22
I803	$003508	Conversion Table Line 7	I2403	$003518	Conversion Table Line 23
I903	$003509	Conversion Table Line 8	I2503	$003519	Conversion Table Line 24
I1003	$00350A	Conversion Table Line 9	I2603	$00351A	Conversion Table Line 25
I1103	$00350B	Conversion Table Line 10	I2703	$00351B	Conversion Table Line 26
I1203	$00350C	Conversion Table Line 11	I2803	$00351C	Conversion Table Line 27
I1303	$00350D	Conversion Table Line 12	I2903	$00351D	Conversion Table Line 28
I1403	$00350E	Conversion Table Line 13	I3003	$00351E	Conversion Table Line 29
I1503	$00350F	Conversion Table Line 14	I3103	$00351F	Conversion Table Line 30
I1603	$003510	Conversion Table Line 15	I3203	$003520	Conversion Table Line 31

Turbo PMAC2 Ultralite Ixx03 Defaults

Ixx03	Value	Register	Ixx03	Value	Register
I103	$003502	Conversion Table Line 1	I1703	$003522	Conversion Table Line 33
I203	$003504	Conversion Table Line 3	I1803	$003524	Conversion Table Line 35
I303	$003506	Conversion Table Line 5	I1903	$003526	Conversion Table Line 37
I403	$003508	Conversion Table Line 7	I2003	$003528	Conversion Table Line 39
I503	$00350A	Conversion Table Line 9	I2103	$00352A	Conversion Table Line 41
I603	$00350C	Conversion Table Line 11	I2203	$00352C	Conversion Table Line 43
I703	$00350E	Conversion Table Line 13	I2303	$00352E	Conversion Table Line 45
I803	$003510	Conversion Table Line 15	I2403	$003530	Conversion Table Line 47
I903	$003512	Conversion Table Line 17	I2503	$003532	Conversion Table Line 49
I1003	$003514	Conversion Table Line 19	I2603	$003534	Conversion Table Line 51
I1103	$003516	Conversion Table Line 21	I2703	$003536	Conversion Table Line 53
I1203	$003518	Conversion Table Line 23	I2803	$003538	Conversion Table Line 55
I1303	$00351A	Conversion Table Line 25	I2903	$00353A	Conversion Table Line 57
I1403	$00351C	Conversion Table Line 27	I3003	$00353C	Conversion Table Line 59
I1503	$00351E	Conversion Table Line 29	I3103	$00353E	Conversion Table Line 61
I1603	$003520	Conversion Table Line 31	I3203	$003540	Conversion Table Line 63

Ixx03 tells the Turbo PMAC where to look for its position feedback value to close the position loop for Motor xx. It contains the address of the register where the motor will read its position feedback value.

Usually this is a result register in the "Encoder Conversion Table", where raw feedback values have been pre-processed at the beginning of each servo cycle. Feedback data is expected in units of 1/32 count (5 bits of fractional data). The result registers in the Encoder Conversion Table are located at addresses X:$003501 to X:$0035C0, corresponding to table setup I-variables I8000 to I8191, respectively.

For a control loop with dual feedback, motor and load, use Ixx03 to point to the encoder on the load, and Ixx04 to point to the encoder on the motor.

Note:

To use Turbo PMAC's hardware position-capture feature for homing search moves or other types of automatic move-until-trigger (Ixx97=0), the encoder channel number addressed by Ixx03 through the Encoder Conversion Table must match the channel number of the flags addressed by Ixx25.

Ixx04 Motor xx Velocity Loop Feedback Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC Addresses

Turbo PMAC/PMAC2 Ixx04 Defaults

Ixx04	Value	Register	Ixx04	Value	Register
I104	$003501	Conversion Table Line 0	I1704	$003511	Conversion Table Line 16
I204	$003502	Conversion Table Line 1	I1804	$003512	Conversion Table Line 17
I304	$003503	Conversion Table Line 2	I1904	$003513	Conversion Table Line 18
I404	$003504	Conversion Table Line 3	I2004	$003514	Conversion Table Line 19
I504	$003505	Conversion Table Line 4	I2104	$003515	Conversion Table Line 20
I604	$003506	Conversion Table Line 5	I2204	$003516	Conversion Table Line 21
I704	$003507	Conversion Table Line 6	I2304	$003517	Conversion Table Line 22
I804	$003508	Conversion Table Line 7	I2404	$003518	Conversion Table Line 23
I904	$003509	Conversion Table Line 8	I2504	$003519	Conversion Table Line 24
I1004	$00350A	Conversion Table Line 9	I2604	$00351A	Conversion Table Line 25
I1104	$00350B	Conversion Table Line 10	I2704	$00351B	Conversion Table Line 26
I1204	$00350C	Conversion Table Line 11	I2804	$00351C	Conversion Table Line 27
I1304	$00350D	Conversion Table Line 12	I2904	$00351D	Conversion Table Line 28
I1404	$00350E	Conversion Table Line 13	I3004	$00351E	Conversion Table Line 29
I1504	$00350F	Conversion Table Line 14	I3104	$00351F	Conversion Table Line 30
I1604	$003510	Conversion Table Line 15	I3204	$003520	Conversion Table Line 31

Turbo PMAC2 Ultralite Ixx04 Defaults

Ixx04	Value	Register	Ixx04	Value	Register
I104	$003502	Conversion Table Line 1	I1704	$003522	Conversion Table Line 33
I204	$003504	Conversion Table Line 3	I1804	$003524	Conversion Table Line 35
I304	$003506	Conversion Table Line 5	I1904	$003526	Conversion Table Line 37
I404	$003508	Conversion Table Line 7	I2004	$003528	Conversion Table Line 39
I504	$00350A	Conversion Table Line 9	I2104	$00352A	Conversion Table Line 41
I604	$00350C	Conversion Table Line 11	I2204	$00352C	Conversion Table Line 43
I704	$00350E	Conversion Table Line 13	I2304	$00352E	Conversion Table Line 45
I804	$003510	Conversion Table Line 15	I2404	$003530	Conversion Table Line 47
I904	$003512	Conversion Table Line 17	I2504	$003532	Conversion Table Line 49
I1004	$003514	Conversion Table Line 19	I2604	$003534	Conversion Table Line 51
I1104	$003516	Conversion Table Line 21	I2704	$003536	Conversion Table Line 53
I1204	$003518	Conversion Table Line 23	I2804	$003538	Conversion Table Line 55
I1304	$00351A	Conversion Table Line 25	I2904	$00353A	Conversion Table Line 57
I1404	$00351C	Conversion Table Line 27	I3004	$00353C	Conversion Table Line 59
I1504	$00351E	Conversion Table Line 29	I3104	$00353E	Conversion Table Line 61
I1604	$003520	Conversion Table Line 31	I3204	$003540	Conversion Table Line 63

Ixx04 tells the Turbo PMAC where to look for its *position* feedback value to close the *velocity* loop for Motor xx. It contains the address of the register where the motor will read its position feedback value.

Usually this is a result register in the "Encoder Conversion Table", where raw feedback values have been pre-processed at the beginning of each servo cycle. Feedback data is expected in units of 1/32 count (5 bits of fractional data). The result registers in the Encoder Conversion Table are located at addresses X:$003501 to X:$0035C0, corresponding to table setup I-variables I8000 to I8191, respectively.

For a control-loop with only a single feedback device – the usual case – Ixx03 and Ixx04 will have the same value, so the same register is used for both position and velocity loops. For a control loop with dual feedback, motor and load, use Ixx03 to point to the encoder on the load for the position loop, and Ixx04 to point to the encoder on the motor for the velocity loop. If the velocity loop uses feedback with different resolution from the position loop, the Ixx09 velocity-loop scale factor should be different from the Ixx08 position-loop scale factor.

Ixx05 Motor xx Master Position Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC 'X' Addresses

Default: $0035C0 (end of conversion table)

WARNING:

Never use the same register for master position and feedback position for the same motor. A dangerous runaway condition may result.

Ixx05 specifies the address of the register for "master" position information of Motor xx for the position following, or electronic gearing, function. Typically, this is a register in the encoder conversion table (addresses $003501 to $0035C0), where processed input position data resides.

The position following function is only enabled if Ixx06 is set to 1 or 3.

Ixx06 Motor xx Position Following Enable & Mode

Range: 0 - 3

Units: none

Default: 0

Ixx06 controls the position following function for Motor xx. It determines whether following is enabled or disabled, and whether the following function is in "normal" mode or "offset" (superimpose) mode.

Normal Mode: In normal following mode, motor position changes due to following are reported when the motor position is queried, and subsequent programmed moves for the motor cancel out the position changes due to the following function.

Offset Mode: In offset following mode, motor position changes due to following are not reported when the motor position is queried (the position reference is effectively offset for the motor), and subsequent programmed moves are added on top of the position changes due to the following function. This permits the superimposition of programmed and following moves in offset mode.

Ixx06 is a two-bit value. Bit 0 controls the enabling of the following function (0 = disabled, 1 = enabled). Bit 1 controls the following mode (0 = normal mode, 1 = offset mode). This yields four possible values for Ixx06:

- Ixx06 = 0: Following disabled, normal mode
- Ixx06 = 1: Following enabled, normal mode
- Ixx06 = 2: Following disabled, offset mode
- Ixx06 = 3: Following enabled, offset mode

Note:

The following mode can be important even when following is disabled, because it affects how subsequent programmed moves are calculated. If the following mode is ever changed, a **PMATCH** position-matching command must be executed before the next programmed move is calculated. Otherwise, that move will use the wrong value for its starting position, and a potentially dangerous jump will occur. (**PMATCH** is automatically executed on an **R** (run) or **S** (step) command.)

Ixx07 Motor xx Master (Handwheel) Scale Factor

Range: -8,388,608 - 8,388,607

Units: none

Default: 96

Ixx07 controls with what scaling the master (handwheel) register gets multiplied when extended into the full-length register. In combination with Ixx08, it controls the following ratio of Motor xx for position following (electronic gearing) according to the equation:

$$\Delta MotorPosition = \frac{Ixx07}{Ixx08} \Delta MasterPosition$$

For this position-following function, Ixx07 and Ixx08 can be thought of as the number of teeth on meshing gears in a mechanical coupling.

Ixx07 may be changed on the fly to permit real-time changing of the following ratio, but Ixx08 may not. Ixx08 should therefore be set to a large enough value to get the required fineness of ratio changes.

Ixx08 Motor xx Position Scale Factor

Range: 0 - 8,388,607

Units: none

Default: 96

Ixx08 specifies the multiplication scale factor for the internal position registers for Motor xx. Source position registers are multiplied by Ixx08 as the get extended into the full-length motor position registers. For most purposes, this is transparent to the user and Ixx08 does not need to be changed from the default.

There are two reasons that the user might want to change this from the default value. First, because it is involved in the "gear ratio" of the position following function -- the ratio is Ixx07/Ixx08 – the value of Ixx08 might be changed (usually raised) to get a more precise ratio.

The second reason to change this parameter (usually lowering it) is to prevent internal saturation at very high gains or count rates (velocity). PMAC's filter will saturate when the velocity in counts/sec multiplied by Ixx08 exceeds 768M (805,306,368). This only happens in very rare applications -- the count rate must exceed 8.3 million counts per second before the default value of Ixx08 gives a problem.

Note:

When changing this parameter, make sure the motor is killed (disabled). Otherwise, a sudden jump will occur, because the internal position registers will have changed. This means that this parameter should not be changed in the middle of an application. If a real-time change in the position-following gear ratio is desired, Ixx07 should be changed.

In most practical cases, Ixx08 should not be set above 1000 because higher values can make the servo filter saturate too easily. If Ixx08 is changed, Ixx30 should be changed inversely to keep the same servo performance (e.g. if Ixx08 is doubled, Ixx30 should be halved).

Ixx09 Motor xx Velocity-Loop Scale Factor

Range: 0 - 8,388,607

Units: none

Default: 96

Ixx09 specifies the multiplication scale factor for the internal actual velocity registers for Motor xx. Source position registers for the velocity loop are multiplied by Ixx09 before they are compared and used in the velocity loop. For most purposes, this is transparent to the user and Ixx09 does not need to be changed from the default.

This parameter should not be changed in the middle of an application, because it scales many internal values. If the same sensor is used to close both the position and velocity loops (Ixx03=Ixx04), Ixx09 should be set equal to Ixx08.

If different sensors are used, Ixx09 should be set such that the ratio of Ixx09 to Ixx08 is inversely proportional to the ratio of the velocity sensor resolution (at the load) to the position sensor resolution. If the value computed this way for Ixx09 does not come to an integer, use the nearest integer value.

Example:

If a 5000 line/inch (20,000 cts/in) linear encoder is used for position feedback, and a 500 line/rev (2000 cts/rev) rotary encoder is used for velocity loop feedback, and there is a 5-pitch screw, the effective resolution of the velocity encoder is 10,000 cts/in (2000*5), half of the position sensor resolution, so Ixx09 should be set to twice Ixx08.

Ixx10 Motor xx Power-On Servo Position Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC or Multiplexer Port Addresses

Default: $0

Ixx10 controls whether Turbo PMAC reads an absolute position sensor for Motor xx on power-up/reset and/or with the **$*** or **$$*** commands. If an absolute position read is to be done, Ixx10 specifies what register is read for that absolute position data. Ixx95 specifies how the data in this register is interpreted.

If Ixx10 is set to 0, no absolute power-on/reset position read is performed. The power-on/reset position is considered to be zero, even if an absolute sensor reporting a non-zero value is used. Ixx10 should be set to 0 when an incremental position sensor is used; a homing search move is typically then executed to establish a position reference.

If Ixx10 is set to a non-zero value, an absolute position read is performed for Motor xx at power-on/reset, from the register whose location is specified in Ixx10 (unless Bit 2 of Ixx80 is set to 1). This is either the address of a Turbo PMAC register, the multiplexed data address on the Multiplexer Port, or the *number* of the MACRO node on the Turbo PMAC, depending on the setting of Ixx95. The motor's position is set to the value read from the sensor location the Ixx26 home offset value.

Ixx10 is used only on power-on/reset, when the **$*** command is issued for the motor, or when the **$$*** command is issued for the coordinate system containing the motor. To get a new value of Ixx10 to take effect, either the **$*** or **$$*** command must be issued, or the value must be stored to non-volatile flash memory with the **SAVE** command, and the board must be reset.

Note:

Variable Ixx81 (with Ixx91) performs the same power-on position read function for the phasing (commutation) algorithm.

R/D Converter Read: If Ixx95 is set to a value from $000000 to $070000, or from $800000 to $870000, the address specified in Ixx10 is a Multiplexer Port address. Turbo PMAC will read the absolute position from an ACC-8D Opt 7 Resolver-to-Digital Converter board at that port address, as set by DIP switches on the board. Ixx95 specifies which R/D converter at that address is read, and whether it is treated as a signed or unsigned value.

If Ixx99 is greater than 0, the next R/D converter at that port address is also read as a second geared-down resolver, with Ixx99 setting the gear ratio. If Ixx98 is also greater than 0, the next R/D converter past that one at the same port address is read as a third geared-down resolver, with Ixx98 setting the gear ratio.

In this mode, bits 1 through 7 of Ixx10 match the settings of DIP-switches SW1-2 through SW1-8, respectively, on the ACC-8D Opt 7 R/D Converter board. A CLOSED (ON) switch represents a 0 value; an OPEN (OFF) switch represents a 1 value. Bit 0 and bits 9 through 23 of Ixx10 are always set to 0 in this mode; bit 8 is only set to 1 if all other bits are 0.

The following table shows the common Multiplexer Port addresses that can be used. Note that address 0 uses an Ixx10 value of $000100, because Ixx10=0 disables the absolute position read function.

Ixx10 for ACC-8D Opt. 7 Resolver/Digital Converter
(Ixx95=$000000 - $070000, $800000 - $870000) Addresses are Multiplexer Port Addresses

Board Mux. Addr.	Ixx10	Board Mux. Addr.	Ixx10	Board Mux. Addr.	Ixx10	Board Mux. Addr.	Ixx10
0	$000100	64	$000040	128	$000080	192	$0000C0
8	$000008	72	$000048	136	$000088	200	$0000C8
16	$000010	80	$000050	144	$000090	208	$0000D0
24	$000018	88	$000058	152	$000098	216	$0000D8
32	$000020	96	$000060	160	$0000A0	224	$0000E0
40	$000028	104	$000068	168	$0000A8	232	$0000E8
48	$000030	112	$000070	176	$0000B0	240	$0000F0
56	$000038	120	$000078	184	$0000B8	248	$0000F8

Parallel Word Read: If Ixx95 is set to a value from $080000 to $300000, from $480000 to $700000, from $880000 to $B00000, or from $C80000 to $F00000, the address specified in Ixx10 is a Turbo PMAC memory-I/O address, and Turbo PMAC will read the parallel word at that address. The least significant bit ("count") is expected at bit 0 of the address. The bit width (8 to 48 bits), the format (signed or unsigned), and the register type (X or Y) are determined by Ixx95.

The common sources for this type of read are ACC-14 parallel I/O expansion boards, and the MLDT timer registers. The following tables show the settings of Ixx10 for these devices.

Ixx10 Values for ACC-14D/V Registers
(Ixx95=$080000 to $300000 [unsigned], $880000 to $B00000 [signed])

Register	ACC-14 Select Jumper	Ixx10	Register	ACC-14 Select Jumper	Ixx10
1st ACC-14D/V Port A	E12	$078A00	4th ACC-14D/V Port A	E15	$078D00
1st ACC-14D/V Port B	E12	$078A01	4th ACC-14D/V Port B	E15	$078D01
2nd ACC-14D/V Port A	E13	$078B00	5th ACC-14D/V Port A	E16	$078E00
2nd ACC-14D/V Port B	E13	$078B01	5th ACC-14D/V Port B	E16	$078E01
3rd ACC-14D/V Port A	E14	$078C00	6th ACC-14D/V Port A	E17	$078F00
3rd ACC-14D/V Port B	E14	$078C01	6th ACC-14D/V Port B	E17	$078F01

Ixx10 for PMAC2-Style MLDT Timer Registers (Ixx95=$180000)

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078008	$078010	$078018	1st IC on board PMAC2, 3U stack
1	$078100	$078108	$078010	$078018	2nd IC on board PMAC2, 3U stack
2	$078200	$078208	$078210	$078218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078300	$078308	$078310	$078318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079200	$079208	$079210	$079218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079300	$079308	$079310	$079318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A200	$07A208	$07A210	$07A218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A300	$07A308	$07A310	$07A318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B200	$07B208	$07B210	$07B218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B300	$07B308	$07B310	$07B318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

It can also be used for registers in the 3U-format ACC-3E1 (for 3U Turbo Stack systems) and ACC-14E (for UMAC Turbo systems) boards. In this case, the last hex digit of Ixx91 must be set to a non-zero value to specify the byte-wide bus of these boards. The following tables show Ixx10 values for these boards.

Ixx10 Values for ACC-3E1 Registers in 3U Turbo Stack Systems
(Ixx95=$08000x to $30000x [unsigned], $88000x to $B0000x [signed])

ACC-3E1 Address Jumper	E1	E2	E3	E4
Ixx10 Value	$07880x	$07890x	$078A0x	$078B0x

Ixx10 Values for ACC-14E Registers in UMAC Turbo Systems
(Ixx95=$08000x to $30000x [unsigned], $88000x to $B0000x [signed])

DIP-Switch Setting	SW1-1 ON (0) SW1-2 ON (0)	SW1-1 OFF (1) SW1-2 ON (0)	SW1-1 ON (0) SW1-2 OFF (1)	SW1-1 OFF (1) SW1-2 OFF (1)
SW1-3 ON (0) SW1-4 ON (0)	$078C0x	$078D0x	$078E0x	$078F0x
SW1-3 OFF (1) SW1-4 ON (0)	$079C0x	$079D0x	$079E0x	$079F0x
SW1-3 ON (0) SW1-4 OFF (1)	$07AC0x	$07AD0x	$07AE0x	$07AF0x
SW1-3 OFF (1) SW1-4 OFF (1)	$07BC0x	$07BD0x	$07BE0x	$07BF0x
SW1-5 & 6 must be ON (0). ON means CLOSED; OFF means OPEN.				

The final digit, represented by an 'x' in both of these tables, can take a value of 0 to 5, depending on which I/O point on the board is used for the least significant bit (LSB):

Ixx10 Last Hex Digit 'x'	Pin Used for LSB	Pin Used for LSB	Pin Used for LSB
x=0	I/O00-07	I/O48-55	I/O96-103
x=1	I/O08-15	I/O56-63	I/O104-111
x=2	I/O16-23	I/O64-71	I/O112-119
x=3	I/O24-31	I/O72-79	I/O120-127
x=4	I/O32-39	I/O80-87	I/O128-135
x=5	I/O40-47	I/O88-95	I/O136-143

ACC-28 A/D Converter Read: If Ixx95 is set to $310000 or $B10000, the address specified by Ixx10 is a Turbo PMAC 'Y' memory-I/O address, and Turbo PMAC will read the data in the high 16 bits of that address as the absolute position (the LSB – one "count" – is in bit 8). This format is intended for the ACC-28A and ACC-28B A/D converters.

The following table shows the settings of Ixx10 for these registers.

Ixx10 Values for PMAC(1)-Style ADC Registers
(Ixx95=$B10000 for ACC-28A, Ixx95=$310000 for ACC-28B)

Register	PMAC	1st ACC-24P/V	2nd ACC-24P/V	3rd ACC-24P/V	4th ACC-24P/V
ADC1	$078006	$078206	$079206	$07A206	$07B206
ADC2	$078007	$078207	$079207	$07A207	$07B207
ADC3	$07800E	$07820E	$07920E	$07A20E	$07B20E
ADC4	$07800F	$07820F	$07920F	$07A20F	$07B20F
ADC5	$078106	$078306	$079306	$07A306	$07B306
ADC6	$078107	$078307	$079307	$07A307	$07B307
ADC7	$07810E	$07830E	$07930E	$07A30E	$07B30E
ADC8	$07810F	$07830F	$07930F	$07A30F	$07B30F

Ixx10 Values for PMAC2-Style ADC Registers using ACC-28B
(Ixx95=$B10000)

Register	PMAC2	1st ACC-24x2	2nd ACC-24x2	3rd ACC-24x2	4th ACC-24x2
ADC 1A	$078005	$078205	$079205	$07A205	$07B205
ADC 1B	$078006	$078206	$079206	$07A206	$07B206
ADC 2A	$07800D	$07820D	$07920D	$07A20D	$07B20D
ADC 2B	$07800E	$07820E	$07920E	$07A20E	$07B20E
ADC 3A	$078015	$078215	$079215	$07A215	$07B215
ADC 3B	$078016	$078216	$079216	$07A216	$07B216
ADC 4A	$07801D	$07821D	$07921D	$07A21D	$07B21D
ADC 4B	$07801E	$07821E	$07921E	$07A21E	$07B21E
ADC 5A	$078105	$078305	$079305	$07A305	$07B305
ADC 5B	$078106	$078306	$079306	$07A306	$07B306
ADC 6A	$07810D	$07830D	$07930D	$07A30D	$07B30D
ADC 6B	$07810E	$07830E	$07930E	$07A30E	$07B30E
ADC 7A	$078115	$078315	$079315	$07A315	$07B315
ADC 7B	$078116	$078316	$079316	$07A316	$07B316
ADC 8A	$07811D	$07831D	$07931D	$07A31D	$07B31D
ADC 8B	$07811E	$07831E	$07931E	$07A31E	$07B31E

Ixx10 Values for ACC-28E Registers in UMAC Turbo Systems
(Ixx95=$B10000)

DIP-Switch Setting	SW1-1 ON (0) SW1-2 ON (0)	SW1-1 OFF (1) SW1-2 ON (0)	SW1-1 ON (0) SW1-2 OFF (1)	SW1-1 OFF (1) SW1-2 OFF (1)
SW1-3 ON (0) SW1-4 ON (0)	$078C0x	$078D0x	$078E0x	$078F0x
SW1-3 OFF (1) SW1-4 ON (0)	$079C0x	$079D0x	$079E0x	$079F0x
SW1-3 ON (0) SW1-4 OFF (1)	$07AC0x	$07AD0x	$07AE0x	$07AF0x
SW1-3 OFF (1) SW1-4 OFF (1)	$07BC0x	$07BD0x	$07BE0x	$07BF0x
SW1-5 & 6 must be ON (0). ON means CLOSED; OFF means OPEN.				

The final digit, represented by an 'x' in both of these tables, can take a value of 0 to 3, depending on which ADC channel on the ACC-28E is used (x = Channel - 1).

Sanyo Absolute Encoder Read: If Ixx95 is set to $320000 or $B20000, the address specified in Ixx10 is a Turbo PMAC memory-I/O address, and Turbo PMAC will read the absolute position from an ACC-49 Sanyo Absolute Encoder Converter board at that address. Ixx95 specifies whether this position is treated as a signed or unsigned value.

The following table shows the possible settings of Ixx10 for ACC-49 Sanyo Absolute Encoder Converter boards.

Ixx10 Values for ACC-49 Sanyo Absolute Encoder Converter (Ixx95=$320000, $B20000)
Addresses are Turbo PMAC Memory-I/O Addresses

Enc. # on Board	Ixx10 for E1 ON	Ixx10 for E2 ON	Ixx10 for E3 ON	Enc. # on Board	Ixx10 for E4 ON	Ixx10 for E5 ON	Ixx10 for E6 ON
Enc. 1	$078A00	$078B00	$078C00	Enc. 3	$078D00	$078E00	$078F00
Enc. 2	$078A04	$078B04	$078C04	Enc. 4	$078D04	$078E04	$078F04

Yaskawa Absolute Encoder Read: If Ixx95 is set to $710000 or $F10000, the address specified in Ixx10 is a Multiplexer Port address, and Turbo PMAC will read the absolute position from an ACC-8D Opt 9 Yaskawa Absolute Encoder Converter board at that port address, as set by DIP switches on the board. Ixx95 specifies whether it is treated as a signed or unsigned value.

In this mode, bits 3 through 7 of Ixx10 match the settings of DIP switches SW1-1 through SW1-5, respectively, of the ACC-8D Opt 9 Yaskawa converter board. A CLOSED switch represents a bit value of 0; an OPEN switch represents a bit value of 1. Bits 0 through 2, and bits 8 though 23, of Ixx10 are always set to 0 in this mode.

The following table shows the Multiplexer Port addresses that can be used and the matching values of Ixx10. Note that address 0 uses an Ixx10 value of $000100, because Ixx10=0 disables the absolute position read function.

Ixx10 for ACC-8D Opt. 9 Yaskawa Absolute Encoder (Ixx95=$710000, $F10000)
Addresses are Multiplexer Port Addresses

Board Mux. Addr.	Ixx10 for Enc. 1	Ixx10 for Enc. 2	Ixx10 for Enc. 3	Ixx10 for Enc. 4	Board Mux. Addr.	Ixx10 for Enc. 1	Ixx10 for Enc. 2	Ixx10 for Enc. 3	Ixx10 for Enc. 4
0	$000100	$000002	$000004	$000006	128	$000080	$000082	$000084	$000086
8	$000008	$00000A	$00000C	$00000E	136	$000088	$00008A	$00008C	$00008E
16	$000010	$000012	$000014	$000016	144	$000090	$000092	$000094	$000096
24	$000018	$00001A	$00001C	$00001E	152	$000098	$00009A	$00009C	$00009E
32	$000020	$000022	$000024	$000026	160	$0000A0	$0000A2	$0000A4	$0000A6
40	$000028	$00002A	$00002C	$00002E	168	$0000A8	$0000AA	$0000AC	$0000AE
48	$000030	$000032	$000034	$000036	176	$0000B0	$0000B2	$0000B4	$0000B6
56	$000038	$00003A	$00003C	$00003E	184	$0000B8	$0000BA	$0000BC	$0000BE
64	$000040	$000042	$000044	$000046	192	$0000C0	$0000C2	$0000C4	$0000C6
72	$000048	$00004A	$00004C	$00004E	200	$0000C8	$0000CA	$0000CC	$0000CE
80	$000050	$000052	$000054	$000056	208	$0000D0	$0000D2	$0000D4	$0000D6
88	$000058	$00005A	$00005C	$00005E	216	$0000D8	$0000DA	$0000DC	$0000DE
96	$000060	$000062	$000064	$000066	224	$0000E0	$0000E2	$0000E4	$0000E6
104	$000068	$00006A	$00006C	$00006E	232	$0000E8	$0000EA	$0000EC	$0000EE
112	$000070	$000072	$000074	$000076	240	$0000F0	$0000F2	$0000F4	$0000F6
120	$000078	$00007A	$00007C	$00007E	248	$0000F8	$0000FA	$0000FC	$0000FE

MACRO Absolute Position Read: If Ixx95 contains a value from $720000 to $740000, or from $F20000 to $F40000, the value specified in Ixx10 is a MACRO node number, and Turbo PMAC will obtain the absolute power-on position through the MACRO ring. Ixx95 specifies what type of position data is used, and whether it is treated as a signed or unsigned value.

The MACRO node number is specified in the last two hex digits of Ixx10. The second-to-last digit specifies the MACRO IC number 0 to 3 (1, 2, and 3 exist only on Ultralite versions of the Turbo PMAC2, or a UMAC Turbo with ACC-5E). Note that the MACRO IC number on the Turbo PMAC does not necessarily match the ring master number for that IC, although it often will. The last digit specifies the MACRO node number 0 to 15 (0 to F hex) in that IC. This function is only supported in nodes 0, 1, 4, 5, 8, 9, 12 (C), and 13 (D).

The following table shows the required values of Ixx10 for all of the MACRO nodes that can be used. Note that MACRO IC 0 Node 0 uses an Ixx10 value of $000100, because Ixx10=0 disables the absolute position read function.

Ixx10 for MACRO Absolute Position Reads
(Ixx95=$720000 - $740000, $F20000 - $F40000)
Addresses are MACRO Node Numbers

MACRO Node Number	Ixx10 for MACRO IC 0	Ixx10 for MACRO IC 1	Ixx10 for MACRO IC 2	Ixx10 for MACRO IC 3
0	$000100	$000010	$000020	$000030
1	$000001	$000011	$000021	$000031
4	$000004	$000014	$000024	$000034
5	$000005	$000015	$000025	$000035
8	$000008	$000018	$000028	$000038
9	$000009	$000019	$000029	$000039
12	$00000C	$00001C	$00002C	$00003C
13	$00000D	$00001D	$00002D	$00003D

If obtaining the absolute position through a Delta Tau MACRO Station or equivalent, MACRO Station setup variable MI11x for the matching node must be set properly to obtain the type of information desired.

Motor Safety I-Variables

Ixx11 Motor xx Fatal Following Error Limit

Range: 0 - 8,388,607

Units: 1/16 count

Default: 32,000 (2000 counts)

Ixx11 sets the magnitude of the following error for Motor xx at which operation will shut down. When the magnitude of the following error exceeds Ixx11, Motor xx is disabled (killed). If the motor's coordinate system is executing a program at the time, the program is aborted. It is optional whether other PMAC motors are disabled when this motor exceeds its following error limit; bits 21 and 22 of Ixx24 control what happens to the other motor (the default is that all PMAC motors are disabled).

A status bit for the motor, and one for the coordinate system (if the motor is in one) are set. On Turbo PMAC(1), if this coordinate system is hardware-selected on JPAN (with I2=0), or software-addressed by the host (with I2=1), the ERLD/ output on JPAN is turned on. On ISA bus cards, the following error input to the interrupt controller is triggered.

Setting Ixx11 to zero disables the fatal-following error limit for the motor. This may be desirable during initial development work, but it is strongly discouraged in an actual application. A fatal following error limit is a very important protection against various types of faults, such as loss of feedback, that cannot be detected directly, and that can cause severe damage to people and equipment.

Note:

The units of Ixx11 are 1/16 of a count. Therefore, this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ixx11 should be set to 16,000.

Ixx12 Motor xx Warning Following Error Limit

Range: 0 - 8,388,607

Units: 1/16 count

Default: 16,000 (1000 counts)

Ixx12 sets the magnitude of the following error for Motor xx at which a warning flag goes true. If this limit is exceeded, status bits are set for the motor and the motor's coordinate system (if any). The coordinate system status bit is the logical OR of the status bits of all the motors in the coordinate system.

Setting this parameter to zero disables the warning following error limit function. If this parameter is set greater than the Ixx11 *fatal* following error limit, the warning status bit will never go true, because the fatal limit will disable the motor first.

If bit 1 of Ixx97 is set to 1, the motor can be triggered for homing search moves, jog-until-trigger moves, and motion program move-until-trigger moves when the following error exceeds Ixx12. This is known as torque-mode triggering, because the trigger will occur at a torque level corresponding to the Ixx12 limit. Bit 0 of Ixx97 should also be set to 1 to enable software position capture, making the value of Ixx97 equal to 3 in this mode.

At any given time, one coordinate system's status bit can be output to several places; which system depends on what coordinate system is hardware-selected on the panel input port if I2=0, or what coordinate system is software-addressed from the host (&n) if I2=1.

The outputs that work in this way are F1LD/ (pin 23 on connector J2 on Turbo PMAC(1) only), F1ER (line IR3 into the programmable interrupt controller (PIC) on Turbo PMAC-PC) and, if E28 connects pins 1 and 2, FEFCO/ (on the JMACH1 connector on Turbo PMAC(1) only).

Note:

The units of Ixx12 are 1/16 of a count. Therefore, this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ixx12 should be set to 16,000.

Ixx13 Motor xx Positive Software Position Limit

Range: -2^{35} - $+2^{35}$

Units: counts

Default: 0 (disabled)

Ixx13 sets the maximum permitted positive position value for Motor xx. It can work in two slightly different ways.

1. Actual position limit: Turbo PMAC's "housekeeping" functions repeatedly compare the actual position of Motor xx to Ixx13. If the motor is closed-loop, and the actual position is greater in an absolute sense (not magnitude) than Ixx13, Turbo PMAC automatically issues an Abort command, which causes this motor to start decelerating to a stop at the rate set by Ixx15. If other motors are in coordinated motion, they are also brought to a stop at their own Ixx15 rate.

Note:

In this mode, the deceleration starts after the limit has been reached, so the motion will end outside the limit.

If the motor is in open-loop enabled mode (from an O-command) when it exceeds the Ixx13 limit, it will be killed (open-loop disabled). If the limit has already been exceeded, no open-loop commands are accepted for this motor, regardless of polarity.

While the Ixx13 limit is exceeded, Turbo PMAC will allow no more positive-direction commands, whether from a programmed move, a jog command, or from the position-following function. However it will allow negative-direction commands of any of these types, permitting a controlled exit from the limit.

2. Desired position limit: If bit 15 of Ixx24 is set to 1, enabling desired position limit checking, Turbo PMAC will compared the desired motor target as calculated by the motion program position – either end of programmed move, or end of intermediate segment – to the limit. If this target position is not calculated within the special lookahead buffer, when this position is greater in an absolute sense (not magnitude) than Ixx13, Turbo PMAC automatically issues an Abort command, which causes this motor to start decelerating to a stop at the rate set by Ixx15. If other motors are in coordinated motion, they are also brought to a stop at their own Ixx15 rate.

If this target position is calculated within the special lookahead buffer, when this position is greater in an absolute sense (not magnitude) than [Ixx13-Ixx41], Turbo PMAC modifies this position to [Ixx13-Ixx41]. Depending on the setting of bit 14 of Ixx24, it either brings the program to a controlled stop at this point (bit 14=0) or continues the program with the motor position saturated to this value (bit 14=1).

If stopped at the limit in lookahead, reversal along the path is possible. Commands for forward execution into the limit will execute one segment at a time in a point-to-point fashion. If the software limit is extended, normal program execution may be resumed. Because program execution is technically only suspended when stopped at the limit in this mode, an Abort command must be issued before another program can be run.

Lookahead is active for **LINEAR** and **CIRCLE** mode moves, provided that the lookahead buffer is defined, and with Isx13 and Isx20 set to values greater than 0.

If Ixx13 is set to 0, there is no positive software limit (if you want 0 as a limit, use 1). This limit is automatically de-activated during homing-search moves, until the home trigger is found. It is active during the post-trigger move.

Ixx13 is referenced to the most recent power-up zero position or homing-move zero position. The physical position at which this limit occurs is not affected by axis-offset commands (e.g. **PSET, {axis}=**), although these commands will change the *reported* position value at which the limit occurs.

Note:

It is possible to set this parameter outside the range $\pm 2^{35}$ ($\pm$64 billion) if a couple of special things are done. First, the Ixx08 scale factor for the motor must be reduced to give the motor the range to use this position (motor range is $\pm 2^{42}$/Ixx08). Second, the variable value must be calculated inside Turbo PMAC, because the command parser cannot accept constants outside the range $+2^{35}$ (e.g. to set I113 to 100 billion, use **I113=1000000000*100**).

Ixx14 Motor xx Negative Software Position Limit

Range: -2^{35} - $+2^{35}$

Units: counts

Default: 0 (disabled)

Ixx14 sets the maximum permitted positive position value for Motor xx. It can work in two slightly different ways.

1. Actual position limit: Turbo PMAC's "housekeeping" functions repeatedly compare the actual position of Motor xx to Ixx14. If the motor is closed-loop, and the actual position is less in an absolute sense (not magnitude) than Ixx14, Turbo PMAC automatically issues an Abort command, which causes this motor to start decelerating to a stop at the rate set by Ixx15. If other motors are in coordinated motion, they are also brought to a stop at their own Ixx15 rate.

Note:

In this mode, the deceleration starts after the limit has been reached, so the motion will end outside the limit.

If the motor is in open-loop enabled mode (from an O-command) when it exceeds the Ixx14 limit, it will be killed (open-loop disabled). If the limit has already been exceeded, no open-loop commands are accepted for this motor, regardless of polarity.

While the Ixx14 limit is exceeded, Turbo PMAC will allow no more negative-direction commands, whether from a programmed move, a jog command, or from the position-following function. However it will allow positive-direction commands of any of these types, permitting a controlled exit from the limit.

2. Desired position limit: If bit 15 of Ixx24 is set to 1, enabling desired position limit checking, Turbo PMAC will compared the desired motor target as calculated by the motion program position – either end of programmed move, or end of intermediate segment – to the limit. If this target position is not calculated within the special lookahead buffer, when this position is less in an absolute sense (not magnitude) than Ixx14, Turbo PMAC automatically issues an Abort command, which causes this motor to start decelerating to a stop at the rate set by Ixx15. If other motors are in coordinated motion, they are also brought to a stop at their own Ixx15 rate.

If this target position is calculated within the special lookahead buffer, when this position is less in an absolute sense (not magnitude) than [Ixx14+Ixx41], Turbo PMAC modifies this position to [Ixx14+Ixx41]. Depending on the setting of bit 14 of Ixx24, it either brings the program to a controlled stop at this point (bit 14=0) or continues the program with the motor position saturated to this value (bit 14=1). If stopped at the limit in lookahead, reversal along the path is possible. Commands for forward execution will execute one segment at a time in a point-to-point fashion.

Lookahead is active for **LINEAR** and **CIRCLE** mode moves, provided that the lookahead buffer is defined, and with Isx13 and Isx20 set to values greater than 0.

If Ixx14 is set to 0, there is no positive software limit (if you want 0 as a limit, use 1). This limit is automatically de-activated during homing-search moves, until the home trigger is found. It is active during the post-trigger move.

Ixx14 is referenced to the most recent power-up zero position or homing-move zero position. The physical position at which this limit occurs is not affected by axis-offset commands (e.g. **PSET, {axis}=**), although these commands will change the *reported* position value at which the limit occurs.

Note:

It is possible to set this parameter outside the range $\pm 2^{35}$ ($\pm$64 billion) if a couple of special things are done. First, the Ixx08 scale factor for the motor must be reduced to give the motor the range to use this position (motor range is $\pm 2^{42}$/Ixx08). Second, the variable value must be calculated inside Turbo PMAC, because the command parser cannot accept constants outside the range $+2^{35}$ (e.g. to set I114 to -100 billion, use **I114=-1000000000*100**).

Ixx15 Motor xx Abort/Limit Deceleration Rate

Range:	Positive Floating-Point
Units:	counts / msec^2
Default:	0.25

CAUTION:

Do not set this parameter to zero, or the motor will continue indefinitely after an abort or limit.

Ixx15 sets the *rate* of deceleration that Motor xx will use if it exceeds a hardware or software limit, or has its motion aborted by command (**A or <CONTROL-A>**). This value should usually be set to a value near the maximum physical capability of the motor. It is not a good idea to set this value past the capability of the motor, because doing so increases the likelihood of exceeding the following error limit, which stops the braking action, and could allow the axis to coast into a hard stop.

Example:

Suppose your motor had 125 encoder lines (500 counts) per millimeter, and you wished it to decelerate at 4000 mm/sec^2. You would set Ixx15 to 4000 mm/sec^2 * 500 cts/mm * sec^2/1,000,000 msec^2 = 2.0 cts/msec^2.

Ixx16 Motor xx Maximum Program Velocity

Range: Positive Floating-Point

Units: counts / msec

Default: 32.0

Ixx16 sets a limit to the magnitude of the commanded velocity for certain programmed moves in certain modes on Turbo PMAC.

1. Non-segmented LINEAR mode moves: If the Isx13 segmentation time parameter for the coordinate system containing Motor xx is set to 0, which takes the coordinate system out of segmentation mode, then Ixx16 serves as the maximum velocity for Motor xx in **LINEAR**-mode moves in the coordinate system. If a **LINEAR** move command in a motion program requests a higher velocity magnitude of this motor, all motors in the coordinate system are slowed down proportionately so that the motor will not exceed this parameter, yet the path will not be changed.

If Isx13 is set to 0, **CIRCLE** mode moves and cutter radius compensation can not be performed.

2. Segmented LINEAR and CIRCLE mode moves with lookahead: If the Isx13 segmentation time parameter for the coordinate system containing Motor xx is set greater than 0, putting the coordinate system in segmentation mode, and the special multi-block lookahead function is active (lookahead buffer defined, and Isx20 greater than 0), then Ixx16 serves as the maximum velocity for Motor xx in all segments of **LINEAR** and **CIRCLE** mode moves in the coordinate system. If a segment of one of these programmed moves requests a higher velocity magnitude of this motor, all motors in the coordinate system are slowed down proportionately so that the motor will not exceed this parameter, yet the path will not be changed.

Note:

Ixx16 is *not* used for segmented **LINEAR** and **CIRCLE** mode moves when the special lookahead buffer is not active.

3. RAPID mode moves: Ixx16 also sets the speed of a programmed **RAPID** mode move for the motor, provided that variable Ixx90 is set to 1 (if Ixx90 is set to 0, jog speed parameter Ixx22 is used instead). This happens regardless of the setting of Isx13.

The Ixx16 velocity limit calculations assume that the coordinate system is operating at the %100 override value (real-time). The true velocity will vary proportionately with the override value.

Ixx17 Motor xx Maximum Program Acceleration

Range: Positive Floating-Point

Units: counts / msec^2

Default: 0.5

Ixx17 sets a limit to the magnitude of the commanded acceleration for certain programmed moves in certain modes on Turbo PMAC.

1. Non-segmented LINEAR mode moves: If the Isx13 segmentation time parameter for the coordinate system containing Motor xx is set to 0, which takes the coordinate system out of segmentation mode, then Ixx17 serves as the maximum velocity for Motor xx in **LINEAR**-mode moves in the coordinate system. If a **LINEAR** move command in a motion program requests a higher acceleration magnitude of this motor given its **TA** and **TS** time settings, the acceleration time for all motors in the coordinate system is extended so that the motor will not exceed this parameter, yet full coordination is maintained.

If Isx13 is set to 0, **CIRCLE** mode moves and cutter radius compensation can not be performed.

In this mode, Turbo PMAC cannot extend the acceleration time to a greater value than the incoming move time, because to go further would require re-calculating already executed moves.

If observing acceleration limits (especially for deceleration) requires acceleration or deceleration over multiple programmed moves, the Ixx17 limit in this mode cannot guarantee that the limits will be observed. Special lookahead is required for this capability.

In this mode, the Ixx17 acceleration limit can lower the speed of short programmed moves, even if they are intended to be blended together at high speed. The algorithm limits the speed of each move so that it can decelerate to a stop within that move. Without special lookahead, it cannot assume that it will blend at full speed into another move.

2. Segmented `LINEAR` and `CIRCLE` mode moves with lookahead: If the Isx13 segmentation time parameter for the coordinate system containing Motor xx is set greater than 0, putting the coordinate system in segmentation mode, and the special multi-block lookahead function is active (lookahead buffer defined, and Isx20 greater than 0), then Ixx17 serves as the maximum acceleration for Motor xx in all segments of **`LINEAR`** and **`CIRCLE`** mode moves in the coordinate system. If a segment of one of these programmed moves requests a higher acceleration magnitude of this motor, the segment time for all motors in the coordinate system is extended so that the motor will not exceed this parameter, yet full coordination is maintained. Furthermore, the Turbo PMAC will work back through already calculated, but not yet executed, segments, to make sure the change in this segment does not cause violations in any of those segments.

Note:

Ixx17 is *not* used for segmented **`LINEAR`** and **`CIRCLE`** mode moves when the special lookahead buffer is not active.

The Ixx17 acceleration limit calculations assume that the coordinate system is operating at the %100 override value (real-time). The true acceleration will vary proportionately with the square of the override value.

The use of the Ixx17 limit permits the setting of very small TA and/or TS values (Ixx87 and Ixx88 by default). Do not set both of these values to 0, or a division-by-zero calculation error could occur. It is advised that you set your TA time no smaller the minimum programmed move block time that you want to occur.

Example:

Given axis definitions of **`#1->10000X`**, **`#2->10000Y`**, Isx13=0 and Ixx17 for each motor of 0.25, and the following motion program segment:

```
INC F10 TA200 TS0
X20
Y20
```

The rate of acceleration from the program at the corner for motor #2 (X) is ((0-10)units/sec * 10000 cts/unit * sec/1000msec) / 200 msec = -0.5 cts/msec^2. The acceleration of motor #2 (Y) is +0.5 cts/msec^2. Since this is twice the limit, the acceleration will be slowed so that it takes 400 msec.

With the same setup parameters and the following program segment:

```
INC F10 TA200 TS0
X20 Y20
X-20 Y20
```

The rate of acceleration from the program at the corner for motor #1 (X) is ((-7.07-7.07)units/sec * 10000 cts/unit * sec/1000msec) / 200 msec = -0.707 cts/msec^2. The acceleration of motor #2 (Y) is 0.0. Since motor #1 exceeds its limit, the acceleration time will be lengthened to 200 * 0.707/0.25 = 707 msec.

Note:

In the second case, the acceleration time is made longer (the corner is made larger) for what is an identically shaped corner (90°). In a contouring XY application, this parameter should not be relied upon to produce consistently sized corners without the special lookahead algorithm.

Ixx19 Motor xx Maximum Jog/Home Acceleration

Range: Positive Floating-Point

Units: counts / msec^2

Default: 0.15625

Ixx19 sets a limit to the commanded acceleration magnitude for jog and home moves, and for **RAPID**-mode programmed moves, of Motor xx. If the acceleration times in force at the time (Ixx20 and Ixx21) request a higher rate of acceleration, this rate of acceleration will be used instead. The calculation does not take into account any feedrate override (%value other than 100).

Since jogging moves are usually not coordinated between motors, many people prefer to specify jog acceleration by rate, not time. To do this, simply set Ixx20 and Ixx21 low enough that the Ixx19 limit is always used. Do not set both Ixx20 and Ixx21 to 0, or a division-by-zero error will result in the move calculations, possibly causing erratic operations. The minimum acceleration *time* settings that should be used are Ixx20=1 and Ixx21=0.

The default limit of 0.015625 counts/msec^2 is quite low and will probably limit acceleration to a lower value than is desired in most systems; most users will eventually raise this limit. This low default was used for safety reasons.

Example:
With Ixx20 (acceleration time) at 100 msec, Ixx21 (S-curve time) at 0, and Ixx22 (jog speed) at 50 counts/msec, a jog command from stop would request an acceleration of (50 cts/msec) / 100 msec, or 0.5 cts/msec^2. If Ixx19 were set to 0.25, the acceleration would be done in 200 msec, not 100 msec.

With the same parameters in force, an on-the-fly reversal from positive to negative jog would request an acceleration of (50-(-50) cts/msec) / 100 msec, or 1.0 cts/msec^2. The limit would extend this acceleration period by a factor of 4, to 400 msec.

Motor Motion I-Variables

Ixx20 Motor xx Jog/Home Acceleration Time

Range: 0 - 8,388,607

Units: msec

Default: 0 (so Ixx21 controls)

Ixx20 establishes the time spent in acceleration in a jogging, homing, or programmed **RAPID**-mode move (starting, stopping, and changing speeds). However, if Ixx21 (jog/home S-curve time) is greater than half this parameter, the total time spent in acceleration will be 2 times Ixx21. Therefore, if Ixx20 is set to 0, Ixx21 alone controls the acceleration time in "pure" S-curve form. In addition, if the maximum acceleration rate set by these times exceeds what is permitted for the motor (Ixx19), the time will be increased so that Ixx19 is not exceeded.

Note:

Do not set both Ixx20 and Ixx21 to 0 simultaneously, even if you are relying on Ixx19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.

A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. **J=**), or at least the end of the jog (**J/**).

Ixx21 Motor xx Jog/Home S-Curve Time

Range: 0 - 8,388,607

Units: msec

Default: 50

Ixx21 establishes the time spent in each "half" of the "S" for S-curve acceleration in a jogging, homing, or **RAPID**-mode move (starting, stopping, and changing speeds). If this parameter is more than half of Ixx20, the total acceleration time will be 2 times Ixx21, and the acceleration time will be "pure" S-curve (no constant acceleration portion). If the maximum acceleration rate set by Ixx20 and Ixx21 exceeds what is permitted for the motor (Ixx19), the time will be increased so that Ixx19 is not exceeded.

Note:

Do not set both Ixx20 and Ixx21 to 0 simultaneously, even if you are relying on Ixx19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.

A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. **J=**), or at least the end of the jog (**J/**).

Ixx22 Motor xx Jog Speed

Range: Positive Floating Point

Units: counts / msec

Default: 32.0

Ixx22 establishes the commanded speed of a jog move, or a programmed **RAPID**-mode move (if Ixx90=0) for Motor xx. Direction of the jog move is controlled by the jog command.

A change in this parameter will not take effect until the next move command. For instance, if you wanted to change the jog speed on the fly, you would start the jog move, change this parameter, then issue a new jog command.

Ixx23 Motor xx Home Speed and Direction

Range: Floating Point

Units: counts / msec

Default: 32.0

Ixx23 establishes the commanded speed and direction of a homing-search move for Motor xx. Changing the sign reverses the direction of the homing move -- a negative value specifies a home search in the negative direction; a positive value specifies the positive direction.

Ixx24 Motor xx Flag Mode Control

Range: $000000 - $FFFFFF

Units: none

Default: $000000 (Turbo PMAC(1) boards)
$000001 (non-Ultralite Turbo PMAC2 boards)
$840001 (Turbo PMAC2 Ultralite boards)

Ixx24 specifies how the flag information in the register(s) specified by Ixx25, Ixx42, and Ixx43 is used. Ixx24 is a set of 24 individual control bits – bits 0 to 23. Currently bits 0 and 11 to 23 are used.

Note:

It is easier to specify this parameter in hexadecimal form. With I9 at 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

Bit 0: Flag Register Type Bit: If bit 0 is set to zero, the Turbo PMAC expects the flag registers to be in the format of a PMAC(1)-style Servo IC. Bit 0 should be set to 0 for any flags on-board a Turbo PMAC(1), an ACC-24P, or an ACC24V.

If bit 0 is set to one, the Turbo PMAC expects the flag registers to be in the format of a PMAC2-style Servo IC. Bit 0 should be set to 1 for any flag register on-board a Turbo PMAC2, an ACC-24P2, an ACC-24V2, an ACC-24E2, or coming from a MACRO Station.

If multiple flag registers are specified by non-zero settings of Ixx42 and/or Ixx43, all registers must be of the same format.

Bit 11: Capture with High-Resolution Feedback Bit: If bit 11 is set to zero when hardware position capture is used in a triggered move such as a homing-search move, the captured data (whether whole-count only or including sub-count data) is processed to match servo feedback of "normal" resolution (5 bits of fractional count data per hardware whole count). This setting is appropriate for digital quadrature feedback or for "low-resolution" interpolation of a sinusoidal encoder.

If bit 11 (value $800, or 2,048) is set to one when hardware position capture is used in a triggered move, the captured data (whether whole-count only or including sub-count data) is processed to match servo feedback of "high" resolution (10 bits of fractional count data per hardware whole count). This setting is appropriate for "high-resolution" interpolation of a sinusoidal encoder through an ACC-51x interpolator.

Bit 12: Sub-Count Capture Enable Bit: If bit 12 is set to zero when hardware position capture is used in a triggered move such as a homing-search move, only the whole-count captured position register is used to establish the trigger position. This setting must be used with PMAC(1)-style Servo ICs, and with PMAC2-style Servo ICs older than Revision "D" (Revision "D" ICs started shipping in early 2002).

If bit 12 (value $1000, or 4,096) is set to one when hardware position capture is used in a triggered move, both the whole-count captured position register and the estimated sub-count position register are used to establish the trigger position. A PMAC2-style Servo IC of Revision "D" or newer must be used for this mode, and I7mn9 for the channel used must be set to 1 to enable the hardware sub-count estimation. This setting is typically used for registration or probing triggered moves with interpolated sinusoidal encoder feedback. (Even with interpolated sinusoidal encoder feedback, homing search moves will probably be done without sub-count captured data, to force a home position referenced to one of the four "zero-crossing" positions of the sine/cosine signals.)

Bit 13 Error Saturation Control Bit: If bit 13 is set to zero, when the motor's following error exceeds the Ixx67 position-error limit, the error is simply truncated by the limit parameter.

If bit 13 (value $2000, or 8,192) is set to 1, when the motor's following error exceeds the Ixx67 position-error limit, the excess is put in the "master position" register for the motor, so it is eventually recoverable.

Bit 14: Continue on Desired Position Limit Bit: If bit 14 is set to zero when desired position limits are enabled (bit 15=1), and desired position within the lookahead buffer exceeds a position limit, Turbo PMAC will stop execution of the program at the point where the motor reaches the limit.

If bit 14 (value $4000, or 16,384) is set to one when desired position limits are enabled (bit 15=1) (e.g. I224=$C000), and desired position within the lookahead buffer exceeds a position limit, Turbo PMAC will continue execution of the program past the point where the motor reaches the limit, but will not let the desired motor position exceed the limit.

Bit 15: Desired Position Limit Enable Bit: If bit 15 is set to zero, Turbo PMAC does not check to see whether the desired position for this motor exceeds software overtravel limits.

If bit 15 (value $8000, or 32,768) is set to one (e.g. I324=$8001), Turbo PMAC will check desired position values for this motor against the software overtravel limits as set by Ixx13, Ixx14, and Ixx41.

If inside the special lookahead buffer, Turbo PMAC will either come to a controlled stop along the path at the point where the desired position reaches the limit, or continue the program with desired position saturated at the limit, depending on the setting of bit 14. If not inside the special lookahead buffer, Turbo PMAC will issue an Abort command when it sees that the desired position has exceeded a position limit.

Bit 16: Amplifier Enable Use Bit: With bit 19 equal to zero – the normal case – the AENAn output is used as an amplifier-enable line: off when the motor is "killed", on when it is enabled.

If bit 16 (value $10000, or 65,536) is set to one (e.g. I1924=$10001), this output is not used as an amplifier-enable line. On PMAC(1)-style channels, it could then be used as a direction output for magnitude and direction command format if Ixx96 is set to 1. Also, by assigning an M-variable to the AENAn output bit, general-purpose use of the this output is possible on either Turbo PMAC(1) or PMAC2 if this bit is set.

Bit 17: Overtravel Limit Use Bit: With bit 17 equal to zero – the normal case – the two hardware overtravel limit inputs must read 0 (drawing current) to permit commanded motion in the appropriate direction. If there are not actual (normally closed or normally conducting) limit switches, the inputs must be hardwired to ground.

If bit 17 (value $20000, or 131,072) is set to one (e.g. I1924=$20000), Motor xx does not use these inputs as overtravel limits. This can be done temporarily, as when using a limit as a homing flag. If the hardware overtravel limit function is not used at all, these inputs can be used as general-purpose inputs by assigning M-variables to them.

Bits 18 and 19: MACRO Node Use Bits: Bits 18 (value $40000, or 262,144) and 19 (value ($80000, or 524,288) of Ixx24 specify what flag information is connected directly to Turbo PMAC hardware channels, and what information comes through the MACRO ring into a MACRO auxiliary register. The following table shows the possible settings of these two bits and what they specify:

Bit 19	Bit 18	Capture Flags	Amp Flags	Limit Flags
0	0	Direct	Direct	(don't care)
0	1	Thru MACRO	Thru MACRO	(don't care)
1	0	Direct	Thru MACRO	(don't care)
1	1	Thru MACRO	Direct	(don't care)

If the amplifier flags are connected through the MACRO ring, bit 23 of Ixx24 must be set to 1 to designate a high-true amplifier fault, which is the MACRO standard. When using a MACRO auxiliary register for the flags, Ixx25, Ixx42, or Ixx43 should contain the address of a holding register in RAM, not the actual MACRO register. Refer to the descriptions of those variables for a list of the holding register addresses. Turbo PMAC firmware automatically copies between the holding registers and the MACRO registers as enabled by I70, I72, I74 and I76, for MACRO ICs 0, 1, 2, and 3, respectively. I71, I73, I75, and I77 must be set properly to determine whether the Type 0 or Type 1 MACRO protocol is being used on the particular node (all Delta Tau products use Type 1).

Bit 20: Amplifier Fault Use Bit: If bit 20 of Ixx24 is 0, the amplifier-fault input function through the FAULTn input is enabled. If bit 20 (value $100000, or 1,048,576) is 1 (e.g. I1924=$100000), this function is disabled. General-purpose use of this input is then possible by assigning an M-variable to the input.

Bits 21 & 22: Action-on-Fault Bits: Bits 21 (value $200000, or 2,097,152) and 22 (value $400000, or 4,194,304) of Ixx24 control what action is taken on an amplifier fault for the motor, or on exceeding the fatal following error limit (as set by Ixx11) for the motor:

Bit 22	Bit 21	Function
Bit 22=0	Bit 21=0:	Kill all PMAC motors
Bit 22=0	Bit 21=1:	Kill all motors in same coordinate system
Bit 22=1	Bit 21=0:	Kill only this motor
Bit 22=1	Bit 21=1:	(Reserved for future use)

Regardless of the setting of these bits, a program running in the coordinate system of the offending motor will be halted on an amplifier fault or the exceeding of a fatal following error.

Bit 23: Amplifier-Fault Polarity Bit: Bit 23 (value $800000, or 8,388,608) of Ixx24 controls the polarity of the amplifier-fault input. A zero in this bit specifies that a zero read in the fault bit means a fault; a one in this bit specifies that a one read in the fault bit means a fault. The actual state of the input circuitry for a fault depends on the actual interface circuitry used. If a Delta Tau-provided optically isolated fault interface is used, when the fault driver from the amplifier is drawing current through the isolator, either sinking or sourcing, the fault bit will read as zero; when it is not drawing current through the isolator, the fault bit will read as one.

In both the standard direct-PWM interface and the standard MACRO interface, bit 23 should be set to one, to specify that a one in the fault bit means a fault. (The actual polarity of the signal into the remote MACRO Station is programmable at the station).

Bit 23 is only used if bit 20 of Ixx24 is set to 0, telling Turbo PMAC to use the amplifier fault input.

Ixx25 Motor xx Flag Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC Addresses

Default:

Turbo PMAC(1) Ixx25 Defaults

Ixx25	Value	Register	Ixx25	Value	Register
I125	$078000	PMAC Flag Set 1	I1725	$079200	2nd ACC-24P/V Flag Set 1
I225	$078004	PMAC Flag Set 2	I1825	$079204	2nd ACC-24P/V Flag Set 2
I325	$078008	PMAC Flag Set 3	I1925	$079208	2nd ACC-24P/V Flag Set 3
I425	$07800C	PMAC Flag Set 4	I2025	$07920C	2nd ACC-24P/V Flag Set 4
I525	$078100	PMAC Flag Set 5	I2125	$079300	2nd ACC-24P/V Flag Set 5
I625	$078104	PMAC Flag Set 6	I2225	$079304	2nd ACC-24P/V Flag Set 6
I725	$078108	PMAC Flag Set 7	I2325	$079308	2nd ACC-24P/V Flag Set 7
I825	$07810C	PMAC Flag Set 8	I2425	$07930C	2nd ACC-24P/V Flag Set 8
I925	$078200	1st ACC-24P/V Flag Set 1	I2525	$07A200	3rd ACC-24P/V Flag Set 1
I1025	$078204	1st ACC-24P/V Flag Set 2	I2625	$07A204	3rd ACC-24P/V Flag Set 2
I1125	$078208	1st ACC-24P/V Flag Set 3	I2725	$07A208	3rd ACC-24P/V Flag Set 3
I1225	$07820C	1st ACC-24P/V Flag Set 4	I2825	$07A20C	3rd ACC-24P/V Flag Set 4
I1325	$078300	1st ACC-24P/V Flag Set 5	I2925	$07A300	3rd ACC-24P/V Flag Set 5
I1425	$078304	1st ACC-24P/V Flag Set 6	I3025	$07A304	3rd ACC-24P/V Flag Set 6
I1525	$078308	1st ACC-24P/V Flag Set 7	I3125	$07A308	3rd ACC-24P/V Flag Set 7
I1625	$07830C	1st ACC-24P/V Flag Set 8	I3225	$07A30C	3rd ACC-24P/V Flag Set 8

Turbo PMAC2 Ixx25 Defaults

Ixx25	Value	Register	Ixx25	Value	Register
I125	$078000	PMAC2 Flag Set 1	I1725	$079200	2nd ACC-24P/V2 Flag Set 1
I225	$078008	PMAC2 Flag Set 2	I1825	$079208	2nd ACC-24P/V2 Flag Set 2
I325	$078010	PMAC2 Flag Set 3	I1925	$079210	2nd ACC-24P/V2 Flag Set 3
I425	$078018	PMAC2 Flag Set 4	I2025	$079218	2nd ACC-24P/V2 Flag Set 4
I525	$078100	PMAC2 Flag Set 5	I2125	$079300	2nd ACC-24P/V2 Flag Set 5
I625	$078108	PMAC2 Flag Set 6	I2225	$079308	2nd ACC-24P/V2 Flag Set 6
I725	$078110	PMAC2 Flag Set 7	I2325	$079310	2nd ACC-24P/V2 Flag Set 7
I825	$078118	PMAC2 Flag Set 8	I2425	$079318	2nd ACC-24P/V2 Flag Set 8
I925	$078200	1st ACC-24P/V2 Flag Set 1	I2525	$07A200	3rd ACC-24P/V2 Flag Set 1
I1025	$078208	1st ACC-24P/V2 Flag Set 2	I2625	$07A208	3rd ACC-24P/V2 Flag Set 2
I1125	$078210	1st ACC-24P/V2 Flag Set 3	I2725	$07A210	3rd ACC-24P/V2 Flag Set 3
I1225	$078218	1st ACC-24P/V2 Flag Set 4	I2825	$07A218	3rd ACC-24P/V2 Flag Set 4
I1325	$078300	1st ACC-24P/V2 Flag Set 5	I2925	$07A300	3rd ACC-24P/V2 Flag Set 5
I1425	$078308	1st ACC-24P/V2 Flag Set 6	I3025	$07A308	3rd ACC-24P/V2 Flag Set 6
I1525	$078310	1st ACC-24P/V2 Flag Set 7	I3125	$07A310	3rd ACC-24P/V2 Flag Set 7
I1625	$078318	1st ACC-24P/V2 Flag Set 8	I3225	$07A318	3rd ACC-24P/V2 Flag Set 8

Turbo PMAC2 Ultralite Ixx25 Defaults

Ixx25	Value	Register	Ixx25	Value	Register
I125	$003440	MACRO Flag Register Set 0	I1725	$003460	MACRO Flag Register Set 32
I225	$003441	MACRO Flag Register Set 1	I1825	$003461	MACRO Flag Register Set 33
I325	$003444	MACRO Flag Register Set 4	I1925	$003464	MACRO Flag Register Set 36
I425	$003445	MACRO Flag Register Set 5	I2025	$003465	MACRO Flag Register Set 37
I525	$003448	MACRO Flag Register Set 8	I2125	$003468	MACRO Flag Register Set 40
I625	$003449	MACRO Flag Register Set 9	I2225	$003469	MACRO Flag Register Set 41
I725	$00344C	MACRO Flag Register Set 12	I2325	$00346C	MACRO Flag Register Set 44
I825	$00344D	MACRO Flag Register Set 13	I2425	$00346D	MACRO Flag Register Set 45
I925	$003450	MACRO Flag Register Set 16	I2525	$003470	MACRO Flag Register Set 48
I1025	$003451	MACRO Flag Register Set 17	I2625	$003471	MACRO Flag Register Set 49
I1125	$003454	MACRO Flag Register Set 20	I2725	$003474	MACRO Flag Register Set 52
I1225	$003455	MACRO Flag Register Set 21	I2825	$003475	MACRO Flag Register Set 53
I1325	$003458	MACRO Flag Register Set 24	I2925	$003478	MACRO Flag Register Set 56
I1425	$003459	MACRO Flag Register Set 25	I3025	$003479	MACRO Flag Register Set 57
I1525	$00345C	MACRO Flag Register Set 28	I3125	$00347C	MACRO Flag Register Set 60
I1625	$00345D	MACRO Flag Register Set 29	I3225	$00347D	MACRO Flag Register Set 61

UMAC Turbo Ixx25 Defaults

Ixx02	Value	Register	Ixx02	Value	Register
I102	$078200	1st ACC-24E2x (IC 2) Flag Set 1	I1702	$07A200	5th ACC-24E2x (IC 6) Flag Set 1
I202	$078208	1st ACC-24E2x (IC 2) Flag Set 2	I1802	$07A208	5th ACC-24E2x (IC 6) Flag Set 2
I302	$078210	1st ACC-24E2x (IC 2) Flag Set 3	I1902	$07A210	5th ACC-24E2x (IC 6) Flag Set 3
I402	$078218	1st ACC-24E2x (IC 2) Flag Set 4	I2002	$07A218	5th ACC-24E2x (IC 6) Flag Set 4
I502	$078300	2nd ACC-24E2x (IC 3) Flag Set 1	I2102	$07A300	6th ACC-24E2x (IC 7) Flag Set 1
I602	$078308	2nd ACC-24E2x (IC 3) Flag Set 2	I2202	$07A308	6th ACC-24E2x (IC 7) Flag Set 2
I702	$078310	2nd ACC-24E2x (IC 3) Flag Set 3	I2302	$07A310	6th ACC-24E2x (IC 7) Flag Set 3
I802	$078318	2nd ACC-24E2x (IC 3) Flag Set 4	I2402	$07A318	6th ACC-24E2x (IC 7) Flag Set 4
I902	$079200	3rd ACC-24E2x (IC 4) Flag Set 1	I2502	$07B200	7th ACC-24E2x (IC 8) Flag Set 1
I1002	$079208	3rd ACC-24E2x (IC 4) Flag Set 2	I2602	$07B208	7th ACC-24E2x (IC 8) Flag Set 2
I1102	$079210	3rd ACC-24E2x (IC 4) Flag Set 3	I2702	$07B210	7th ACC-24E2x (IC 8) Flag Set 3
I1202	$079218	3rd ACC-24E2x (IC 4) Flag Set 4	I2802	$07B218	7th ACC-24E2x (IC 8) Flag Set 4
I1302	$079300	4th ACC-24E2x (IC 5) Flag Set 1	I2902	$07B300	8th ACC-24E2x (IC 9) Flag Set 1
I1402	$079308	4th ACC-24E2x (IC 5) Flag Set 2	I3002	$07B308	8th ACC-24E2x (IC 9) Flag Set 2
I1502	$079310	4th ACC-24E2x (IC 5) Flag Set 3	I3102	$07B310	8th ACC-24E2x (IC 9) Flag Set 3
I1602	$079318	4th ACC-24E2x (IC 5) Flag Set 4	I3202	$07B318	8th ACC-24E2x (IC 9) Flag Set 4

Ixx25 tells Turbo PMAC what registers it will access for its position-capture flags, and possibly its overtravel-limit input flags and amplifier enable/fault flags, for Motor xx. If Ixx42 is set to 0, Ixx25 specifies the address of the amplifier flags; if Ixx42 is set to a non-zero value, Ixx42 specifies the address of the amplifier flags. If Ixx43 is set to 0, Ixx25 specifies the address of the overtravel limit flags; if Ixx43 if set to a non-zero value, Ixx43 specifies the address of the overtravel limit flags. Variable Ixx24 tells which of the flags from the specified register(s) are to be used, and how they are to be used.

The addresses for the standard flag registers are given in the default table, above. The following tables show settings by register if you wish to change from the default.

Ixx25 Addresses for PMAC(1)-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078004	$078008	$07800C	1st IC on board PMAC
1	$078100	$078104	$078108	$07810C	2nd IC on board PMAC
2	$078200	$078204	$078208	$07820C	1st IC on 1st ACC-24P/V
3	$078300	$078304	$078308	$07830C	2nd IC on 1st ACC-24P/V
4	$079200	$079204	$079208	$07920C	1st IC on 2nd ACC-24P/V
5	$079300	$079304	$079308	$07930C	2nd IC on 2nd ACC-24P/V
6	$07A200	$07A204	$07A208	$07A20C	1st IC on 3rd ACC-24P/V
7	$07A300	$07A304	$07A308	$07A30C	2nd IC on 3rd ACC-24P/V
8	$07B200	$07B204	$07B208	$07B20C	1st IC on 4th ACC-24P/V
9	$07B300	$07B304	$07B308	$07B30C	2nd IC on 4th ACC-24P/V
Bit 0 of Ixx24 must be set to 0 to use PMAC(1)-style Servo ICs.					

Ixx25 Addresses for PMAC2-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078008	$078010	$078018	1st IC on board PMAC2, 3U stack
1	$078100	$078108	$078010	$078018	2nd IC on board PMAC2, 3U stack
2	$078200	$078208	$078210	$078218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078300	$078308	$078310	$078318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079200	$079208	$079210	$079218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079300	$079308	$079310	$079318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A200	$07A208	$07A210	$07A218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A300	$07A308	$07A310	$07A318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B200	$07B208	$07B210	$07B218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B300	$07B308	$07B310	$07B318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2
Bit 0 of Ixx24 must be set to 1 to use PMAC2-style Servo ICs.					

Ixx25 Addresses for MACRO Flag Holding Registers

IC Node #	MACRO IC 1	MACRO IC 2	MACRO IC 3	MACRO IC 4	Notes
0	$003440	$003450	$003460	$003470	MACRO Flag Register Sets 0, 16, 32, 48
1	$003441	$003451	$003461	$003471	MACRO Flag Register Sets 1, 17, 33, 49
4	$003444	$003454	$003464	$003474	MACRO Flag Register Sets 4, 20, 36, 52
5	$003445	$003455	$003465	$003475	MACRO Flag Register Sets 5, 21, 37, 53
8	$003448	$003458	$003468	$003478	MACRO Flag Register Sets 8, 24, 40, 56
9	$003449	$003459	$003469	$003479	MACRO Flag Register Sets 9, 25, 41, 57
12	$00344C	$00345C	$00346C	$00347C	MACRO Flag Register Sets 12, 28, 44, 60
13	$00344D	$00345D	$00346D	$00347D	MACRO Flag Register Sets 13, 29, 45, 61
Bit 0 of Ixx24 must be set to 1 to use MACRO flag holding registers Bits 18 and 19 of Ixx24 specify what flag information comes directly into Turbo PMAC and what comes through the MACRO ring. The following table explains the possible settings:					

Bit 19	Bit 18	Capture Flags	Amp Flags	Limit Flags
0	0	Direct	Direct	(don't care)
0	1	Thru MACRO	Thru MACRO	(don't care)
1	0	Direct	Thru MACRO	(don't care)
1	1	Thru MACRO	Direct	(don't care)

Typically, the position-capture flags will be on the same hardware channel as the position feedback encoder for the motor. If you wish to use the hardware-captured position for a Turbo PMAC triggered-move function such as a homing search move, Ixx25 must specify flags of the same hardware channel as the position feedback encoder specified with Ixx03 through the encoder conversion table, whether digital quadrature feedback, or interpolated sinusoidal feedback.

In the case of sinusoidal-encoder feedback through an ACC-51x high-resolution interpolator, if hardware position-capture capability is desired, the position-capture flags will be specified as being on the ACC-51x using Ixx25, and the amplifier flags will be specified as being on the output channel using Ixx42; the overtravel-limit flags will probably be specified as being on the same channel as the outputs, using Ixx43.

For the position-capture function, variables I7mn2 and I7mn3 for Servo IC *m* Channel *n* of the channel selected (or node-specific variables MI912 and MI913 on a MACRO Station) specify which edges of which signal(s) for the channel will cause the position-capture trigger.

The overtravel-limit inputs specified by Ixx25 or Ixx43 must read as 0 in order for Motor xx to be able to command movement in the direction of the limit unless bit 17 of Ixx24 is set to 1 to disable their action. With Delta Tau interface circuitry with optical isolation on the flags, this means that the switches must be drawing current through the opto-isolators, whether sinking or sourcing.

Whether the address of the amplifier flags is specified with Ixx25 or Ixx42, the polarity of the amplifier-fault input is determined by bit 23 of Ixx24 and the polarity of the amplifier-enable output must be determined with the hardware interface.

Ixx26 Motor xx Home Offset

Range: -8,388,608 - 8,388,607

Units: 1/16 count

Default: 0

Ixx26 specifies the difference between the zero position of sensor(s) for the motor and the motor's own zero "home" position. For a motor that establishes its position reference with a homing search move, this is the difference between the home trigger position and the motor zero position. For a motor that establishes its position reference with an absolute position read (Ixx10 > 0), this is the difference between the absolute sensor's zero position and the motor zero position.

In a homing search move, Ixx26 specifies the distance between the *actual* position at which the home trigger is found, and the *commanded* end of the post-trigger move, where the motor will come to a stop. The commanded end position of the post-trigger move is considered motor position zero. (It is possible to use other offsets to create a different *axis* position zero for programming purposes.)

A difference between the trigger position and the motor zero position is particularly useful when using an overtravel limit as a home flag (offsetting out of the limit before re-enabling the limit input as a limit). If Ixx26 is large enough (greater than 1/2 times home speed times acceleration time), it permits a homing search move without any reversal of direction.

In an absolute position read done on board reset, the **$*** command, or the **$$*** command, Ixx26 specifies the difference between the position read from the sensor as specified by Ixx10 and Ixx95, and the actual motor position set as a result of this read. Ixx26 is *subtracted from* the sensor position to calculate motor position. This offset is particularly useful when the absolute sensor's zero position is outside the range of travel for the motor, as with an MLDT sensor.

Note:

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts between the trigger position and the home zero position.

Example:

If you wish your motor zero position to be 500 counts in the negative direction from the home trigger position, you would set Ixx26 to -500 * 16 = -8000.

Ixx27 Motor xx Position Rollover Range

Range: -2^{35} - $+2^{35}$

Units: counts

Default: 0

Ixx27 permits either of two position rollover modes on a Turbo PMAC rotary axis assigned to Motor xx by telling Turbo PMAC how many encoder counts are in one revolution of the rotary axis. This lets Turbo PMAC handle rollover properly. If Ixx27 is set to the default value of 0, no rollover mode is active, and the axis is treated as a linear axis.

If Ixx27 is greater than zero, and Motor xx is assigned to a rotary axis (A, B, or C), the standard rollover mode is active. With standard rollover active, for a programmed *axis* move in absolute (**ABS**) mode, the motor will take the shortest path around the circular range defined by Ixx27 to get to the destination point. No absolute-mode move will be longer than half of a revolution (Ixx27/2) with standard rollover.

If Ixx27 is set to a negative number, an alternate rollover mode for the rotary axis assigned to the motor is activated that uses the sign of the commanded destination in absolute mode to specify the direction of motion to that destination. In this mode, all moves are less than one revolution (with the size of the revolution specified by the magnitude of Ixx27), but can be greater than one-half revolution. This mode also does not affect the action of incremental-mode moves.

The sign of the commanded absolute destination in this mode is also part of the destination value. So a command of **A-90** in this mode is a command to go to -90 degrees (= +270 degrees) in the negative direction. For commands to move in the positive direction, the + sign is not required, but it is permitted (e.g. to command a move to 90 degrees in the positive direction, either **A90** or **A+90** can be used).

PMAC cannot store the difference between a +0 and a –0 destination command, so a command with a tiny non-zero magnitude for the end position must be used (e.g. **A+0.0000001** and **A-0.0000001**). This increment can be small enough not to have any effect on the final destination.

If the distance of the move commanded in alternate rollover mode is less than the size of the in-position band defined for the motor with Ixx28, no move will be executed. This means that the minimum distance for a move in this mode is Ixx28, and the maximum distance is 360 degrees minus Ixx28.

If using commands from a similar mode in which only the magnitude, and not the sign, of the value specifies the destination position, then the destination values for negative-direction moves must be modified so that the magnitude is 360 degrees minus the magnitude in the other mode. For example, if the command were C-120, specifying a move to (+)120 degrees in the negative direction, the command would have to be modified for PMAC to C-240, which specifies a move to -240 degrees (= +120 degrees) in the negative direction. Commands for positive-direction moves do not have to be modified.

Axis moves in incremental (**INC**) mode are not affected by either rollover mode. Rollover should not be attempted for axes other than A, B, or C. Jog moves are not affected by rollover. Reported motor position is not affected by rollover. (To obtain motor position information rolled over to within one motor revolution, use the modulo (remainder) operator, either in PMAC or in the host computer: e.g. **P4=(M462/(I408*32))%I427).**

Note:

It is possible to set this parameter outside the range -2^{35} to $+2^{35}$ ($\pm$64 billion) if a couple of special things are done. First, the Ixx08 scale factor for the motor must be reduced to give the motor the range to use this position (motor range is $\pm 2^{42}$/Ixx08). Second, the variable value must be calculated inside Turbo PMAC, because the command parser cannot accept constants outside the range $\pm 2^{35}$ (e.g. to set I127 to 100 billion, use **I127=1000000000*100**).

Example:
Motor #4 drives a rotary table with 36,000 counts per revolution. It is defined to the A-axis with **#4->100A** (A is in units of degrees). I427 is set to 36000. With motor #4 at zero counts (A-axis at zero degrees), an **A270** move in a program is executed in Absolute mode. Instead of moving the motor from 0 to 27,000 counts, which it would have done with I427=0, PMAC moves the motor from 0 to -9,000 counts, or -90 degrees, which is equivalent to +270 degrees on the rotary table.

Ixx28 Motor xx In-Position Band

Range: 0 - 8,388,607

Units: 1/16 count

Default: 160 (10 counts)

Ixx28 specifies the magnitude of the maximum following error at which Motor xx will be considered "in position" when not performing a move.

Several things happen when the motor is "in-position". First, a status bit in the motor status word (bit 0 of Y:$0000C0 for Motor 1) is set. Second, if all other motors in the same coordinate system are also "in-position", a status bit in the coordinate system status word (bit 17 of Y:$00203F for C.S. 1) is set.

Third, for the hardware-selected (FPD0/-FPD3/) coordinate system -- if I2=0 (Turbo PMAC(1) only) -- or for the software addressed (&n) coordinate system -- if I2=1 -- outputs to the control panel port (Turbo PMAC(1) only) and to the interrupt controller are set.

Technically, five conditions must be met for a motor to be considered "in-position":

1. The motor must be in closed-loop control;
2. The desired velocity must be zero;
3. The magnitude of the following error must be less than this parameter;
4. The move timer must not be active;
5. The above four conditions must all be true for (Ixx88+1) consecutive scans.

The over timer is active (the motor "running a program/definite-time move" status bit is 1) during any programmed or non-programmed move, including **DWELL**s and **DELAY**s in a program -- if you wish this bit to come true during a program, you must do an indefinite wait between some moves by keeping the program trapped in a **WHILE** loop that has no moves or **DWELL**s.

If you just want a status bit indicating whether the magnitude of the following error is above or below a threshold (condition 3 only), you can use the "warning following error" status bit with Ixx12 as the threshold.

If global variable I13 is set to 1, Turbo PMAC also performs an in-position check every servo cycle as part of the foreground tasks. In this check, it only evaluates the first 4 conditions listed above. This task controls a separate motor status bit: "foreground in-position" (bit 13 of Y:$0000C0 for Motor 1). This function can be used when the background in-position check is not fast enough.

Note:

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the in-position band.

Example:

The following motion program segment shows how the in-position function could be used in a program to set an output after coming in-position at a programmed point. M140 represents Motor 1's in-position status bit (see suggested M-variable definitions).

```
X10                          ; Commanded move
DWELL0                       ; Stop lookahead in motion programs
WHILE (M140=0) WAIT          ; Loop while not in position
M1=1                         ; Set output
```

Ixx29 Motor xx Output/First Phase Offset

Range: -32,768 - 32,767

Units: 16-bit DAC/ADC bit equivalent

Default: 0

Ixx29 serves as an output or feedback offset for Motor xx; its exact use depends on the mode of operation as described below. In any of the modes, it effectively serves as the digital equivalent of an offset pot.

Mode 1*:* When Turbo PMAC is not commutating Motor xx (Ixx01 Bit 0 = 0), Ixx29 serves as the offset for the single command output value, usually a DAC command. Ixx29 is added to the output command value before it is written to the command output register.

Mode 2: When Turbo PMAC (PMAC(1)-style Servo ICs only) is not commutating Motor xx (Ixx01 Bit 0 = 0) but is in sign-and-magnitude output mode (Ixx96 = 1), Ixx29 is the offset of the command output value before the absolute value is taken (Ixx79 is the offset after the absolute value is taken). Ixx29 is typically left at zero in this mode, because it cannot compensate for real circuitry offsets.

Mode 3: When Turbo PMAC is commutating Motor xx (Ixx01 Bit 0 = 1) but not closing the current loop (Ixx82 = 0), Ixx29 serves as the offset for the first of two phase command output values (Phase A), for the address specified by Ixx02; Ixx79 serves the same purpose for the second phase (Phase B). Ixx29 is added to the output command value before it is written to the command output register.

When commutating from a PMAC(1)-style Servo IC, Phase A is output on the *higher*-numbered of the two DACs (e.g. DAC2), Phase B on the *lower*-numbered (e.g. DAC1). When commutating from a PMAC2-style Servo IC, Phase A is output on the A-channel DAC (e.g. DAC1A), Phase B on the B-channel DAC (e.g. DAC1B).

As an output command offset, Ixx29 is always in units of a 16-bit register, even if the actual output device is of a different resolution. For example, if a value of 60 had to be written into an 18-bit DAC to create a true zero command, this would be equivalent to a value of 60/4=15 in a 16-bit DAC, so Ixx29 would be set to 15 to cancel the offset.

Mode 4: When Turbo PMAC is commutating (Ixx01 Bit 0 = 1) and closing the current loop for Motor xx (Ixx82 > 0), Ixx29 serves as an offset that is added to the phase current reading from the ADC for the first phase (Phase A), at the address specified by Ixx82 minus 1. Ixx79 performs the same function for the second phase. The sum of the ADC reading and Ixx29 is used in the digital current loop algorithms.

As an input feedback offset, Ixx29 is always in units of a 16-bit ADC, even if the actual ADC is of a different resolution. For example, if a 12-bit ADC reported a value of -5 when no current was flowing in the phase, this would be equivalent to a value of -5*16=-80 in a 16-bit ADC, so Ixx29 would be set to 80 to compensate for this offset.

Motor xx PID Servo Setup I-Variables

Note:

PID Servo Gains Ixx30 – Ixx40 are only used if supplementary motor I-variable Iyy00/Iyy50 is set to its default value of 0. If Iyy00/Iyy50 is set to 1, the Extended Servo Algorithm gains in Iyy10-39/Iyy60-89 are used instead.

Ixx30 Motor xx PID Proportional Gain

Range: -8,388,608 - 8,388,607

Units: (Ixx08/2^{19}) 16-bit output bits / count

Default: 2000

WARNING:

Changing the sign of Ixx30 on a motor that has been closing a stable servo loop will cause an unstable servo loop, leading to a probable runaway condition.

Ixx30 provides a control output proportional to the position error (commanded position minus actual position) of Motor xx. It acts effectively as an electronic spring. The higher Ixx30 is, the stiffer the "spring" is. Too low a value will result in sluggish performance. Too high a value can cause a "buzz" from constant over-reaction to errors.

If Ixx30 is set to a negative value, this has the effect of inverting the command output polarity for motors not commutated by PMAC, when compared to a positive value of the same magnitude. This can eliminate the need to exchange wires to get the desired polarity. On a motor that is commutated by PMAC, changing the sign of Ixx30 has the effect of changing the commutation phase angle by 180^{o}. Negative values of Ixx30 currently cannot be used with the auto tuning programs in the PMAC Executive program.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Note:

The default value of 2000 for this parameter is exceedingly "weak" for most systems (all but the highest resolution velocity-loop systems), causing sluggish motion and/or following error failure. Most users will immediately want to raise this parameter significantly even before starting serious tuning.

If the servo update time is changed, Ixx30 will have the same effect for the same numerical value. However, smaller update times (faster update rates) should permit higher values of Ixx30 (stiffer systems) without instability problems.

Ixx30 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx31 Motor xx PID Derivative Gain

Range: -8,388,608 - 8,388,607

Units: (Ixx30*Ixx09)/2^{26} 16-bit output bits / (counts/servo update)

Default: 1280

Ixx31 subtracts an amount from the control output proportional to the measured velocity of Motor xx. It acts effectively as an electronic damper. The higher Ixx31 is, the heavier the damping effect is.

If the motor is driving a properly tuned velocity-loop amplifier, the amplifier will provide sufficient damping, and Ixx31 should be set to zero. If the motor is driving a current-loop (torque) amplifier, or if PMAC is commutating the motor, the amplifier will provide no damping, and Ixx31 must be greater than zero to provide damping for stability.

On a typical system with a current-loop amplifier and PMAC's default servo update time (~440 µsec), an Ixx31 value of 2000 to 3000 will provide a critically damped step response.

If the servo update time is changed, Ixx31 must be changed proportionately in the opposite direction to keep the same damping effect. For instance, if the servo update time is cut in half, from 440 µsec to 220 µsec, Ixx31 must be doubled to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx31 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx32 Motor xx PID Velocity Feedforward Gain

Range: -8,388,608 - 8,388,607

Units: (Ixx30*Ixx08)/2^{26} 16-bit output bits / (counts/servo update)

Default: 1280

Ixx32 adds an amount to the control output proportional to the desired velocity of Motor xx. It is intended to reduce tracking error due to the damping introduced by Ixx31, analog tachometer feedback, or physical damping effects.

If the motor is driving a current-loop (torque) amplifier, Ixx32 will usually be equal to (or slightly greater than) Ixx31 to minimize tracking error. If the motor is driving a velocity-loop amplifier, Ixx32 will typically be substantially greater than Ixx31 to minimize tracking error.

If the servo update time is changed, Ixx32 must be changed proportionately in the opposite direction to keep the same effect. For instance, if the servo update time is cut in half, from 440 µsec to 220 µsec, Ixx32 must be doubled to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx32 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx33 Motor xx PID Integral Gain

Range: 0 - 8,388,607

Units: (Ixx30*Ixx08)/2^{42} 16-bit output bits / (counts*servo update)

Default: 1280

Ixx33 adds an amount to the control output proportional to the time integral of the position error for Motor xx. The magnitude of this integrated error is limited by Ixx63. With Ixx63 at a value of zero, the contribution of the integrator to the output is zero, regardless of the value of Ixx33.

No further errors are added to the integrator if the output saturates (if output equals Ixx69), and, if Ixx34=1, when a move is being commanded (when desired velocity is not zero). In both of these cases, the contribution of the integrator to the output remains constant.

If the servo update time is changed, Ixx33 must be changed proportionately in the same direction to keep the same effect. For instance, if the servo update time is cut in half, from 440 µsec to 220 µsec, Ixx33 must be cut in half to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx33 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx34 Motor xx PID Integration Mode

Range: 0 - 1

Units: none

Default: 1

Ixx34 controls when the position-error integrator is turned on. If it is 1, position error integration is performed only when Motor xx is not commanding a move (when desired velocity is zero). If it is 0, position error integration is performed all the time.

If Ixx34 is 1, it is the *input* to the integrator that is turned off during a commanded move, which means the *output* control effort of the integrator is kept constant during this period (but is generally not zero). This same action takes place whenever the total control output saturates at the Ixx69 value.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. When performing the feedforward tuning part of that utility, it is important to set Ixx34 to 1 so the dynamic behavior of the system may be observed without integrator action. Ixx34 may be changed on the fly at any time to create types of adaptive control.

Ixx34 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx35 Motor xx PID Acceleration Feedforward Gain

Range: -8,388,608 - 8,388,607

Units: (Ixx30*Ixx08)/2^{26} 16-bit output bits / (counts/servo update^2)

Default: 0

Ixx35 adds an amount to the control output proportional to the desired acceleration for Motor xx. It is intended to reduce tracking error due to inertial lag.

If the servo update time is changed, Ixx35 must be changed by the inverse square to keep the same effect. For instance, if the servo update time is cut in half, from 440 µsec to 220 µsec, Ixx35 must be quadrupled to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx35 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx36 Motor xx PID Notch Filter Coefficient N1

Range: -2.0 - 2.0

Units: none (unit-less z-transform coefficient)

Default: 0.0

Ixx36, along with Ixx37-Ixx39, is part of the 2nd-order "notch filter" for Motor xx, whose main purpose is to damp out a resonant mode in the motor/load dynamics. This filter can also be used as a low-pass filter and a velocity-loop integrator. This parameter can be set according to instructions in the Servo Loop Features section of the manual.

The notch filter parameters Ixx36-Ixx39 are 24-bit variables, with 1 sign bit, 1 integer bit, and 22 fractional bits, providing a range of -2.0 to +2.0.

The equation for the notch filter is:

$$F(z) = \frac{1 + N1z^{-1} + N2z^{-2}}{1 + D1z^{-1} + D2z^{-2}}$$

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx36 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx37 Motor xx PID Notch Filter Coefficient N2

Range: -2.0 - 2.0

Units: none (unit-less z-transform coefficient)

Default: 0.0

Ixx37 is part of the "notch filter" for Motor xx. See Ixx36 and the Servo Loop Features section of the manual for details.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx37 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx38 Motor xx PID Notch Filter Coefficient D1

Range: -2.0 - 2.0

Units: none (unit-less z-transform coefficient)

Default: 0.0

Ixx38 is part of the "notch filter" for Motor xx. See Ixx36 and the Servo Loop Features section of the manual for details.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx38 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx39 Motor xx PID Notch Filter Coefficient D2

Range: -2.0 - 2.0

Units: none (unit-less z-transform coefficient)

Default: 0.0

Ixx39 is part of the "notch filter" for Motor xx. See Ixx36 and the Servo Loop Features section of the manual for details.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Ixx39 is not used if Iyy00/50 for the motor has been set to 1 to enable the Extended Servo Algorithm (ESA) for the motor.

Ixx40 Motor xx Net Desired Position Filter Gain

Range: 0.0 – 0.999999

Units: none

Default: 0.0

Ixx40 permits the introduction of a first-order low-pass filter on the net desired position for Motor xx. This can be useful to smooth motion that comes from a "rough" source, such as master following from a noisy sensor, or quantization error in very closely spaced programmed points that are commonly found in lookahead applications.

If Ixx40 is set to its default value of 0.0, this filter function is disabled. If Ixx40 is set to any value greater than 0.0, the filter is enabled.

Ixx40 can be expressed in terms of the filter time constant by the following equation:

where T_f is the filter time constant, and T_s is the servo update time.

$$Ixx40 = \frac{T_f}{T_s + T_f}$$

The filter time constant can be expressed in terms of Ixx40 by the following equation:

$$T_f = \frac{Ixx40 * T_s}{1 - Ixx40}$$

Filter time constants can range from a fraction of a servo cycle (when Ixx40 ~ 0) to infinite (when Ixx40 ~ 1). As with any low-pass filter, there is a fundamental trade-off between smoothness and delay. Generally, when the filter is used, filter time constants of a few milliseconds are set. In an application where multiple motors are executing a path, the same time constant should be used for all of the motors.

Example:

To set a filter time constant of 2 msec on a system with the default servo update time of 442 μsec, Ixx40 can be computed as:

$$Ixx40 = \frac{2}{0.442 + 2} = 0.819$$

Ixx41 Motor xx Desired Position Limit Band

Range: 0 – 8,388,607

Units: counts

Default: 0

Ixx41 specifies the difference between the software position limits using desired position at lookahead time and those using actual position at move execution time.

Turbo PMAC will check the motor *desired* position at lookahead time if bit 15 of Ixx24 is set to 1. If the lookahead desired position is greater than (Ixx13-Ixx41), or less than (Ixx14+Ixx41), Turbo PMAC will limit the desired position at this value, and either stop the program on the path at this point or continue the program while saturating the motor position at this value, depending on the setting of bit 14 of Ixx24.

Turbo PMAC also checks the motor *actual* position at move execution time. If the actual position is greater than Ixx13, or less than Ixx14, Turbo PMAC issues an Abort command, bringing the motors to a stop, but not along the path. This checking is done even if Turbo PMAC is already stopping on the path because lookahead desired position was exceeded.

The purpose of Ixx41 is to permit the lookahead desired position limit to operate, stopping or limiting the program in a recoverable fashion, without also tripping the actual position limit and creating an unrecoverable stop. If the two limits are the same, a slight overshoot during the deceleration for desired position limit would trip the actual position limit. Ixx41 should be set slightly greater than the magnitude of the largest following error expected when decelerating at the Ixx17 maximum deceleration rate.

Ixx42 Motor xx Amplifier Flag Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC Addresses

Default: $0

Ixx42, if set to a non-zero value, specifies the address of the amplifier-enable output flag and amplifier-fault input flag, independently of position-capture flags and overtravel-limit flags, for Motor xx. If Ixx42 is set to 0, Ixx25 specifies the address of the amplifier flags as well as the position-capture flags, and possibly the overtravel-limit flags, for Motor xx. This maintains backward compatibility with older firmware revisions in which Ixx42 was not implemented.

Whether the address of the amplifier flags is specified with Ixx25 or Ixx42, the polarity of the amplifier-fault input is determined by bit 23 of Ixx24 and the polarity of the amplifier-enable output must be determined with the hardware interface.

If amplifier flags are specified separately using Ixx42, they must use the same type of ICs as does Ixx25, those specified by bit 0 of Ixx24.

Bits 18 and 19 of Ixx24 specify whether the amplifier flags and the capture flags are connected directly to Turbo PMAC circuitry, or interface to it through the MACRO ring as shown in the following table:

Bit 19	Bit 18	Capture Flags	Amp Flags
0	0	Direct	Direct
0	1	Thru MACRO	Thru MACRO
1	0	Direct	Thru MACRO
1	1	Thru MACRO	Direct

The following tables show the standard addresses that can be used for Ixx42.

Ixx42 Addresses for PMAC(1)-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078004	$078008	$07800C	1st IC on board PMAC
1	$078100	$078104	$078108	$07810C	2nd IC on board PMAC
2	$078200	$078204	$078208	$07820C	1st IC on 1st ACC-24P/V
3	$078300	$078304	$078308	$07830C	2nd IC on 1st ACC-24P/V
4	$079200	$079204	$079208	$07920C	1st IC on 2nd ACC-24P/V
5	$079300	$079304	$079308	$07930C	2nd IC on 2nd ACC-24P/V
6	$07A200	$07A204	$07A208	$07A20C	1st IC on 3rd ACC-24P/V
7	$07A300	$07A304	$07A308	$07A30C	2nd IC on 3rd ACC-24P/V
8	$07B200	$07B204	$07B208	$07B20C	1st IC on 4th ACC-24P/V
9	$07B300	$07B304	$07B308	$07B30C	2nd IC on 4th ACC-24P/V

Bit 0 of Ixx24 must be set to 0 to use PMAC(1)-style Servo ICs.

Ixx42 Addresses for PMAC2-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078008	$078010	$078018	1st IC on board PMAC2, 3U stack
1	$078100	$078108	$078010	$078018	2nd IC on board PMAC2, 3U stack
2	$078200	$078208	$078210	$078218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078300	$078308	$078310	$078318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079200	$079208	$079210	$079218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079300	$079308	$079310	$079318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A200	$07A208	$07A210	$07A218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A300	$07A308	$07A310	$07A318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B200	$07B208	$07B210	$07B218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B300	$07B308	$07B310	$07B318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

Bit 0 of Ixx24 must be set to 1 to use PMAC2-style Servo ICs.

Ixx42 Addresses for MACRO Flag Holding Registers

IC Node #	MACRO IC 1	MACRO IC 2	MACRO IC 3	MACRO IC 4	Notes
0	$003440	$003450	$003460	$003470	MACRO Flag Register Sets 0, 16, 32, 48
1	$003441	$003451	$003461	$003471	MACRO Flag Register Sets 1, 17, 33, 49
4	$003444	$003454	$003464	$003474	MACRO Flag Register Sets 4, 20, 36, 52
5	$003445	$003455	$003465	$003475	MACRO Flag Register Sets 5, 21, 37, 53
8	$003448	$003458	$003468	$003478	MACRO Flag Register Sets 8, 24, 40, 56
9	$003449	$003459	$003469	$003479	MACRO Flag Register Sets 9, 25, 41, 57
12	$00344C	$00345C	$00346C	$00347C	MACRO Flag Register Sets 12, 28, 44, 60
13	$00344D	$00345D	$00346D	$00347D	MACRO Flag Register Sets 13, 29, 45, 61

Bit 0 of Ixx24 must be set to 1 to use MACRO flag holding registers.

Ixx43 Motor xx Overtravel-Limit Flag Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC Addresses

Default: $0

Ixx43, if set to a non-zero value, specifies the address of the overtravel-limit input flags, independently of position-capture flags and amplifier flags, for Motor xx. If Ixx43 is set to 0, Ixx25 specifies the address of the overtravel-limit flags as well as the position-capture flags, and possibly the amplifier flags, for Motor xx. This maintains backward compatibility with older firmware revisions in which Ixx43 was not implemented.

If overtravel limit flags are specified separately using Ixx43, they must use the same type of ICs as Ixx25, as specified by bit 0 of Ixx24.

The following tables show the standard addresses that can be used for Ixx42.

Ixx43 Addresses for PMAC(1)-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078004	$078008	$07800C	1st IC on board PMAC
1	$078100	$078104	$078108	$07810C	2nd IC on board PMAC
2	$078200	$078204	$078208	$07820C	1st IC on 1st ACC-24P/V
3	$078300	$078304	$078308	$07830C	2nd IC on 1st ACC-24P/V
4	$079200	$079204	$079208	$07920C	1st IC on 2nd ACC-24P/V
5	$079300	$079304	$079308	$07930C	2nd IC on 2nd ACC-24P/V
6	$07A200	$07A204	$07A208	$07A20C	1st IC on 3rd ACC-24P/V
7	$07A300	$07A304	$07A308	$07A30C	2nd IC on 3rd ACC-24P/V
8	$07B200	$07B204	$07B208	$07B20C	1st IC on 4th ACC-24P/V
9	$07B300	$07B304	$07B308	$07B30C	2nd IC on 4th ACC-24P/V

Bit 0 of Ixx24 must be set to 0 to use PMAC(1)-style Servo ICs.

Ixx43 Addresses for PMAC2-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078008	$078010	$078018	1st IC on board PMAC2, 3U stack
1	$078100	$078108	$078010	$078018	2nd IC on board PMAC2, 3U stack
2	$078200	$078208	$078210	$078218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078300	$078308	$078310	$078318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079200	$079208	$079210	$079218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079300	$079308	$079310	$079318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A200	$07A208	$07A210	$07A218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A300	$07A308	$07A310	$07A318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B200	$07B208	$07B210	$07B218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B300	$07B308	$07B310	$07B318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

Bit 0 of Ixx24 must be set to 1 to use PMAC2style Servo ICs.

Ixx43 Addresses for MACRO Flag Holding Registers

IC Node #	MACRO IC 1	MACRO IC 2	MACRO IC 3	MACRO IC 4	Notes
0	$003440	$003450	$003460	$003470	MACRO Flag Register Sets 0, 16, 32, 48
1	$003441	$003451	$003461	$003471	MACRO Flag Register Sets 1, 17, 33, 49
4	$003444	$003454	$003464	$003474	MACRO Flag Register Sets 4, 20, 36, 52
5	$003445	$003455	$003465	$003475	MACRO Flag Register Sets 5, 21, 37, 53
8	$003448	$003458	$003468	$003478	MACRO Flag Register Sets 8, 24, 40, 56
9	$003449	$003459	$003469	$003479	MACRO Flag Register Sets 9, 25, 41, 57
12	$00344C	$00345C	$00346C	$00347C	MACRO Flag Register Sets 12, 28, 44, 60
13	$00344D	$00345D	$00346D	$00347D	MACRO Flag Register Sets 13, 29, 45, 61

Bit 0 of Ixx24 must be set to 1 to use MACRO flag holding registers.

Motor Servo and Commutation Modifiers

Ixx55 Motor xx Commutation Table Address Offset

Range: $000000 - $FFFFFF

Units: Turbo PMAC Address Offsets from $003800

Default: $0

Ixx55 permits the user to create and use a custom commutation table for Motor xx in Turbo PMAC, instead of using the default commutation sine/cosine table. Ixx55 contains the offset from the start of the default table at address $003800 to the start of the user's custom table.

Custom tables are usually located in the UBUFFER at the end of flash-backed memory ($0107FF for standard data memory configuration, $03FFFF for the extended data memory configuration). Alternately, they can be located in the optional battery-backed memory ($050000 - $053FFF for the basic option, $050000 - $05FFFF for the extended option), but access is significantly slower to the battery-backed memory.

Custom tables must occupy 2048 consecutive double words of memory, covering the 360 degrees of the commutation cycle. The first register (lowest-numbered address) is the entry for 0 degrees. The address of this register must be divisible by $800, which means that the last three hex digits of this address must be $000 or $800, and the last three hex digits of Ix55 must be $800 or $000. The signed 24-bit X registers contain cosine-type values multiplied by 2^{23}; the signed 24-bit Y-registers contain sine-type values multiplied by 2^{23}.

Examples:

The custom commutation table for Motor 1 is located in the UBUFFER from $00F800 to $00FFFF. I155 should be set to $00F800 - $003800 = $00C000.

The custom commutation table for Motor 12 is located in battery-backed RAM from $052000 to $0527FF. I1255 should be set to $052000 - $003800 = $048800.

Ixx56 Motor xx Commutation Delay Compensation

Range: 0.0 – 1.0

Units: (Ixx09*32/2048) commutation cycles/(counts/servo update)

Default: 0

Ixx56 permits the Turbo PMAC to compensate lags in the electrical circuits of the motor phases, and/or for calculation delays in the commutation of Motor xx, therefore improving high-velocity performance. The compensation is simply Ixx56 multiplied by the motor velocity.

Ixx56 is only used if Turbo PMAC is commutating Motor xx (Ixx01=1). It should be only be used for the commutation of synchronous motors (Ixx78=0) such as permanent magnet brushless motors. Ixx56 should be set to 0 for asynchronous motors (Ixx78>0) such as AC induction motors.

If Turbo PMAC is commutating Motor xx, but not closing the current loop (Ixx82=0), Ixx56 can improve performance typically starting at a few thousand RPM, because it compensates for inductive lags in the motor windings. If Turbo PMAC is also closing the current loop for Motor xx (Ixx82>0; Turbo PMAC2 only), the DC field-frame current loop closure compensates for inductive lags, and only small calculation delays need to be compensated; these are usually not significant until well over 10,000 rpm.

This parameter is best set experimentally by running the motor at high speeds, and finding the setting that minimizes the current draw of the motor.

Ixx57 Motor xx Continuous Current Limit

Range: -32,768 – 32,767

Units: 16-bit DAC/ADC bit equivalent

Default: 0

Ixx57 sets the magnitude of the maximum continuous current limit for Turbo PMAC's integrated current limiting function, when that function is active (Ixx58 must be greater than 0 for the integrated current limit to be active). If Turbo PMAC is closing a digital current loop for the motor, it uses actual current measurements for this function; otherwise, it uses commanded current values. If the magnitude of the actual or commanded current level from Turbo PMAC is above the magnitude of Ixx57 for a significant period of time, as set by Ixx58, Turbo PMAC will trip this motor on an integrated-current amplifier fault condition.

The integrated current limit can either be an I^2T ("I-squared-T") limit, or an |I|T ("I-T") limit. If Ixx57 is set to a positive value, Turbo PMAC performs I^2T limiting, squaring the value of current before integrating and comparing to Ixx58. If Ixx57 is set to a negative value, Turbo PMAC performs |I|T limiting, just taking the absolute value of the current before integrating and comparing to Ixx58.

I^2T limiting is best used if the system device with the shortest thermal time constant is resistive (and so has I^2R heating), as in motor windings and MOSFET drivers. |I|T limiting is best used if the system device with the shortest thermal time constant has a fixed voltage drop (and so has IV heating), as in IGBT drivers.

Ixx57 is in units of a 16-bit DAC or ADC (maximum possible value of 32,767), even if the actual output or input device has a different resolution. Typically, Ixx57 will be set to between 1/3 and 1/2 of the Ixx69 (instantaneous) output limit. Consult your amplifier and motor documentation for their specifications on instantaneous and continuous current limits.

Technically, Ixx57 is the continuous limit of the vector sum of the quadrature and direct currents. The quadrature (torque-producing) current is the output of the position/velocity-loop servo. The direct (magnetization) current is set by Ixx77.

In sine-wave output mode (Ixx01 bit 0 = 1, Ixx82 = 0), amplifier gains are typically given in amperes of phase current per volt of PMAC output, but motor and amplifier limits are typically given in RMS amperage values. In this case, it is important to realize that peak phase current values are √2 (1.414) times greater than the RMS values.

In direct-PWM mode (Ixx01 bit 0 = 1, Ixx82 > 0) of 3-phase motors (Ixx72 = 683 or 1365), the corresponding top values of the sinusoidal phase-current ADC readings will be $1/\cos(30^o)$, or 1.15, times greater than the vector sum of quadrature and direct current. Therefore, once you have established the top values you want to see in the A/D converters your phase currents on a continuous basis, this value should be multiplied by $\cos(30^o)$, or 0.866, to get your value for Ixx57. Remember that if current limits are given as RMS values, you should multiply these by √2 (1.414) to get peak phase current values.

Examples:

1. Turbo PMAC Motor 1 is driving a torque-mode DC brush-motor amplifier that has a gain of 3 amps/volt with a single analog output voltage. The amplifier has a continuous current rating of 10 amps; the motor has a continuous current rating of 12 amps.
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 30 amps.
   - The amplifier has the lower continuous current rating, so we use its limit of 10 amps.
   - I157 is set to 32,768 * 10 / 30 = 10,589.

2. Motor 3 is driving a self-commutating brushless-motor amplifier in current (torque) mode with a single analog output. The amplifier has a gain of 5 amps(RMS)/volt and an continuous current limit of 20 amps (RMS). The motor has a continuous current limit of 25 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 50 amps (RMS).
   - The amplifier has the lower continuous current rating, so we use its limit of 20 amps (RMS).
   - I357 is set to 32,768 * 20/50 = 13,107.
3. Turbo PMAC Motor 4 is driving a sine-wave mode amplifier that has a gain for each phase input of 5 amps/volt. The amplifier has a continuous rating of 20 amps (RMS); the motor has a continuous rating of 22 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 50 amps peak in a phase.
   - The amplifier has the lower continuous current rating, so we use its limit of 20 amps (RMS).
   - 20 amps (RMS) corresponds to peak phase currents of 20*1.414 = 28.28 amps.
   - I457 is set to 32,768 * 28.28 / 50 = 18,534.
4. Turbo PMAC Motor 6 is driving a direct-PWM power block amplifier for an AC motor. The A/D converters in the amplifier are scaled so that a maximum reading corresponds to 50 amps of current in the phase. The amplifier has a continuous current rating of 20 amps (RMS), and the motor has a continuous rating of 15 amps (RMS).
   - PMAC's maximum ADC phase reading of 32,768 corresponds to 50 amps.
   - The motor has the lower continuous current rating, so we use its limit of 15 amps (RMS).
   - 15 amps (RMS) corresponds to peak phase currents of 15*1.414 = 21.21 amps.
   - 21.21 amps corresponds to an ADC reading of 32,768 * 21.21/50 = 13,900.
   - I657 should be set to 13,900 * 0.866 = 12,037.

See Also:
Integrated Current Protection (Making Your Application Safe)
I-Variables Ixx58, Ixx69

Ixx58 Motor xx Integrated Current Limit

Range: 0 - 8,388,607

Units: 2^{30} $(\text{DAC bits})^2$ • servo cycles
{Bits of a 16-bit DAC/ADC}

Default: 0

Ixx58 sets the maximum integrated current limit for Turbo PMAC's I^2T or |I|T integrated current limiting function. If Ixx58 is 0, the I^2T limiting function is disabled. If Ixx58 is greater than 0, Turbo PMAC will compare the time-integrated difference between the commanded or actual current and the Ixx57 continuous current limit to Ixx58. If Ixx57 is greater than 0, Turbo PMAC uses the squares of these current values for I^2T limiting; if Ixx57 is less than 0, Turbo PMAC uses the absolute value of these current values for |I|T limiting. If the integrated value exceeds the limit set by Ixx58, then Turbo PMAC faults the motor just as it would for receiving an amplifier fault signal, setting both the amplifier-fault and the integrated-current-fault motor status bits.

The Ixx58 limit is typically set by taking the relationship between the instantaneous current limit (Ixx69 on Turbo PMAC, in units of a 16-bit DAC), the magnetization current (commanded by Ixx77; typically 0 except for vector control of induction motors) and the continuous current limit (|Ixx57| on Turbo PMAC, in units of a 16-bit DAC) and multiplying by the time permitted at the instantaneous limit.

When using I^2T limiting (Ixx57 > 0), the formula is:

$$Ixx58 = \frac{Ixx69^2 + Ixx77^2 - Ixx57^2}{32768^2} * ServoUpdateRate(Hz) * PermittedTime(sec)$$

When using |I|T limiting (Ixx57 < 0), the formula is:

$$Ixx58 = \frac{\sqrt{Ixx69^2 + Ixx77^2} - |Ixx57|}{32768} * ServoUpdateRate(Hz) * PermittedTime(sec)$$

Refer to the section *Making Your Application Safe* in the User's Guide for a more detailed explanation of I^2T and |I|T protection.

Example:
For I^2T limiting, with the instantaneous current limit Ixx69 at 32,767, the magnetization current Ixx77 at 0, the continuous current limit Ixx57 at 10,589 (1/3 of max), the time permitted with maximum current at 1 minute, and the servo update rate at the default of 2.25 kHz, Ixx58 would be set as

$$Ixx58 = (1.0^2 + 0.0^2 - 0.33^2) * 2250 * 60 = 120000$$

For |I|T limiting, with the instantaneous current limit Ixx69 at 24,576, the magnetization current Ixx77 at 0, the continuous current limit at 8192 (Ixx57 = -8192), the time permitted with maximum current at 3 seconds, and the servo update rate at 4 kHz, Ixx58 would be set as:

$$Ixx58 = (\sqrt{0.75^2 + 0.0^2} - 0.25) * 4000 * 3 = 6000$$

Ixx59 Motor xx User-Written Servo/Phase Enable

Range: 0 - 3

Units: none

Default: 0

Ixx59 controls whether the built-in servo and commutation routines, or user-written servo and commutation routines, are used for Motor xx. The following table shows the possible values of Ixx59 and their effects:

Ixx59	Servo Algorithm	Commutation Algorithm
0	Built-in (PID or ESA)	Built-in
1	User-written	Built-in
2	Built-in (PID or ESA)	User-written
3	User-written	User-written

Any user-written servo or commutation (phase) algorithms will have been coded and cross-assembled in a host computer, and downloaded into PMAC's program memory. These algorithms are retained by the battery on battery-backed RAM versions, or saved into flash memory on flash-backed versions.

Ixx00 must be 1 in order for the user-written servo to execute. Ixx01 must be 1 or 3 in order for the user-written commutation to execute. The servo algorithm can be changed immediately between the built-in algorithm and a user-written algorithm by changing Ixx59. PMAC only selects the phasing algorithm to be used at power-on reset, so in order to change the commutation algorithm, Ixx59 must be changed, this new value stored to non-volatile memory with the **SAVE** command, and the board reset.

It is possible to use the user-written algorithms for purposes other than servo or commutation, making them essentially very fast and efficient PLC programs. This is very useful for fast, position-based outputs. Simply load the code, activate an extra "motor" with Ixx00 and/or Ixx01, and set Ixx59 for this pseudo-motor to use this algorithm.

Ixx60 Motor xx Servo Cycle Period Extension Period

Range: 0 - 255

Units: Servo Interrupt Periods

Default: 0

Ixx60 permits an extension of the servo update time for Motor xx beyond a single servo interrupt period. The servo loop will be closed every (Ixx60 + 1) servo interrupts. With the default value of zero, the loop will be closed every servo interrupt. An extended servo update time can be useful for motors with slow dynamics, and/or limited feedback resolution. It can also be useful if the control loop is used for a slow process-control function.

On Turbo PMAC(1) boards, the servo interrupt period is controlled by hardware settings (jumpers E3-E6, E29-E33, and E98). On Turbo PMAC2 boards, it is controlled by I-variables (I7000, I7001, and I7002 for non-Ultralite boards; I6800, I6801, and I6802 for Ultralite boards).

Other update times, including trajectory update, and phase update, are not affected by Ixx60. I10 does *not* need to be changed with Ixx60.

The filtered motor velocity values reported with the **V** and **<CTRL-V>** commands are not affected by Ixx60. They still will report in counts per servo interrupt. However, the raw actual velocity register will store velocity in terms of counts per servo loop closure.

Ixx61 Motor xx Current-Loop Integral Gain

Range: 0.0 - 1.0 (24-bit resolution)

Units: Output = 8 * Ixx61 * Sum [i=0 to n] ($I_{cmd}[i]$-$I_{act}[i]$)

Default: 0

Ixx61 is the integral gain term of the digital current loops, multiplying the difference between the commanded and actual current levels and adding the result into a running integrator that adds into the command output. It is only used if Ixx82>0 to activate digital current loop execution.

Ixx61 can be used with either Ixx62 forward-path proportional gain, or Ixx76 back-path proportional gain. If used with Ixx62, the value can be quite low, because Ixx62 provides the quick response, and Ixx61 just needs to correct for biases. If used with Ixx76, Ixx61 is the only gain that responds directly to command changes, and it must be significantly higher to respond quickly.

Ixx61 is typically set using the current loop auto-tuner or interactive tuner in the Turbo PMAC Executive or Setup program. Typical values of Ixx61 are near 0.02.

Digital current loop closure on the Turbo PMAC requires a set of three consecutive command output registers. Generally, this requires writing to either a PMAC2-style Servo IC or a MACRO IC.

Ixx61 is only used if Ixx82>0 to activate digital current-loop execution.

Ixx62 Motor xx Current-Loop Forward-Path Proportional Gain

Range: 0.0 - 2.0 (24-bit resolution)

Units: Output = 4 * Ixx62 * (I_{cmd} - I_{act})

Default: 0

Ixx62 is the proportional gain term of the digital current loops that is in the "forward path" of the loop, multiplying the difference between the commanded and actual current levels. Either Ixx62 or Ixx76 (back path proportional gain) must be used to close the current loop. Generally, only one of these proportional gain terms is used, although both can be. Ixx62 is only used if Ixx82>0 to activate digital current loop execution.

Ixx62 can provide more responsiveness to command changes from the position/velocity loop servo, and therefore a higher current loop bandwidth, than Ixx76. However, if the command value is very noisy, which can be the case with a low-resolution position sensor, using Ixx76 instead can provide better filtering of the noise.

Typically, Ixx62 is set using the current loop auto-tuner or interactive tuner in the Turbo PMAC Executive or Setup program. Typical values of Ixx62, when used, are around 0.9.

Digital current loop closure on the Turbo PMAC requires a set of three consecutive command output registers. Generally, this requires writing to either a PMAC2-style Servo IC or a MACRO IC.

Ixx62 is only used if Ixx82>0 to activate digital current-loop execution.

Ixx63 Motor xx Integration Limit

Range: -8,388,608 - 8,388,607

Units: (Ixx33 / 2^{19}) counts * servo cycles

Default: 4,194,304

Ixx63 limits the magnitude of the integrated position error (the *output* of the integrator) for the PID servo algorithm, which can be useful for "anti-windup" protection, when the servo loop output saturates. The default value of Ixx63 provides essentially no limitation. (The integral gain Ixx33 controls how fast the error is integrated.)

A value of zero in Ixx63 forces a zero output of the integrator, effectively disabling the integration function in the PID filter. This can be useful during periods when you are applying a constant force and are expecting a steady-state position error. (In contrast, setting Ixx33 to 0 prevents further inputs to the integrator, but maintains the output.)

The Ixx63 integration limit can also be used to create a fault condition for the motor. If Ixx63 is set to a negative number, then PMAC will also check as part of its following error safety check whether the magnitude of integrated following error has saturated at the magnitude of Ixx63. With Ixx63 negative, if the integrator has saturated, PMAC will trip (kill) the motor with a following error fault. Both the normal fatal following error motor status bit and the integrated following error status bit are set when this fault occurs. If Ixx63 is 0 or positive, the motor cannot trip on integrated following error fault.

To set Ixx63 to a value such that the integrator saturates at the same point that its contribution to the command output causes saturation at the Ixx69 level, use the following formula:

$$Ixx63 = \pm\left(\frac{Ixx69 * 2^{23}}{Ixx08 * Ixx30}\right)$$

To cause trips, the magnitude of Ixx63 must be set to less than this value due to other potential contributions to the output. Remember that the integrator stops increasing when the output saturates at Ixx69.

Ixx63 is not used if the Extended Servo Algorithm for Motor xx is being executed (Iyy00=1).

Ixx64 Motor xx Deadband Gain Factor

Range: -32,768 - 32,767

Units: none

Default: 0 (no gain adjustment)

Ixx64 is part of the PMAC feature known as deadband compensation, which can be used to create or cancel deadband. It controls the effective gain within the deadband zone (see Ixx65). When the magnitude of the following error is less than the value of Ixx65, the proportional gain (Ixx30) is multiplied by (Ixx64+16)/16. At a value of -16, Ixx64 provides true deadband.

Values between -16 and 0 yield reduced gain within the deadband. Ixx64 = 0 disables any deadband effect.

Values of Ixx64 greater than 0 yield increased gain within the deadband; a value of 16 provides double gain in the deadband. A small band of increased gain can be used to reduce errors while holding position, without as much of a threat to make the system unstable. It is also useful in compensating for physical deadband in the system.

Note:

Values of Ixx64 less than -16 will cause negative gain inside the "deadband", making it impossible for the system to settle inside the band. These settings have no known useful function.

Outside the deadband, gain asymptotically approaches Ixx30 as the following error increases.

Ixx64 is not used if the Extended Servo Algorithm for Motor xx is being executed (Iyy00/50=1).

Ixx65 Motor xx Deadband Size

Range: -32,768 - 32,767

Units: 1/16 count

Default: 0

Ixx65 defines the size of the position error band, measured from zero error, within which there will be changed or no control effort, for the PMAC feature known as "deadband compensation". Ixx64 controls the effective gain relative to Ix30 within the "deadband".

Note:

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the deadband. For example, if modified gain is desired in the range of +/-5 counts of following error, Ixx65 should be set to 80.

Ixx65 is not used if the Extended Servo Algorithm for Motor xx is being executed (Iyy00/50=1).

Ixx66 Motor xx PWM Scale Factor

Range: 0 - 32,767

Units: PWM_CLK cycles

Default: 6527

Ixx66 multiplies the output of the digital current loops for Motor x (which are values between -1.0 and 1.0) before they are written to the PWM output registers. As such, it determines the maximum value that can be written to the PWM output register. Ixx66 is only used if Ixx82>0 to activate digital current loop execution.

The PWM output value for each phase is compared digitally to the PWM up-down counter, which increments or decrements once per PWM_CLK cycle to determine whether the outputs are on or off. The limits of the up-down counter are set by the PWM maximum count variable, I6m00 for Servo IC m on PMAC, and MI900, MI906, or MI992 on the MACRO Station.

Generally, Ixx66 is set to about 10% above the PWM maximum count value. This permits a full-on command of the phase for a substantial fraction of the commutation cycle, providing maximum possible utilization of the power devices at maximum command. If Ixx66 is set to a smaller value than PWM maximum count, it serves as a voltage limit for the motor (Vmax = VDC * PWM_Max_Count / Ixx66). Note that Ixx69 serves as the current limit.

Digital current loop closure on the Turbo PMAC requires a set of three consecutive command output registers. Generally, this requires writing to either a PMAC2-style Servo IC or a MACRO IC.

Ixx67 Motor xx Position Error Limit

Range: 0 – 8,388,607

Units: 1/16 count

Default: 4,194,304 (= 262,144 counts)

Ixx67 defines the biggest position error that will be allowed into the servo filter. This is intended to keep extreme conditions from upsetting the stability of the filter. However, if it is set too low, it can limit the response of the system to legitimate commands (this situation can particularly be noticed on very fine resolution systems).

If pure velocity control is desired for the motor, Ixx67 can be set to 0, effectively disabling the position loop.

This parameter is not to be confused with Ixx11 or Ixx12, the following error. Those parameters take action outside the servo loop based on the real (before limiting) following error.

Note:

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the limit.

Ixx67 is not used if the Extended Servo Algorithm for Motor xx is being executed (Iyy00=1).

Ixx68 Motor xx Friction Feedforward

Range: 0 .. 32,767

Units: 16-bit DAC bits

Default: 0

Ixx68 adds a bias term to the servo loop output of Motor xx that is proportional to the *sign* of the commanded velocity. That is, if the commanded velocity is positive, Ixx68 is added to the output. If the commanded velocity is negative, Ixx68 is subtracted from the output. If the commanded velocity is zero, no value is added to or subtracted from the output.

This parameter is intended primarily to help overcome errors due to mechanical friction. It can be thought of as a "friction feedforward" term. Because it is a feedforward term that does not utilize any feedback information, it has no direct effect on system stability. It can be used to correct the error resulting from friction, especially on turnaround, without the time constant and potential stability problems of integral gain.

Ixx68 is used with both the PID servo algorithm executed if Iyy00=0, and the Extended Servo Algorithm executed if Iyy00=1. If Turbo PMAC is commutating this motor, the Ixx68 bias is applied before the commutation algorithm, and so will affect the magnitude of both analog outputs.

Note:

This direction-sensitive bias term is independent of the constant bias introduced by Ixx29 and/or Ixx79.

Example:

For a control loop with ±10V analog output, starting with a motor at rest, if Ixx68 = 1600, then as soon as a commanded move in the positive direction is started, a value of +1600 (~0.5V) is added to the servo loop output. As soon as the commanded velocity goes negative, a value of -1600 is added to the output. When the commanded velocity becomes zero again, no bias is added to the servo output as a result of this term.

Ixx69 Motor xx Output Command Limit

Range: 0 .. 32,767 (0 to 10V or equivalent)

Units: 16-bit DAC bits

Default: 20,480 (6.25V or equivalent)

Ixx69 defines the magnitude of the largest output that can be sent from Turbo PMAC's PID position/velocity servo loop. If the servo loop computes a larger value, Turbo PMAC clips it to this value. When the PID output has saturated at the Ixx69 limit, the integrated error value will not increase, providing "anti-windup" protection.

For the Extended Servo Algorithm (ESA) that is enabled if Iyy00/50 for the motor is set to 1, Ixx69 is used to multiply a normalized command (-1.0 <= Normalized Command < +1.0) before outputting it or using it for commutation. As such, it acts as both a scale factor and an output command limit for the ESA.

Ixx69 is always in units of a 16-bit DAC, even if the actual output device is of a different resolution, or the command value is used for Turbo PMAC's own internal current loop commands.

If you are using differential analog outputs (DAC+ and DAC-), the voltage between the two outputs is twice the voltage between an output and AGND, so the Ixx69 value should be set to half of what it would be for a single-ended analog output.

This parameter provides a torque (current) limit in systems with current-loop amplifiers, or when using Turbo PMAC's internal commutation; it provides a velocity limit with velocity-mode amplifiers. Note that if this limit "kicks in" for any amount of time, the following error will start increasing.

Use when Commutating: When Turbo PMAC is commutating Motor xx, Ixx69 corresponds to *peak* values of the sinusoidal phase currents. Motor and amplifier current limits are usually given as RMS values. Peak phase values are √2, or 1.414, times greater than RMS values. For instance if an amplifier has a 10 amp (RMS) instantaneous current limit, the instantaneous limit for the peak of the phase currents is 14.14 amps.

Use with Magnetization Current: When commutating (Ixx01 bit 0 = 1), Ixx69 is technically the limit of only the quadrature, or torque-producing, current. Ixx77 sets the magnitude of the direct, or magnetization current, and the total current limit is the vector sum of these two variables. If the Ixx77 magnetization current for the motor is set to a value other than 0, Ixx69 should be set such that:

$$\sqrt{Ixx69^2 + Ixx77^2} \le I_{max} \le 32{,}767$$

Use in Direct-PWM Mode: When commutating (Ixx01 bit 0 = 1) and closing the current loop (Ixx82 > 0) of a 3-phase motor (Ixx72 = 683 or 1365), it is important to understand the relationship between the quadrature current limited by Ixx69 and the phase currents measured by the A/D converters. This difference is due to the nature of the conversion between direct and quadrature current components, which are 90° apart, and the phase currents, which are 120° apart. This difference introduces a factor of cos(30°) into the calculations.

For a given level of DC quadrature current with zero direct (magnetization) current, the peak value of AC sinusoidal current measured in the phases will be 1/cos(30°), or 1.15 times, greater. When quadrature current is commanded at its limit of Ixx69, the peak phase currents can be 15% higher that this value. For instance, with Ixx69 at 10,000, and Ixx77 at 0, the A/D converters can provide readings (normalized to 16-bit resolution) up to 11,547.

With non-zero direct current, the peak value of AC sinusoidal current measured in the phases will be 1.15 times greater than the vector sum of the direct and quadrature currents. Therefore, in order not to saturate the current in the phases, Ixx69 should be set such that:

$$\sqrt{Ixx69^2 + Ixx77^2} \le I_{max} \cos(30^o) \le 32{,}767 * 0.866 \le 28{,}377$$

Examples:

1. Motor 1 is driving a velocity-mode amplifier with differential analog inputs that are limited to +/-10V between the inputs. This means that the PMAC outputs should each be limited to +/-5V with respect to the AGND reference. I169 should therefore be limited to 32,768/2 = 16,384.
2. Motor 3 is driving a DC brush motor amplifier in current (torque) mode with an analog output. The amplifier has a gain of 2 amps/volt and an instantaneous current limit of 20 amps. The motor has an instantaneous current limit of 15 amps.
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 20 amps.
   - The motor has the lower instantaneous current rating, so we use its limit of 15 amps.
   - I369 is set to 32,768 * 15/20 = 24,576.
3. Motor 5 is driving a self-commutating brushless-motor amplifier in current (torque) mode with a single analog output. The amplifier has a gain of 5 amps(RMS)/volt and an instantaneous current limit of 50 amps (RMS). The motor has an instantaneous current limit of 60 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 50 amps (RMS).
   - The amplifier has the lower instantaneous current rating, so we use its limit of 50 amps (RMS).
   - I569 is set to 32,768 * 50/50 = 32,767 (note that the maximum permitted value is 32,767).
4. Motor 7 is driving a "sine-wave" amplifier for a brushless servo motor with two analog outputs. The Ixx77 magnetization current limit is set to 0. The amplifier has a gain on each phase of 4 amps/volt. The amplifier has an instantaneous current limit of 25 amps (RMS). The motor has an instantaneous current limit of 30 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 40 amps peak in the phase.
   - The amplifier has the lower instantaneous current rating, so we use its limit of 25 amps (RMS).
   - 25 amps (RMS) corresponds to peak phase currents of 25*1.414 = 35.35 amps.
   - I769 is set to 32,768 * 35.35/40 = 28,958.
5. Motor 9 is driving a direct-PWM "power-block" amplifier and an AC induction motor. The Ixx77 magnetization current parameter is set to 3000. The A/D converters in the amplifier are scaled so that a maximum reading corresponds to 100 amps of current in the phase. The amplifier has an instantaneous current limit of 60 amps (RMS), and the motor has an instantaneous current limit of 75 amps (RMS).
   - PMAC's maximum ADC phase reading of 32,768 corresponds to 100 amps in the phase.
   - The amplifier has the lower instantaneous current rating, so we use its limit of 60 amps (RMS).
   - 60 amps (RMS) corresponds to peak phase currents of 60*1.414 = 84.84 amps.
   - 84.84 amps corresponds to an ADC reading of 32,768 * 84.84/100 = 27,800.
   - The vector sum of Ixx69 and Ixx77 should equal 27,800 * 0.866 = 24,075.
   - I969 should be set to sqrt($24{,}075^2$-$3{,}000^2$) = 23,887.

Motor Commutation Setup I-Variables

Ixx70 Motor xx Number of Commutation Cycles (N)

Range: 0 – 255

Units: Commutation Cycles

Default: 1

For a PMAC-commutated motor (Ixx01=1), Ixx70 is used in combination with Ixx71 to define the size of the commutation cycle, as Ixx71/Ixx70 counts. Usually, Ixx70 is set to one, and Ixx71 represents the number of counts in a single commutation cycle. However, many people will use Ixx70 to represent the number of commutation cycles (pole pairs) per mechanical revolution, and Ixx71 to represent the counts per mechanical revolution. Ixx70 only *needs* to be set greater than one if the number of counts in a single cycle is not an integer.

A commutation cycle, or electrical cycle, consists of two poles (one pole pair) of a multiphase motor.

Setting Ixx70 to 0 effectively defeats the creation of the AC commutation cycle. This setting can be useful when doing direct PWM control of DC brush motors which requires the use of the Turbo PAMC commutation algorithms, but cannot use an AC output.

Example:
A 6-pole brushless motor has three commutation cycles per mechanical revolution. If a feedback device with 4096 counts per mechanical revolution (a number not divisible by three) is used, Ixx70 should be set to 3, and Ixx71 to 4096.

See Also:
I-variables Ixx01, Ixx71-Ixx83
Setting Up PMAC Commutation

Ixx71 Motor xx Counts per N Commutation Cycles

Range: 0 – 16,777,215

Units: counts

Default: 1000

For a Turbo PMAC-commutated motor, this parameter defines the size of a commutation cycle in conjunction with Ixx70 (counts/cycle = Ixx71/Ixx70). The meaning of a "count" used in this parameter is defined by the encoder-decode variable I7mn0 for the commutation feedback device. If a "times-4" decode is used, a *count* is one-fourth of an encoder *line*.

When the commutation position feedback is received over the MACRO ring, the units of the feedback are typically 1/32 of a count, so Ixx71 should be in units of 1/32 count in this case.

A commutation cycle, or electrical cycle, consists of two poles (one pole pair) of a multiphase motor.

> ***Note***
>
> In firmware revisions V1.938 and older, the maximum value of Ixx71 was 8,388,607.

Examples:

1. A four-pole brushless motor with a 1000-line-per-revolution encoder and "times-4" decode has 2 commutation cycles per revolution and 4000 counts per revolution. Therefore, either Ixx70=2 and Ixx71=4000 could be used, or Ixx70=1 and Ixx71=2000.
2. A linear motor has a 60.96-mm (2.4-inch) electrical cycle. An encoder with a 40 micron pitch is wired directly into PMAC and "times-4" decode is used. Ixx70 can be set to 1 and Ixx71 can be calculated as:

$$Ixx71 = 60.96 \frac{mm}{cycle} * \frac{line}{0.04mm} * 4 \frac{counts}{line} = 6096 \frac{counts}{cycle}$$

3. An 8-pole brushless motor has an 8192-line encoder that is wired into a Compact MACRO Station with "times-4" decode. The position data is sent back to PMAC in the MACRO "Type 1" protocol, with units of 1/32 count. If Ixx70 is set to 4 (for 4 electrical cycles per revolution), Ixx71 can be calculated as:

$$Ixx71 = 8192 \frac{lines}{rev} * \frac{rev}{4 - cycles} * 4 \frac{counts}{line} * 32 \frac{(1/32 count)}{count} = 262{,}144 \frac{(1/32 count)}{line}$$

See Also:
I-variables Ixx01, Ixx70, Ixx72-Ixx83
Setting Up Turbo PMAC Commutation

Ixx72 Motor xx Commutation Phase Angle

Range: 0 – 2047

Units: 360/2048 elec. deg. (1/2048 commutation cycle)

Default: 1365 (= -120°e or 240°e)

For a Turbo PMAC-commutated motor, Ixx72 sets the angular distance between the phases of a multiphase motor. The units of Ixx72 are 1/2048 of an electrical cycle. The usual values to be used are:

3-phase: 683 or 1365 (+/- 120°e)

2- or 4-phase: 512 or 1536 (+/- 90°e)

For a given number of phases, the proper choice of the two possible values is determined by the polarity match between the output commands and the feedback, as detailed below. Typically the choice is made automatically by the "Turbo Setup" expert-system program on the PC.

Ixx72 is used slightly differently depending on whether Turbo PMAC is performing current-loop calculations as well as commutation. Both cases are explained below:

1. **Turbo PMAC performing commutation, but not current loop:** When Turbo PMAC *is not* performing digital current loop closure for Motor xx (Ixx82=0), the output direction sense determined by this parameter and the motor and amplifier phase wiring must match the feedback direction sense as determined by the encoder-decode variable I7mn0 and the encoder wiring. If the direction senses do not match, proper commutation and servo control will be impossible; the motor will lock into a given position.

   For these systems, changing between the two values for a given number of phases has the same effect as exchanging motor leads, which changes the motor's direction of rotation for a given sign of a PMAC2 torque command.

   Refer to the section Setting Up Turbo PMAC Commutation for tests to determine the proper Ixx72 setting. For systems without Turbo PMAC digital current loop closure, once this commutation/feedback polarity has been properly matched, the servo/feedback polarity will automatically be properly matched.

2. **Turbo PMAC performing commutation and current loop:** When Turbo PMAC (PMAC2-style interface only) *is* performing digital current loop closure for Motor xx (Ixx82 > 0), the output direction sense determined by this parameter must match the polarity of the phase current sensors and the analog-to-digital conversion (ADC) circuitry that brings this data into Turbo PMAC. It is independent of motor or amplifier phase wiring, encoder wiring, and Turbo PMAC encoder-decode direction sense.

WARNING:

Do not attempt to close the digital current loops on Turbo PMAC (O commands or closing the position loop) until you are sure of the proper sense of the Ixx72 setting. An Ixx72 setting of the wrong sense will cause positive feedback in the current loop, leading to saturation of the PMAC outputs and possible damage to the motor and or amplifier.

For these systems with a Turbo PMAC digital current loop, if the phase-current ADC registers report a positive value for current flowing *into* the phase (i.e. the PWM voltage command value and the current feedback value have the same sign), Ixx72 must be set to a value greater than 1024 (usually 1365 for a 3-phase motor, or 1536 for a 2- or 4-phase motor).

If the phase-current ADC registers report a positive value for current flowing *out of* the phase (i.e. the PWM voltage command value and the current feedback value have opposite signs), Ixx72 must be set to a value less than 1024 (usually 683 for a 3-phase motor, or 512 for a 2- or 4-phase motor).

For systems with Turbo PMAC digital current loop closure, the commutation/feedback polarity match is independent of the servo/feedback polarity. Once Ixx72 has been set for proper commutation/feedback polarity, the proper position-loop servo/feedback polarity must still be established.

Ixx73 Motor xx Phase Finding Output Value

Range: -32,768 – 32,767

Units: bits of 16-bit DAC

Default: 0

WARNING:

An unreliable phasing search method can lead to a runaway condition. Test your phasing search method carefully to make sure it works properly under all conceivable conditions. Make sure your Ix11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Ixx73 defines the magnitude of the open-loop output to be used if a power-on phasing search is done for a Turbo PMAC-commutated motor (Ixx01 bit 0 = 1). A phasing search is required for a synchronous motor (Ixx78=0) such as a permanent-magnet brushless motor with no absolute position sensor (Ixx81=0). The phasing search is done automatically as part of the power-on phasing search if Ixx80 is 1 or 3; if Ixx80 is 0 or 2, the on-line **$** or **$$** command must be used must be used to initiate the phasing search.

Ixx73 is in units of a 16-bit DAC, so that 32,767 represents full current command to the phases, even if a different output device and/or different resolution is used.

If Ixx80 is 0 or 1, the "two-guess" phasing search is used, and Ixx73 controls the "vector" magnitude of the open-loop output that is distributed among the phases according to the guessed phasing angle.

If Ixx80 is 2 or 3, the "stepper-motor" phasing search is used, and Ixx73 controls the magnitude of current forced into individual phase(s) to lock the motor to a position like a stepper motor. In this method, if the Turbo PMAC is not performing current loop closure for the motor (Ixx82=0) and Ixx72 > 1024, then Ixx73 should be set to a negative number of the desired magnitude. In all other cases it should be set to a positive number. If the sign of Ixx73 is wrong for your setup, the motor will run away when the loop is closed.

A negative value of Ixx73 must be used for sinewave-output commutation (Ixx82 = 0) with Ixx72 > 1024 and the stepper-motor phase search method (Ixx80 bit 0 = 1).

Typically, values of magnitude 2000 to 6000 are used for Ixx73 in either method.

See Also:
Power-Up Phasing Search (Setting Up PMAC Commutation)
I-Variables Ixx01, Ixx74, Ixx78, Ixx80, Ixx81

Ixx74 Motor xx Phase Finding Time

Range: 0 – 255

Units: Servo Interrupt Cycles (for Ixx80 = 0 or 1)
or
Servo Interrupt Cycles * 256 (for Ixx80 = 2 or 3)

Default: 0

WARNING

An unreliable phasing search method can lead to a runaway condition. Test your phasing search method carefully to make sure it works properly under all conceivable conditions. Make sure your Ixx11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Ixx74 defines the time that an open-loop output is to be used if a power-on phasing search is done for a PMAC-commutated motor (Ixx01 bit 0 = 1). A phasing search is required for a synchronous motor (Ixx78=0) such as a permanent-magnet brushless motor with no absolute position sensor (Ixx81=0). The phasing search is done automatically as part of the power-on phasing search if bit 0 of Ixx80 is 1; if bit 0 of Ixx80 is 0, the on-line **$** or **$$** command must be used must be used to initiate the phasing search.

If Ixx74 is set to 0, no phasing search move will be done, even if one is requested and required. In this case, the "phase reference error" motor status bit will stay set, preventing the servo loop from closing.

If bit 1 of Ixx80 is 0 (Ixx80 = 0 or 1), the "two-guess" phasing search is used; Ixx74 has units of servo cycles and controls the time for the open-loop command at each "guess" of the phase angle. Typical values are 3 to 10 servo cycles; a value of 5 is a good starting point.

If bit 1 of Ixx80 is 1 (Ixx80 = 2 or 3), the "stepper-motor" phasing search is used; Ixx74 has units of (servo cycles*256) and controls the time current is forced into each phase and Turbo PMAC waits for the motor to settle into the "step" position. With the default servo cycle rate of 2.25 kHz, each unit of Ixx74 represents about 0.1 seconds in this mode; typical values are 10 to 20.

See Also:
Power-Up Phasing Search (Setting Up PMAC Commutation)
I-Variables Ixx01, Ixx73, Ixx78, Ixx80, Ixx81

Ixx75 Motor xx Phase Position Offset

Range: 0 – Ixx71 (up to 16,777,215)

Units: Counts * Ixx70

Default: 0

Ixx75 tells Turbo PMAC the distance between the zero position of an absolute sensor used for power-on phase position (specified by Ixx81 and Ixx91) and the zero position of Turbo PMAC's commutation cycle.

It is used to reference the phasing algorithm for a PMAC-commutated motor with an absolute sensor (Ixx81 > 0). If Ixx80 bit 0 is 1 (Ixx80 = 1 or 3), this is done automatically during the power-up/reset cycle. It will also be done in response to a **$** on-line command to the motor, or a **$$** on-line command to the coordinate system containing the motor.

Ixx75 is also used by the **SETPHASE** command (on-line, motion-program, or PLC-program). When the **SETPHASE** command is given, the value of Ixx75 is immediately copied directly into the motor's phase position register. This operation is typically used to correct the phasing, usually at the encoder index pulse, after an initial rough phasing (e.g. from Hall commutation sensors).

The proper value for this parameter can be found with a simple procedure that should be done with an unloaded motor, after satisfactory operation has been achieved using a power-on phasing search.

- Define an M-variable to the absolute sensor if you have one.
- Define an M-variable to the internal phase position register. Mxx71 is the suggested M-variable.
- Give the motor an **O0** command.
- Put a bias (a magnitude of 2000 is usually good) on the A phase (higher-numbered DAC of a pair for Turbo PMAC(1)) by setting Ixx29; use a positive bias if Ixx82>0 for digital current loop closure, or if Ixx82=0 and Ixx72>1024 (e.g. 1365 or 1536); use a negative bias if Ixx82=0 and Ixx72<1024 (e.g. 683 or 512).
- Also, put a bias in the opposite direction of the same magnitude on the B phase by setting Ixx79. The motor should lock in on a position like a stepper motor.
- Now remove the A-phase bias by setting Ixx29 back to zero, or at least to the value you have found to force zero current in the phase, and the motor should lock in on another position. This position is the zero position of the phasing cycle.
- If you have an absolute sensor, after you are sure the motor has settled, read the position of the absolute sensor by querying its M-variable value. Then:
- Take the negative of this value, multiply it by Ixx70, and put the resulting value in Ixx75.
- Now, with Ixx79 returned to zero or the proper bias, and Ixx81 pointing to the absolute sensor, give the motor a **$** command. The motor should be properly phased.
- If you are doing this so you can use the **SETPHASE** command at a known position such as the index, set the internal phase position register to 0 with Mxx71. Then:
- Return Ixx79 to zero or the proper bias, and close the loop with a **J/** command.
- Now move to the reference position (e.g. do a homing search move with the index pulse as the trigger) and make sure you are settled there with minimal following error (some integral gain should be used).
- Read the value of Mxx71 at this point and set Ixx75 to this value.
- Remember to save these variable values before doing a full reset on the card.

See Also
I-variables Ixx01, Ixx70 – Ixx74, Ixx76 – Ixx83
Setting Up Turbo PMAC Commutation

Ixx76 Motor xx Current-Loop Back-Path Proportional Gain

Range: 0.0 – 2.0 (24-bit resolution)

Units: PWMout = -4 * Ixx62 * (Iact)

Default: 0.0

Ixx76 is the proportional gain term of the digital current loop that is in the "back path" of the loop, multiplying the actual current level, and subtracting the result from the command output. Either Ixx76 or Ixx62 (forward path proportional gain) must be used to close the current loop. Generally, only one of these proportional gain terms is used, although both can be.

If Ixx76 is used as the only proportional gain term, only the Ixx61 integral gain term reacts directly to command changes. The act of integration acts as a low-pass filter on the command, which eliminates a lot of noise, but lowers the responsiveness to real changes. Generally, Ixx76 is only used when the command value from the position/velocity loop servo have high noise levels (usually due to low position resolution), and the actual current measurements have low noise levels.

Ixx76 is typically set using the current loop auto-tuner or interactive tuner in the Turbo PMAC Executive Program. Typical values of Ixx76, when used, are around 0.9.

Digital current loop closure on the Turbo PMAC requires a set of three consecutive command output registers. Generally, this requires writing to either a PMAC2-style Servo IC or a MACRO IC.

Ixx76 is only used if Ixx82>0 to activate digital current loop execution.

Ixx77 Motor xx Magnetization Current

Range: -32,768 – 32,767

Units: Bits of a 16-bit DAC

Default: 0

This parameter is used in induction motors to provide a stator current component parallel to the estimated rotor magnetic field (the "direct" current -- the control loop determines the magnitude of the "quadrature" current perpendicular to this component). This should generally be set to zero for non-induction motors, unless advanced "field weakening" algorithms are desired.

The proper value for an induction motor is system dependent, but 2500 is a good starting value for most motors. Refer to the *Setting Up Commutation* section of the manual for instructions in optimizing the setting of this parameter. The "Turbo Setup" expert-system program for PCs is typically used to set the proper value of Ixx77 for induction motors.

If Ixx77 is set to a non-zero value, Ixx69 should be reduced from what it would be with Ixx77 set to 0. The effective current limit is:

$$I_{max} = \sqrt{Ixx69^2 + Ixx77^2}$$

See Also:
Setting Induction Motor Parameters (Setting Up PMAC Commutation)
I-variables Ixx01, Ixx70-Ixx72, Ixx78

Ixx78 Motor xx Slip Gain

Range: 0.0 – 1.0 (24-bit resolution)

Units: none (ratio of times)

Default: 0.0

Ixx78 controls the relationship between the torque command and the slip frequency of magnetic field on the rotor of an AC asynchronous (induction) motor. While it is usually set experimentally, it can be calculated as the ratio between the phase update period and the rotor (not stator) L/R electrical time constant.

Turbo PMAC computes the slip frequency each phase update by multiplying the torque command from the position/velocity-loop servo (or O-command magnitude) by Ixx78 and dividing by the magnetization current value controlled by Ixx77.

Ixx78 is typically set through use of the "Turbo Setup" expert-system program running on PCs. This program excites the motor and analyzes its response to derive an optimum Ixx78 value.

Ixx78 can also be set experimentally by giving the motor an O-command and watching the velocity response, probably with the data-gathering feature. As the velocity saturates because the back EMF reaches the supply voltage, the velocity should fall back about 5% to reach a steady-state value. If it falls back more than this, the slip time constant is too high; if it falls back less than this, or not at all, the slip time constant is too low.

0.00015 is a typical value of Ixx78 for a standard induction motor at a phase update rate of about 9 kHz.

Ixx78 is only active if Ixx01 is set to 1 to specify Turbo PMAC commutation of Motor xx. It should be set to 0 for AC synchronous motors such as permanent-magnet brushless motors and switched (variable) reluctance motors.

Ixx79 Motor xx Second Phase Offset

Range: -32,768 – 32,767

Units: 16-bit DAC/ADC bit equivalent

Default: 0

Ixx79 serves as an output or feedback offset for Motor xx; its exact use depends on the mode of operation as described below:

Mode 1: When Turbo PMAC is not commutating Motor xx (Ixx01 bit 0 = 0), Ixx79 is not used. Ixx29 is the offset for this mode.

Mode 2: When Turbo PMAC is not commutating Motor xx (Ixx01 bit 0 = 0) but is in sign-and-magnitude output mode (Ixx96 = 1 – PMAC(1)-style outputs only), Ixx79 is the offset of the command output value after the absolute value is taken (Ixx29 is the offset before the absolute value is taken). Ixx79 is typically used in this mode to compensate for analog offsets in interface circuitry, either in DACs or in voltage-to-frequency converters.

Mode 3: When Turbo PMAC is commutating Motor xx (Ixx01 bit 0 = 1) but not closing the current loop (Ixx82 = 0), Ixx79 serves as the offset for the second of two phase command output values (Phase B), for the address specified by Ixx02 plus 1; Ixx29 serves the same purpose for the first phase. Ixx79 is added to the output command value before it is written to the command output register.

When commutating from a PMAC(1)-style Servo IC, Phase A is output on the *higher*-numbered of the two DACs (e.g. DAC2), Phase B on the *lower*-numbered (e.g. DAC1). When commutating from a PMAC2-style Servo IC, Phase A is output on the A-channel DAC (e.g. DAC1A), Phase B on the B-channel DAC (e.g. DAC1B).

As an output command offset, Ixx79 is always in units of a 16-bit register, even if the actual output device is of a different resolution. For example, if a value of 60 had to be written into an 18-bit DAC to create a true zero command, this would be equivalent to a value of 60/4=15 in a 16-bit DAC, so Ixx79 would be set to 15 to cancel the offset.

Mode 4: When Turbo PMAC is commutating (Ixx01 bit 0 = 1) and closing the current loop for Motor xx (Ixx82 > 0), Ixx79 serves as an offset that is added to the phase current reading from the ADC for the second phase (Phase B), at the address specified by Ixx82. Ixx29 performs the same function for the first phase. The sum of the ADC reading and Ixx79 is used in the digital current loop algorithms.

As an input feedback offset, Ixx79 is always in units of a 16-bit ADC, even if the actual ADC is of a different resolution. For example, if a 12-bit ADC reported a value of -5 when no current was flowing in the phase, this would be equivalent to a value of -5*16=-80 in a 16-bit ADC, so Ixx79 would be set to 80 to compensate for this offset.

Ixx80 Motor xx Power-Up Mode

Range: 0 – 7

Units: none

Default: 0

Ixx80 controls the power-up mode, including the phasing search method (if used), for Motor xx. It consists of 3 independent control bits, each determining one aspect of the state of the motor at power-up or full board reset:

- Bit 0 controls whether the motor is enabled at power-up/reset or not. If bit 0 is set to 0, the motor is left in the "killed" (disabled) state at power-up/reset, and a command must be issued to the motor to enable it. If bit 0 is set to 1, the motor is automatically enabled at power-up/reset, and if a phasing search move is required to establish the commutation position reference, this is automatically done.
- Bit 1 controls what type of phasing search move is performed, if one is required (Ixx01 bit 0 = 1, Ixx78 = 0, Ixx74 > 0), either during power-up/reset, or on a subsequent **$** motor reset command. If bit 1 is 0 and a phasing search move is required, Turbo PMAC will use the "two-guess" phasing search method. If bit 1 is 1 and a phasing search move is required, Turbo PMAC will use the "stepper-motor" phasing search method. The state of bit 1 does not matter unless a phasing search move is to be done.
- Bit 2 controls whether an absolute position read for the motor is done at power-up/reset or not, if one is required (Ixx10 > 0). If bit 2 is set to 0 and an absolute position read is specified, this read operation will be performed automatically at the board power-up/reset. If bit 2 is set to 1 and an absolute position read is specified, this read operation will not be done automatically at power-up/reset, and the **$*** or **$$*** command must be issued to perform the absolute position read. The state of bit 2 does not matter unless an absolute position read is to be done.

The possible values of Ixx80 and the function of each are described in the following table:

Ixx80	Absolute Position Read at Power-up/Reset?	Phasing Search Method	Power-up/Reset Enable State
0	Yes	Two-Guess	Disabled
1	Yes	Two-Guess	Enabled
2	Yes	Stepper-Motor	Disabled
3	Yes	Stepper-Motor	Enabled
4	No	Two-Guess	Disabled
5	No	Two-Guess	Enabled
6	No	Stepper-Motor	Disabled
7	No	Stepper-Motor	Enabled

Power-up/reset enable state: If the motor is not automatically enabled at power-up/reset, a command must be used subsequently to enable the motor. If Turbo PMAC is commutating the motor (Ixx01 bit 0 = 1) and it is a synchronous motor (Ixx78 = 0), a phase reference must be established with the **$** or **$$** command as part of the enabling process. The motor cannot be enabled before a successful phase reference is established, because the motor "phase reference error" status bit that is automatically set on power-up/reset will not have been cleared.

If the motor is either not commutated by Turbo PMAC (Ixx01 bit 0 =0) or it is not a synchronous motor (Ixx78 > 0), a simple enabling command can be used. The **J/** command enables a single motor; the **A** command enables all of the motors in a coordinate system; the **<CTRL-A>** command enables all of the motors on Turbo PMAC.

The phase reference, whether executed at power-up/reset or on the **$** command, can be done either by reading an absolute position sensor (Ixx81 > 0) or by a phasing search move (Ixx74 > 0) if only an incremental sensor is used.

WARNING:

An unreliable phasing search method can lead to a runaway condition. Test your phasing search method carefully to make sure it works properly under all conceivable conditions. Make sure your Ixx11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Phasing search move method: The two-guess phasing search is very quick and requires little movement, but can be adversely affected if there are significant external loads such as friction and gravity. The stepper-motor phasing search takes more time and causes more movement, but it is more reliable in the presence of significant external loads.

Absolute motor position read: If Ixx10 is set to 0, the position reference for a motor comes from a homing search move. If Ixx10 is greater than 0, the position reference comes from reading an absolute position sensor at the address specified by Ixx10 and with the format specified by Ixx95. In this case, Ixx80 bit 2 specifies whether this read is done automatically at power-up/reset.

If the absolute position read is not done automatically at power-up/reset, the motor position will be set to 0 at this time. This does not prevent full operation of the motor. The **$*** or **$$*** command *must* be used later to read the sensor and establish absolute position. Even if the absolute position is read automatically at power-up/reset, it *may* be read again later with the **$*** or **$$*** command.

See Also:
Power-Up Phasing Search (Setting Up PMAC Commutation)
On-line commands **$**, **$$**, **$* $$***, **$$$**
I-Variables Ixx01, Ixx73, Ixx74, Ixx78, Ixx81

Ixx81 Motor xx Power-On Phase Position Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC or multiplexer-port addresses

Default: 0

WARNING:

An unreliable phasing reference method can lead to a runaway condition. Test your phasing reference method carefully to make sure it works properly under all conceivable conditions. Make sure your Ixx11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Ixx81 tells Turbo PMAC what address to read for absolute power-on phase-position information for Motor xx, if such information is present. This can be a different address from that of the ongoing phase position information, which is specified by Ixx83, but it must have the same resolution and direction sense. Ixx81 is set to zero if no special power-on phase position reading is desired, as is the case for an incremental encoder.

If Ixx81 is set to zero, a power-on phasing search routine is required for synchronous "fixed-field" brushless motors (permanent magnet, and switched reluctance); those that have a slip gain (Ixx78) of zero. Turbo PMAC's automatic phasing search routines based on Ixx73 and Ixx74 can be used, or a custom power-on PLC routine can be written.

Note:

Ixx81 is used for PMAC's commutation algorithms alone, to locate position within one electrical cycle of the motor. It is not used for any servo loop position information, even for power-up. Ixx10 and Ixx95 are used for that purpose.

Ixx91 tells how the data at the address specified by Ixx81 is to be interpreted. It also determines whether the location specified by Ixx81 is a multiplexer ("thumbwheel") port address, an address in Turbo PMAC's own memory and I/O space, or a MACRO node *number*.

Note:

It is easier to specify this parameter in hexadecimal form ($ prefix). If I9 is set to 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

R/D Converter Read: If Ixx91 contains a value from $000000 to $070000, Ixx81 contains the multiplexer port address of an ACC-8D Opt. 7 R/D-converter board. The value of Ixx81 matches the base address of the board (0 to 248 decimal, $0 to $F8 hex) on the port as set by its DIP switches. If the base address is 0, Ixx81 should be set to $100, because a value of 0 in Ixx81 disables the absolute read. The following table lists the possible values of Ixx81 in this mode.

Ixx81 for ACC-8D Opt. 7 Resolver/Digital Converter
(Ixx91=$000000 - $070000)
Addresses are Multiplexer Port Addresses

Board Mux. Addr.	Ixx81	Board Mux. Addr.	Ixx81	Board Mux. Addr.	Ixx81	Board Mux. Addr.	Ixx81
0	$000100	64	$000040	128	$000080	192	$0000C0
8	$000008	72	$000048	136	$000088	200	$0000C8
16	$000010	80	$000050	144	$000090	208	$0000D0
24	$000018	88	$000058	152	$000098	216	$0000D8
32	$000020	96	$000060	160	$0000A0	224	$0000E0
40	$000028	104	$000068	168	$0000A8	232	$0000E8
48	$000030	112	$000070	176	$0000B0	240	$0000F0
56	$000038	120	$000078	184	$0000B8	248	$0000F8

Parallel Data Read: If Ixx91 contains a value from $080000 to $180000 or from $480000 to $580000, Ixx81 specifies the address of a Turbo PMAC memory or I/O register where it will read the power-on phase position. Bits 16 to 21 of Ixx91, which can take a value of $08 to $18 (8 to 24) in this mode, specify the number of bits, starting at bit 0, of the register to be read for the absolute position.

Bit 22 of Ixx91 controls whether the address specified in Ixx81 is an X-register or a Y-register. If bit 22 of Ixx91 is set to 0, it is a Y-register. If bit 22 of Ixx91 is set to 1, it is an X-register.

There are four common sources of parallel data for absolute power-on phase position read. The first source is an ACC-14D/V parallel I/O board. ACC-14D/V boards map into Y-registers, so bit 22 of Ixx91 is set to 0. The settings of Ixx81 for each port of each possible ACC-14DV board are shown in the following table:

Ixx81 Values for ACC-14D/V Registers
(Ixx91=$080000 to $18000)

Register	Ixx81	Register	Ixx81
1st ACC-14D/V Port A	$078A00	4th ACC-14D/V Port A	$078D00
1st ACC-14D/V Port B	$078A01	4th ACC-14D/V Port B	$078D01
2nd ACC-14D/V Port A	$078B00	5th ACC-14D/V Port A	$078E00
2nd ACC-14D/V Port B	$078B01	5th ACC-14D/V Port B	$078E01
3rd ACC-14D/V Port A	$078C00	6th ACC-14D/V Port A	$078F00
3rd ACC-14D/V Port B	$078C01	6th ACC-14D/V Port B	$078F01

The second common source of parallel data for an absolute power-on phase position read is the encoder counter "phase position" register when an ACC-8D Option 9 Yaskawa Absolute Encoder converter board is used. This board synthesizes quadrature signals into the Turbo PMAC at power-on until the power-on position within one revolution is reached, so the value of the encoder counter can simply be read.

Encoder phase position counters map into X-registers, so bit 22 of Ixx91 is set to 1. The settings of Ixx81 for typical encoder registers on Turbo PMAC(1) and PMAC2 boards are shown in the following tables:

Turbo PMAC(1) Ixx81 Encoder Register Settings
(Ixx91=$480000 - $580000)

Encoder Register Channel #	PMAC	1st ACC-24P/V	2nd ACC-24P/V	3rd ACC-24P/V	4th ACC-24P/V
Channel 1	$078001	$078201	$079201	$07A201	$07B201
Channel 3	$078009	$078209	$079209	$07A209	$07B209
Channel 5	$078101	$078301	$079301	$07A301	$07B301
Channel 7	$078109	$078309	$079309	$07A309	$07B309

Turbo PMAC2 Ixx81 Typical Encoder Register Settings
(Ix91=$480000 - $580000)

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078001	$078009	$078011	$078019	1st IC on board PMAC2, 3U stack
1	$078101	$078109	$078011	$078019	2nd IC on board PMAC2, 3U stack
2	$078201	$078209	$078211	$078219	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078301	$078309	$078311	$078319	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079201	$079209	$079211	$079219	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079301	$079309	$079311	$079319	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A201	$07A209	$07A211	$07A219	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A301	$07A309	$07A311	$07A319	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B201	$07B209	$07B211	$07B219	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B301	$07B309	$07B311	$07B319	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

The third common source of parallel data for power-on phasing is the ACC-49 Sanyo Absolute Encoder Converter Board. The ACC-49 maps into Turbo PMAC's expansion port, at the addresses shown in the following table.

Ixx81 Values for ACC-49 Sanyo Absolute Encoder Converter (Ixx91=$0D0000)
Addresses are Turbo PMAC Memory-I/O Addresses

Enc. # on Board	Ixx10 for E1 ON	Ixx10 for E2 ON	Ixx10 for E3 ON	Enc. # on Board	Ixx10 for E4 ON	Ixx10 for E5 ON	Ixx10 for E6 ON
Enc. 1	$078A00	$078B00	$078C00	Enc. 3	$078D00	$078E00	$078F00
Enc. 2	$078A04	$078B04	$078C04	Enc. 4	$078D04	$078E04	$078F04

The fourth common source is a register in the 3U-format ACC-3E1 (for 3U Turbo Stack systems) or ACC-14E (for UMAC Turbo systems) board. In this case the last hex digit of Ixx91 must be set to a non-zero value to specify the byte-wide bus of these boards. The following tables show Ixx81 values for these boards.

Ixx81 Values for ACC-3E1 Registers in 3U Turbo Stack Systems
(Ixx91=$08000x to $18000x [unsigned], $88000x to $98000x [signed])

ACC-3E1 Address Jumper	E1	E2	E3	E4
Ixx81 Value	$07880x	$07890x	$078A0x	$078B0x

Ixx81 Values for ACC-14E Registers in UMAC Turbo Systems
(Ixx91=$08000x to $18000x [unsigned], $88000x to $98000x [signed])

DIP-Switch Setting	SW1-1 ON (0) SW1-2 ON (0)	SW1-1 OFF (1) SW1-2 ON (0)	SW1-1 ON (0) SW1-2 OFF (1)	SW1-1 OFF (1) SW1-2 OFF (1)
SW1-3 ON (0) SW1-4 ON (0)	$078C0x	$078D0x	$078E0x	$078F0x
SW1-3 OFF (1) SW1-4 ON (0)	$079C0x	$079D0x	$079E0x	$079F0x
SW1-3 ON (0) SW1-4 OFF (1)	$07AC0x	$07AD0x	$07AE0x	$07AF0x
SW1-3 OFF (1) SW1-4 OFF (1)	$07BC0x	$07BD0x	$07BE0x	$07BF0x
SW1-5 & 6 must be ON (0). ON means CLOSED; OFF means OPEN.				

The final digit, represented by an 'x' in both of these tables, can take a value of 0 to 5, depending on which I/O point on the board is used for the least significant bit (LSB):

Ixx10 Last Hex Digit 'x'	Pin Used for LSB	Pin Used for LSB	Pin Used for LSB
x=0	I/O00-07	I/O48-55	I/O96-103
x=1	I/O08-15	I/O56-63	I/O104-111
x=2	I/O16-23	I/O64-71	I/O112-119
x=3	I/O24-31	I/O72-79	I/O120-127
x=4	I/O32-39	I/O80-87	I/O128-135
x=5	I/O40-47	I/O88-95	I/O136-143

Hall Sensor Read: If Ixx91 contains a value from $800000 to $FF0000, Ixx81 specifies the address of a Turbo PMAC X-memory or I/O register where it will read the power-on phase position in bits 20, 21, and 22 of the register. It is expecting these 3 bits to be encoded as U, V, and W hall sensors with 120°e spacing. Typically, Ixx81 will contain the address of a flag register of a Servo IC.

Note:

Hall-style commutation sensors give only an approximate phase position, with a +/-30°e error. It is generally necessary to correct the phase position value at a known position such as the encoder's index pulse, either using the **SETPHASE** command or by writing directly into the phase position register (suggested M-variable Mxx71).

If the flag register is in a PMAC(1)-style Servo IC, the flags used are HMFLn, +LIMn, and -LIMn. Usually, the flag register is for the "spare" (even-numbered) set of flags corresponding to the second DAC output used for the commutation. The following table shows the values of Ixx81 used here.

Turbo PMAC(1) Ixx81 Typical Hall Phasing Settings
(Ixx91=$800000 - $FF0000)

Hall Flag Channel #	PMAC	1st ACC-24P/V	2nd ACC-24P/V	3rd ACC-24P/V	4th ACC-24P/V
Channel 2	$078004	$078204	$079204	$07A204	$07B204
Channel 4	$07800C	$07820C	$07920C	$07A20C	$07B20C
Channel 6	$078104	$078304	$079304	$07A304	$07B304
Channel 8	$07810C	$07830C	$07930C	$07A30C	$07B30C

If the flag register is in a PMAC2-style Servo IC, the flags used are CHUn, CHVn, and CHWn. Usually the flag register is the same register as used for the "main" flags as specified by Ixx25.

The following table shows the values of Ixx81 used here.

Turbo PMAC2 Ixx81 Typical Hall Phasing Settings
(Ix91=$800000 - $FF0000)

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$078000	$078008	$078010	$078018	1st IC on board PMAC2, 3U stack
1	$078100	$078108	$078010	$078018	2nd IC on board PMAC2, 3U stack
2	$078200	$078208	$078210	$078218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$078300	$078308	$078310	$078318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$079200	$079208	$079210	$079218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$079300	$079308	$079310	$079318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$07A200	$07A208	$07A210	$07A218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$07A300	$07A308	$07A310	$07A318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$07B200	$07B208	$07B210	$07B218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$07B300	$07B308	$07B310	$07B318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

If the flag register is obtained through the MACRO ring, Ixx81 will contain the address of a MACRO auxiliary image register in RAM. The following table shows the typical values of Ixx81 used here.

Turbo PMAC2 Ultralite Ixx81 Typical Hall Phasing Settings
(Ixx91=$800000 - $FF0000)

Ixx81	Value	Register	Ixx81	Value	Register
I181	$003440	MACRO Flag Register Set 0	I1781	$003460	MACRO Flag Register Set 32
I281	$003441	MACRO Flag Register Set 1	I1881	$003461	MACRO Flag Register Set 33
I381	$003444	MACRO Flag Register Set 4	I1981	$003464	MACRO Flag Register Set 36
I481	$003445	MACRO Flag Register Set 5	I2081	$003465	MACRO Flag Register Set 37
I581	$003448	MACRO Flag Register Set 8	I2181	$003468	MACRO Flag Register Set 40
I681	$003449	MACRO Flag Register Set 9	I2281	$003469	MACRO Flag Register Set 41
I781	$00344C	MACRO Flag Register Set 12	I2381	$00346C	MACRO Flag Register Set 44
I881	$00344D	MACRO Flag Register Set 13	I2481	$00346D	MACRO Flag Register Set 45
I981	$003450	MACRO Flag Register Set 16	I2581	$003470	MACRO Flag Register Set 48
I1081	$003451	MACRO Flag Register Set 17	I2681	$003471	MACRO Flag Register Set 49
I1181	$003454	MACRO Flag Register Set 20	I2781	$003474	MACRO Flag Register Set 52
I1281	$003455	MACRO Flag Register Set 21	I2881	$003475	MACRO Flag Register Set 53
I1381	$003458	MACRO Flag Register Set 24	I2981	$003478	MACRO Flag Register Set 56
I1481	$003459	MACRO Flag Register Set 25	I3081	$003479	MACRO Flag Register Set 57
I1581	$00345C	MACRO Flag Register Set 28	I3181	$00347C	MACRO Flag Register Set 60
I1681	$00345D	MACRO Flag Register Set 29	I3281	$00347D	MACRO Flag Register Set 61

Because phase position needs only to be known within a single revolution, any geared-down secondary absolute sensors are not relevant for this purpose. They may still be used for power-on position information for the servo loop, with Ixx10, Ixx99, and Ixx98

In general, the zero position of the absolute sensor will not be the same as the zero position of the commutation cycle. Parameter Ixx75 is used to hold the offset between these two reference positions.

MACRO Absolute Position Read: If Ixx91 contains a value from $720000 to $740000, the value specified in Ixx81 is a MACRO node number, and Turbo PMAC will obtain the absolute power-on position through the MACRO ring. Ixx91 specifies what type of position data is used.

The MACRO node number is specified in the last two hex digits of Ixx81. The second-to-last digit specifies the MACRO IC number 0 to 3 (1, 2, and 3 exist only on Ultralite versions of the Turbo PMAC2, or on UMAC Turbo systems with ACC-5E).

Note that the MACRO IC number on the Turbo PMAC does not necessarily match the ring master number for that IC, although it often will. The last digit specifies the MACRO node number 0 to 15 (0 to F hex) in that IC. This function is only supported in nodes 0, 1, 4, 5, 8, 9, 12 (C), and 13 (D).

The following table shows the required values of Ixx81 for all of the MACRO nodes that can be used. Note that MACRO IC 0 Node 0 uses an Ixx81 value of $000100, because Ixx81=0 disables the absolute position read function.

Ixx81 for MACRO Absolute Position Reads
(Ixx91=$720000 - $740000)
Addresses are MACRO Node Numbers

MACRO Node Number	Ixx81 for MACRO IC 0	Ixx81 for MACRO IC 1	Ixx81 for MACRO IC 2	Ixx81 for MACRO IC 3
0	$000100	$000010	$000020	$000030
1	$000001	$000011	$000021	$000031
4	$000004	$000014	$000024	$000034
5	$000005	$000015	$000025	$000035
8	$000008	$000018	$000028	$000038
9	$000009	$000019	$000029	$000039
12	$00000C	$00001C	$00002C	$00003C
13	$00000D	$00001D	$00002D	$00003D

If obtaining the absolute position through a Delta Tau MACRO Station or equivalent, MACRO Station setup variable MI11x for the matching node must be set properly to obtain the type of information desired.

Ixx82 Motor xx Current-Loop Feedback Address

Range: $000000 – $FFFFFF

Units: Turbo PMAC Y-addresses

Default: $0

Ixx82 tells Turbo PMAC which addresses to read to get its current feedback values for Motor xx if Turbo PMAC is closing the current loop for this motor. Turbo PMAC must be performing the commutation for the motor (Ixx01=1) if it is to close the current loop as well.

A zero value for Ixx82 tells PMAC not to close the current loop for this motor. In this case, PMAC outputs either one velocity or torque command value (Ixx01 bit 0 = 0), or two phase-current command values (Ixx01 bit 0 = 1), usually represented as analog voltages.

A non-zero value for Ixx82 automatically triggers current loop execution in the phase interrupt using the current values found in the registers specified by Ixx82. Typically, these registers are analog-to-digital converter (ADC) registers in a PMAC2-style Servo IC, or MACRO feedback registers containing copies of ADC registers in a MACRO Station.

Digital current loop closure on the Turbo PMAC requires a set of three consecutive command output registers. Generally, this requires writing to either a PMAC2-style Servo IC or a MACRO IC.

When Ixx01 is set to 1, Turbo PMAC performs the phase commutation for this motor, computing two phase current commands based on the position/velocity servo command and the magnetization current value. If Ixx82>0, these commands are compared to the two actual current values read from the address specified by Ixx82, and the next *lower* address. It executes a PI filter on the current loops and outputs three voltage command values to the address specified by Ixx02 and the next two higher addresses. These are typically the PWM commands for the three half-bridges of a brushless motor power stage.

When the digital current loop is used for drives connected directly to the Turbo PMAC2, the typical values for Ixx82 are:

Turbo PMAC2 Ixx82 Typical Settings

Ixx82	Value	Register	Ixx82	Value	Register
I182	$078006	PMAC2 ADC1B	I1782	$079206	2nd ACC-24x2 ADC1B
I282	$07800E	PMAC2 ADC2B	I1882	$07920E	2nd ACC-24x2 ADC2B
I382	$078016	PMAC2 ADC3B	I1982	$079216	2nd ACC-24x2 ADC3B
I482	$07801E	PMAC2 ADC4B	I2082	$07921E	2nd ACC-24x2 ADC4B
I582	$078106	PMAC2 ADC5B	I2182	$079306	2nd ACC-24x2 ADC5B
I682	$07810E	PMAC2 ADC6B	I2282	$07930E	2nd ACC-24x2 ADC6B
I782	$078116	PMAC2 ADC7B	I2382	$079316	2nd ACC-24x2 ADC7B
I882	$07811E	PMAC2 ADC8B	I2482	$07931E	2nd ACC-24x2 ADC8B
I982	$078206	1st ACC-24x2 ADC1B	I2582	$07A206	3rd ACC-24x2 ADC1B
I1082	$07820E	1st ACC-24x2 ADC2B	I2682	$07A20E	3rd ACC-24x2 ADC2B
I1182	$078216	1st ACC-24x2 ADC3B	I2782	$07A216	3rd ACC-24x2 ADC3B
I1282	$07821E	1st ACC-24x2 ADC4B	I2882	$07A21E	3rd ACC-24x2 ADC4B
I1382	$078306	1st ACC-24x2 ADC5B	I2982	$07A306	3rd ACC-24x2 ADC5B
I1482	$07830E	1st ACC-24x2 ADC6B	I3082	$07A30E	3rd ACC-24x2 ADC6B
I1582	$078316	1st ACC-24x2 ADC7B	I3182	$07A316	3rd ACC-24x2 ADC7B
I1682	$07831E	1st ACC-24x2 ADC8B	I3282	$07A31E	3rd ACC-24x2 ADC8B

When the digital current loop is used for drives connected to the Turbo PMAC2 Ultralite through a MACRO station, the typical values for Ixx82 are:

Turbo PMAC2 Ultralite Ixx82 Typical Settings

Ixx82	Value	Register	Ixx82	Value	Register
I182	$078422	MACRO IC 0 Node 0 Reg. 2	I1782	$07A422	MACRO IC 2 Node 0 Reg. 2
I282	$078426	MACRO IC 0 Node 1 Reg. 2	I1882	$07A426	MACRO IC 2 Node 1 Reg. 2
I382	$07842A	MACRO IC 0 Node 4 Reg. 2	I1982	$07A42A	MACRO IC 2 Node 4 Reg. 2
I482	$07842E	MACRO IC 0 Node 5 Reg. 2	I2082	$07A42E	MACRO IC 2 Node 5 Reg. 2
I582	$078432	MACRO IC 0 Node 8 Reg. 2	I2182	$07A432	MACRO IC 2 Node 8 Reg. 2
I682	$078436	MACRO IC 0 Node 9 Reg. 2	I2282	$07A436	MACRO IC 2 Node 9 Reg. 2
I782	$07843A	MACRO IC 0 Node 12 Reg. 2	I2382	$07A43A	MACRO IC 2 Node 12 Reg. 2
I882	$07843E	MACRO IC 0 Node 13 Reg. 2	I2482	$07A43E	MACRO IC 2 Node 13 Reg. 2
I982	$079422	MACRO IC 1 Node 0 Reg. 2	I2582	$07B422	MACRO IC 3 Node 0 Reg. 2
I1082	$079426	MACRO IC 1 Node 1 Reg. 2	I2682	$07B426	MACRO IC 3 Node 1 Reg. 2
I1182	$07942A	MACRO IC 1 Node 4 Reg. 2	I2782	$07B42A	MACRO IC 3 Node 4 Reg. 2
I1282	$07942E	MACRO IC 1 Node 5 Reg. 2	I2882	$07B42E	MACRO IC 3 Node 5 Reg. 2
I1382	$079432	MACRO IC 1 Node 8 Reg. 2	I2982	$07B432	MACRO IC 3 Node 8 Reg. 2
I1482	$079436	MACRO IC 1 Node 9 Reg. 2	I3082	$07B436	MACRO IC 3 Node 9 Reg. 2
I1582	$07943A	MACRO IC 1 Node 12 Reg. 2	I3182	$07B43A	MACRO IC 3 Node 12 Reg. 2
I1682	$07943E	MACRO IC 1 Node 13 Reg. 2	I3282	$07B43E	MACRO IC 3 Node 13 Reg. 2

UMAC Turbo Ixx82 Typical Settings

Ixx82	Value	Register	Ixx82	Value	Register
I182	$078206	1st ACC-24E2 ADC1B	I1782	$07A206	5th ACC-24E2 ADC1B
I282	$07820E	1st ACC-24E2 ADC2B	I1882	$07A20E	5th ACC-24E2 ADC2B
I382	$078216	1st ACC-24E2 ADC3B	I1982	$07A216	5th ACC-24E2 ADC3B
I482	$07821E	1st ACC-24E2 ADC4B	I2082	$07A21E	5th ACC-24E2 ADC4B
I582	$078306	2nd ACC-24E2 ADC1B	I2182	$07A306	6th ACC-24E2 ADC1B
I682	$07830E	2nd ACC-24E2 ADC2B	I2282	$07A30E	6th ACC-24E2 ADC2B
I782	$078316	2nd ACC-24E2 ADC3B	I2382	$07A316	6th ACC-24E2 ADC3B
I882	$07831E	2nd ACC-24E2 ADC4B	I2482	$07A31E	6th ACC-24E2 ADC4B
I982	$079206	3rd ACC-24E2 ADC1B	I2582	$07B206	7th ACC-24E2 ADC1B
I1082	$07920E	3rd ACC-24E2 ADC2B	I2682	$07B20E	7th ACC-24E2 ADC2B
I1182	$079216	3rd ACC-24E2 ADC3B	I2782	$07B216	7th ACC-24E2 ADC3B
I1282	$07921E	3rd ACC-24E2 ADC4B	I2882	$07B21E	7th ACC-24E2 ADC4B
I1382	$079306	4th ACC-24E2 ADC1B	I2982	$07B306	8th ACC-24E2 ADC1B
I1482	$07930E	4th ACC-24E2 ADC2B	I3082	$07B30E	8th ACC-24E2 ADC2B
I1582	$079316	4th ACC-24E2 ADC3B	I3182	$07B316	8th ACC-24E2 ADC3B
I1682	$07931E	4th ACC-24E2 ADC4B	I3282	$07B31E	8th ACC-24E2 ADC4B

If Ixx82>0, the following variables must be set properly for correct operation of the digital current loop:

- Ixx61: Current-Loop Integral Gain
- Ixx62: Current-Loop Forward-Path Proportional Gain
- Ixx66: PWM Scale Factor
- Ixx72: Commutation Phase Angle
- Ixx76: Current-Loop Back-Path Proportional Gain
- Ixx84: Current-Loop Feedback Mask Word

Ixx83 Motor xx Commutation Position Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC addresses

Default values:

Turbo PMAC(1) Ixx83 Defaults

Ixx83	Value	Register	Ixx83	Value	Register
I183	$078001	PMAC Encoder 1	I1783	$079201	2nd ACC-24P/V Encoder 1
I283	$078005	PMAC Encoder 2	I1883	$079205	2nd ACC-24P/V Encoder 2
I383	$078009	PMAC Encoder 3	I1983	$079209	2nd ACC-24P/V Encoder 3
I483	$07800D	PMAC Encoder 4	I2083	$07920D	2nd ACC-24P/V Encoder 4
I583	$078101	PMAC Encoder 5	I2183	$079301	2nd ACC-24P/V Encoder 5
I683	$078105	PMAC Encoder 6	I2283	$079305	2nd ACC-24P/V Encoder 6
I783	$078109	PMAC Encoder 7	I2383	$079309	2nd ACC-24P/V Encoder 7
I883	$07810D	PMAC Encoder 8	I2483	$07930D	2nd ACC-24P/V Encoder 8
I983	$078201	1st ACC-24P/V Encoder 1	I2583	$07A201	3rd ACC-24P/V Encoder 1
I1083	$078205	1st ACC-24P/V Encoder 2	I2683	$07A205	3rd ACC-24P/V Encoder 2
I1183	$078209	1st ACC-24P/V Encoder 3	I2783	$07A209	3rd ACC-24P/V Encoder 3
I1283	$07820D	1st ACC-24P/V Encoder 4	I2883	$07A20D	3rd ACC-24P/V Encoder 4
I1383	$078301	1st ACC-24P/V Encoder 5	I2983	$07A301	3rd ACC-24P/V Encoder 5
I1483	$078305	1st ACC-24P/V Encoder 6	I3083	$07A305	3rd ACC-24P/V Encoder 6
I1583	$078309	1st ACC-24P/V Encoder 7	I3183	$07A309	3rd ACC-24P/V Encoder 7
I1683	$07830D	1st ACC-24P/V Encoder 8	I3283	$07A30D	3rd ACC-24P/V Encoder 8

Turbo PMAC2 (Non-Ultralite) Ixx83 Defaults

Ixx83	Value	Register	Ixx83	Value	Register
I183	$078001	PMAC2 Encoder 1	I1783	$079201	2nd ACC-24P/V2 Encoder 1
I283	$078009	PMAC2 Encoder 2	I1883	$079209	2nd ACC-24P/V2 Encoder 2
I383	$078011	PMAC2 Encoder 3	I1983	$079211	2nd ACC-24P/V2 Encoder 3
I483	$078019	PMAC2 Encoder 4	I2083	$079219	2nd ACC-24P/V2 Encoder 4
I583	$078101	PMAC2 Encoder 5	I2183	$079301	2nd ACC-24P/V2 Encoder 5
I683	$078109	PMAC2 Encoder 6	I2283	$079309	2nd ACC-24P/V2 Encoder 6
I783	$078111	PMAC2 Encoder 7	I2383	$079311	2nd ACC-24P/V2 Encoder 7
I883	$078119	PMAC2 Encoder 8	I2483	$079319	2nd ACC-24P/V2 Encoder 8
I983	$078201	1st ACC-24P/V2 Encoder 1	I2583	$07A201	3rd ACC-24P/V2 Encoder 1
I1083	$078209	1st ACC-24P/V2 Encoder 2	I2683	$07A209	3rd ACC-24P/V2 Encoder 2
I1183	$078211	1st ACC-24P/V2 Encoder 3	I2783	$07A211	3rd ACC-24P/V2 Encoder 3
I1283	$078219	1st ACC-24P/V2 Encoder 4	I2883	$07A219	3rd ACC-24P/V2 Encoder 4
I1383	$078301	1st ACC-24P/V2 Encoder 5	I2983	$07A301	3rd ACC-24P/V2 Encoder 5
I1483	$078309	1st ACC-24P/V2 Encoder 6	I3083	$07A309	3rd ACC-24P/V2 Encoder 6
I1583	$078311	1st ACC-24P/V2 Encoder 7	I3183	$07A311	3rd ACC-24P/V2 Encoder 7
I1683	$078319	1st ACC-24P/V2 Encoder 8	I3283	$07A319	3rd ACC-24P/V2 Encoder 8

Turbo PMAC2 Ultralite Ixx83 Defaults

Ixx83	Value	Register	Ixx83	Value	Register
I183	$078420	MACRO IC 0 Node 0 Reg. 0	I1783	$07A420	MACRO IC 2 Node 0 Reg. 0
I283	$078424	MACRO IC 0 Node 1 Reg. 0	I1883	$07A424	MACRO IC 2 Node 1 Reg. 0
I383	$078428	MACRO IC 0 Node 4 Reg. 0	I1983	$07A428	MACRO IC 2 Node 4 Reg. 0
I483	$07842C	MACRO IC 0 Node 5 Reg. 0	I2083	$07A42C	MACRO IC 2 Node 5 Reg. 0
I583	$078430	MACRO IC 0 Node 8 Reg. 0	I2183	$07A430	MACRO IC 2 Node 8 Reg. 0
I683	$078434	MACRO IC 0 Node 9 Reg. 0	I2283	$07A434	MACRO IC 2 Node 9 Reg. 0
I783	$078438	MACRO IC 0 Node 12 Reg. 0	I2383	$07A438	MACRO IC 2 Node 12 Reg. 0
I883	$07843C	MACRO IC 0 Node 13 Reg. 0	I2483	$07A43C	MACRO IC 2 Node 13 Reg. 0
I983	$079420	MACRO IC 1 Node 0 Reg. 0	I2583	$07B420	MACRO IC 3 Node 0 Reg. 0
I1083	$079424	MACRO IC 1 Node 1 Reg. 0	I2683	$07B424	MACRO IC 3 Node 1 Reg. 0
I1183	$079428	MACRO IC 1 Node 4 Reg. 0	I2783	$07B428	MACRO IC 3 Node 4 Reg. 0
I1283	$07942C	MACRO IC 1 Node 5 Reg. 0	I2883	$07B42C	MACRO IC 3 Node 5 Reg. 0
I1383	$079430	MACRO IC 1 Node 8 Reg. 0	I2983	$07B430	MACRO IC 3 Node 8 Reg. 0
I1483	$079434	MACRO IC 1 Node 9 Reg. 0	I3083	$07B434	MACRO IC 3 Node 9 Reg. 0
I1583	$079438	MACRO IC 1 Node 12 Reg. 0	I3183	$07B438	MACRO IC 3 Node 12 Reg. 0
I1683	$07943C	MACRO IC 1 Node 13 Reg. 0	I3283	$07B43C	MACRO IC 3 Node 13 Reg. 0

UMAC Turbo Ixx83 Defaults

Ixx82	Value	Register	Ixx82	Value	Register
I182	$078201	1st ACC-24E2 Encoder 1	I1782	$07A201	5th ACC-24E2 Encoder 1
I282	$078209	1st ACC-24E2 Encoder 2	I1882	$07A209	5th ACC-24E2 Encoder 2
I382	$078211	1st ACC-24E2 Encoder 3	I1982	$07A211	5th ACC-24E2 Encoder 3
I482	$078219	1st ACC-24E2 Encoder 4	I2082	$07A219	5th ACC-24E2 Encoder 4
I582	$078301	2nd ACC-24E2 Encoder 1	I2182	$07A301	6th ACC-24E2 Encoder 1
I682	$078309	2nd ACC-24E2 Encoder 2	I2282	$07A309	6th ACC-24E2 Encoder 2
I782	$078311	2nd ACC-24E2 Encoder 3	I2382	$07A311	6th ACC-24E2 Encoder 3
I882	$078319	2nd ACC-24E2 Encoder 4	I2482	$07A319	6th ACC-24E2 Encoder 4
I982	$079201	3rd ACC-24E2 Encoder 1	I2582	$07B201	7th ACC-24E2 Encoder 1
I1082	$079209	3rd ACC-24E2 Encoder 2	I2682	$07B209	7th ACC-24E2 Encoder 2
I1182	$079211	3rd ACC-24E2 Encoder 3	I2782	$07B211	7th ACC-24E2 Encoder 3
I1282	$079219	3rd ACC-24E2 Encoder 4	I2882	$07B219	7th ACC-24E2 Encoder 4
I1382	$079301	4th ACC-24E2 Encoder 1	I2982	$07B301	8th ACC-24E2 Encoder 1
I1482	$079309	4th ACC-24E2 Encoder 2	I3082	$07B309	8th ACC-24E2 Encoder 2
I1582	$079311	4th ACC-24E2 Encoder 3	I3182	$07B311	8th ACC-24E2 Encoder 3
I1682	$079319	4th ACC-24E2 Encoder 4	I3282	$07B319	8th ACC-24E2 Encoder 4

For a motor commutated by Turbo PMAC (Ixx01 = 1 or 3), Ixx83 tells Turbo PMAC where to read its commutation (phasing) position information for Motor xx every commutation cycle. This can be a different address from that used for power-on/reset phasing position, which is determined by Ixx81. If Turbo PMAC is not commutating Motor xx (Ixx01 = 0 or 2), Ixx83 is not used.

Ixx83 contains the address of the register to be read. If Ixx01 bit 1 is set to 0 (Ixx01 = 1), the register is the X-register at that address. If Ixx01 bit 1 is set to 1 (Ixx01 = 3), the register is the Y-register at that address.

For Turbo PMAC boards with on-board encoder circuitry, Ixx83 typically contains the address of the "phase position" encoder register for encoder *x*; this is the default. Since these registers have 'X' addresses, Ixx01 is set to 1.

For Turbo PMAC2 Ultralite boards, Ixx83 typically contains the address of a MACRO node's position feedback register; this is the default. Since PMAC2 can only commutate over MACRO using nodes with 'Y' addresses, Ixx01 is set to 3 in these cases.

Ixx84 Motor xx Current-Loop Feedback Mask Word

Range: $000000 - $FFFFFF

Units: Bit mask

Default: $FFF000 (12-bit ADCs)

Ixx84 tells Turbo PMAC what bits of the 24-bit current feedback word(s) to use as actual the actual current value in the current loop equations. It is only used if Ixx82>0, enabling current loop closure in Motor xx of the Turbo PMAC.

Turbo PMAC supports interface to serial analog-to-digital converters of many resolutions through a PMAC2-style "DSPGATE1" Servo IC, either on the PMAC, on an ACC-24 axis expansion board, or at a remote MACRO node. The data is received in 18-bit shift registers in the ASIC, which are read as the high end of a 24-bit word, with the number "left-justified" to the most significant bit.

Ixx84 specifies a 24-bit mask word that is combined with the feedback word through a logical AND operation to produce the value that is used in the current loop equations. There should be a 1 in every bit that is used, and a 0 in every bit that is not. Since the data is left justified, Ixx84 should start with 1s and end with 0s. Usually Ixx84 is represented as a hexadecimal number, with 4 bits per digit, and a total of six digits

Some amplifiers will transmit status and fault information on the end of the serial data stream for the ADC, and it is important to mask out these values from the current loop equations.

Examples:
For a 10-bit ADC: Ixx84=$FFC000
For a 12-bit ADC: Ixx84=$FFF000
For a 16-bit ADC: Ixx84=$FFFF00

Further Motor I-Variables

Ixx85 Motor xx Backlash Take-up Rate

Range: 0 - 8,388,607

Units: 1/16 count / background cycle

Default: 0

Ixx85 determines how fast backlash is "taken up" on direction reversal. The size of the backlash is determined by Ixx86, and possibly by the backlash compensation table for the motor. Turbo PMAC will "take up" the backlash at the Ixx85 rate whenever the commanded or Master Handwheel position for the motor reverses by more than the amount set by Ixx87 the backlash hysteresis parameter. If Ixx85 is zero, backlash is effectively disabled. Ixx85 is usually set interactively and experimentally to as high a value as possible without creating dynamic problems.

Ixx86 Motor xx Backlash Size

Range: 0 - 8,388,607

Units: 1/16 count

Default: 0

Ixx86 allows PMAC to compensate for backlash in the motor's coupling by adding or subtracting (depending on the new direction) the amount specified in the parameter to the commanded position on direction reversals (this offset will not appear when position is queried or displayed). A value of zero means no backlash. The rate at which this backlash is added or subtracted ("taken up") is determined by Ixx85.

Variable Ixx87, Backlash Hysteresis, determines the amount of reversal in desired position that is required before backlash will start to be introduced or removed.

If backlash tables are used, Ixx86 represents the backlash at motor zero position; values in the table should represent the difference between the backlash at a given position and Ixx86.

Note:

The units of this parameter are 1/16 of a count so the value should be 16 times the number of counts of backlash compensation desired.

Example:
If you find that you have a backlash on reversal of motor direction of 7.5 encoder counts, you would set Ixx86 to 7.5 * 16 = 120.

Ixx87 Motor xx Backlash Hysteresis

Range: 0 - 8,388,607

Units: 1/16 count

Default: 64 (= 4 counts)

Ixx87 controls the size of the direction reversal in motor commanded position that must occur on Motor xx before Turbo PMAC starts to add the programmed backlash (Ixx86) in the direction of motion. The purpose of this variable is to allow the customer to ensure that a very small direction reversal (e.g. from the dithering of a master encoder) does not cause the backlash to "kick in". Ixx87 thus provides a hysteresis in the backlash function.

The units of Ixx87 are 1/16 of a count. Therefore, this parameter must hold a value 16 times larger than the number of counts reversal at which backlash is introduced. For example, if backlash is to be introduced after 5 counts of reversal, Ixx87 should be set to 80.

Example:
With a system in which one count of the master encoder creates 10 counts of movement in the slave motor, it is desired that a single count reversal of the master not trigger backlash reversal. Therefore, the backlash hysteresis is set to 15 counts, and Ixx87 is set to 15*16=240.

Ixx88 Motor xx In-Position Number of Scans

Range: 0 - 255

Units: Background computation cycles (minus one)

Default: 0

Ixx88 permits the user to define the number of consecutive scans that Turbo PMAC Motor xx must satisfy all "in-position" conditions before the motor in-position status bit is set true. This permits the user to ensure that the motor is truly settled in the end position before executing the next operation, on or off Turbo PMAC. The number of consecutive scans required is equal to Ixx88 + 1.

Turbo PMAC scans for the in-position condition of each active motor during the "housekeeping" part of every background cycle, which occurs between each scan of each enabled background PLC (PLC 1-31). All motors in a coordinate system must have true in-position bits for the coordinate-system in-position bit to be set true.

In non-Turbo PMACs, this function is controlled by global I-variable I7.

Ixx90 Motor xx Rapid Mode Speed Select

Range: 0 - 1

Units: None

Default: 1

Ixx90 determines which variable is used for the speed of a **RAPID** mode move. When Ixx90 is set to 0, the jog speed parameter Ixx22 is used. When Ixx90 is set to the default of 1, the maximum program speed parameter Ixx16 is used. Regardless of the setting of Ixx90, the jog acceleration parameters Ixx19 - Ixx21 control the acceleration and deceleration of a **RAPID** mode move.

In non-Turbo PMACs, this function is controlled by global I-variable I50.

Ixx91 Motor xx Power-On Phase Position Format

Range: $000000 - $FFFFFF

Units: None

Default: 0

Ixx91 specifies how the power-on phase-position data, if any, for Motor xx is interpreted. Ixx81 specifies the address of the register containing this position data; Ixx91 controls how that data is read. This permits the use of a wide variety of absolute position sensors with the Turbo PMAC.

Ixx91 is used only on power-on/reset or on the **$** or **$$** on-line reset commands. To get a new value of Ixx91 to take effect, the **$** or **$$** command must be issued, or the value of Ixx91 must be stored to non-volatile flash memory with the **SAVE** command, and the board must be reset.

Ixx91 is a 24-bit value; currently only bits 16-23, which comprise the first two of six hex digits, are used. Ixx91 is only used if Ixx81 is set to a non-zero value.

The possible values of Ixx91 and the position sources they specify are summarized in the following table:

Ixx91 Value Range	Absolute Position Source	Ixx81 Address Type
$000000 - $070000	ACC-8D Opt 7 R/D Converter	Multiplexer Port
$080000 - $180000	Parallel Data Y-Register	Turbo PMAC Memory-I/O
$480000 - $580000	Parallel Data X-Register	Turbo PMAC Memory-I/O
$730000	MACRO Station R/D Converter	MACRO Node Number
$740000	MACRO Station Parallel Read	MACRO Node Number
$800000 - $FF0000	Hall Sensor Read	Turbo PMAC Memory-I/O

R/D Converter: If Ixx91 contains a value from $000000 to $070000, Motor xx will expect its absolute power-on phase position from an ACC-8D Opt. 7 R/D converter board. Ixx81 should contain the address of the board on the multiplexer port, as set by the DIP switches on the board.

The second hex digit of Ixx91, which can take a value from 0 to 7 in this mode, specifies the number of the individual R/D converter at that multiplexer port address. This is a function of the DIP switch setting on the board and the location of the converter on the board, as specified in the following table:

Ixx91 Value	ACC-8D Opt. 7 SW1-1 Setting	# of R/D Converter on ACC-8D Opt. 7
$000000	CLOSED (0)	1
$010000	CLOSED (0)	2
$020000	CLOSED (0)	3
$030000	CLOSED (0)	4
$040000	OPEN (1)	1
$050000	OPEN (1)	2
$060000	OPEN (1)	3
$070000	OPEN (1)	4

Parallel Data Read: If Ixx91 contains a value from $08000n to $18000n, or from $48000n to $58000n, Motor xx will do a parallel data read of the Turbo PMAC memory or I/O register at the address specified by Ixx81.

In this mode, bits 16 to 21 specify the number of bits to be read. If the last hex digit of Ixx91 is 0, consecutive bits will be read from the address specified by Ixx81, with the least significant bit read from bit 0. This format is used for registers and I/O devices with 24-bit interfaces.

If the last hex digit of Ixx91 is 4, 5, or 6, data will be read in byte-wide pieces, with the least significant byte at the address specified in Ixx81, the next byte at one address higher, and the next byte (if used) at one more address higher. This format is intended for getting parallel data from the ACC-3E 3U-format stack I/O board or the ACC-14E 3U-format pack (UMAC) I/O board, which have byte-wide interfaces. For this format, the last hex digit of Ixx91 determines which byte of the 24-bit word is used, according to the following table:

Ixx91 Last Digit	Byte	Bits
4	Low	0 – 7
5	Middle	8 – 15
6	High	16 – 23

In this mode, bit 22 of Ixx91 specifies whether a Y-register is to be read, or an X-register. A value of 0 in this bit, yielding Ixx91 values from $080000 to $180000, specifies a Y-register; a value of 1, yielding Ixx91 values from $480000 to $580000, specifies an X-register.

For the ACC-8D Option 9 Yaskawa Absolute Encoder Converter, Turbo PMAC's 24-bit encoder phase position register, an X-register, is read, so Ixx91 is set to $580000 ($180000 + $400000).

For the ACC-49 Sanyo Absolute Encoder Converter, the encoder provides a 13-bit value within one motor revolution, and the data is read from a Y-register, so Ixx91 is set to $0D0000.

Example: If Ixx81=$078D01 and Ixx91=$140000, Turbo PMAC would read 20 bits (bits 0 – 19) from Y:$078D01.

Example: If Ixx81=$078C00 and Ixx91=$100004, Turbo PMAC would read 16 bits, with the low 8 bits from the low byte of Y:$078C00, and the high 8 bits from the low byte of Y:$078C01.

Example: If Ixx81=$079E03 and Ixx91=$120005, Turbo PMAC would read 18 bits, with the low 8 bits from the middle byte of Y:$079E03, and the next 8 bits from the middle byte of Y:$079E04, and the high 2 bits from the first 2 bits of the middle byte of Y:$079E05.

MACRO R/D Read: If Ixx91 contains a value of $730000, Motor xx will read the absolute phase position from an ACC-8D Opt. 7 Resolver-to-Digital Converter through a MACRO Station or compatible device.

In this mode, Ixx81 specifies the MACRO node number. MACRO Station setup variable MI11x for the matching node must be set to read the R/D converter.

MACRO Parallel Read: If Ixx91 contains a value of $740000, Motor xx will read the absolute phase position from a parallel data source through a MACRO Station or compatible device.

In this mode, Ixx81 specifies the MACRO node number. MACRO Station setup variable MI11x for the matching node must be set to read the parallel data source.

Hall Sensor Read: If Ixx91 contains a value from $800000 to $FF0000 (bit 23 set to 1), Motor xx will read bits 20 through 22 of the Turbo PMAC memory or I/O register at the address specified by Ixx81. It will expect these three bits to be encoded as the U, V, and W "hall-effect" commutation signals with 120°e spacing for the absolute power-on phase position. In this mode, the address specified in Ixx81 is usually that of a flag register.

Note:

Hall-style commutation sensors give only an approximate phase position, with a +/-30°e error. It is generally necessary to correct the phase position value at a known position such as the encoder's index pulse, either using the **SETPHASE** command or by writing directly into the phase position register (suggested M-variable Mxx71).

If the flag register is in a PMAC(1)-style Servo IC, the flag inputs for bits 20, 21, and 22, representing W, V, and U, are +LIMn, -LIMn, and HMFLn, respectively. In a typical application, Ixx81 specifies that these inputs be used from the "spare" flag register matching the second DAC channel used for commutation.

If the flag register is in a PMAC2-style Servo IC, the input flags for bits 20, 21, and 22, representing W, V, and U, are CHWn, CHVn, and CHUn, respectively. In a typical application, these inputs are used from the same flag register addressed by Ixx25 for the main flags.

In this mode, bit 22 of Ixx91 allows for reversal of the sense of the hall-effect sensors. If W (bit 20 of the register; HMFLn or CHWn) leads V (bit 21; -LIMn or CHVn), and V leads U (bit 22; +LIMn or CHUn) as the commutation cycle counts up, then bit 22 of Ixx91 should be set to 0. If U leads V and V leads W as the commutation cycle counts up, then bit 22 of Ixx91 should be set to 1.

In this mode, bits 16 to 21 of Ixx91 together form an offset value from 0 to 63 representing the difference between PMAC's commutation cycle zero and the hall-effect sensor zero position, which is defined as the transition of the V signal when U is low. This offset has units of 1/64 of a commutation cycle, or 5.625°e. Typically, one of the transitions will be at PMAC's commutation zero point, so the desired offset values will be 0°, 60°, 120°, 180°, 240°, and 300°, approximated by values of 0, 11($0B), 21($15), 32($20), 43($2B), and 53($35).

This operation can handle hall-effect sensors separated by 120°e. The following table gives the Ixx91 settings for bits 16 to 23 for all of the common cases of hall-effect settings as they relate to the PMAC commutation cycle.

Ixx91 Values for UVW Hall States (120°e Spacing)

0 to 60 deg	60 to 120deg	120 to 180 deg	180 to -120 deg	-120 to -60 deg	-60 to 0 deg	Ixx91
011	010	110	100	101	001	$800000
001	011	010	110	100	101	$8B0000
101	001	011	010	110	100	$950000
100	101	001	011	010	110	$A00000
110	100	101	001	011	010	$AB0000
010	110	100	101	001	011	$B50000
001	101	100	110	010	011	$C00000
011	001	101	100	110	010	$CB0000
010	011	001	101	100	110	$D50000
110	010	011	001	101	100	$E00000
100	110	010	011	001	101	$EB0000
101	100	110	010	011	001	$F50000

Ixx92 Motor xx Jog Move Calculation Time

Range: 1 - 8,388,607

Units: msec

Default: 10

Ixx92 controls how much time is allotted to calculate an on-line jog move, a homing search move, or a motion-program **RAPID**-mode move for Motor xx. It also determines the delay in the trajectory's reaction to an altered destination or the trigger condition in any type of move-until-trigger: a homing search move, an on-line jog-until-trigger, or a motion-program **RAPID**-mode move-until-trigger. If the motor is sitting still at the beginning of this time, it will continue to sit for this time. If it is executing a trajectory, it will continue on the present trajectory for this time before changing to the trajectory of the new command or post-trigger move.

This parameter should rarely need to be changed from the default of 10 msec. It should not be set to 0 for any reason, or PMAC will not be able to perform any of these types of moves. The minimum practical value for this parameter is 2 or 3.

In non-Turbo PMACs, this function is controlled by global I-variable I12.

Ixx95 Motor xx Power-On Servo Position Format

Range: $000000 - $FFFFFF

Units: none

Default: $000000

Ixx95 specifies how the absolute power-on servo-position data, if any, for Motor xx is interpreted. Ixx10 specifies the address of the register containing this position data; Ixx95 controls how that data is read. This permits the use of a wide variety of absolute position sensors with the Turbo PMAC.

Ixx95 is used only on power-on/reset or on the **$*** or **$$*** command. To get a new value of Ixx95 to take effect, either the **$*** or **$$*** command must be issued, or the value must be stored to non-volatile flash memory with the **SAVE** command, and the board must be reset.

Ixx95 is a 24-bit value; currently bits 16-23, which comprise the first two of six hex digits, are used. Ixx95 is only used if Ixx10 is set to a non-zero value.

The possible values of Ixx95 and the absolute position feedback devices they reference are summarized in the following table:

Ixx95 Value Range	Absolute Position Source	Ixx10 Address Type	Format
$000000 - $070000	ACC-8D Opt 7 R/D Converter	Multiplexer Port	Unsigned
$080000 - $300000	Parallel Data Y-Register	Turbo PMAC Memory-I/O	Unsigned
$310000	ACC-28 A/D Converter	Turbo PMAC Memory-I/O	Unsigned
$320000	ACC-49 Sanyo Abs. Encoder	Turbo PMAC Memory-I/O	Unsigned
$480000 - $700000	Parallel Data X-Register	Turbo PMAC Memory-I/O	Unsigned
$710000	ACC-8D Opt 9 Yaskawa Abs. Enc.	Multiplexer Port	Unsigned
$720000	MACRO Station Yaskawa Abs. Enc.	MACRO Node Number	Unsigned
$730000	MACRO Station R/D Converter	MACRO Node Number	Unsigned
$740000	MACRO Station Parallel Read	MACRO Node Number	Unsigned
$800000 - $870000	ACC-8D Opt 7 R/D Converter	Multiplexer Port	Signed
$880000 - $B00000	Parallel Data Y-Register	Turbo PMAC Memory-I/O	Signed
$B10000	ACC-28 A/D Converter	Turbo PMAC Memory-I/O	Signed
$B20000	ACC-49 Sanyo Abs. Encoder	Turbo PMAC Memory-I/O	Signed
$C80000 - $F00000	Parallel Data X-Register	Turbo PMAC Memory-I/O	Signed
$F10000	ACC-8D Opt 9 Yaskawa Abs. Enc.	Multiplexer Port	Signed
$F20000	MACRO Station Yaskawa Abs. Enc.	MACRO Node Number	Signed
$F30000	MACRO Station R/D Converter	MACRO Node Number	Signed
$F40000	MACRO Station Parallel Read	MACRO Node Number	Signed

The following section provides details for each type of position feedback.

R/D Converter: If Ixx95 contains a value from $000000 to $070000, or from $800000 to $870000, Motor xx will expect its absolute power-on position from an ACC-8D Opt. 7 R/D converter board. Ixx10 should contain the address of the board on the multiplexer port, as set by the DIP switches on the board.

The first hex digit of Ixx95, which can take a value of 0 or 8 in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8).

The second hex digit of Ixx95, which can take a value from 0 to 7 in this mode, specifies the number of the individual R/D converter at that multiplexer port address.

The following table shows the Ixx95 values for this mode and the R/D converter each specifies at the Ixx10 address:

Ixx95 Value for Unsigned Position	Ixx95 Value for Signed Position	ACC-8D Opt. 7 SW1-1 Setting	# of R/D Converter on ACC-8D Opt. 7
$000000	$800000	CLOSED (0)	1
$010000	$810000	CLOSED (0)	2
$020000	$820000	CLOSED (0)	3
$030000	$830000	CLOSED (0)	4
$040000	$840000	OPEN (1)	1
$050000	$850000	OPEN (1)	2
$060000	$860000	OPEN (1)	3
$070000	$870000	OPEN (1)	4

If Ixx99 is set greater than 0, the next higher numbered R/D converter at the same multiplexer port address is also read and treated as a geared-down resolver, with Ixx99 specifying the gear ratio. Ixx98 is also set greater than 0, the following R/D converter at the same multiplexer port address is read and treated as a third resolver geared down from the second, with Ixx98 specifying that gear ratio.

Parallel Data Read: If Ixx95 contains a value from $080000 to $300000, from $480000 to $700000, from $880000 to $B00000, or from $C80000 to $F00000, Motor xx will do a parallel data read of the Turbo PMAC memory or I/O register at the address specified by Ixx10. It expects to find the least significant bit of the feedback in Bit 0 of this register.

In this mode, bits 16 to 21 specify the number of bits to be read. If the last hex digit of Ixx95 is 0, consecutive bits will be read from the address specified by Ixx81, with the least significant bit read from bit 0. If the number of bits is greater than 24, the high bits are read from the register at the next higher-numbered address. This format is used for registers and I/O devices with 24-bit interfaces.

If the last hex digit of Ixx95 is 4, 5, or 6, data will be read in byte-wide pieces, with the least significant byte at the address specified in Ixx81, the next byte at one address higher, and so on, up to a possible 6 consecutive addresses. This format is intended for getting parallel data from the ACC-3E 3U-format stack I/O board or the ACC-14E 3U-format pack (UMAC) I/O board, which have byte-wide interfaces. For this format, the last hex digit of Ixx95 determines which byte of the 24-bit word is used, according to the following table:

Ixx95 Last Digit	Byte	Bits
4	Low	0 – 7
5	Middle	8 – 15
6	High	16 – 23

In this mode, bits 16 to 21 of Ixx95 specify the number of bits to be read, starting with bit 0 at the specified address. In this mode, they can take a value from $08 to $30 (8 to 48). If the number of bits is greater than 24, the high bits are read from the register at the next higher-numbered address.

In this mode, bit 22 of Ixx95 specifies whether a Y-register is to be read, or an X-register. A value of 0 in this bit specifies a Y-register; a value of 1 specifies an X-register. Almost all common sources of absolute position information are located in Y-registers, so this digit is usually 0.

In this mode, bit 23 of Ixx95 specifies whether the position is interpreted as an unsigned or a signed value. If the bit is set to 0, it is interpreted as an unsigned value, if the bit is 1, it is interpreted as a signed value.

Combining these components, Ixx95 values in this mode can be summarized as:

- $08000n - $30000n: Parallel Y-register read, unsigned value, 8 to 48 bits
- $48000n - $70000n: Parallel X-register read, unsigned value, 8 to 48 bits
- $88000n - $B0000n: Parallel Y-register read, signed value, 8 to 48 bits
- $C8000n - $F0000n: Parallel X-register read, signed value, 8 to 48 bits

Example: If Ixx10=$078D00 and Ixx95=$200000, Turbo PMAC would read 32 bits, the low 24 bits from Y:$078D00, and the high 8 bits from the low 8 bits of Y:$078D01.

Example: If Ixx10=$078C00 and Ixx95=$100004, Turbo PMAC would read 16 bits, with the low 8 bits from the low byte of Y:$078C00, and the high 8 bits from the low byte of Y:$078C01.

Example: If Ixx10=$079E03 and Ixx95=$120005, Turbo PMAC would read 18 bits, with the low 8 bits from the middle byte of Y:$079E03, and the next 8 bits from the middle byte of Y:$079E04, and the high 2 bits from the first 2 bits of the middle byte of Y:$079E05.

Example: If Ixx10=$078000 and Ixx95=$540000, Turbo PMAC would read 20 bits from X:$078000 (timer register for Channel 1). This type of setting is used for MLDT feedback.

ACC-28 A/D Converter Read: If Ixx95 is set to $310000 or $B10000, Motor xx will expect its power-on position in the upper 16 bits of the Turbo PMAC Y-memory or I/O register specified by Ixx10. This format is intended for ACC-28 A/D converters.

Bit 23 of Ixx95 specifies whether the position is interpreted as an unsigned or a signed value. If the bit is set to 0, it is interpreted as an unsigned value, if the bit is 1, it is interpreted as a signed value. Because ACC-28A produces signed values, Ixx95 should be set to $B10000 when using ACC-28A. ACC-28B produces unsigned values, so Ixx95 should be set to $310000 when using ACC-28B.

Sanyo Absolute Encoder Read: If Ixx95 is set to $320000 or $B20000, Motor xx will expect its power-on position from the ACC-49 Sanyo Absolute Encoder converter board at the Turbo PMAC address specified by Ixx10.

The first hex digit of Ixx95, which can take a value of 3 or B in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8). Set Ixx95 to $320000 for unsigned, or to $B20000 for signed.

Yaskawa Absolute Encoder Read: If Ixx95 is set to $710000 or $F10000, Motor xx will expect its power-on position from the Yaskawa Absolute Encoder converter board at the multiplexer port address specified by Ixx10.

The first hex digit of Ixx95, which can take a value of 7 or F in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8). Set Ixx95 to $710000 for unsigned, or to $F10000 for signed.

In this mode, Ixx99 specifies the number of bits per revolution for a single turn of the Yaskawa absolute encoder. It must be set greater than 0 to use the multi-turn absolute capability of this encoder.

MACRO Station Yaskawa Absolute Encoder Read: If Ixx95 is set to $720000 or $F20000, Motor xx will expect its power-on position from a Yaskawa Absolute Encoder through a MACRO Station. In this mode, Ixx10 specifies the MACRO node number at which the position value will be read by Turbo PMAC itself. Set-up variable MI11x for the MACRO Station tells the Station how to read the Yaskawa Encoder converter connected to its own multiplexer port or serial port.

The first hex digit of Ixx95, which can take a value of 7 or F in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8). Set Ixx95 to $720000 for unsigned, or to $F20000 for signed.

In this mode, Ixx99 specifies the number of bits per revolution for a single turn of the Yaskawa absolute encoder. It must be set greater than 0 to use the multi-turn absolute capability of this encoder.

MACRO Station R/D Converter Read: If Ixx95 is set to $730000 or $F30000, Motor xx will expect its power-on position from an R/D converter through a MACRO Station or compatible device. In this mode, Ixx10 specifies the MACRO node number at which Turbo PMAC will read the position value itself. Set-up variable MI11x for the matching node on the MACRO Station tells the Station how to read the R/D converter connected to its own multiplexer port.

The first hex digit of Ixx95, which can take a value of 7 or F in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8). Set Ixx95 to $730000 for unsigned, or to $F30000 for signed.

If Ixx99 is set greater than 0, the next higher numbered R/D converter at the same multiplexer port address is also read and treated as a geared-down resolver, with Ixx99 specifying the gear ratio. Ixx98 is also set greater than 0, the following R/D converter at the same multiplexer port address is read and treated as a third resolver geared down from the second, with Ixx98 specifying that gear ratio.

MACRO Station Parallel Data Read: If Ixx95 is set to $740000 or $F40000, Motor xx will expect its power-on position from a parallel data source through a MACRO Station or compatible device. In this mode, Ixx10 specifies the MACRO node number at which Turbo PMAC will read the position value itself. Set-up variable MI11x for the matching node on the MACRO Station tells the Station how to read the parallel data source connected to it.

The first hex digit of Ixx95, which can take a value of 7 or F in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8). Set Ixx95 to $740000 for unsigned, or to $F40000 for signed.

In non-Turbo PMACs, bits 16-23 of Ix10 control this function.

Ixx96 Motor xx Command Output Mode Control

Range: 0 - 1

Units: none

Default: 0

Ixx96 controls how Turbo PMAC writes to the command output register(s) specified in Ixx02.

If bit 0 of Ixx01 is set to 0 (no Turbo PMAC commutation for Motor xx), and Ixx96 is set to 0, the single command value from the Turbo PMAC servo is written to the register specified by Ixx02 as a signed (bipolar) value.

For PMAC(1)-style Servo ICs only, if bit 0 of Ixx01 is set to 0 and Ixx96 is set to 1, then the command output value is the absolute value (magnitude) of what the servo calculates, and the sign (direction) is output on the AENAn/DIRn line of the set of flags addressed by Ixx25 (polarity determined by jumper E17 or E17x). In this case, bit 16 of Ixx24 should also be set to 1 to *disable* the amplifier-enable function for that line. For PMAC2-style Servo ICs, this sign-and-magnitude mode is not supported.

If bit 0 of Ixx01 is set to 1 (Turbo PMAC commutation enabled for Motor xx), Ixx82 is set to 0 (Turbo PMAC current loop disabled for Motor xx), and Ixx96 is set to 0, Turbo PMAC will perform the normal closed-loop commutation for Motor xx. If bit 0 of Ixx01 is set to 1, Ixx82 is set to 0, and Ixx96 is set to 1, then Turbo PMAC's commutation performs the special "direct microstepping" algorithm. In this algorithm, the magnitude of the command from the servo does not affect the magnitude of the phase command outputs; it simply controls their frequency.

If bit 0 of Ixx01 is set to 1 (Turbo PMAC commutation enabled for Motor xx), Ixx82 is set to a value greater than 0 (Turbo PMAC current loop enabled for Motor xx), and Ixx96 is set to 0, Turbo PMAC will perform the normal direct-PWM control with both direct and quadrature current loops closed, for a 3-phase motor. If bit 0 of Ixx01 is set to 1, Ixx82 is set to a value greater than 0, and Ixx96 is set to 1, Turbo PMAC will perform direct-PWM control for a brush motor, truly closing only the quadrature current loop, and repeatedly zeroing the direct current-loop registers.

In non-Turbo PMACs, this function is controlled by bit 16 of Ix02.

Ixx97 Motor xx Position Capture & Trigger Mode

Range: 0 - 3

Units: none

Default: 0

Ixx97 controls the triggering function and the position capture function for triggered moves on Motor xx. These triggered moves include homing search moves, on-line jog-until-trigger moves, and motion program **RAPID**-mode move-until-trigger. Ixx97 is a 2-bit value: bit 0 controls the how the capture of the trigger position is done (the post-trigger move is relative to the trigger position), and bit 1 specifies what the trigger condition is.

Hardware Capture: If Ixx97 is set to 0 or 2 (bit 0 = 0), Turbo PMAC will use the hardware-captured position in the Servo IC as the trigger position. This is the "flag-capture" register associated with the flag set used for the motor, as specified for Ixx25. In order for this to work properly, the position-loop feedback for Motor xx, as specified by Ixx03, and the conversion table, must be received through the encoder counter of the same hardware interface channel as used for the flag set (e.g. if flag set 2 is used, encoder 2 must be used for position-loop feedback). The advantage of the hardware position capture is that it is immediate, and accurate to the exact count at any speed.

Software Capture: If Ixx97 is set to 1 or 3 (bit 0 = 1), Turbo PMAC will use a software-captured position for the trigger position. In this case, Turbo PMAC uses the register whose address is specified by Ixx03, usually a register in the encoder conversion table, for the trigger position. The advantage of software capture is that it can be used with any type of feedback, or when the position encoder channel is not the same as the flag channel. The disadvantage is that the software capture can have up to 1 background cycle delay (typically 2-3 msec), which limits the accuracy of the capture.

Input Trigger: If Ixx97 is set to 0 or 1, (bit 1 = 0), Turbo PMAC will use the input capture trigger flag in the Servo IC flag register addressed by Ixx25 as the trigger for the move. This input trigger is created by an edge of the index input and a flag input for the channel as specified by I6mn2 and I6mn3 for the selected Channel n of Servo IC m, or if a MACRO flag register is selected by Ixx25, with bit 18 of Ixx25 set to 1, the input trigger condition is set by MI-variables on the MACRO station.

Error Trigger: If Ixx97 is set to 2 or 3, (bit 1 = 1), Turbo PMAC will use the "warning following error" status bit in the motor status word as the trigger for the move. When this bit changes from 0 to 1 because the magnitude of the following error for the motor has exceeded the warning limit in Ixx11, Turbo PMAC will consider this the trigger condition for the triggered move. Because there is nothing in this mode that can create a hardware capture, only software capture should be used with error trigger (Ixx97 = 3).

Summarizing the values of Ixx97, and their effect:

- Ixx97 = 0: Input trigger, hardware position capture
- Ixx97 = 1: Input trigger, software position capture
- Ixx97 = 2: Error trigger, hardware position capture (not useful!)
- Ixx97 = 3: Error trigger, software position capture

In non-Turbo PMACs, this function is controlled by bits 16 and 17 of Ix03.

Ixx98 Motor xx Third-Resolver Gear Ratio

Range: 0 - 4095

Units: Second-resolver turns per third resolver turn

Default: 0

Ixx98 tells Turbo PMAC the gear ratio between the second (medium) and third (coarse) resolvers for a triple-resolver setup for Motor xx. It is expressed as the number of turns (electrical cycles) the second resolver makes in one full turn (electrical cycle) of the third resolver.

This parameter is used only during Turbo PMAC's power-up/reset cycle to establish absolute power-on servo position. Therefore, the parameter must be set, the value stored in non-volatile flash memory with the **SAVE** command, and the card reset before it takes effect.

If there is no geared third resolver on Motor xx, or if absolute power-on position is not desired, Ixx98 should be set to zero. If either Ixx10 (for the primary resolver) or Ixx99 (for the secondary resolver) is set to zero, Ixx98 is not used.

The third resolver must be connected to the next higher numbered R/D converter at the same multiplexer address than the second resolver, which must be connected to the next higher numbered converter at the same multiplexer address than the primary resolver. There can be up to eight R/D converters on two ACC-8D Option 7 boards at one multiplexer address.

In non-Turbo PMACs, this function is controlled by I-variable I8x.

Example:
Motor 3 has a triple resolver, with each resolver geared down by a ratio of 16:1 from the resolver before it. The fine resolver is connected to R/D converter 4 at multiplexer address 0 (the first R/D converter on the second ACC-8D Option 7 at address 0). The medium resolver is connected to R/D converter 5 at this address, and the coarse resolver is connected to R/D converter 6. The following I-variable values should be used:

```
I310=$000100 .      ; The $000100 specifies multiplexer address 0
I395=$040000 .      ; the 4 in the high 8 bits of Ixx95
.....               ; specifies R/D converter 4 at this address.
I399=16 .....       ; Specifies 16:1 ratio between medium and fine
I398=16 .....       ; Specifies 16:1 ratio between coarse and medium
```

Ixx99 Motor xx Second-Resolver Gear Ratio

Range: 0 - 4095

Units: Primary resolver turns per second-resolver turn

Default: 0

Ixx99 tells PMAC the gear ratio between the first (fine, or primary) and second (coarse or medium) resolvers for a double- or triple-resolver setup for Motor xx. It is expressed as the number of turns (electrical cycles) the first resolver makes in one full turn (electrical cycle) of the second resolver.

This parameter is used only during Turbo PMAC's power-up/reset cycle to establish absolute power-on servo position. Therefore, the parameter must be set, the value stored in non-volatile flash memory with the **SAVE** command, and the card reset before it takes effect.

If there is no geared second resolver on Motor xx, or if absolute power-on position is not desired, Ixx99 should be set to zero. If Ixx10 (for the primary resolver) is set to zero, Ixx99 is not used. In a triple-resolver system, Ixx99 must be set greater than zero in order for Ixx88 (third-resolver gear ratio) to be used.

The second resolver must be connected to the next higher numbered R/D converter at the same multiplexer address than the first resolver. If there is a third resolver, it must be connected to the next higher numbered converter at the same multiplexer address than the second resolver. There can be up to eight R/D converters on two ACC-8D Option 7 boards at one multiplexer address.

If Ixx10 is set up for an ACC-8D Option 9 Yaskawa encoder converter, Ixx99 represents the counts per revolution (including x2 or x4 quadrature decode, if used) of the encoder; effectively it is the "gear ratio" between the encoder and the revolution counter.

In non-Turbo PMACs, this function is controlled by I-variable I9x.

Example:
Motor 1 has a double resolver with the fine resolver connected to the R/D converter at location 2 on an ACC-8D Option 7 board set to multiplexer address 4, and the coarse resolver, geared down at a 36:1 ratio from the fine resolver, connected to the R/D converter at location 3 on the same board. The following I-variable settings should be used:

```
I110=$000004 .      ; Value of $0004 specifies multiplexer address 4
I118=$020000 .      ; $02 in high 8 bits of I118
.....               ; specifies R/D at location 2 of this address
```

```
I199=36 .....          ; Specify 36 turns of fine resolver per turn of
.....                  ; coarse resolver; R/D must be at location 3
.....                  ; of multiplexer address 4
I198=0 .....           ; No third resolver
```

Supplemental Motor Setup I-Variables

`Iyy00 – Iyy49/Iyy50 – Iyy99` Supplemental Motor I-Variables
`yy = 33 – 48`
Motor Number xx = 2 * (yy - 32) - 1 for Iyy00 – Iyy49 (odd-numbered motors)
Motor Number xx = 2 * (yy - 32) for Iyy50 – Iyy99 (even-numbered motors)

Motor #	Supplemental I-Variables	Motor #	Supplemental I-Variables	Motor #	Supplemental I-Variables	Motor #	Supplemental I-Variables
1	I3300 - I3349	9	I3700 - I3749	17	I4100 - I4149	25	I4500 - I4549
2	I3350 - I3399	10	I3750 - I3799	18	I4150 - I4199	26	I4550 - I4599
3	I3400 - I3449	11	I3800 - I3849	19	I4200 - I4249	27	I4600 - I4649
4	I3450 - I3499	12	I3850 - I3899	20	I4250 - I4299	28	I4650 - I4699
5	I3500 - I3549	13	I3900 - I3949	21	I4300 - I4349	29	I4700 - I4749
6	I3550 - I3599	14	I3950 - I3999	22	I4350 - I4399	30	I4750 - I4799
7	I3600 - I3649	15	I4000 - I4049	23	I4400 - I4449	31	I4800 - I4849
8	I3650 - I3699	16	I4050 - I4099	24	I4450 - I4499	32	I4850 - I4899

Iyy00/50 Motor xx Extended Servo Algorithm Enable

Range: 0 - 1

Units: none

Default: 0

Iyy00 or Iyy50 controls whether the matching Motor xx uses the PID servo algorithm or the Extended Servo Algorithm (ESA). If Iyy00/50 is set to the default value of 0, Motor xx uses the PID servo algorithm, whose gains are determined by Ixx30-39 and Ixx63-69. If Iyy00/50 is set to 1, Motor xx uses the ESA, whose gains are determined by Iyy10/60 to Iyy39/89.

The motor should be "killed" when changing which servo algorithm is used by changing Iyy00/50. The loop should not be closed again until the gain variables for the selected servo algorithm are basically set up properly.

The following servo control I-variables are only used if Iyy00/50 is set to 0:

- Ixx30-39, Ixx63-65, Ixx67

The following servo control I-variables are only used if Iyy00/50 is set to 1:

- Iyy10-39 / Iyy60 – 89

Note:

These I-variables are "disabled" if Iyy00/50 for the motor is set to 0. No value can be written to them, and if queried, they will report a value of 0.

The following servo control I-variables are used regardless of the setting of Iyy00/50:

- Ixx59, Ixx60, Ixx68, Ixx69

Iyy10 – Iyy39/Iyy60 – Iyy89 Motor xx Extended Servo Algorithm Gains

Iyy10 through Iyy39 (for odd-numbered motors), and Iyy60 through Iyy89 (for even-numbered motors) are the gains for the Extended Servo Algorithm (ESA). The following table lists the function of each variable; refer to the User's Guide for a detailed description and diagram of the algorithm structure.

I-Var. for Odd-Numbered Motors	I-Var. for Even-Numbered Motors	Gain Name	Range	I-Var. for Odd-Numbered Motors	I-Var. for Even-Numbered Motors	Gain Name	Range
Iyy10	Iyy60	s0	-1.0≤Var<+1.0	Iyy25	Iyy75	TS	-2^{23}≤Var<2^{23}
Iyy11	Iyy61	s1	-1.0≤Var<+1.0	Iyy26	Iyy76	L1	-1.0≤Var<+1.0
Iyy12	Iyy62	f0	-1.0≤Var<+1.0	Iyy27	Iyy77	L2	-1.0≤Var<+1.0
Iyy13	Iyy63	f1	-1.0≤Var<+1.0	Iyy28	Iyy78	L3	-1.0≤Var<+1.0
Iyy14	Iyy64	h0	-1.0≤Var<+1.0	Iyy29	Iyy79	k0	-1.0≤Var<+1.0
Iyy15	Iyy65	h1	-1.0≤Var<+1.0	Iyy30	Iyy80	k1	-1.0≤Var<+1.0
Iyy16	Iyy66	r1	-1.0≤Var<+1.0	Iyy31	Iyy81	k2	-1.0≤Var<+1.0
Iyy17	Iyy67	r2	-1.0≤Var<+1.0	Iyy32	Iyy82	k3	-1.0≤Var<+1.0
Iyy18	Iyy68	r3	-1.0≤Var<+1.0	Iyy33	Iyy83	KS	-2^{23}≤Var<2^{23}
Iyy19	Iyy69	r4	-1.0≤Var<+1.0	Iyy34	Iyy84	d1	-1.0≤Var<+1.0
Iyy20	Iyy70	t0	-1.0≤Var<+1.0	Iyy35	Iyy85	d2	-1.0≤Var<+1.0
Iyy21	Iyy71	t1	-1.0≤Var<+1.0	Iyy36	Iyy86	g0	-1.0≤Var<+1.0
Iyy22	Iyy72	t2	-1.0≤Var<+1.0	Iyy37	Iyy87	g1	-1.0≤Var<+1.0
Iyy23	Iyy73	t3	-1.0≤Var<+1.0	Iyy38	Iyy88	g2	-1.0≤Var<+1.0
Iyy24	Iyy74	t4	-1.0≤Var<+1.0	Iyy39	Iyy89	GS	-2^{23}≤Var<2^{23}

The ESA gains that these I-variables represent are usually set using the Auto-tuning function of the Servo Evaluation Package (SEP).

Note:

These I-variables are "disabled" if Iyy00/50 for the motor is set to 0. No value can be written to them, and if queried, they will report a value of 0.

System Configuration Reporting

I4900 Servo ICs Present

Range: $000000 – $0FFFFF

Units: none (individual bits)

Default: --

I4900 is a read-only status I-variable that reports which Servo ICs are present in a Turbo PMAC system. It is provided for user setup and diagnostic purposes only. On power-up/reset, Turbo PMAC automatically queries for the presence of each possible Servo IC and reports what it has found in I4900. It also enables the set-up I-variables for each IC that it has found.

I4900 is a 20-bit value with each individual bit representing each possible Servo IC that could be present in the system. The bit is set to 0 if the IC is not present; it is set to 1 if the IC is present.

The following table shows the Servo IC each bit of I4900 represents:

I4900 Bit #	Bit Value	Servo IC #	Ident I-var	I-vars	Location
0	$1	0	x	I7000 – I7049	On-board or stack
1	$2	1	x	I7100 – I7149	On-board or stack
2	$4	2	I4910	I7200 – I7249	Exp. port accessory
3	$8	3	I4911	I7300 – I7349	Exp. port accessory
4	$10	4	I4914	I7400 – I7449	Exp. port accessory
5	$20	5	I4915	I7500 – I7549	Exp. port accessory
6	$40	6	I4918	I7600 – I7649	Exp. port accessory
7	$80	7	I4919	I7700 – I7749	Exp. port accessory
8	$100	8	I4922	I7800 – I7849	Exp. port accessory
9	$200	9	I4923	I7900 – I7949	Exp. port accessory
10	$400	0*	x	I7050 – I7099	(none)
11	$800	1*	x	I7150 – I7199	(none)
12	$1000	2*	I4912	I7250 – I7299	Exp. port accessory
13	$2000	3*	I4913	I7350 – I7399	Exp. port accessory
14	$4000	4*	I4916	I7450 – I7499	Exp. port accessory
15	$8000	5*	I4917	I7550 – I7599	Exp. port accessory
16	$10000	6*	I4920	I7650 – I7699	Exp. port accessory
17	$20000	7*	I4921	I7750 – I7799	Exp. port accessory
18	$40000	8*	I4924	I7850 – I7899	Exp. port accessory
19	$80000	9*	I4925	I7950 – I7999	Exp. port accessory

Note:

In firmware versions older than 1.936, bits 20 through 23 of I4900 reported the presence of the four possible MACRO ICs. With versions 1.936 and newer, there is support for more than four MACRO ICs, and their presence is reported in I4902.

I4901 Servo IC Type

Range: $000000 – $0FFFFF

Units: none (individual bits)

Default: --

I4901 is a read-only status I-variable that reports which types of Servo ICs are present in a Turbo PMAC system. It is provided for user setup and diagnostic purposes only. On power-up/reset, Turbo PMAC queries for the presence and type of each possible Servo IC automatically and reports the types it has found in I4901. It also enables the appropriate group set-up I-variables for each IC found, depending on the type.

I4901 is a 20-bit value with each individual bit representing each possible Servo that could be present in the system. The table shown in the I4900 description, above, lists which IC is represented by each bit.

A bit of I4901 is set to 0 if a "Type 0" PMAC(1)-style DSPGATE Servo IC is found at the appropriate address slot, or if no Servo IC is found there. The bit is set to 1 if a "Type 1" PMAC2-style DSPGATE1 Servo IC is found there.

I4902 MACRO ICs Present

Range: $000000 – $0FFFF

Units: none (individual bits)

Default: --

I4902 is a read-only status I-variable that reports which MACRO ICs are present in a Turbo PMAC system. It is provided for user setup and diagnostic purposes only. On power-up/reset, Turbo PMAC automatically queries for the presence of each possible MACRO IC and reports what it has found in I4902.

I4902 is a 16-bit value with each individual bit representing each possible MACRO IC that could be present in the system. (Only a UMAC system can have more than 4 MACRO ICs present.) The bit is set to 0 if the IC is not present; it is set to 1 if the IC is present.

The following table shows the MACRO IC each bit of I4902 represents:

I4902 Bit #	Bit Value	Base Address	Ident I-var	I4902 Bit #	Bit Value	Base Address	Ident I-var
0	$1	$078400	I4926	8	$100	$078600	I4934
1	$2	$079400	I4927	9	$200	$079600	I4935
2	$4	$07A400	I4928	10	$400	$07A600	I4936
3	$8	$07B400	I4929	11	$800	$07B600	I4937
4	$10	$078500	I4930	12	$1000	$078700	I4938
5	$20	$079500	I4931	13	$2000	$079700	I4939
6	$40	$07A500	I4932	14	$4000	$07A700	I4940
7	$80	$07B500	I4933	15	$8000	$07B700	I4941

Which of these ICs is assigned as MACRO IC 0, 1, 2, and 3 for firmware support issues is dependent on the settings of I20, I21, I22, and I23, respectively.

Note:

In firmware versions older than 1.936, bits 20 through 23 of I4900 reported the presence of the four possible MACRO ICs. With versions 1.936 and newer, there is support for more than four MACRO ICs, and their presence is reported in I4902.

I4903 MACRO IC Types

Range: $000000 – $00FFFF

Units: none (individual bits)

Default: --

I4903 is a read-only status I-variable that reports which types of MACRO ICs are present in a Turbo PMAC system. It is provided for user setup and diagnostic purposes only. On power-up/reset, Turbo PMAC automatically queries for the presence and type of each possible MACRO IC and reports the types it has found in I4903.

I4903 is a 16-bit value with each individual bit representing each possible Servo that could be present in the system. The table shown in the I4902 description, above, lists which IC is represented by each bit.

A bit of I4903 is set to 1 if a "DSPGATE2" MACRO IC is found at the appropriate address slot. The bit is set to 0 if a "MACROGATE" MACRO IC is found there, or if no MACRO IC is found there.

I4904 Dual-Ported RAM ICs Present

Range: $000000 – $FF8000

Units: none (individual bits)

Default: --

I4904 is a read-only status I-variable that reports which dual-ported RAM ICs are present in a Turbo PMAC system. It is provided for user setup and diagnostic purposes only. On power-up/reset, Turbo PMAC automatically queries for the presence of each possible DPRAM IC and reports what it has found in I4904.

I4904 is a 24-bit value with the 9 high bits currently used. Each individual bit used represents each possible DPRAM IC that could be present in the system. The bit is set to 0 if the IC is not present; it is set to 1 if the IC is present.

UMAC accessory boards with DPRAM, such as the ACC-54E UBUS/Ethernet board, provide identification information in variables I4942 – I4949, depending on their base address.

The following table shows the DPRAM IC each bit of I4904 represents, and the matching identification I-variable:

I4904 Bit #	Bit Value	Base Address	Ident I-var
15	$8000	$060000	None
16	$10000	$06C000	I4942
17	$20000	$074000	I4943
18	$40000	$06D000	I4944
19	$80000	$075000	I4945
20	$100000	$06E000	I4946
21	$200000	$076000	I4947
22	$400000	$06F000	I4948
23	$800000	$077000	I4949

I24 contains the address of the DPRAM IC that is to be used for the automatic communications functions. The value of I24 at power-up/reset sets the pointers for these automatic communications functions.

I4904 also contains information about the flash memory (this information is contained in I4909 as well). Bits 0 – 2 of I4904, which contain a value from 0 to 7, report which type of flash-memory IC is present in the system. Since bit 3 is not used, these bits form the last hex digit of I4909. The following list shows what each value of this digit means:

- 0: Unknown flash IC (cannot save)
- 1: Intel 28F004S3 512k x 8 flash IC
- 2: Intel 28F008S3 1M x 8 flash IC (Opt 5x0)
- 3: Intel 28F016S3 2M x 8 flash IC (Opt 5x1,2)
- 4: Intel 28F160S3 2M x 8 flash IC (Opt 5x1,2)
- 5: Intel 28F320S3 4M x 8 flash IC (Opt 5x3)
- 6: Intel 28F320J5 4M x 8 flash IC (Opt 5x3)
- 7: Intel 28F640J5 8M x 8 flash IC

In addition, I4904 contains the status of the eight locking bits that an application can use with the **LOCK** and **UNLOCK** commands to make sure that tasks of different priorities do not overwrite each other. The following table shows how the eight locking bits are stored. Each bit is a 0 if unlocked; it is a 1 if locked.

I4904 Bit #	Bit Value	Locking Bit #
4	$10	0
5	$20	1
6	$40	2
7	$80	3
8	$100	4
9	$200	5
10	$400	6
11	$800	7

I4908 End of Open Memory

Range: $006000 – $040000

Units: none (individual bits)

Default: --

I4908 is a read-only status I-variable that reports the end of the open active memory that can be used for most programs and buffers. It returns the address of the register one number higher than the last register than can be used for these programs and buffers.

The value returned for I4908 is a function of two things: the size of the "user data" memory, and the declared size of the "UBUFFER" user buffer. If no UBUFFER has been declared, I4908 will return $010800 for the standard user data memory (Option 5x0 or 5x2). Starting in V1.937, Turbo PMACs with the extended user data memory (Option 5x1 or 5x3) by default have a 65,536-word ($10000) UBUFFER declared, occupying addresses $030000 - $03FFF. In these systems, I4908 will return a value of $030000. It is possible to declare a smaller or non-existent UBUFFER in these systems with an explicit **DEFINE UBUFFER** command. With no UBUFFER, a Turbo PMAC with the extended user data memory option will report an I4908 value of $040000.

If a UBUFFER has been declared, the value returned for I4908 will be reduced by an amount equivalent to the size of the UBUFFER.

Example:

```
$$$***          ; Re-initialize card, clearing all buffers
I4908            ; Request value of I4908
$010800          ; Value for standard data memory, no UBUFFER
DEF UBUF 512     ; Reserve 512 ($200) words for user buffer
I4908            ; Request value of I4908
$010600          ; Value reduced by 512 ($200)
```

I4909 Turbo CPU ID Configuration

Range: $000000000 – $FFFFFFFFF

Units: none (individual bits)

Default: --

I4909 is a read-only status I-variable that reports configuration information for the Turbo PMAC CPU section. I4909 is a 36-bit value that contains vendor ID, option data, CPU type, and card ID. All of it is reported if I39 is set to 0; individual parts are reported if I39>0.

The following table shows what each part of I4909 returns and what each part means.

I4909 Bit #(s)	Bit Value(s)	Meaning
0 – 7 (I39=0) 0 – 7 (I39=1)	$FF (I39=0) $FF (I39=1)	=1: Vendor is Delta Tau
8 (I39=0) 0 (I39=2)	$100 (I39=0) $1 (I39=2)	=0: Standard (128k x 24) user data memory (Opt 5x0,2) =1: Extended (512k x 24) user data memory (Opt 5x1,3)
9 (I39=0) 1 (I39=2)	$200 (I39=0) $2 (I39=2)	=0: Standard (128k x 24) program memory (Opt 5x0,1) =1: Extended (512k x 24) program memory (Opt 5x2,3)
10,11 (I39=0) 2,3 (I39=2)	$C00 (I39=0) $C (I39=2)	=0: No dual-ported RAM =1: 8k x 16 dual-ported RAM (Opt 2x) =3: 32k x 16 dual-ported RAM (Opt 2x)
12,13 (I39=0) 4,5 (I39=2)	$3000 (I39=0) $30 (I39=2)	=0: No battery-backed RAM =1: 32k x 24 battery-backed RAM (Opt 16A) =3: 128k x 24 battery-backed RAM (Opt 16B)
14,15,16 (I39=0) 6,7,8 (I39=2)	$1C000 (I39=0) $1C0 (I39=2)	=0: Unknown flash IC (cannot save) =1: Intel 28F004S3 512k x 8 flash IC =2: Intel 28F008S3 1M x 8 flash IC (Opt 5x0) =3: Intel 28F016S3 2M x 8 flash IC (Opt 5x1,2) =4: Intel 28F160S3 2M x 8 flash IC (Opt 5x1,2) =5: Intel 28F320S3 4M x 8 flash IC (Opt 5x3) =6: Intel 28F320J5 4M x 8 flash IC (Opt 5x3) =7: Intel 28F640J5 8M x 8 flash IC
17 (I39=0) 9 (I39=2)	$20000 (I39=0) $200 (I39=2)	=0: Aux. RS232 not present or not active =1: Aux. RS232 present (Opt 9T) and active (I53>0)
18 – 21 (I39=0) 0 – 3 (I39=3)	$3C0000 (I39=0) $F (I39=3)	=0: DSP56303 CPU =1: DSP56309 CPU >1: (Reserved)
22 – 35 (I39=0) 0 – 13 (I39=4)	$FFFC00000 (I39=0) $3FFF (I39=4)	(Last 4 digits of card part number)

I4910 – I4925 Servo IC Card Identification

Range: $000000000 – $FFFFFFFFF

Units: none (individual bits)

Default: --

I4910 – I4925 are read-only status I-variables that report configuration information for UMAC accessory boards that contain Servo ICs, such as the ACC-24E2 family and the ACC-51E.

The following table shows which variable corresponds to which card:

Ident I-var	Servo IC #	I4900 Bit #	Board DIP Switch 4,3,2,1 Setting[1]	Board Base Address	Board Setup I-variables	Board Ident. Info Address[2]
I4910	2	2	0000 (0)	$078200	I7200 – I7249	$078F08
I4911	3	3	0001 (1)	$078300	I7300 – I7349	$078F0C
I4912	2*	12	0010 (2)	$078220	I7250 – I7299	$078F28
I4913	3*	13	0011 (3)	$078320	I7350 – I7399	$078F2C
I4914	4	4	0100 (4)	$079200	I7400 – I7449	$079F08
I4915	5	5	0101 (5)	$079300	I7500 – I7549	$079F0C
I4916	4*	14	0110 (6)	$079220	I7450 – I7499	$079F28
I4917	5*	15	0111 (7)	$079320	I7550 – I7599	$079F2C
I4918	6	6	1000 (8)	$07A200	I7600 – I7649	$07AF08
I4919	7	7	1001 (9)	$07A300	I7700 – I7749	$07AF0C
I4920	6*	16	1010 (10)	$07A220	I7650 – I7699	$07AF28
I4921	7*	17	1011 (11)	$07A320	I7750 – I7799	$07AF2C
I4922	8	8	1100 (12)	$07B200	I7800 – I7849	$07BF08
I4923	9	9	1101 (13)	$07B300	I7900 – I7949	$07BF0C
I4924	8*	18	1110 (14)	$07B220	I7850 – I7899	$07BF28
I4925	9*	19	1111 (15)	$07B320	I7950 – I7999	$07BF2C

Notes:

1. Board DIP-switches SW1-1 to SW1-4 are currently used to set the addresses of the boards on the UBUS backplane. A '0' is ON (CLOSED); a '1' is OFF (OPEN). The E1 jumper on the back of the ACC-Ux UBUS backplane board must be ON to use the DIP-switch addressing. SW1-5 and SW1-6 must be ON (CLOSED).
2. For diagnostic purposes only. The four "Y" registers starting at this address contain the information used to assemble this I-variable.

I4910 to I4925 have multiple "fields" of information, which can be reported individually or in groups, depending on the setting of I39. The following table shows what each field reports.

Information	Reported when:	Bits when I39=0	Bits when I39>0
Vendor ID	I39 = 0 or 1	0 – 7	0 – 7
Options	I39 = 0 or 2	8 – 17	0 – 9
Revision #	I39 = 0 or 3	18 – 21	0 – 3
Card ID	I39 = 0 or 4	22 – 35	0 – 13
Base Address	I39 = 5	--	0 - 18

- The Vendor ID field is an 8-bit value that reports the manufacturer of the board. Delta Tau boards report a value of 1 in this field.
- The Options field is a 10-bit field that is typically used as a set of individual bits to report which options are present on the board. The meaning of each bit is board-dependent.
- The Revision Number field is a 4-bit value that represents the design revision (0 to 15) of the board. For Delta Tau boards, this value matches the "x" in the "–10x" part number suffix for the board.
- The Card ID field is a 14-bit value that represents the part number of the board. For Delta Tau boards, this value matches the "xxxx" in the "60xxxx" (decimal) main part number for the board.
- The Base Address field is a 19-bit value that represents the starting address of the board in the Turbo PMAC's address space.

I4926 – I4941 MACRO IC Card Identification

Range: $000000000 – $FFFFFFFFF

Units: none (individual bits)

Default: --

I4926 – I4941 are read-only status I-variables that report configuration information for UMAC accessory boards that contain MACRO ICs, such as the ACC-5E. Which of these ICs is assigned as MACRO IC 0, 1, 2, or 3 for firmware support issues is dependent on the settings of I20, I21, I22, and I23, respectively. The following table shows which variable corresponds to which card:

Ident I-var	I4902 Bit #	Board DIP Switch 4,3,2,1 Setting[1]	Board Base Address	Board Ident. Info Address[2]
I4926	0	0000 (0)	$078400	$078F10
I4927	1	0001 (1)	$079400	$079F10
I4928	2	0010 (2)	$07A400	$07AF10
I4929	3	0011 (3)	$07B400	$07BF10
I4930	4	0100 (4)	$078500	$078F14
I4931	5	0101 (5)	$079500	$079F14
I4932	6	0110 (6)	$07A500	$07AF14
I4933	7	0111 (7)	$07B500	$07BF14
I4934	8	1000 (8)	$078600	$078F18
I4935	9	1001 (9)	$079600	$079F18
I4936	10	1010 (10)	$07A600	$07AF18
I4937	11	1011 (11)	$07B600	$07BF18
I4938	12	1100 (12)	$078700	$078F1C
I4939	13	1101 (13)	$079700	$079F1C
I4940	14	1110 (14)	$07A700	$07AF1C
I4941	15	1111 (15)	$07B700	$07BF1C

Notes:

1. Board DIP-switches SW1-1 to SW1-4 are currently used to set the addresses of the boards on the UBUS backplane. A '0' is ON (CLOSED); a '1' is OFF (OPEN). The E1 jumper on the back of the ACC-Ux UBUS backplane board must be ON to use the DIP-switch addressing. SW1-5 and SW1-6 must be ON (CLOSED).
2. For diagnostic purposes only. The four Y registers starting at this address contain the information used to assemble this I-variable.

I4926 to I4941 have multiple fields of information, which can be reported individually or in groups, depending on the setting of I39. The following table shows what each field reports.

Information	Reported when:	Bits when I39=0	Bits when I39>0
Vendor ID	I39 = 0 or 1	0 – 7	0 – 7
Options	I39 = 0 or 2	8 – 17	0 – 9
Revision #	I39 = 0 or 3	18 – 21	0 – 3
Card ID	I39 = 0 or 4	22 – 35	0 – 13
Base Address	I39 = 5	--	0 - 18

- The Vendor ID field is an 8-bit value that reports the manufacturer of the board. Delta Tau boards report a value of 1 in this field.
- Typically, the Options field is a 10-bit field that is used as a set of individual bits to report which options are present on the board. The meaning of each bit is board-dependent.
- The Revision Number field is a 4-bit value that represents the design revision (0 to 15) of the board. For Delta Tau boards, this value matches the "x" in the "–10x" part number suffix for the board.

- The Card ID field is a 14-bit value that represents the part number of the board. For Delta Tau boards, this value matches the "xxxx" in the "60xxxx" (decimal) main part number for the board.
- The Base Address field is a 19-bit value that represents the starting address of the board in the Turbo PMAC's address space.

I4942 – I4949 DPRAM IC Card Identification

Range: $000000000 – $FFFFFFFFF

Units: none (individual bits)

Default: --

I4942 – I4949 are read-only status I-variables that report configuration information for UMAC accessory boards that contain DPRAM ICs, such as the ACC-54E. The following table shows which variable corresponds to which card:

Ident I-var	I4904 Bit #	Board DIP Switch 4,3,2,1 Setting[1]	Board Base Address	Board Ident. Info Address[2]
I4942	16	0000 (0) or 0001 (1)	$06C000	$078F20
I4943	17	0010 (2) or 0011 (3)	$074000	$078F24
I4944	18	0100 (4) or 0101 (5)	$06D000	$079F20
I4945	19	0110 (6) or 0011 (7)	$075000	$079F24
I4946	20	1000 (8) or 1001 (9)	$06E000	$07AF20
I4947	21	1010 (10) or 1011 (11)	$076000	$07AF24
I4948	22	1100 (12) or 1101 (13)	$06F000	$07BF20
I4949	23	1110 (14) or 1011 (15)	$077000	$07BF24

Notes:

1. Board DIP-switches SW1-1 to SW1-4 are currently used to set the addresses of the boards on the UBUS backplane. A '0' is ON (CLOSED); a '1' is OFF (OPEN). The E1 jumper on the back of the ACC-Ux UBUS backplane board must be ON to use the DIP-switch addressing. SW1-5 and SW1-6 must be ON (CLOSED).
2. For diagnostic purposes only. The four "Y" registers starting at this address contain the information used to assemble this I-variable.

I4942 to I4949 have multiple fields of information, which can be reported individually or in groups, depending on the setting of I39. The following table shows what each field reports.

Information	Reported when:	Bits when I39=0	Bits when I39>0
Vendor ID	I39 = 0 or 1	0 – 7	0 – 7
Options	I39 = 0 or 2	8 – 17	0 – 9
Revision #	I39 = 0 or 3	18 – 21	0 – 3
Card ID	I39 = 0 or 4	22 – 35	0 – 13
Base Address	I39 = 5	--	0 - 18

- The Vendor ID field is an 8-bit value that reports the manufacturer of the board. Delta Tau boards report a value of 1 in this field.
- The Options field is a 10-bit field that is typically used as a set of individual bits to report which options are present on the board. The meaning of each bit is board-dependent.
- The Revision Number field is a 4-bit value that represents the design revision (0 to 15) of the board. For Delta Tau boards, this value matches the "x" in the "–10x" part number suffix for the board.

- The Card ID field is a 14-bit value that represents the part number of the board. For Delta Tau boards, this value matches the "xxxx" in the "60xxxx" (decimal) main part number for the board.
- The Base Address field is a 19-bit value that represents the starting address of the board in the Turbo PMAC's address space.

I4950 – I4965 I/O IC Card Identification

Range: $000000000 – $FFFFFFFFF

Units: none (individual bits)

Default: --

I4950 – I4965 are read-only status I-variables that report configuration information for UMAC accessory boards that contain I/O ICs, such as the ACC-14E, 65E, 66E, and 67E digital I/O boards. The following table shows which variable corresponds to which card:

Ident I-var	Board DIP Switch 4,3,2,1 Setting[1]	Board Base Address	Board Ident. Info Address[2]
I4950	0000 (0)	$078C00	$078F30
I4951	0001 (1)	$078D00	$078F34
I4952	0010 (2)	$078E00	$078F38
I4953	0011 (3)	$078F00	$078F3C
I4954	0100 (4)	$079C00	$079F30
I4955	0101 (5)	$079D00	$079F34
I4956	0110 (6)	$079E00	$079F38
I4957	0111 (7)	$079F00	$079F3C
I4958	1000 (8)	$07AC00	$07AF30
I4959	1001 (9)	$07AD00	$07AF34
I4960	1010 (10)	$07AE00	$07AF38
I4961	1011 (11)	$07AF00	$07AF3C
I4962	1100 (12)	$07BC00	$07BF30
I4963	1101 (13)	$07BD00	$07BF34
I4964	1110 (14)	$07BE00	$07BF38
I4965	1111 (15)	$07BF00	$07BF3C

Notes:

1. Currently, board DIP-switches SW1-1 to SW1-4 are used to set the addresses of the boards on the UBUS backplane. A '0' is ON (CLOSED); a '1' is OFF (OPEN). The E1 jumper on the back of the ACC-Ux UBUS backplane board must be ON to use the DIP-switch addressing. SW1-5 and SW1-6 must be ON (CLOSED).
2. For diagnostic purposes only. The four "Y" registers starting at this address contain the information used to assemble this I-variable.

Note

The ACC-9E, 10E, 11E and 12E I/O boards do not report identification information for this variable.

I4950 to I4965 have multiple fields of information, which can be reported individually or in groups, depending on the setting of I39. The following table shows what each field reports.

Information	Reported when:	Bits when I39=0	Bits when I39>0
Vendor ID	I39 = 0 or 1	0 – 7	0 – 7
Options	I39 = 0 or 2	8 – 17	0 – 9
Revision #	I39 = 0 or 3	18 – 21	0 – 3
Card ID	I39 = 0 or 4	22 – 35	0 – 13
Base Address	I39 = 5	--	0 - 18

- The Vendor ID field is an 8-bit value that reports the manufacturer of the board. Delta Tau boards report a value of 1 in this field.
- The Options field is a 10-bit field that is typically used as a set of individual bits to report which options are present on the board. The meaning of each bit is board-dependent.
- The Revision Number field is a 4-bit value that represents the design revision (0 to 15) of the board. For Delta Tau boards, this value matches the "x" in the "–10x" part number suffix for the board.
- The Card ID field is a 14-bit value that represents the part number of the board. For Delta Tau boards, this value matches the "xxxx" in the "60xxxx" (decimal) main part number for the board.
- The Base Address field is a 19-bit value that represents the starting address of the board in the Turbo PMAC's address space.

Data Gathering I-Variables

I5000 Data Gathering Buffer Location and Mode

Range: 0 - 3

Units: none

Default: 0

I5000 controls where the data gathering buffer will be located when it is defined, and whether it will wrap around when it is filled. It can take the following values:

- 0: Locate buffer in regular RAM. Do not permit wrap-around (stop gathering when end of buffer is reached). This setting must be used for Turbo PMAC Executive program data gathering and tuning routines.
- 1: Locate buffer in regular RAM. Permit wrap-around upon reaching end of buffer. Note: wrap-around feature not supported by Turbo PMAC Executive program data gathering and tuning routines.
- 2: Locate buffer in dual-ported RAM (Turbo PMAC Option 2 required). Do not permit wrap-around. Not very useful.
- 3: Locate buffer in dual-ported RAM (Turbo PMAC Option 2 required). Permit wrap-around upon reaching end of buffer (usual mode for dual-ported RAM).

Note:

Normally, this parameter is set automatically by the PMAC Executive Program's gathering and tuning routines.

I5001 – I5048 Data Gathering Source 1-48 Address

Range: $000000 - $C7FFFF

Units: Modified Turbo PMAC Addresses

Default: $000000

I5001 through I5048 specify the addresses of the 48 possible sources to be read by the data gathering function. I50*nn* specifies the address of source number *nn*.

These variables are 24-bit values, usually represented by 6 hexadecimal digits. The last 5 digits (bits 0 to 19; bit 19 must be 0) represent the numerical address of the register.

The first hex digit controls which part of the address is to be read. It can take one of 4 possible values:

- $0: Y-register only (24 bits)
- $4: X-register only (24 bits)
- $8: X/Y double register (48 bits), Executive program interprets as integer
- $C: X/Y double register (48 bits), Executive program interprets as floating-point

The address specified by one of these variables is only gathered if the I5050 or I5051 selection mask enables the gathering of that particular source.

Note:

Normally, these parameters are set automatically by the PMAC Executive Program's gathering and tuning routines.

I5049 Data Gathering Period

Range: 0 - 8,388,607

Units: Servo Cycles

Default: 1

I5049 controls how often data is logged from source addresses when data gathering is enabled, in units of servo interrupt cycles. If I5049 is set to 0, data is logged only once per data gathering command (single-shot mode).

Note:

Normally, this parameter is set automatically by the PMAC Executive Program's gathering and tuning routines.

I5050 Data Gathering Selection Mask 1

Range: $000000 - $FFFFFF

Units: Individual Bits

Default: $000000

I5050 controls which of the 24 potential data sources specified by I5001 to I5024 will be gathered when data gathering is performed. It is a 24-bit value and each bit controls one potential data source. A '1' in the I5050 bit enables the gathering of the data source; a '0' in the I5050 bit disables the gathering of the data source.

The following table shows the relationship between bits of I5050 and the data gathering source address I-variables:

Bit #	Value	I-Variable Enabled	Bit #	Value	I-Variable Enabled
0	$1	I5001	12	$1000	I5013
1	$2	I5002	13	$2000	I5014
2	$4	I5003	14	$4000	I5015
3	$8	I5004	15	$8000	I5016
4	$10	I5005	16	$10000	I5017
5	$20	I5006	17	$20000	I5018
6	$40	I5007	18	$40000	I5019
7	$80	I5008	19	$80000	I5020
8	$100	I5009	20	$100000	I5021
9	$200	I5010	21	$200000	I5022
10	$400	I5011	22	$400000	I5023
11	$800	I5012	23	$800000	I5024

Note:

Normally, this parameter is set automatically by the PMAC Executive Program's gathering and tuning routines.

I5051 Data Gathering Selection Mask 2

Range: $000000 - $FFFFFF

Units: Individual Bits

Default: $000000

I5051 controls which of the 24 potential data sources specified by I5025 to I5048 will be gathered when data gathering is performed. It is a 24-bit value and each bit controls one potential data source. A '1' in the I5051 bit enables the gathering of the data source; a '0' in the I5051 bit disables the gathering of the data source. The following table shows the relationship between bits of I5051 and the data gathering source address I-variables:

Bit #	Value	I-Variable Enabled	Bit #	Value	I-Variable Enabled
0	$1	I5025	12	$1000	I5037
1	$2	I5026	13	$2000	I5038
2	$4	I5027	14	$4000	I5039
3	$8	I5028	15	$8000	I5040
4	$10	I5029	16	$10000	I5041
5	$20	I5030	17	$20000	I5042
6	$40	I5031	18	$40000	I5043
7	$80	I5032	19	$80000	I5044
8	$100	I5033	20	$100000	I5045
9	$200	I5034	21	$200000	I5046
10	$400	I5035	22	$400000	I5047
11	$800	I5036	23	$800000	I5048

Note:

Normally, this parameter is set automatically by the PMAC Executive Program's gathering and tuning routines.

A/D Processing Table I-Variables

I5060 A/D Processing Ring Size

Range: 0 - 16

Units: Number of A/D Pairs

Default: 0

I5060 controls the number of pairs of multiplexed A/D converters, either from on-board Option 12 ADCs, or off-board ACC-36 ADCs, that are processed and "de-multiplexed" into individual registers. If I5060 is set to 0, none of these A/D converters is processed automatically.

If I5060 is set to a value greater than 0, it specifies the number of pairs of ADCs in the automatic processing ring. Each phase clock cycle, one pair is processed, and the values copied into image registers in RAM.

Global I-variable I7 permits most phasing tasks, such as motor commutation and digital current loop closure, to skip some phase clock cycles. This A/D de-multiplexing occurs every phase clock cycle, regardless of the setting of I7. This permits the de-multiplexing to occur at a very high frequency with out overloading Turbo PMAC with phase calculations.

For each pair enabled, one of the A/D ring slot pointer I-variables I5061-I5076 and one of the A/D ring convert code I-variables I5081-I5096 must be set properly. If I5060 is set to 1, then I5061 and I5081 must be set properly; if I5060 is set to 2, then I5061, I5062, I5081, and I5082 must be set properly.

I5060 is actually used at power-on/reset only, so to make a change in the A/D de-multiplexing ring size, including activating or de-activating the function, change the value of I5060, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command.

I5061-I5076 A/D Ring Slot Pointers

Range: $000000 - $7FFFFF

Units: Turbo PMAC Addresses

Default: $0 (specifies address $078800)

I5061 through I5076 control which of the multiplexed A/D converters are read in the A/D ring table, as enabled by I5060. These I-variables contain the Turbo PMAC addresses where these ADCs can reside.

If the A/D converters are in the on-board Option 12 or 12A (Turbo PMAC2 only), or if they are in the ACC-1E or 6E 3U stack board, they reside at address $078800, and the I-variable pointing to them can be set either to $0 or to $078800. If they are in the on-board Option 12 or 12A on a Turbo PMAC(1)-PCI, they reside at address $078008.

The following table shows the proper value of I5061-I5076 for A/D converters on ACC-36P and 36V accessory boards:

ADC Location	Board # Jumper ON	Board Letter Jumper ON	I5061 – I5076 Address	ADC Location	Board # Jumper ON	Board Letter Jumper ON	I5061 – I5076 Address
ACC-36 #1A	E1	E7	$078A00	ACC-36 #4A	E4	E7	$078D00
ACC-36 #1B	E1	E8	$078A02	ACC-36 #4B	E4	E8	$078D02
ACC-36 #1C	E1	E9	$078A04	ACC-36 #4C	E4	E9	$078D04
ACC-36 #1D	E1	E10	$078A06	ACC-36 #4D	E4	E10	$078D06
ACC-36 #2A	E2	E7	$078B00	ACC-36 #5A	E5	E7	$078E00
ACC-36 #2B	E2	E8	$078B02	ACC-36 #5B	E5	E8	$078E02
ACC-36 #2C	E2	E9	$078B04	ACC-36 #5C	E5	E9	$078E04
ACC-36 #2D	E2	E10	$078B06	ACC-36 #5D	E5	E10	$078E06
ACC-36 #3A	E3	E7	$078C00	ACC-36 #6A	E6	E7	$078F00
ACC-36 #3B	E3	E8	$078C02	ACC-36 #6B	E6	E8	$078F02
ACC-36 #3C	E3	E9	$078C04	ACC-36 #6C	E6	E9	$078F04
ACC-36 #3D	E3	E10	$078C06	ACC-36 #6D	E6	E10	$078F06

The following table shows the values for A/D converters on UMAC ACC-36E and ACC-59E boards, based on the settings of the address DIP switches SW1-*n* on those boards:

SW1-1	SW1-2	SW1-3	SW1-4	I5061-I5076 Address
ON	ON	ON	ON	$078C00
OFF	ON	ON	ON	$078D00
ON	OFF	ON	ON	$078E00
OFF	OFF	ON	ON	$078F00
ON	ON	OFF	ON	$079C00
OFF	ON	OFF	ON	$079D00
ON	OFF	OFF	ON	$079E00
OFF	OFF	OFF	ON	$079F00
ON	ON	ON	OFF	$07AC00
OFF	ON	ON	OFF	$07AD00
ON	OFF	ON	OFF	$07AE00
OFF	OFF	ON	OFF	$07AF00
ON	ON	OFF	OFF	$07BC00
OFF	ON	OFF	OFF	$07BD00
ON	OFF	OFF	OFF	$07BE00
OFF	OFF	OFF	OFF	$07BF00
Note: SW1-5 and SW1-6 must be ON to enable this addressing.				

Each variable I5061 – I5076 is matched with the I-variable numbered 20 higher (e.g. I5081 for I5061) to specify which channel of the muxed A/D-converter is to be used, and how that channel is to be read. Up to 8 of these I-variable pairs must be used to read all 8 channels of a muxed A/D converter – the eight variables in the I5061 – I5076 range will all contain the same address.

The results of the A/D tables are placed in registers at addresses Y:$003400 to Y:$00341F, using bits 12 to 23 of these registers. The value of the A/D converter found in the low 12 bits of the source register is placed in the register with the even-numbered address; the value of the A/D converter found in the high 12 bits of the source register is placed in the register with the odd-numbered address. The following table shows the matching between the A/D pointer I-variables and the addresses of the result registers.

I-Variable	Result Address for Low ADC	Result Address for High ADC	I-Variable	Result Address for Low ADC	Result Address for High ADC
I5061	Y:$003400	Y:$003401	I5069	Y:$003410	Y:$003411
I5062	Y:$003402	Y:$003403	I5070	Y:$003412	Y:$003413
I5063	Y:$003404	Y:$003405	I5071	Y:$003414	Y:$003415
I5064	Y:$003406	Y:$003407	I5072	Y:$003416	Y:$003417
I5065	Y:$003408	Y:$003409	I5073	Y:$003418	Y:$003419
I5066	Y:$00340A	Y:$00340B	I5074	Y:$00341A	Y:$00341B
I5067	Y:$00340C	Y:$00340D	I5075	Y:$00341C	Y:$00341D
I5068	Y:$00340E	Y:$00340F	I5076	Y:$00341E	Y:$00341F

I5061 – I5076 are actually used at power-on/reset only, so to make a change in the A/D de-multiplexing sources, change the values of I5061 – I5076, store these new values to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command.

I5080 A/D Ring Convert Enable

Range: 0 - 1

Units: none

Default: 1

I5080 controls whether the A/D-converter demultiplexing algorithm specified by I5060 – I5076 and I5081 – I5096 is enabled or not. If I5080 is set to 1, the algorithm is enabled; if I5080 is set to 0, the algorithm is disabled.

If the saved value of I5060 is greater than 0, specifying that some demultiplexing is to be done, then I5080 is automatically set to 1 on power-up/reset, so the algorithms are automatically running. By subsequently setting I5080 to 0, the user can suspend the execution of these algorithms, to be resumed by setting I5080 back to 1. If the save value of I5060 is 0, then I5080 is automatically set to 0 on power-up/reset.

I5081-I5096 A/D Ring Convert Codes

Range: $000000 - $00F00F

Units: None

Default: $000000

I5081 through I5096 contain the convert codes written to the multiplexed A/D converters that are read in the A/D ring table, as enabled by I5060. The convert codes control which of the multiplexed ADCs at the address is to be read, and the range of the analog input for that ADC. The ADCs can be on-board the Turbo PMAC with Option 12 and 12A, or off-board with an ACC-36P/V. The Turbo PMAC address of the ADC to be read is set by the I-variable number 20 less (e.g. I5061 determines the address of the ADC whose convert code is set by I5081). The number of ADC converters in the ring is determined by I5060.

I5081-I5096 are 24-bit values, represented by 6 hexadecimal digits. Legitimate values are of the format $00*m*00*n*, where *m* and *n* can take any hex value from 0 through F.

For the on-board Option 12 & 12A ADCs on a Turbo PMAC2, the *m* value determines which of the inputs ANAI08 to ANAI15 that come with Option 12A is to be read, and how it is to be converted, according to the following formulas:

$m = ANAI\#-8$; 0 to +5V unipolar input

$m = ANAI\#$; -2.5V to +2.5V bipolar input

For the on-board Option 12 & 12A ADCs on a Turbo PMAC2, the *n* value determines which of the inputs ANAI00 to ANAI07 that come with Option 12A is to be read, and how it is to be converted, according to the following formulas:

$n = ANAI\#$; 0V to +5V unipolar input

$n = ANAI\#+8$; -2.5V to +2.5V bipolar input

For example, to read ANAI02 from Option 12 and ANAI10 from Option 12A, both as +/-2.5V inputs, into the first slot in the ring, m would be set to A (10) and n would be set to A (10), so I5081 would be set to $00A00A.

For the off-board ACC-36P/V ADCs, the *m* value is always 0, and the *n* value determines which pair of ADCs is to be read, and how they are to be converted, according to the following formulas:

$n = ADC\#-1, ADC\#-9$; 0 to +10V (between + and -) unipolar inputs

$n = ADC\#+7, ADC\#-1$; -5V to +5V (between + and -) bipolar inputs

For example, to read ADC3 and ADC11 of an ACC-36 as 0-10V inputs into the second slot in the ring, *n* would be set to 2, so I5082 would be set to $000002.

I5081 – I5096 are actually used at power-on/reset only, so to make a change in the A/D de-multiplexing codes, change the values of I5081 – I5096, store these new values to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command.

Coordinate System I-Variables

I-Variables in the I5100s through the I6600s control the setup of the 16 possible coordinate systems on a Turbo PMAC, Coordinate System 1 through Coordinate System 16. Each group of 100 I-variables is reserved for a specific coordinate system: the I5100s for C.S. 1, the I5200s for C.S. 2, and so on, to the I6600s for C.S. 16. The following table lists the I-variables used for each coordinate system:

C.S. #	I-Variables	C.S. #	I-Variables	C.S. #	I-Variables	C.S. #	I-Variables
1	I5100-5199	5	I5500-5599	9	I5900-5999	13	I6300-6399
2	I5200-5299	6	I5600-5699	10	I6000-6099	14	I6400-6499
3	I5300-5399	7	I5700-5799	11	I6100-6199	15	I6500-6599
4	I5400-5499	8	I5800-5899	12	I6200-6299	16	I6600-6699

In the generic description of these I-variables, the "thousands" digit is represented by the letter 's', and the "hundreds" digit by the letter 'x'; for example, Isx11. 's' and 'x' can take the following values:
's' is equal to '5' for Coordinate Systems 1 – 9;
's' is equal to '6' for Coordinate Systems 10 – 16;
'x' is equal to the coordinate system number for Coordinate Systems 1 – 9;
'x' is equal to the (coordinate system number minus 10) for Coordinate Systems 10 – 16.

The descriptions of the variables refer to "Coordinate System 'x'" generically, even though the coordinate system number is equal to (x + 10) for Coordinate Systems 10 – 16.

Isx11 Coordinate System 'x' User Countdown Timer 1

Range: -8,388,608 - 8,388,607

Units: servo cycles

Default: 0

Isx11 provides an automatic countdown timer for user convenience. If Coordinate System 'x' is activated by I68, Isx11 will count down one unit per servo cycle. The user may write to this variable at any time, and it will count down from that value. Typically user software will then wait until the variable is less than another value, usually zero. The software accessing Isx11 does not have to be associated with Coordinate System 'x'.

Isx11 is a signed 24-bit variable, providing a range of -2^{23} (-8,388,608) to $+2^{23}-1$ (8,388,607). If active, it counts down continually until it reaches its maximum negative value of -8,388,608. It will not roll over. Most people will just use the positive range, writing a number representing the number of servo cycles for the period to the variable, then waiting for it to count down past 0.

The following code shows how Isx11 could be used in a PLC to turn on an output for a fixed period of time.

```
M1=1                ; Set the output
I5111=2259          ; Set the timer to 1 second (2259 servo cycles)
WHILE (I5111>0)     ; Wait for timer to count down
ENDWHILE
M1=0                ; Clear the output
```

Isx12 Coordinate System 'x' User Countdown Timer 2

Range: -8,388,608 - 8,388,607

Units: servo cycles

Default: 0

Isx12 provides an automatic countdown timer for user convenience. If Coordinate System 'x' is activated by I68, Isx12 will count down one unit per servo cycle. The user may write to this variable at any time, and it will count down from that value. Typically user software will then wait until the variable is less than another value, usually zero. The software accessing Isx12 does not have to be associated with Coordinate System 'x'.

Isx12 is a signed 24-bit variable, providing a range of -2^{23} (-8,388,608) to $+2^{23}-1$ (8,388,607). If active, it counts down continually until it reaches its maximum negative value of -8,388,608. It will not roll over. Most people will just use the positive range, writing a number representing the number of servo cycles for the period to the variable, then waiting for it to count down past 0.

The following code shows how Isx12 could be used in a PLC to turn on an output for a fixed period of time.

```
M1=1                ; Set the output
I5112=2259          ; Set the timer to 1 second (2259 servo cycles)
WHILE (I5112>0)     ; Wait for timer to count down
ENDWHILE
M1=0                ; Clear the output
```

Isx13 Coordinate System 'x' Segmentation Time

Range: 0 - 255

Units: msec

Default: 0

Isx13 controls whether Coordinate System 'x' is in "segmentation mode" or not, and if it is, what the "segmentation time" is in units of milliseconds.

If Isx13 is greater than zero, Coordinate System 'x' is in segmentation mode, and all **LINEAR** and **CIRCLE** mode trajectories are created by computing intermediate "segment" points with a coarse interpolation algorithm every Isx13 milliseconds, then executing a fine interpolation using a cubic spline algorithm every servo cycle.

While it is possible to execute programmed moves ("blocks") of a shorter time than the Isx13 segmentation time, the segmentation algorithm will automatically skip over these blocks, effectively performing a smoothing function over multiple blocks.

This coarse/fine interpolation method activated by putting the coordinate system into segmentation mode is required for the coordinate system to be able to use any of the following features:

- Circular interpolation
- Cutter radius compensation
- '/' Program stop command
- '\' Quick-stop command
- Rotary buffer blend-on-the-fly
- Multi-block lookahead
- Inverse kinematics

If none of these features is required in the coordinate system, it is usually best to leave Isx13 at the default value of 0 to free up calculation time for other tasks. If Isx13 is 0, **CIRCLE** mode moves are executed as **LINEAR** moves, cutter radius compensation is not performed, '/' commands are executed as '**Q**' quit commands, '\' commands are executed as '**H**' feed-hold commands.

Typical values of Isx13 for segmentation mode are 5 to 20 msec. The smaller the value, the tighter the fit to the true curve, but the more computation is required for the moves, and the less is available for background tasks. If Isx13 is set too low, Turbo PMAC will not be able to do all of its move calculations in the time allotted, and it will stop the motion program with a run-time error.

The formula for the interpolation error introduced on a curved path by the segmentation mode is:

$$E = \frac{V^2 T^2}{6R}$$

where V is the velocity, T is the segmentation time, and R is the local radius, all expressed in consistent units. On a straight-line path, R is infinite, making the error equal to 0. If the velocity is expressed as a feedrate F, in units per minute, the formula is:

$$E = \frac{F^2\left(\frac{units^2}{min^2}\right)*\left(\frac{min^2}{60{,}000^2\, m\,sec^2}\right)*(Isx13)^2\left(m\,sec^2\right)}{6R(units)} = \frac{F^2(Isx13)^2}{2.16\times 10^{10} R}$$

On non-Turbo PMACs, this function is controlled by global I-variable I13.

Example:

At a feedrate of 5000 mm/min (200 in/min), and a radius of 50 mm (2 in), a value of Isx13 of 10 msec produces an interpolation error of 2.3 μm (0.00009 in).

Isx20 Coordinate System 'x' Lookahead Length

Range: 0 – 65,535

Units: Isx13 segmentation periods

Default: 0

Isx20 controls the enabling of the lookahead buffering function for Coordinate System 'x', and if enabled, determines how far ahead the buffer will look ahead.

If Isx20 is set to 0 (the default), the buffered lookahead function is not used, even if a lookahead buffer has been defined.

If Isx20 is set to 1, points are stored in the lookahead buffer as they are calculated, but no lookahead velocity or acceleration-limiting calculations are done. The stored points can then be used to back up along the path as necessary.

If Isx20 is set to a value greater than 1, PMAC will look Isx20 segments ahead on LINEAR and CIRCLE mode moves, provided that the coordinate system is in segmentation mode (Isx13 > 0) and a lookahead buffer has been defined. The lookahead algorithm can extend the time for each segment in the buffer as needed to keep velocities under the Ixx16 limits and the accelerations under the Ixx17 limits.

For proper lookahead control, Isx20 must be set to a value large enough so that PMAC looks ahead far enough that it can create a controlled stop from the maximum speed within the acceleration limit. This required stopping time for a motor could be expressed as:

$$StopTime = \frac{V_{max}}{A_{max}} = \frac{Ixx16}{Ixx17}$$

All motors in the coordinate system should be evaluated to see which motor has the longest stopping time. This motor's stopping time will be used to compute Isx20.

The average speed during this stopping time is $V_{max}/2$, so as the moves enter the lookahead algorithm at V_{max} (the worst case), the required time to look ahead is *StopTime*/2. Therefore, the required number of segments always corrected in the lookahead buffer can be expressed as:

$$SegmentsAhead = \frac{StopTime(msec)/2}{SegTime(msec/seg)} = \frac{Ixx16}{2 * Ixx17 * Isx13}$$

Because Turbo PMAC does not completely correct the lookahead buffer as each segment is added, the lookahead distance specified by Isx20 must be slightly larger than this. The formula for the minimum value of Isx20 that guarantees sufficient lookahead for the stopping distance is:

$$Isx20 = \frac{4}{3} * SegmentsAhead$$

If a fractional value results, round up to the next integer. A value of Isx20 less than this amount will not result in velocity or acceleration limits being violated; however, the algorithm will not permit maximum velocity to be reached, even if programmed.

Isx20 should not be set greater than the number of segments reserved in the **DEFINE LOOKAHEAD** command. If the lookahead algorithm runs out of buffer space, Turbo PMAC will automatically reduce Isx20 to reflect the amount of space that is available

Example:
The axes in a system have a maximum speed of 24,000 mm/min, or 400 mm/sec (900 in/min or 15 in/sec). They have a maximum acceleration of 0.1g or 1000 mm/sec^2 (40 in/sec^2), and a count resolution of 1μm. A maximum block rate of 200 blocks/sec is desired, so Isx13 is set to 5 msec. The parameters can be computed as:

- Ixx16 = 400 mm/sec * 0.001 sec/msec * 1000 cts/mm = 400 cts/msec
- Ixx17 = 1000 mm/sec^2 * 0.001^2 sec^2/msec^2 * 1000 cts/mm = 1.0 cts/msec^2
- Isx20 = [4/3] * [400 cts/msec / (2 * 1.0 cts/msec^2 *5 msec/seg)] = 54 segments

Isx21 Coordinate System 'x' Lookahead State Control

Range: 0 – 15

Units: none

Default: 0

Isx21 permits direct control of the state of lookahead execution, without going through Turbo PMAC's background command interpreter. This is useful for applications such as wire EDM, which can require very quick stops and reversals.

- Setting Isx21 to 4 is the equivalent of issuing the **** quick-stop command.
- Setting Isx21 to 6 is the equivalent of resuming forward motion with the **>** resume forward command.
- Setting Isx21 to 7 is the equivalent of issuing the **<** back-up command.
- Setting Isx21 to 14 requests execution of a single segment in the forward direction.
- Setting Isx21 to 15 requests execution of a single segment in the reverse direction.

If you are monitoring Isx21 at other times, you will see that the "4's" bit is cleared after the command has been processed. Therefore, you will see the following values:

- Isx21 = 0 when stopped with a quick-stop command.
- Isx21 = 2 when running forward in lookahead.
- Isx21 = 3 when running reversed in lookahead.

- Isx21 = 10 when has executed a single forward segment in lookahead.
- Isx21 = 11 when has executed a single reverse segment in lookahead.

Note:

In preliminary versions of the Turbo PMAC firmware, Isx21 served a different function. That variable value is now a constant value (3) set by the firmware.

Isx50 Coordinate System 'x' Kinematic Calculations Enable

Range: 0 – 1

Units: none

Default: 0

Isx50 controls whether the special forward-kinematic and inverse-kinematic program buffers for Coordinate System 'x' are used to relate the motor and axis positions for the coordinate system.

If Isx50 is set to 0 (the default), Turbo PMAC will use the relationships set up in the axis-definition statements for the coordinate system to compute the relationship between motor positions and axis positions. The inverse of the axis-definition equations is used to compute the starting axis positions on an **R** (run), **S** (step), or **PMATCH** command. The axis-definition equations are used to convert programmed axis positions to motor positions each programmed move or move segment. Even if the forward-kinematic and inverse-kinematic programs have been loaded for the coordinate system, they will not be used.

If Isx50 is set to 1, Turbo PMAC will use the relationships set up in the special kinematic program buffers to compute the relationship between motor positions and axis positions. The forward-kinematic program is used to compute the starting axis positions on an **R** (run), **S** (step), or **PMATCH** command. The inverse-kinematic program is used to convert programmed axis positions to motor positions each programmed move or move segment for each motor defined as an inverse-kinematic axis (**#xx->I**). Motors in the coordinate system not defined as inverse-kinematic axes still use axis-definition equations to convert programmed axis positions to motor positions.

Isx53 Coordinate System 'x' Step Mode Control

Range: 0 - 1

Units: none

Default: 0

Isx53 controls the action of a Step (**S**) command in Coordinate System 'x'. At the default Isx53 value of 0, a Step command causes program execution through the next move, **DELAY**, or **DWELL** command in the motion program, even if this takes multiple program lines.

When Isx53 is set to 1, a Step command causes execution of only a single program line, even if there is no move, **DELAY**, or **DWELL** command on that line. If there is more than one **DWELL** or **DELAY** command, a single Step command will only execute one of the **DWELL** or **DELAY** commands.

Regardless of the setting of Isx53, if program execution on a Step command encounters a **BLOCKSTART** statement in the program, execution will continue until a **BLOCKSTOP** statement is encountered.

On non-Turbo PMACs, this function is controlled by global I-variable I53.

Isx86 Coordinate System 'x' Alternate Feedrate

Range: positive floating point

Units: (user position units) / (Isx90 feedrate time units)

Default: 1000.0

Isx86 can control the speed of motion for a feedrate-specified move when the motion of "non-feedrate" axes is predominant. "Feedrate", or "vector-feedrate" axes are those specified by the **FRAX** command; X, Y, and Z are the feedrate axes by default.

If Isx86 is greater than 0, PMAC compares the move time for the vector feedrate axes, computed as the vector distance of the feedrate axes divided by the specified feedrate (the **F** value in the program or Ix89), to the move time for the non-feedrate axes, computed as the longest distance for these axes divided by Isx86. It then uses the longer of these two times as the move time for all axes, feedrate and non-feedrate.

If Isx86 is 0, and PMAC sees a feedrate-specified move in which the vector distance is zero (i.e. no motion of the vector feedrate axes), PMAC computes the move time as the longest distance of the non-feedrate axes on the line divided by the program feedrate.

Isx86 has two main uses. First, it automatically controls the motion of "non-feedrate" axes when they are commanded alone on a line in feedrate mode. Typically, these are rotary axes in a combined linear/rotary system where only the linear axes are vector feedrate axes.

Second, it permits a fast "dry-run" mode in which the programmed feedrates are ignored. If no axes in the coordinate system are vector feedrate axes (implemented with the **NOFRAX** command), then Isx86 will be used for all moves, regardless of the **F** values in the program.

Example:

```
I5190=1000      ; Speeds are specified as per-second
I5186=5         ; Alternate feedrate of 5 user units per second
INC             ; Moves specified by distance
X20 F10         ; Move time = 20 units / 10 (units/sec) = 2 sec
X10 C20         ; Move time = 10 units / 10 (units/sec) = 1 sec
C20             ; Move time = 20 units / 5 (units/sec) = 4 sec
```

See Also:

I-variables Isx89, Isx98

On-line commands **FRAX**, **NOFRAX**

Motion program commands **F**, **FRAX**, **NOFRAX**

Isx87 Coordinate System 'x' Default Program Acceleration Time

Range: 0 - 8,388,607

Units: msec

Default: 0 (so Isx88 controls)

Isx87 sets the default time for commanded acceleration for programmed blended **LINEAR** and **CIRCLE** mode moves in Coordinate System 'x'. If global variable I42 is set to 1, it also sets the default time for **PVT** and **SPLINE** mode moves. The first use of a **TA** statement in a program overrides this value.

Note:

Even though this parameter makes is possible not to specify acceleration time in the motion program, you are strongly encouraged to use **TA** in the program and not rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 "G-Codes "). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

If the specified S-curve time (see Isx88, below) is greater than half the specified acceleration time, the time used for commanded acceleration in blended moves will be twice the specified S-curve time.

The acceleration time is also the minimum time for a blended move; if the distance on a feedrate-specified (F) move is so short that the calculated move time is less than the acceleration time, or the time of a time-specified (TM) move is less than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move. If the acceleration time is 0 because both TA and TS are set to 0, the minimum move time is 0.5 msec.

Note:

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ixx17) for a programmed **LINEAR**-mode move with Isx13=0 (no move segmentation).

When polled, Isx87 will report the value from the most recently executed **TA** command in that coordinate system.

Isx88 Coordinate System 'x' Default Program S-Curve Time

Range: 0 - 8,388,607

Units: msec

Default: 50

Isx88 sets the default time in each "half" of the "S" in S-curve acceleration for programmed blended **LINEAR** and **CIRCLE** mode moves in coordinate system 'x'. It does not affect **SPLINE**, **PVT**, or **RAPID** mode moves. The first use of a **TS** statement in a program overrides this value.

Note:

Even though this parameter makes is possible not to specify acceleration time in the motion program, you are strongly encouraged to use **TS** in the program and not rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 "G-Codes"). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

If Isx88 is zero, the acceleration is constant throughout the Isx87 time and the velocity profile is trapezoidal.

If Isx88 is greater than zero, the acceleration will start at zero and linearly increase through Isx88 time, then stay constant (for time TC) until Isx87-Isx88 time, and linearly decrease to zero at Isx87 time (that is Isx87=2*Isx88 - TC).

If Isx88 is equal to Isx87/2, the entire acceleration will be spec in S-curve form (Isx88 values greater than Isx87/2 override the Isx87 value; total acceleration time will be 2*Isx88).

Note:

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ixx17) for a programmed LINEAR-mode move with Isx13=0 (no move segmentation).

When polled, Isx88 will report the value from the most recently executed **TS** command in that coordinate system.

Isx89 Coordinate System 'x' Default Program Feedrate/Move Time

Range: positive floating point

Units: (user position units) / (Isx90 feedrate time units) or msec

Default: 1000.0

Isx90 sets the default feedrate (commanded speed) for programmed **LINEAR** and **CIRCLE** mode moves in coordinate system 'x'. The first use of an **F** or **TM** statement in a motion program overrides this value. The velocity units are determined by the position and time units, as defined by axis definition statements and Isx90. After power-up/reset, the coordinate system is in feedrate mode, not move-time mode.

Note:

You are strongly encouraged *not* to rely on this parameter and to declare your feedrate in the program. This will keep your move parameters with your move commands, lessening the chances of future errors, and making debugging easier.

When polled, Isx89 will report the value from the most recently executed **F** or **TM** command in that coordinate system.

Isx90 Coordinate System 'x' Feedrate Time Units

Range: positive floating point

Units: msec

Default: 1000.0

Isx90 defines the time units used in commanded velocities (feedrates) in motion programs executed by Coordinate System 'x'. Velocity units are comprised of length or angle units divided by time units. The length/angle units are determined in the axis definition statements for the coordinate system.

Isx90 sets the time units. Isx90 itself has units of milliseconds, so if Isx90 is 60,000, the time units are 60,000 milliseconds, or minutes. The default value of Ix90 is 1000 msec, specifying velocity time units of seconds.

This affects two types of motion program values: **F** values (feedrate) for **LINEAR**- and **CIRCLE**-mode moves; and the velocities in the actual move commands for **PVT**-mode moves.

Isx91 Coordinate System 'x' Default Working Program Number

Range: 0 - 32,767

Units: Motion Program Numbers

Default: 0

Isx91 tells Turbo PMAC which motion program to run in Coordinate System 'x' when commanded to run from the control-panel input (START/ or STEP/ line taken low, or its equivalent in DPRAM). It performs the same function for a hardware run command as the **B** command does for a software run command (**R**). It is intended primarily for stand-alone Turbo PMAC applications. The first use of a **B** command from a host computer for this coordinate system overrides this parameter.

Isx92 Coordinate System 'x' Move Blend Disable

Range: 0 - 1

Units: none

Default: 0

Isx92 controls whether programmed moves for Coordinate System 'x' are automatically blended or not. If this parameter set to 0, programmed moves -- **LINEAR**, **SPLINE**, and **CIRCLE**-mode -- are blended together with no intervening stop. Upcoming moves are calculated during the current moves. If this parameter is set to 1, there is a brief stop in between each programmed move (it effectively adds a **DWELL 0** command), during which the next move is calculated. The calculation time for the next move is determined by Isx11.

This parameter is only acted upon when the **R** or **S** command is given to start program execution. To change the mode of operation while the program is running the "continuous motion request" coordinate system status bit must be changed. The 0/1 polarity of this bit is opposite that of Isx92.

Isx93 Coordinate System 'x' Time Base Control Address

Range: $000000 - $FFFFFF

Units: Turbo PMAC X-Addresses

Default:

I-Var.	Default	Register	I-Var.	Default	Register
I5193	$002000	C.S.1 '%' Cmd. Reg.	I5993	$002800	C.S.9 '%' Cmd. Reg.
I5293	$002100	C.S.2 '%' Cmd. Reg.	I6093	$002900	C.S.10 '%' Cmd. Reg.
I5393	$002200	C.S.1 '%' Cmd. Reg.	I6193	$002A00	C.S.11 '%' Cmd. Reg.
I5493	$002300	C.S.2 '%' Cmd. Reg.	I6293	$002B00	C.S.12 '%' Cmd. Reg.
I5593	$002400	C.S.5 '%' Cmd. Reg.	I6393	$002C00	C.S.13 '%' Cmd. Reg.
I5693	$002500	C.S.6 '%' Cmd. Reg.	I6493	$002D00	C.S.14 '%' Cmd. Reg.
I5793	$002600	C.S.7 '%' Cmd. Reg.	I6593	$002E00	C.S.15 '%' Cmd. Reg.
I5893	$002700	C.S.8 '%' Cmd. Reg.	I6693	$002F00	C.S.16 '%' Cmd. Reg.

Isx93 tells Coordinate System 'x' where to look for its time base control ("feedrate override") information by specifying the address of the register that will be used. The default value of this parameter for each coordinate system (see above) specifies the register that responds to on-line **%** commands. If the time base is left alone, or is under host or programmatic control, this parameter should be left at the default.

Alternatively, if the time base is controlled externally from a frequency or voltage, usually the register containing the time-base information will be in the conversion table (which starts at address $003500).

If another register is to be used for the time base, it must have the units of I10 so that 8,388,608 (2^{23}) indicates 1 msec between servo interrupts. See instructions for using an external time base, under Synchronizing PMAC to External Events.

Note:

Isx93 contains the address of the register that holds the time-base value (it is a pointer to that register). Isx93 does not contain the time-base value itself.

Isx94 Coordinate System 'x' Time Base Slew Rate

Range: 0 - 8,388,607

Units: 2^{-23}msec / servo cycle

Default: 1644

Isx94 controls the rate of change of the time base for Coordinate System 'x'. It effectively works in two slightly different ways, depending on the source of the time base information. If the source of the time base is the "%" command register, then Isx94 defines the rate at which the "%" (actual time base) value will slew to a newly commanded value. If the rate is too high, and the % value is changed while axes in the coordinate system are moving, there will be a virtual step change in velocity. For this type of application, Isx94 is set relatively low (often 1000 to 5000) to provide smooth changes.

Note:

The default Isx94 value of 1644, when used on a card set up with the default servo cycle time of 442 µsec, provides a transition time between %0 and %100 of one second.

If there is a hardware source (as defined by Isx93), the commanded time-base value changes every servo cycle, and the rate of change of the commanded value is typically limited by hardware considerations (e.g. inertia). In this case, Isx94 effectively defines the *maximum* rate at which the "%" value can slew to the new hardware-determined value, and the actual rate of change is determined by the hardware. If you wish to keep synchronous to a hardware input frequency, as in a position-lock cam, Isx94 should be set high enough that the limit is never activated. However, following motion can be smoothed significantly with a lower limit if total synchronicity is not required.

Isx95 Coordinate System 'x' Feed Hold Slew Rate

Range: 0 - 8,388,607

Units: 2^{-23}msec / servo cycle

Default: 1644

Isx95 controls the rate at which the axes of Coordinate System 'x' stop if a feed hold command (**H**) is given, and the rate at which they start up again on a succeeding run command (**R** or **S**). A feed hold command is equivalent to a **%0** command except that it uses Isx95 for its slew rate instead of Isx94. Having separate slew parameters for normal time-base control and for feed hold commands allows both responsive ongoing time-base control (Ix94 relatively high) and well-controlled holds (Ix95 relatively low).

Note:

The default Isx95 value of 1644, when used on a card set up with the default servo cycle time of 442 µsec, provides a transition time between %100 and %0 (feed hold) of one second.

Isx96 Coordinate System 'x' Circle Error Limit

Range: Positive floating-point

Units: User length units

Default: 0 (function disabled)

In a circular arc move, a move distance that is more than twice the specified radius will cause a computation error because a proper path cannot be found. Sometimes, due to round-off errors, a distance slightly larger than twice the radius is given (for a half-circle move), and it is desired that this not create an error condition.

Isx96 allows the user to set an error limit on the amount the move distance is greater than twice the radius. If the move distance is greater than 2R, but by less than this limit, the move is done in a spiral fashion to the endpoint, and no error condition is generated. If the distance error is greater than this limit, the program will stop at the beginning of the move. Turbo PMAC will set this coordinate system's "circle radius error" status bit.

If Isx96 is set to 0, the error generation is disabled and any move distance greater than 2R is done in a spiral fashion to the endpoint.

Example:

Given the program segment

```
INC CIRCLE1 F2
X7.072 Y7.072 R5
```

Technically no circular arc path can be found, because the distance is SQRT(7.072^2+7.072^2) = 10.003, which is greater than twice the radius of 5. However, as long as Isx96 is greater than 0.003, PMAC will create a near-circular path to the end point.

Isx97 Coordinate System 'x' Minimum Arc Length

Range: Non-negative floating-point

Units: Semi-circles (π radians; 180 degrees)

Default: 0 (sets 2^{-20})

Isx97 sets the threshold between a short arc and a full circle for CIRCLE mode moves in Turbo PMAC's Coordinate System 'x'. Isx97 is expressed as an angle, with units that represent a fraction of a half-circle. It represents the smallest angle that can be covered by a programmed arc move.

Any programmed CIRCLE-mode move with an IJK-vector representation of the center that covers an angle smaller than Isx97 is executed as a full circle plus the programmed angle change. Any such move which covers an angle greater than Isx97 is executed as an arc smaller than a full circle.

The purpose of Isx97 is to support the circle programming standard that permits a full-circle move to be commanded simply by making the end point equal to the starting point (0-degree arc), yet allow for round-off errors.

Most users will be able to leave Isx97 at the default value of one-millionth of a semi-circle. This was formerly the fixed threshold value. However, some users may want to enlarge the threshold to compensate for round-off errors, particularly when using cutter-radius compensation in conjunction with full-circle moves. Remember that no arc covering an angle less than Isx97 can be executed.

If a full-circle move is commanded with cutter compensation on, and the blending from the previous move or into the next move creates a compensated outside corner without adding an arc (see Isx99), PMAC will extend the compensated move past a full circle. If Isx97 is too small, it may execute this as a very short arc, appearing to miss the move completely. Isx97 may have to be increased from its effective default value to cover this case.

For backward compatibility reasons, if Isx97 is set to 0, a threshold value of 2^{-20} (about one-millionth) of a semi-circle will be used.

Isx98 Coordinate System 'x' Maximum Feedrate

Range: Non-negative floating-point

Units: (user position units) / (Isx90 feedrate time units)

Default: 1000.0

Isx98 permits a maximum feedrate to be set for a coordinate system, preventing a program from accidentally exceeding a specified value. If Isx98 is greater than 0.0, Turbo PMAC will compare each commanded vector feedrate value from an **F** command in a motion program to Ix98. If the commanded feedrate is greater than Isx98, it will use Isx98 instead. This variable permits the system integrator to place a limit on the speed that a part programmer can command.

Other possible sources of commanded feedrate values, such as Isx86, Isx89, and **TM** commands, are not checked against Isx98.

This check of commanded feedrate against Isx98 is done at the programmed move block calculation time, before any lookahead calculations are done. Lookahead calculations compare individual motor velocities against Ixx16 limits; they do not check vector velocities.

If Isx98 is set to 0.0, Turbo PMAC will not check the programmed feedrate value against a limit.

See Also:
I-variables Isx86, Isx89
On-line command **FRAX**
Motion program commands **F**, **FRAX**, **TM**

Isx99 Coordinate System 'x' Cutter-Comp Outside Corner Break Point

Range: -1.0 - 0.9999

Units: cosine

Default: 0.998 (cos 1^{o})

Isx99 controls the threshold in Coordinate System 'x' between outside corner angles for which an extra arc move is added in cutter compensation, and those for which the incoming and outgoing moves are directly blended together.

Isx99 is expressed as the cosine of the change in directed angle between the incoming and outgoing moves. As such, it can take a value between -1.0 and +1.0. If the two moves have the same directed angle at the move boundary (i.e. they are moving in the same direction), the change in directed angle is 0, and the cosine is 1.0.

As the change in directed angle increases, the corner gets sharper, and the cosine of the change in directed angle decreases. For a total reversal, the change in directed angle is 180^{o}, and the cosine is -1.0.

If the cosine of the change in directed angle of an outside corner is less than Isx99 (a large change in directed angle; a sharp corner), PMAC will automatically add an arc move with a radius equal to the cutter radius to join the incoming and outgoing moves. This prevents the cutter from moving too far out when going around the outside of a sharp corner.

If the cosine of the change in directed angle of an outside corner is greater than Isx99 (a small change in directed angle; a gradual corner), PMAC will directly blend the incoming and outgoing

moves with its normal blending algorithms. This can provide increased speed on small angle changes, because an extra segment of minimum TA or 2*TS time is not added.

Isx99 does not affect the behavior at inside corners, where the incoming and outgoing moves are always blended directly together, regardless of the change in directed angle.

Example:

If it is desired that an arc only be added if the change in directed angle is greater than 45^o, then Isx99 should be set to 0.707, because $\cos \Delta\theta = \cos 45^o = 0.707$

Turbo PMAC2 MACRO IC I-Variables

I-Variables numbered in the I6800s and I6900s control hardware aspects of the MACRO ICs 0 to 3 of a Turbo PMAC2. These ICs control the operation of the MACRO ring on all PMAC2 boards. MACRO IC 0, a "DSPGATE2" IC, also controls operation of the general-purpose I/O and two supplemental servo channels. On the Ultralite versions of the Turbo PMAC2, this IC also controls the frequency of the clock signals on the board, because the "DSPGATE1" Servo ICs are not present.

A UMAC Turbo system may have up to 16 MACRO ICs present, but only 4 of these can be supported by automatic firmware functions at any given time.

Starting in V1.936 firmware, I20 through I23 must contain the base addresses of MACRO ICs 0 through 3, respectively. If these are not set correctly, the automatic firmware functions associated with these ICs, including the I-variables I6800 – I6999, will not function.

Configuration status variable I4902 tells where MACRO ICs are present; I4903 tells whether these ICs are "MACROGATE" ICs or "DSPGATE2" ICs. Some functions and there supporting I-variables are only available on "DSPGATE2" ICs.

The numbering scheme for the MACRO IC I-Variables is as follows:

- MACRO IC 0: I6800 – I6849
- MACRO IC 1: I6850 – I6899
- MACRO IC 2 I6900 – I6949
- MACRO IC 3 I6590 – I6999

Only the Ultralite and 3U versions of the Turbo PMAC2 may contain MACRO ICs 1, 2, and 3, and these ICs are optional. MACRO ICs 1, 2, and 3 are "MACROGATE" ICs that only have the MACRO ring functionality.

I6800/I6850/I6900/I6950 MACRO IC MaxPhase/PWM Frequency Control

Range:	0 - 32767
Units:	MaxPhase Frequency = 117,964.8 kHz / [2*I6800+3] PWM Frequency = 117,964.8 kHz / [4*I6800+6]
Default:	6527 MaxPhase Frequency = 117,964.8 / 13,057 = 9.0346 kHz PWM Frequency = 117,964.8 / 26,114 = 4.5173 kHz

I6800, I6850, I6900, and I6950 control the internal "MaxPhase" clock frequency, and the PWM frequency for the two machine interface channels (if present), on MACRO ICs 0, 1, 2, and 3, respectively. The internally generated Phase and Servo clocks on a MACRO IC are derived from the MaxPhase clock.

If the IC is used to generate the Phase and Servo clocks for the PMAC system (as set by I6807 etc.), this variable is part of the control for the frequency of these clocks.

On a Turbo PMAC2 Ultralite board, MACRO IC 0 provides the Phase and Servo clock signals for the entire board, so I6800 is used to derive the Phase clock and Servo clock frequencies for the board, along with I6801 and I6802. (On Turbo PMAC2 boards that are not Ultralite, this function is typically controlled by I7000, I7001, and I7002, because Servo IC 0 usually controls the board clock frequencies on these boards.) In a UMAC Turbo system, the MACRO IC on an ACC-5E board can be used to control these clocks.

MACROGATE ICs, commonly used as MACRO ICs 1, 2, and 3, generate no PWM signals and no Servo clock signal. Therefore, they cannot be used as the source of the system Phase and Servo clocks, and the only purpose of this variable is for control of the internal Phase clock signal.

I6800 (etc.) controls these frequencies by setting the limits of the PWM up-down counter, which increments and decrements at the PWMCLK frequency of 117,964.8 kHz (117.9648 MHz). The PWM frequency of MACRO IC 0 determines the frequency of the two single-phase PWM outputs on the JHW "Handwheel" connector.

The actual phase clock frequency is divided down from the maximum phase clock according to the setting of I6801 (etc.). On the falling edge of the phase clock, PMAC2 samples any serial analog-to-digital converters connected to its MACRO ICs (as for phase current measurement), and interrupts the processor to start any necessary phase commutation and digital current-loop algorithms. Even if phasing and current-loop algorithms are not used, the MaxPhase and Phase Clock frequencies are important because the servo clock is derived from the phase clock.

The PWM frequency determines the actual switching frequency of amplifiers connected to any of four machine interface channels with the direct PWM command. It is only important if the direct PWM command signal format is used.

The maximum value that can be written into a PWM command register without full saturation is I6800+1 on the positive end, and -I6800-2 on the negative end.

If the MACRO IC is not used to generate the system clocks, this variable for the IC is generally set to the same value as the comparable variable on the Servo IC (I7000, etc.) or MACRO IC (I6800, etc.) that is used. The only time a different setting should be used is if it is desired that a different PWM frequency be generated on the two channels ("DSPGATE2" ICs only) from that of the variable controlling the system clocks. Certain different frequencies are possible, but they are restricted to the cases where:

$$\frac{2 * PWMFreq(kHz)}{PhaseFreq} = \{ Integer \}$$

This will keep the PWM hardware on channels 1* and 2* in synchronization with the software algorithms driven by the system's Phase clock. For example if the system Phase clock frequency is 10 kHz, the PWM frequency for channels from a different IC can be 5, 10, 15, 20, (etc.) kHz.

To set I6800 (etc.) for a desired PWM frequency, the following formula can be used:

$$I6800 = \frac{117{,}964.8(kHz)}{4 * PWM\_Freq(kHz)} - 1 \text{ (rounded down)}$$

To set I6800 (etc.) for a desired "maximum phase" clock frequency, the following formula can be used:

$$I6800 = \frac{117{,}964.8(kHz)}{2 * MaxPhaseFreq(kHz)} - 1 \text{ (rounded down)}$$

Example:

To set a PWM frequency of 10 kHz and therefore a MaxPhase clock frequency of 20 kHz:

I6800 = (117,964.8 kHz / [4*10 kHz]) - 1 = 2948

To set a PWM frequency of 7.5 kHz and therefore a MaxPhase clock frequency of 15 kHz:

I6800 = (117,964.8 kHz / [4*7.5 kHz]) - 1 = 3931

See Also:

I7, I10, I67, I6801, I6802, I7000, I7001, I7002

I6801/I6851/I6901/I6951 MACRO IC Phase Clock Frequency Control

Range: 0 - 15

Units: none

Default: 0

I6801, I6851, I6901, and I6951, in conjunction with I6800, I6850, I6900, and I6950, determine the frequency of the Phase clock generated inside MACRO ICs 0, 1, 2, and 3, respectively. However, the internal clocks on the IC are only used if the clock-direction control I-variable on the IC (I6807 I6857, I6907, or I6957) is set to 0, specifying that this IC uses its own internal clocks. If this is the case, the IC outputs the clock signals, and these variables determine the phase clock frequency for the entire PMAC2 system.

On a Turbo PMAC2 Ultralite board, MACRO IC 0 typically provides the Phase clock signal for the entire board, so that I6800 and I6801 usually control the Ultralite Phase clock frequency.

Specifically, I6801 (etc.) controls how many times the internally generated Phase clock frequency is divided down from the "MaxPhase" clock, whose frequency is set by I6800 (etc.). The Phase clock frequency is equal to the MaxPhase clock frequency divided by (I6801+1). I6801 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I6801 is:

$$I6801 = \frac{MaxPhaseFreq(kHz)}{PhaseFreq(kHz)} - 1$$

The ratio of MaxPhase frequency to Phase Clock frequency must be an integer.

The main software tasks performed on the Phase clock interrupt – commutation and current-loop closure – are executed every (I7 + 1) Phase clock cycles. With I7 at the default value of 0, they are executed every cycle. In MACRO systems where the Turbo PMAC is closing the current loo, it can be useful to send MACRO data twice per phase software update by setting I7 to 1.

Note:

If the phase clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and the Phase clock frequency can be set to a supportable value.

I6802/I6852/I6902/I6952 MACRO IC Servo Clock Frequency Control

Range: 0 - 15

Units: Servo Clock Frequency = Phase Clock Frequency / (I6802+1)

Default: 3
Servo Clock Frequency = 9.0346 kHz / (3+1) = 2.2587 kHz
(with default values of I6800 and I6801 [etc.])

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I6802, I6852, I6902, and I6952, in conjunction with I6800 and I6801 (etc.), determine the frequency of the Servo clock generated inside MACRO ICs 0, 1, 2, and 3, respectively. However, the internal clocks on the IC are only used if the clock-direction control I-variable on the IC (I6807 I6857, I6907, or I6957) is set to 0, specifying that this IC uses its own internal clocks. If this is the case, the IC outputs the clock signals, and these variables determine the phase clock frequency for the entire PMAC2 system.

On a Turbo PMAC2 Ultralite board, MACRO IC 0 typically provides the Servo clock signal for the entire board, so that I6800, I6801, and I6802 control the Ultralite Servo clock frequency.

Specifically, I6802 controls how many times the Servo clock frequency is divided down from the Phase clock, whose frequency is set by I6801 and I6800. The Servo clock frequency is equal to the Phase clock frequency divided by (I6802+1). I6802 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I6802 is:

$$I6802 = \frac{PhaseFreq(kHz)}{ServoFreq(kHz)} - 1$$

The ratio of Phase Clock frequency to Servo Clock frequency must be an integer.

For execution of trajectories at the proper speed, I10 must be set properly to tell the trajectory generation software what the Servo clock cycle time is. The formula for I10 is:

$$I10 = \frac{8{,}388{,}608}{ServoFreq(kHz)}$$

In terms of the variables that determine the Servo clock frequency on a Turbo PMAC2 Ultralite board, the formula for I10 is:

$$I10 = \frac{640}{9}(2*I6800+3)(I6801+1)(I6802+1)$$

At the default servo clock frequency, I10 should be set to 3,713,707 in order that PMAC's interpolation routines use the proper servo update time.

Note:

If the servo clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and the Servo clock frequency can be set to a supportable value.

Example:
With a 6.67 kHz Phase Clock frequency established by I6800 and I6801, and a desired 3.33 kHz Servo Clock frequency:

I6802 = (6.67 / 3.33) - 1 = 2 - 1 = 1

See Also:
I10, I19, I6800, I6801, I7000, I7001, I7002

I6803/I6853/I6903/I6953 MACRO IC Hardware Clock Control

Range: 0 - 4095

Units: Individual Clock Dividers
I6803 = Encoder SCLK Divider
+ 8 * PFM_CLK Divider
+ 64 * DAC_CLK Divider
+ 512 * ADC_CLK Divider
where:
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ Encoder SCLK Divider)
PFM_CLK Frequency = 39.3216 MHz / (2 ^ PFM_CLK Divider)
DAC_CLK Frequency = 39.3216 MHz / (2 ^ DAC_CLK Divider)
ADC_CLK Frequency = 39.3216 MHz / (2 ^ ADC_CLK Divider)

Default: 2258 = 2 + (8 * 2) + (64 * 3) + (512 * 4)
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
PFM_CLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
DAC_CLK Frequency = 39.3216 MHz / (2 ^ 3) = 4.9152 MHz
ADC_CLK Frequency = 39.3216 MHz / (2 ^ 4) = 2.4576 MHz

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I6803, I6853, I6903, and I6953 control the frequency of four hardware clock signals -- SCLK, PFM_CLK, DAC_CLK, and ADC_CLK -- for the two supplemental machine interface channels of MACRO ICs 0, 1, 2, and 3, respectively, provided they are "DSPGATE2" ICs. These are 12-bit variables consisting of four independent 3-bit controls, one for each of the clocks. Each of these clock frequencies can be divided down from a starting 39.3216 MHz frequency by powers of 2, 2^N, from 1 to 128 times (N=0 to 7). This means that the possible frequency settings for each of these clocks are:

Frequency	Divide by	Divider N in $1/2^N$
39.3216 MHz	1	0
19.6608 MHz	2	1
9.8304 MHz	4	2
4.9152 MHz	8	3
2.4576 MHz	16	4
1.2288 MHz	32	5
614.4 kHz	64	6
307.2 kHz	128	7

Very few Turbo PMAC2 users will be required to change the setting of these variables from their default values.

Note:

In firmware versions V1.933 and older, bit *m* of I67 must be set to 1 in order to be able to access this variable on MACRO IC *m*. In V1.934 and V1.935, Turbo PMAC automatically enables this variable for any IC present at power-up/reset. In V1.936 and newer, I2*m* must contain the base address of MACRO IC *m* in order to access this variable on the IC.

SCLK: The encoder sample clock signal SCLK controls how often the MACRO IC's digital hardware looks at the encoder and flag inputs. The MACRO IC can take at most one count per SCLK cycle, so the SCLK frequency is the absolute maximum encoder count frequency. SCLK also controls the signal propagation through the digital delay filters for the encoders and flags; the lower the SCLK frequency, the greater the noise pulse that can be filtered out. The SCLK frequency should optimally be set to the lowest value that can accept encoder counts at the maximum possible rate.

PFM_CLK: The pulse-frequency-modulation clock PFM_CLK controls the PFM circuitry that is commonly used for stepper drives. The maximum pulse frequency possible is 1/4 of the PFM_CLK frequency. The PFM_CLK frequency should optimally be set to the lowest value that can generate pulses at the maximum frequency required.

DAC_CLK: The DAC_CLK controls the serial data frequency into D/A converters. If these converters are on Delta Tau-provided accessories, the DAC_CLK setting should be left at the default value.

ADC_CLK: The ADC_CLK controls the serial data frequency from A/D converters. If these converters are on Delta Tau-provided accessories, the ADC_CLK setting should be left at the default value.

Note:

By default, the DAC and ADC circuits of a MACRO IC are not used on a Turbo PMAC2. The DAC and ADC lines are the "alternate" uses of pins on the Multiplexer and I/O ports, respectively.

To determine the clock frequencies set by a given value of I6803 (etc.), use the following procedure:

1. Divide I6803 by 512 and round down to the nearest integer. This value N1 is the ADC_CLK divider.
2. Multiply N1 by 512 and subtract the product from I6803 to get I6803'. Divide I6803' by 64 and round down to the nearest integer. This value N2 is the DAC_CLK divider.
3. Multiply N2 by 64 and subtract the product from I6803' to get I6803". Divide I6803" by 8 and round down to the nearest integer. This value N3 is the PFM_CLK divider.
4. Multiply N3 by 8 and subtract the product from I6803". The resulting value N4 is the SCLK divider.

Examples:

The maximum encoder count frequency in the application is 800 kHz, so the 1.2288 MHz SCLK frequency is chosen. A pulse train up to 500 kHz needs to be generated, so the 2.4576 MHz PFM_CLK frequency is chosen. The default serial DACs and ADCs provided by Delta Tau are used, so the default DAC_CLK frequency of 4.9152 MHz and the default ADC_CLK frequency of 2.4576 MHz are chosen. From the table:

SCLK Divider N: 5
PFM_CLK Divider N: 4
DAC_CLK Divider N: 3
ADC_CLK Divider N: 4
I6803 = 5 + (8 * 4) + (64 * 3) + (512 * 4) = 5 + 32 + 192 + 2048 = 2277

I6803 has been set to 3429. What clock frequencies does this set?

N1 = INT (3429/512) = 6	ADC_CLK = 611.44 kHz
I6803' = 3429 - (512*6) = 357	
N2 = INT (357/64) = 5	DAC_CLK = 1.2288 MHz
I6803" = 357 - (64*5) = 37	
N3 = INT (37/8) = 4	PFM_CLK = 2.4576 MHz
N4 = 37 - (8*4) = 5	SCLK = 1.2288 MHz

See Also:
I-variables I7m03, I7m53

I6804/I6854/I6904/I6954 MACRO IC PWM Deadtime / PFM Pulse Width Control

Range: 0 - 255

Units: 16*PWM_CLK cycles / PFM_CLK cycles
PWM Deadtime = [16 / PWM_CLK (MHz)] * I6804 = 0.135 usec * I6804
PFM Pulse Width = [1 / PFM_CLK (MHz)] * I6804
= PFM_CLK_period (usec) * I6804

Default: 15
PWM Deadtime = 0.135 usec * 15 = 2.03 µsec
PFM Pulse Width = [1 / 9.8304 MHz] * 15 = 1.526 µsec (with default I6803)

Note:

This I-variable is only active if the MACRO IC is present, and a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I6804, I6854, I6904, and I6954 control the deadtime period between top and bottom on-times in PMAC2's automatic PWM generation for the two supplemental machine interface channels of MACRO ICs 0, 1, 2, and 3, respectively, provided they are "DSPGATE2" ICs. In conjunction with I6803 I6853, I6903, and I6953, they also control the pulse width for PMAC2's automatic pulse-frequency modulation generation for the two machine interface channels on the DSPGATE2 MACRO IC.

The PWM deadtime, which is the delay between the top signal turning off and the bottom signal turning on, and vice versa, is specified in units of 16 PWM_CLK cycles. This means that the deadtime can be specified in increments of 0.135 µsec. The equation for I6804 (etc.) as a function of PWM deadtime is:

$$I6804 = \frac{DeadTime(\mu sec)}{0.135\mu sec}$$

The PFM pulse width is specified in PFM_CLK cycles, as defined by I6803 (etc.). The equation for I6804 (etc.) as a function of PFM pulse width and PFM_CLK frequency is:

$$I6804 = PFM\_CLK\_Freq(MHz) * PFM\_Pulse\_Width(\mu sec)$$

In PFM pulse generation, the minimum off time between pulses is equal to the pulse width. This means that the maximum PFM output frequency is

$$PFM\_Max\_Freq(MHz) = \frac{PFM\_CLK\_Freq(MHz)}{2 * I6804}$$

Examples:
A PWM deadtime of approximately 1 microsecond is desired:

I6804 ≅ 1 µsec / 0.135 µsec ≅ 7

With a 2.4576 MHz PFM_CLK frequency, a pulse width of 0.4 µsec is desired:

I6804 ≅ 2.4576 MHz * 0.4 µsec ≅ 1

See Also:

I-variables I6803 (etc.), I7m03, I7m04

I6805/I6855/I6905/I6955 MACRO IC DAC Strobe Word

Range: $000000 - $FFFFFF

Units: Serial Data Stream (MSB first, starting on rising edge of phase clock)

Default: $7FFFC0 (for 18-bit DACs)

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I6805, I6855, I6905, and I6955 control the DAC strobe signal for the two supplemental machine interface channels of MACRO ICs 0, 1, 2, and 3, respectively, provided they are "DSPGATE2" ICs.

The 24-bit word set by this variable for the IC is shifted out serially on the DAC_STROB lines, MSB first, one bit per DAC_CLK cycle starting on the rising edge of the phase clock. The value in the LSB is held until the next phase clock cycle.

For a typical *n*-bit DACs, the strobe line is held high for *n*-1 clock cycles. Therefore, the common settings of this variable are:

- 18-bit DACs: $7FFFC0 (high for 17 clock cycles)
- 16-bit DACs: $7FFF00 (high for 15 clock cycles)
- 12-bit DACs: $7FF000 (high for 11 clock cycles)

Note:

By default, the DAC circuitry of a MACRO IC is not used on a Turbo PMAC2. The DAC lines are the "alternate" use of lines on the I/O port.

I6806/I6856/I6906/I6956 MACRO IC ADC Strobe Word

Range: $000000 - $FFFFFF

Units: Serial Data Stream (MSB first, starting on rising edge of phase clock)

Default: $FFFFFE

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I6806, I6856, I6906, and I6956 control the ADC strobe signal for the two supplemental machine interface channels of MACRO ICs 0, 1, 2, and 3, respectively, provided they are "DSPGATE2" ICs. The 24-bit word set by this variable for the IC is shifted out serially on the ADC_STROB lines, MSB first, one bit per DAC_CLK cycle starting on the rising edge of the phase clock. The value in the LSB is held until the next phase clock cycle.

The first '1' creates a rising edge on the ADC_STROB output that is typically used as a "start-convert" signal. Some A/D converters just need this rising edge for the conversion; others need the signal to stay high all of the way through the conversion. The LSB of I6806 should always be set to 0 so that a rising edge is created on the next cycle. The default I6806 value of $FFFFFE is suitable for virtually all A/D converters.

The A/D converters used on matching Delta Tau products just need the rising edge at the start of a conversion cycle; this permits intermediate bits in the data stream to be used as special control bits. Delta Tau's ACC-8T Supplemental Flag Multiplexer Board uses these bits to control the multiplexing; Delta Tau's ACC-8K1 Fanuc C/S-Series PWM Interface Board uses these bits to control the magnetic contactors on the drives.

Note:

By default, the ADC circuitry on a MACRO IC is not used on a Turbo PMAC2. The ADC lines are the "alternate" use of pins on the Multiplexer port.

I6807/I6857/I6907/I6957 MACRO IC Clock Direction Control

Range: 0 – 3 (DSPGATE2 IC); 0 – 1 (MACROGATE IC)

Units: none

Default: System dependent

I6807, I6857, I6907, and I6957 control whether MACRO ICs 0, 1, 2, and 3, respectively, use their own internally generated servo and phase clock signals, or whether they use servo and phase clock signals from a source external to them (usually MACRO IC 0 or Servo IC 0).

In any Turbo PMAC2 system, there must be one and only one source of servo and phase clock signals for the system – either one of the Servo ICs or MACRO ICs, or a source external to the system. Only in a 3U-format Turbo PMAC2 system (UMAC Turbo or 3U Turbo Stack) can the system clock signals come from an accessory board. In all other Turbo PMAC2 systems, the system clock signals must come from and IC on the base PMAC2 boards, or be brought from an external source through the serial port.

These variables are 2-bit values on "DSPGATE2" MACRO ICs, but only 1-bit values on MACROGATE MACRO ICs. Bit 0 is set to 0 for the IC to use its own Phase clock signal and output it; it is set to 1 to use an externally input Phase clock signal. Bit 1 ("DSPGATE2 only) is set to 1 for the IC to use its own Servo clock signal and output it; it is set to 1 to use an externally input Servo clock signal. This yields 4 possible values for I6807 (etc.):

- I6807 = 0: Internal Phase clock; internal Servo clock
- I6807 = 1: External Phase clock; internal Servo clock
- I6807 = 2: Internal Phase clock; external Servo clock
- I6807 = 3: External Phase clock; external Servo clock

In all normal use, I6807 (etc.) is either set to 0 (on at most one IC) or 3 (on all the other ICs – 1 on MACROGATE ICs).

In typical use of the Turbo PMAC2 Ultralite, MACRO IC 0, whose Phase clock frequency is controlled by I6800 and I6801, will generate the Phase clock signal for the entire board, so I6807 is set to 0, and I6857, I6907, and I6957 should all be set to 1.

Note:

A "MACROGATE" MACRO IC cannot generate a servo clock signal internally. Therefore, it cannot be used to provide the system clocks for the Turbo PMAC2 system.

During re-initialization, Turbo PMAC2 determines which IC it will use as the source of its system Phase and Servo clock signals, setting I19 to the number of the clock-direction I-variable whose IC is selected as the source. This clock-direction I-variable is then automatically set to 0; all other clock-direction I-variables are set to 1 or 3. Most users will never change these settings.

When a clock-direction I-variable is commanded to its default value (e.g. **I6857=***), Turbo PMAC2 looks to the value of I19 to determine whether this I-variable is set to 0 or 3 (0 or 1 on a MACROGATE IC).

On the reset of a 3U-format Turbo PMAC2 system (UMAC Turbo or 3U Turbo Stack), the values set for these I-variables are determined by the saved value of I19, and not by the saved values of these I-variables themselves. On these systems, if you wish to change which IC is the source of the system clocks, change the value of I19, save this setting, and reset the card.

In other Turbo PMAC2 systems, if you wish to change which IC is the source of the system clocks, it is best to change both clock-direction I-variables on a single command line (e.g. **I6807=1 I7007=0**), then **SAVE** these new settings.

If all of the Servo ICs and MACRO ICs in a Turbo PMAC2 system have been set up for external phase and servo clocks, but these clock signals are not provided, the Turbo PMAC2 will immediately trip its watchdog timer.

Channel-Specific MACRO IC I-variables

(For MACRO IC Channel n*, where n* = 1 to 2)

I-Variables in the I6810s, I6820s, I6910s, and I6920s control the hardware aspects of the MACRO IC "DSPGATE2" ASIC that provides the machine interface for supplemental channels 1 and 2. Note that few of these functions are normally used on the Turbo PMAC2s. By default, only the two encoder inputs and the two C-channel PWM/PFM outputs are used. These I-variables are not active if the MACRO IC is not present, or is a "MACROGATE" IC.

I68n0/I69n0 MACRO IC Channel n* Encoder/Timer Decode Control

Range: 0 - 15

Units: None

Default: 7

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*0 and I69*n*0 control how the encoder input signal for Channel *n** (*n** = 1 to 2) on a DSPGATE2 MACRO IC is decoded into counts. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6810 controls MACRO IC 0 Channel 1; I6970 controls MACRO IC 3 Channel 2). As such, this defines the sign and magnitude of a "count". The following settings may be used to decode an input signal.

- I68n0/I69n0 = 0: Pulse and direction CW
- I68n0/I69n0 = 1: x1 quadrature decode CW
- I68n0/I69n0 = 2: x2 quadrature decode CW
- I68n0/I69n0 = 3: x4 quadrature decode CW
- I68n0/I69n0 = 4: Pulse and direction CCW
- I68n0/I69n0 = 5: x1 quadrature decode CCW
- I68n0/I69n0 = 6: x2 quadrature decode CCW
- I68n0/I69n0 = 7: x4 quadrature decode CCW
- I68n0/I69n0 = 8: Internal pulse and direction
- I68n0/I69n0 = 9: Not used
- I68n0/I69n0 = 10: Not used
- I68n0/I69n0 = 11: x6 hall format decode CW*

- I68n0/I69n0 = 12: MLDT pulse timer control
  (internal pulse resets timer; external pulse latches timer)
- I68n0/I69n0 = 13: Not used
- I68n0/I69n0 = 14: Not used
- I68n0/I69n0 = 15: x6 hall format decode CCW*

*requires version B or newer of the DSPGATE2 MACRO IC.

In any of the quadrature decode modes, the MACRO IC is expecting two input waveforms on CHAn and CHBn, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. "Times-one" (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The "clockwise" (CW) and "counterclockwise" (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa).

WARNING

Changing the direction sense of the decode for the feedback encoder of a motor that is operating properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes. The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

In the pulse-and-direction decode modes, the MACRO IC is expecting the pulse train on CHAn, and the direction (sign) signal on CHBn. If the signal is unidirectional, the CHBn line can be allowed to pull up to a high state, or it can be hardwired to a high or low state.

If I68n0/I69n0 is set to 8, the decoder inputs the pulse and direction signal generated by Channel n's pulse frequency modulator (PFM) output circuitry. This permits the PMAC2 to create a phantom closed loop when driving an open-loop stepper system. *No jumpers or cables are needed to do this; the connection is entirely within the MACRO IC.* The counter polarity automatically matches the PFM output polarity.

If I68n0/I69n0 is set to 11 or 15, Channel n is expecting three Hall-sensor format inputs on CHAn, CHBn, and CHCn, each with approximately 50% duty cycle, and approximately one-third (120°e) of a cycle out of phase with each other. The decode circuitry will generate one count on each edge of each signal, yielding 6 counts per signal cycle ("x6 decode"). The difference between 11 and 15 is which direction of signal causes the counter to count up.

If I68n0/I69n0 is set to 12, the timer circuitry is set up to read magnetostrictive linear displacement transducers (MLDTs) such as Temposonics™. In this mode, the timer is cleared when the PFM circuitry sends out the excitation pulse to the sensor on PULSEn, and it is latched into the memory-mapped register when the excitation pulse is received on CHAn.

I68n1/I69n1 MACRO IC Channel n* Position Compare Channel Select

Range: 0 - 1

Units: None

Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*1 and I69*n*1 control which channel's encoder counter is tied to the position compare circuitry for Channel *n** (*n** = 1 to 2) on a "DSPGATE2" MACRO IC is decoded into counts. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6811 controls MACRO IC 0 Channel 1; I6971 controls MACRO IC 3 Channel 2). They have the following possible settings:

- I68n1/I69n1 = 0: Use Channel n* encoder counter for position compare function
- I68n1/I69n1 = 1: Use Channel 1* encoder counter on IC for position compare function

When I68n1/I69n1 is set to 0, Channel n*'s position compare registers tied to the channel's own encoder counter, and the position compare signal appears only on the EQU output for that channel.

When I68n1/I69n1 is set to 1, the channel's position compare register is tied to the first encoder counter on the MACRO IC, and the position compare signal appears both on Channel n*'s EQU output, and combined into the EQU output for Channel 1* on the MACRO IC (EQU1* on the board); executed as a logical OR.

I68n1 for the first channel performs no effective function, so is always 1. It cannot be set to 0.

Note:

By default, the position compare circuitry on a MACRO IC is not used on Turbo PMAC2 boards. The compare outputs are the "alternate" use of lines on the Multiplexer port.

I68n2/I69n2 MACRO IC Encoder n* Capture Control

Range: 0 - 15

Units: none

Default: 1

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*2 and I69*n*2 determine which input signal or combination of signals, and which polarity, for Channel *n** (*n** = 1 to 2) on a DSPGATE2 MACRO IC triggers a hardware position capture of the counter for Encoder *n**. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6812 controls MACRO IC 0 Channel 1; I6972 controls MACRO IC 3 Channel 2). If a flag input (home, limit, or user) is used, I68n3/I69n3 determines which flag. Proper setup of this variable is essential for a successful homing search move or other move-until-trigger for the Motor xx using Channel n* for its position-loop feedback and flags if the super-accurate hardware position capture function is used. If Ixx97 is at its default value of 0 to select hardware capture and trigger, this variable must be set up properly.

The following settings of I68n2 may be used:

- I68n2 = 0: Continuous capture
- I68n2 = 1: Capture on Index (CHCn) high
- I68n2 = 2: Capture on Flag n high
- I68n2 = 3: Capture on (Index high AND Flag n high)
- I68n2 = 4: Continuous capture
- I68n2 = 5: Capture on Index (CHCn) low
- I68n2 = 6: Capture on Flag n high
- I68n2 = 7: Capture on (Index low AND Flag n high)

- I68n2 = 8: Continuous capture
- I68n2 = 9: Capture on Index (CHCn) high
- I68n2 = 10: Capture on Flag n low
- I68n2 = 11: Capture on (Index high AND Flag n low)
- I68n2 = 12: Continuous capture
- I68n2 = 13: Capture on Index (CHCn) low
- I68n2 = 14: Capture on Flag n low
- I68n2 = 15: Capture on (Index low AND Flag n low)

Only flags and index inputs of the same channel number as the encoder may be used for hardware capture of that encoder's position. This means that to use the hardware capture feature for the homing search move, Ixx25 must use flags of the same channel number as the encoder that Ixx03 uses for position-loop feedback.

The trigger is armed when the position capture register is read. After this, as soon as the MACRO IC hardware sees that the specified input lines are in the specified states, the trigger will occur -- it is level-triggered, not edge-triggered.

Note:

By default, the index-channel and flag inputs of a MACRO IC are not used on a Turbo PMAC2. The index inputs and flag inputs are "alternate" uses of pins on the Multiplexer and I/O ports, respectively.

I68n3/I69n3 MACRO IC Channel n* Capture Flag Select Control

Range: 0 - 3

Units: none

Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a DSPGATE2 IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*3 and I69*n*3 determine which of the Flag inputs will be used for hardware position capture (if one is used) of the encoder counter of Channel *n** (*n** = 1 to 2) on a "DSPGATE2" MACRO IC. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6813 controls MACRO IC 0 Channel 1; I6973 controls MACRO IC 3 Channel 2). I68n2/I69n2 determines whether a flag is used and which polarity of the flag will cause the trigger. The possible values of I68n3/I69n3 and the flag each selects is:

- I68n3/I69n3 = 0: HOMEn (Home Flag n)
- I68n3/I69n3 = 1: PLIMn (Positive End Limit Flag n)
- I68n3/I69n3 = 2: MLIMn (Negative End Limit Flag n)
- I68n3/I69n3 = 3: USERn (User Flag n)

I68n3/I69n3 is typically set to 0 for homing search moves in order to use the home flag for the channel. It is typically set to 3 afterwards to select the User flag if other uses of the hardware position capture function are desired, such as for probing and registration. If you wish to capture on the PLIMn or MLIMn overtravel limit flags, you probably will want to disable their normal functions with Ixx25, or use a channel n where none of the flags is used for the normal axis functions.

Note:

By default, the flag inputs of MACRO IC 0 are not used on a Turbo PMAC2.

I68n4/I69n4 MACRO IC Channel n* Encoder Gated Index Select

Range: 0 - 1

Units: none

Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a DSPGATE2 IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*4 and I69*n*4 control whether the raw encoder index channel input or a version of the input gated by the AB-quadrature state is used for position capture of the encoder counter of Channel *n** (*n** = 1 to 2) on a "DSPGATE2" MACRO IC. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6814 controls MACRO IC 0 Channel 1; I6974 controls MACRO IC 3 Channel 2). They have the following possible settings:

- I68n4/I69n4 = 0: Use ungated index for encoder position capture
- I68n4/I69n4 = 1: Use index gated by quadrature channels for position capture

When I68n4/I69n4 is set to 0, the encoder index channel input (CHCn) is passed directly into the position capture circuitry.

When I68n4/I69n4 is set to 1, the encoder index channel input (CHCn) is logically combined with (gated by) the quadrature signals of Encoder n before going to the position capture circuitry. The intent is to get a gated index signal exactly one quadrature state wide. This provides a more accurate and repeatable capture, and makes the use of the capture function to confirm the proper number of counts per revolution very straightforward.

In order for the gated index capture to work reliably, the index pulse must reliably span one, but only one, high-high or low-low AB quadrature state of the encoder. I68n5/I69n5 allows you to select which of these two possibilities is used.

Note:

If I68n4/I69n4 is set to 1, but I68n2/I69n2 bit 0 is set to 0, so the index is not used in the position capture, then the encoder position is captured on the first edge of any of the U, V, or W flag inputs for the channel. In this case, bits 0, 1, and 2 of the channel status word tell what hall-state edge caused the capture.

Note:

By default, the index channels of a DSPGATE2 MACRO IC are not used on a Turbo PMAC2. The index inputs are the "alternate" uses of pins on the Multiplexer port.

I68n5/I69n5 MACRO IC Channel n* Encoder Index Gate State/Demux Control

Range: 0 - 3
Units: none
Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68n5 and I69n5 are 2-bit variables that control two functions for the index channel of the encoder.

When using the gated index feature of Channel *n** of a DSPGATE2 MACRO IC for more accurate position capture (I68n4/I69n4 = 1), bit 0 of I68*n*5 and I69*n*5 controls whether the raw index-channel signal for Encoder *n** (*n** = 1 to 2) on the MACRO IC is passed through to the position capture signal only on the high-high quadrature state (bit 0 = 0), or only on the "low-low" quadrature state (bit 0 = 1). For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6815 controls MACRO IC 0 Channel 1; I6975 controls MACRO IC 3 Channel 2).

Bit 1 of I68n5 and I69n5 controls whether the Servo IC "de-multiplexes" the index pulse and the 3 hall-style commutation states from the third channel based on the quadrature state, as with Yaskawa incremental encoders. If bit 1 is set to 0, this de-multiplexing function is not performed, and the signal on the C channel of the encoder is used as the index only. If bit 1 is set to 1, the Servo IC breaks out the third-channel signal into four separate values, one for each of the four possible AB-quadrature states. The de-multiplexed hall commutation states can be used to provide power-on phase position using Ixx81 and Ixx91.

Note:

Immediately after power-up, the Yaskawa encoder automatically cycles its AB outputs forward and back through a full quadrature cycle to ensure that all of the hall commutation states are available to the controller before any movement is started. However, if the encoder is powered up at the same time as the Turbo PMAC, this will happen before the Servo IC is ready to accept these signals. Bit 2 of the channel's status word, "Invalid De-multiplex", will be set to 1 if the Servo IC has not seen all of these states when it was ready for them. To use this feature, it is recommended that the power to the encoder be provided through a software-controlled relay to ensure that valid readings of all states have been read before using these signals for power-on phasing.

I68n5 and I69n5 have the following possible settings:

- I68n5/I69n5 = 0: Gate index with "high-high" quadrature state (GI = A & B & C), no demux
- I68n5/I69n5 = 1: Gate index with "low-low" quadrature state (GI = A/ & B/ & C), no demux
- I68n5/I69n5 = 2 or 3: De-multiplex hall and index from third channel, gating irrelevant

Note:

By default, the index channels of a "DSPGATE2" MACRO IC are not used on a Turbo PMAC2. The index inputs are the "alternate" uses of pins on the Multiplexer port.

I68n6/I69n6 MACRO IC Channel n* Output Mode Select

Range: 0 - 3

Units: none

Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a DSPGATE2 IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*6 and I69*n*6 control what output formats are used on the command output signal lines for machine interface channel *n** (*n** = 1 to 2) on a DSPGATE2 MACRO IC. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6816 controls MACRO IC 0 Channel 1; I6976 controls MACRO IC 3 Channel 2). They have the following possible settings:

- I68n6/I69n6 = 0: Outputs A & B are PWM; Output C is PWM
- I68n6/I69n6 = 1: Outputs A & B are DAC; Output C is PWM
- I68n6/I69n6 = 2: Outputs A & B are PWM; Output C is PFM
- I68n6/I69n6 = 3: Outputs A & B are DAC; Output C is PFM

If a three-phase direct PWM command format is desired, I68n6/I69n6 should be set to 0. If signal outputs for (external) digital-to-analog converters are desired, I68n6/I69n6 should be set to 1 or 3. In this case, the C output can be used as a supplemental (non-servo) output in either PWM or PFM form. For example, it can be used to excite an MLDT sensor (e.g. Temposonics™) in PFM form.

Note:

By default, only the C outputs (PWM or PFM) of MACRO IC 0 are used on a Turbo PMAC2. The A and B outputs are the "alternate" use of pins on the I/O port.

I68n7/I69n7 MACRO IC Channel n* Output Invert Control

Range: 0 - 3

Units: none

Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a DSPGATE2 IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*7 and I69*n*7 control the high/low polarity of the command output signals for machine interface channel *n** (*n** = 1 to 2) on a "DSPGATE2" MACRO IC. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6817 controls MACRO IC 0 Channel 1; I6977 controls MACRO IC 3 Channel 2). They have the following possible settings:

- I68n7/I69n7 = 0: Do not invert Outputs A & B; Do not invert Output C
- I68n7/I69n7 = 1: Invert Outputs A & B; Do not invert Output C
- I68n7/I69n7 = 2: Do not invert Outputs A & B; Invert Output C
- I68n7/I69n7 = 3: Invert Outputs A & B; Invert Output C

The default non-inverted outputs are high true. For PWM signals on Outputs A, B, and C, this means that the transistor-on signal is high. Delta Tau PWM-input amplifiers, and most other PWM-input amplifiers, expect this non-inverted output format. For such a 3-phase motor drive, I68n7 should be set to 0.

Note:

If the high/low polarity of the PWM signals is wrong for a particular amplifier, what was intended to be deadtime between top and bottom on-states as set by I6804 becomes overlap. If the amplifier input circuitry does not lock this out properly, this causes an effective momentary short circuit between bus power and ground. This would destroy the power transistors very quickly.

For PFM signals on Output C, non-inverted means that the pulse-on signal is high (direction polarity is controlled by I68n8). During a change of direction, the direction bit will change synchronously with the leading edge of the pulse, which in the non-inverted form is the rising edge. If the drive requires a set-up time on the direction line before the rising edge of the pulse, the pulse output can be inverted so that the rising edge is the trailing edge, and the pulse width (established by I6804) is the set-up time.

For DAC signals on Outputs A and B, non-inverted means that a 1 value to the DAC is high. DACs used on Delta Tau accessory boards, as well as all other known DACs always expect non-inverted inputs, so I68n7 should always be set to 0 or 2 when using DACs on Channel n.

Note:

Changing the high/low polarity of the digital data to the DACs has the effect of inverting the voltage sense of the DACs' analog outputs. This changes the polarity match between output and feedback. If the feedback loop had been stable with negative feedback, this change would create destabilizing positive feedback, resulting in a dangerous runaway condition that would only be stopped when the motor exceeded Ixx11 fatal following error

Note:

By default, only the C outputs (PWM or PFM) of MACRO IC 0 are used on a Turbo PMAC2. The A and B outputs are the "alternate" use of pins on the I/O port.

I68n8/I69n8 MACRO IC Channel n* PFM Direction Signal Invert Control

Range: 0 - 1

Units: none

Default: 0

Note:

This I-variable is only active if the MACRO IC is present, and is a "DSPGATE2" IC. The presence and type of MACRO ICs are reported in I4902 and I4903.

I68*n*8 and I69*n*8 control the polarity of the direction output signal in the pulse-and-direction format for machine interface channel *n** (*n** = 1 to 2) on a "DSPGATE2" MACRO IC. For MACRO ICs 0 and 2, *n* = *n**; for MACRO ICs 1 and 3, *n* = *n** + 5 (i.e. I6818 controls MACRO IC 0 Channel 1; I6978 controls MACRO IC 3 Channel 2). They have the following possible settings:

- I68n8/I69n8 = 0: Do not invert direction signal (+ = low; - = high)
- I68n8/I69n8 = 1: Invert direction signal (- = low; + = high)

If I68n8/I69n8 is set to the default value of 0, a positive direction command provides a low output; if I68n8/I69n8 is set to 1, a positive direction command provides a high output.

I68n9/I69n9 Reserved for future use

MACRO IC Ring Setup I-variables

I6840/I6890/I6940/I6990 MACRO IC Ring Configuration/Status

Range: $0000 - $FFFF (0 - 65,535)

Units: none

Default: 0

I6840, I6890, I6940, and I6990 contain configuration and status bits for MACRO ring operation of MACRO ICs 0, 1, 2, and 3, respectively, on the Turbo PMAC2.

There are 11 configuration bits and 5 status bits, as follows:

Bit #	Value	Type	Function
0	1($1)	Status	Data Overrun Error (cleared when read)
1	2($2)	Status	Byte Violation Error (cleared when read)
2	4($4)	Status	Packet Parity Error (cleared when read)
3	8($8)	Status	Packet Underrun Error (cleared when read)
4	16($10)	Config	Master Station Enable
5	32($20)	Config	Synchronizing Master Station Enable
6	64($40)	Status	Sync Node Packet Received (cleared when read)
7	128($80)	Config	Sync Node Phase Lock Enable
8	256($100)	Config	Node 8 Master Address Check Disable
9	512($200)	Config	Node 9 Master Address Check Disable
10	1024($400)	Config	Node 10 Master Address Check Disable
11	2048($800)	Config	Node 11 Master Address Check Disable
12	4096($1000)	Config	Node 12 Master Address Check Disable
13	8192($2000)	Config	Node 13 Master Address Check Disable
14	16384($4000)	Config	Node 14 Master Address Check Disable
15	32768($8000)	Config	Node 15 Master Address Check Disable

In most applications, the only important configuration bits are bits 4, 5, and 7. In every MACRO ring, there must be one and only one synchronizing master station (each MACRO IC counts as a separate station; only one MACRO IC on any card in the ring can be a synchronizing master station). For this MACRO IC, bits 4 and 5 should be set (1), but bit 7 should be clear (0). On a Turbo PMAC2 Ultralite, this should be MACRO IC 0, for which I6840 should be set to $30, or $xx30 if any of the high bits are to be set.

If there are more than one MACRO ICs acting as masters on the ring, the others should not be synchronizing masters, but they should be set up as masters and enable "sync node phase lock" to stay synchronized with the synchronizing master. For these MACRO ICs, bit 4 should be set (1), bit 5 should be clear (0), and bit 7 should be set (1), so I6890/I6940/I6990 should be set to $90, or $xx90 if any of the high bits are to be set.

Bits 8-15 can be set individually to disable the "master address check" for their corresponding node numbers. This capability is for multi-master broadcast and synchronization. If the master address check is disabled, only the slave node number part of the packet address must match for a packet to be latched in. In this way, the synchronizing master can send the same data packet to multiple other master and slave stations. This common packet can be used to keep multiple stations synchronized using the sync lock function enabled with bit 7 of I6890/I6940/I6990; the packet number is specified in I6891/I6941/I6991 (packet 15 is suggested for this purpose).

I6841/I6891/I6941/I6991 MACRO IC Node Activate Control

Range: $000000 to $FFFFFF (0 to 8,388,607)

Units: none

Default: $0 (all nodes de-activated)

I6841, I6891, I6941, and I6991 control which of the 16 MACRO nodes on MACRO ICs 0, 1, 2, and 3, respectively, are activated. They also control the master station number of the IC, and the node number of the packet that creates a synchronization signal. The bits of these I-variables are arranged as follows:

Bit #	Value	Type	Function
0	1($1)	Config	Node 0 Activate
1	2($2)	Config	Node 1 Activate
2	4($4)	Config	Node 2 Activate
3	8($8)	Config	Node 3 Activate
4	16($10)	Config	Node 4 Activate
5	32($20)	Config	Node 5 Activate
6	64($40)	Config	Node 6 Activate
7	128($80)	Config	Node 7 Activate
8	256($100)	Config	Node 8 Activate
9	512($200)	Config	Node 9 Activate
10	1024($400)	Config	Node 10 Activate
11	2048($800)	Config	Node 11 Activate
12	4096($1000)	Config	Node 12 Activate
13	8192($2000)	Config	Node 13 Activate
14	16384($4000)	Config	Node 14 Activate
15	32768($8000)	Config	Node 15 Activate
16-19	$X0000	Config	Packet Sync Node Slave Address (X=0-F)
20-23	$X00000	Config	Master Station Number (X=0-F)

Bits 0 to 15 are individual control bits for the matching node number 0 to 15. If the bit is set to 1, the node is activated; if the bit is set to 0, the node is de-activated.

Note:

If the use of an activated node n includes auxiliary register functions, including servo flags, bit n of I72 (IC 1), I74 (IC 2), or I76 (IC 3) must also be set to 1, and bit n of I73 (IC 1), I75 (IC 2), or I77 (IC 3) must be set properly to 0 or 1 to define Type 0 or Type 1 auxiliary register functions, respectively.

If MACRO IC m is a master station (likely) as determined by I6840/I6890/I6940/I6990, it will send out a packet for each activated node every ring cycle (every phase cycle). When it receives a packet for an activated node, it will latch in that packet and not pass anything on.

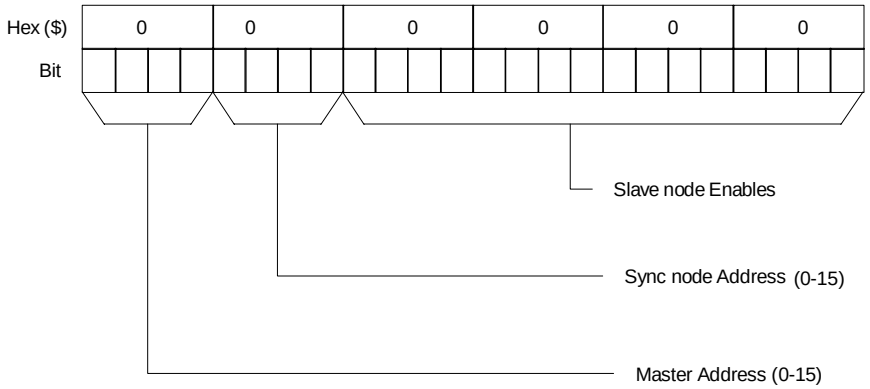
If MACRO IC m is a slave station (unlikely but possible) as determined by I6840/I6890/I6940/I6990, when it receives a packet for an activated node, it will latch in the contents of that packet into its read registers for that node address, and automatically substitute the contents of its write registers into the packet.

If a node is disabled, the PMAC2, whether master or slave, will still latch in the contents of a packet it receives, but it will also pass on the packet unchanged. This feature is particularly useful for the MACRO broadcast feature, in which multiple stations need to receive the same packet.

Bits 16-19 together specify the slave number part of the packet address (0-15) that will cause a sync lock pulse on the card, if this function is enabled by I6890/I6940/I6990. This function is useful for a PMAC2 that is a slave or non-synchronizing master on the ring, to keep it locked to the synchronizing master. If the master address check for this node is disabled with I6890/I6940/I6990, only the slave number must match to create the sync lock pulse. If the master address check is left enabled, the master number part of the packet address must match the master number for the card, as set in bits 20-23 of I6891/I6941/I6991.

If this card is the synchronizing master, this function is not enabled, so the value of these bits does not matter; they can be left at the default of 0.

Bits 20-23 specify the master number for the MACRO IC (0-15). Each MACRO IC on a ring must have a separate master number, even multiple MACRO ICs on the same Turbo PMAC2 Ultralite. The number must be specified whether the card is used as a master or a slave.



If I78 is set greater than 0 to enable Type 1 master-to-slave auxiliary communications, then bit 15 of I6891/I6941/I6991 is set to 1 automatically by the firmware at power-up/reset, regardless of the saved value of I6841/I6891/I6991.

Examples:

Master number 0; Sync node address 0
Activated nodes 0-5; De-activated nodes 6-15:
I6891 =0000 0000 0000 0000 0011 1111 (binary) = $00003F
Master number 1; Sync node address 15 ($F)
Activated nodes 0, 2, 4, 6, 8, 10, 12; other nodes de-activated:
I6941 = 0001 1111 0001 0101 0101 0101 (binary) = $1F1555

Servo IC I-Variables

I-variables in the range I7000 to I7999 control the hardware setup of the "Servo ICs" in a Turbo PMAC system.

There can be up to 10 Servo ICs in a Turbo PMAC system: Servo IC 0 to Servo IC 9; in the I-variable numbering scheme, the Servo IC number determines the 100's digit of the I-variable number, represented by the letter 'm' to refer to any IC generally (e.g. I7m00). Servo ICs 0 and 1 are on board the Turbo PMAC itself, or on piggyback boards in the 3U Turbo Stack; Servo ICs 2 through 9 are "off-board"; on ACC-24 or similar boards with their own Servo ICs.

Servo ICs can be either PMAC(1)-style (DSPGATE) or PMAC2-style (DSPGATE1). The meaning of a particular I-variable number can differ depending on which type of IC is used. The off-board ICs do not have to be of the same type as the on-board ICs.

In firmware versions V1.933 and older, the user had tell Turbo PMAC which off-board Servo ICs were present with I65, and which type they were with I66. In V1.934 and newer, Turbo PMAC automatically detects the presence and type of all Servo ICs present at each power-up/reset, enables the I-variables for those present, and selects the I-variables for type of each IC.

Each Servo IC has 4 channels of servo interface circuitry, numbered IC channels 1 to 4. In the I-variable numbering scheme, the IC channel number determines the 10's digit of the I-variable number, represented by the letter 'n' to refer to any channel generally (e.g. I7mn3).

For even-numbered Servo ICs 0, 2, 4, 6, and 8, the channel numbers 1 – 4 on the IC match the channel numbers 1 – 4 on the board. For odd-numbered Servo ICs 1, 3, 5, 7, and 9, which require the presence of Option 1 on the board, the IC channel numbers 1 – 4 correspond to board channel numbers 5 – 8, respectively.

The following table shows key data about each potential Servo IC in the system:

Servo IC #	Board	Board Channel #s	I-Variables	Base Address	Default Assignment
0	Turbo PMAC	1 – 4	I7000 – I7049	$078000	Motors 1-4
1	Turbo PMAC	5 – 8	I7100 – I7149	$078100	Motors 5-8
2	1st ACC-24	1 – 4	I7200 – I7249	$078200	Motors 9-12
3	1st ACC-24	5 – 8	I7300 – I7349	$078300	Motors 13-16
4	2nd ACC-24	1 – 4	I7400 – I7449	$079200	Motors 17-20
5	2nd ACC-24	5 – 8	I7500 – I7549	$079300	Motors 21-24
6	3rd ACC-24	1 – 4	I7600 – I7649	$07A200	Motors 25-28
7	3rd ACC-24	5 – 8	I7700 – I7749	$07A300	Motors 29-32
8	4th ACC-24	1 – 4	I7800 – I7849	$07B200	none
9	4th ACC-24	5 – 8	I7900 – I7949	$07B300	none

Note:

Some new accessory boards for the UMAC 3U-format Turbo PMAC employ alternate addressing of Servo ICs, labeled Servo ICs 2* through 9*. Servo IC *m** is controlled by I-variables numbered 50 higher than Servo IC *m*,(e.g. I7250 – I7299 for Servo IC 2*) and is addressed $20 higher (e.g. $078220 for Servo IC 2*).

PMAC2-Style Multi-Channel Servo IC I-Variables

I-variables in the range I7m00 to I7m09 control global and multi-channel aspects of the hardware setup using the first "DSPGATE1" Servo IC on the Turbo PMAC2. On Turbo PMAC2 Ultralite boards, there are no DSPGATE1 Servo ICs on board, so these functions are implemented in the DSPGATE2 ASIC, which is controlled by variables in the I6800s.

I7m00 Servo IC m MaxPhase/PWM Frequency Control

Range: 0 - 32767

Units: MaxPhase Frequency = 117,964.8 kHz / [2*I7m00+3]
PWM Frequency = 117,964.8 kHz / [4*I7m00+6]

Default: 6527
MaxPhase Frequency = 117,964.8 / 13057 = 9.0346 kHz
PWM Frequency = 117,964.8 / 26114 = 4.5173 kHz

I7m00 controls the internal "MaxPhase" clock frequency, and the PWM frequency for the four machine interface channels, on PMAC2-style Servo IC m (m = 0 to 9). The internally generated Phase and Servo clocks on Servo IC m are derived from the MaxPhase clock.

If the Servo IC is used to generate the Phase and Servo clocks for the PMAC system (as set by I19 and the I7m07 variables), this variable is part of the control for the frequency of these system clocks.

On Turbo PMAC2 boards that are not "Ultralite", Servo IC 0 typically provides the Phase and Servo clock signals for the entire board (I7007 = 0), so I7000 is used to derive the Phase clock and Servo clock frequencies for the board, along with I7001 and I7002. (On Turbo PMAC2 Ultralite boards, this function is controlled by I6800, I6801, and I6802, because MACRO IC 0 controls the board clock frequencies on these boards.)

I7m00 controls these frequencies by setting the limits of the PWM up-down counter, which increments and decrements at the PWMCLK frequency of 117,964.8 kHz (117.9648 MHz).

The actual Phase clock frequency is divided down from the maximum phase clock according to the setting of I7001. On the falling edge of the phase clock, PMAC2 samples any serial analog-to-digital converters connected to its Servo ICs (as for phase current measurement), and interrupts the processor to start any necessary phase commutation and digital current-loop algorithms. Even if phasing and current-loop algorithms are not used, the MaxPhase and Phase Clock frequencies are important because the servo clock is derived from the phase clock.

The PWM frequency determines the actual switching frequency of amplifiers connected to any of four machine interface channels with the direct PWM command. It is only important if the direct PWM command signal format is used.

The maximum value that can be written into the PWM command register without full saturation is I7m00+1 on the positive end, and –I7m00-2 on the negative end. Generally, the "PWM scale factor" Ixx66 for Motor, which determines the maximum PWM command magnitude, is set to I7m00 + 10%.

Generally I7m00 for Servo IC m that is not controlling the system Phase clock frequency is set to the same value as I7000 or I6800, which controls the board's Phase clock frequency (with I7001 or I6801). If a different PWM frequency is desired for the PWM outputs on Servo IC m, I7m00 should be set so that:

$$\frac{2 * PWMFreq(kHz)}{PhaseFreq} = \{ Integer \}$$

This will keep the PWM hardware on these channels in synchronization with the software algorithms driven by the system Phase clock.. For example, if the phase frequency is 10 kHz, the PWM frequency for channels 5 to 8 can be 5, 10, 15, 20, (etc.) kHz.

To set I7m00 for a desired PWM frequency, the following formula can be used:

$$I7m00 = \frac{117{,}964.8(kHz)}{4 * PWM\_Freq(kHz)} - 1 \text{ (rounded down)}$$

To set I7000 for a desired "maximum phase" clock frequency, the following formula can be used:

$$I7000 = \frac{117{,}964.8(kHz)}{2 * MaxPhaseFreq(kHz)} - 1 \text{ (rounded down)}$$

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m50, not I7m00.

Example:

To set a PWM frequency of 10 kHz and therefore a MaxPhase clock frequency of 20 kHz:

I7000 = (117,964.8 kHz / [4*10 kHz]) - 1 = 2948

To set a PWM frequency of 7.5 kHz and therefore a MaxPhase clock frequency of 15 kHz:

I7000 = (117,964.8 kHz / [4*7.5 kHz]) - 1 = 3931

I7m01 Servo IC m Phase Clock Frequency Control

Range: 0 - 15

Units: Phase Clock Frequency = MaxPhase Frequency / (I7m01+1)

Default: 0
Phase Clock Frequency = 9.0346 kHz / 1 = 9.0346 kHz
(with default value of I7m00)

I7m01, in conjunction with I7m00, determines the frequency of the Phase clock generated inside each PMAC2-style Servo IC m. However, only the Servo IC told to use and output its own Phase clock with I7m07, typically Servo IC 0, uses the Phase clock signal it generates. This means that I7001, in conjunction with I7000, typically controls the Phase clock frequency for the entire Turbo PMAC2 system. (For Turbo PMAC2 Ultralite boards, I6801 and I6800 control this.) Each cycle of the Phase clock, motor phase commutation and digital current-loop algorithms are performed for specified motors.

Specifically, I7m01 controls how many times the Phase clock frequency is divided down from the "maximum phase" clock, whose frequency is set by I7m00. The Phase clock frequency is equal to the "maximum phase" clock frequency divided by (I7m01+1). I7m01 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I7m01 is:

$$I7m01 = \frac{MaxPhaseFreq(kHz)}{PhaseFreq(kHz)} - 1$$

The ratio of MaxPhase Freq. to Phase Clock Freq. must be an integer.

Note:

If the phase clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and I6000 and I6001 can be set to supportable values.

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m51, not I7m01.

Example:
With a 20 kHz MaxPhase Clock frequency established by I7000, and a desired 6.67 kHz PHASE clock frequency, the ratio between MaxPhase and Phase is 3:

I7001 = (20 / 6.67) - 1 = 3 -1 = 2

See Also: I19, I7m00, I7m02, I7m07, I6800, I6801, I6802, I6807

I7m02 Servo IC m Servo Clock Frequency Control

Range: 0 - 15

Units: Servo Clock Frequency = Phase Clock Frequency / (I7m02+1)

Default: 3
Servo Clock Frequency = 9.0346 kHz / (3+1) = 2.2587 kHz
(with default values of I7m00 and I7m01)

I7m02, in conjunction with I7m01 and I7m00, determines the frequency of the Servo clock generated inside each PMAC2-style Servo IC. However, only the Servo IC told to use and output its own Servo clock with I7m07, typically Servo IC 0, uses the Servo clock signal it generates.

This means that I7002, in conjunction with I7001 and I7000, controls the Servo clock frequency for the entire Turbo PMAC2 system. (For Turbo PMAC2 Ultralite boards, I6802, I6801 and I6800 control this.) Each cycle of the Servo clock, Turbo PMAC2 updates the commanded position for each activated motor, and executes the servo algorithm to compute the command to the amplifier or the commutation algorithm.

Specifically, I7m02 controls how many times the Servo clock frequency is divided down from the Phase clock, whose frequency is set by I7m01 and I7m00. The Servo clock frequency is equal to the Phase clock frequency divided by (I7m02+1). I7m02 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I7m02 is:

$$I7m02 = \frac{PhaseFreq(\,kHz\,)}{ServoFreq(\,kHz\,)} - 1$$

The ratio of Phase Clock frequency to Servo Clock frequency must be an integer.

For execution of trajectories at the proper speed, I10 must be set properly to tell the trajectory generation software what the Servo clock cycle time is. The formula for I10 is:

$$I10 = \frac{8{,}388{,}608}{ServoFreq(\,kHz\,)}$$

In terms of the variables that determine the Servo clock frequency on a (non-Ultralite) Turbo PMAC2 board, the formula for I10 is:

$$I10 = \frac{640}{9}(2 * I7000 + 3)(I7001 + 1)(I7002 + 1)$$

At the default servo clock frequency, I10 should be set to 3,713,707 in order that PMAC's interpolation routines use the proper servo update time.

Note:

If the servo clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and I7000, I7001, and I7002 can be set to supportable values.

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m52, not I7m02.

Example:

With a 6.67 kHz Phase Clock frequency established by I7000 and I7001, and a desired 3.33 kHz Servo Clock frequency:

I7002 = (6.67 / 3.33) - 1 = 2 - 1 = 1

See Also: I10, I19, I7m00, I7m01, I7m07, I6800, I6801, I6802, I6807

I7m03 Servo IC m Hardware Clock Control

Range: 0 - 4095

Units: Individual Clock Dividers
I7m03 = Encoder SCLK Divider
+ 8 * PFM_CLK Divider
+ 64 * DAC_CLK Divider
+ 512 * ADC_CLK Divider
where:
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ Encoder SCLK Divider)
PFM_CLK Frequency = 39.3216 MHz / (2 ^ PFM_CLK Divider)
DAC_CLK Frequency = 39.3216 MHz / (2 ^ DAC_CLK Divider)
ADC_CLK Frequency = 39.3216 MHz / (2 ^ ADC_CLK Divider)

Default: 2258 = 2 + (8 * 2) + (64 * 3) + (512 * 4)
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
PFM_CLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
DAC_CLK Frequency = 39.3216 MHz / (2 ^ 3) = 4.9152 MHz
ADC_CLK Frequency = 39.3216 MHz / (2 ^ 4) = 2.4576 MHz

I7m03 controls the frequency of four hardware clock frequencies -- SCLK, PFM_CLK, DAC_CLK, and ADC_CLK -- for the four machine interface channels on PMAC2-Style Servo IC m. It is a 12-bit variable consisting of four independent 3-bit controls, one for each of the clocks. Each of these clock frequencies can be divided down from a starting 39.3216 MHz frequency by powers of 2, 2^N, from 1 to 128 times (N=0 to 7). This means that the possible frequency settings for each of these clocks are:

Frequency	Divide by	Divider N in $1/2^N$
39.3216 MHz	1	0
19.6608 MHz	2	1
9.8304 MHz	4	2
4.9152 MHz	8	3
2.4576 MHz	16	4
1.2288 MHz	32	5
614.4 kHz	64	6
307.2 kHz	128	7

Very few Turbo PMAC2 users will be required to change the setting of I7m03 from the default value.

SCLK: The encoder sample clock signal SCLK controls how often Servo IC m's digital hardware looks at the encoder and flag inputs. The Servo IC can take at most one count per SCLK cycle, so the SCLK frequency is the absolute maximum encoder count frequency. SCLK also controls the signal propagation through the digital delay filters for the encoders and flags; the lower the SCLK frequency, the greater the noise pulse that can be filtered out. The SCLK frequency should optimally be set to the lowest value that can accept encoder counts at the maximum possible rate.

PFM_CLK: The pulse-frequency-modulation clock PFM_CLK controls the PFM circuitry that is commonly used for stepper drives. The maximum pulse frequency possible is 1/4 of the PFM_CLK frequency. The PFM_CLK frequency should optimally be set to the lowest value that can generate pulses at the maximum frequency required.

DAC_CLK: The DAC_CLK controls the serial data frequency into D/A converters. If these converters are on Delta Tau-provided accessories, the DAC_CLK setting should be left at the default value.

ADC_CLK: The ADC_CLK controls the serial data frequency from A/D converters. If these converters are on Delta Tau-provided accessories, the ADC_CLK setting should be left at the default value.

To determine the clock frequencies set by a given value of I7m03, use the following procedure:

1. Divide I7m03 by 512 and round down to the nearest integer. This value N1 is the ADC_CLK divider.
2. Multiply N1 by 512 and subtract the product from I7m03 to get I7m03'. Divide I7m03' by 64 and round down to the nearest integer. This value N2 is the DAC_CLK divider.
3. Multiply N2 by 64 and subtract the product from I7m03' to get I7m03". Divide I7m03" by 8 and round down to the nearest integer. This value N3 is the PFM_CLK divider.
4. Multiply N3 by 8 and subtract the product from I7m03". The resulting value N4 is the SCLK divider.

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m53, not I7m03.

Examples:

The maximum encoder count frequency in the application is 800 kHz, so the 1.2288 MHz SCLK frequency is chosen. A pulse train up to 500 kHz needs to be generated, so the 2.4576 MHz PFM_CLK frequency is chosen. The default serial DACs and ADCs provided by Delta Tau are used, so the default DAC_CLK frequency of 4.9152 MHz and the default ADC_CLK frequency of 2.4576 MHz are chosen.

From the table:

SCLK Divider N: 5
PFM_CLK Divider N: 4
DAC_CLK Divider N: 3
ADC_CLK Divider N: 4
I7m03 = 5 + (8 * 4) + (64 * 3) + (512 * 4) = 5 + 32 + 192 + 2048 = 2277

I7m03 has been set to 3429. What clock frequencies does this set?

N1 = INT (3429/512) = 6	ADC_CLK = 611.44 kHz
I7m03' = 3429 - (512*6) = 357	
N2 = INT (357/64) = 5	DAC_CLK = 1.2288 MHz
I7m03" = 357 - (64*5) = 37	
N3 = INT (37/8) = 4	PFM_CLK = 2.4576 MHz
N4 = 37 - (8*4) = 5	SCLK = 1.2288 MHz

See Also: I6803

I7m04 Servo IC m PWM Deadtime / PFM Pulse Width Control

Range: 0 - 255

Units: 16*PWM_CLK cycles / PFM_CLK cycles
PWM Deadtime = [16 / PWM_CLK (MHz)] * I7m04 = 0.135 usec * I7m04
PFM Pulse Width = [1 / PFM_CLK (MHz)] * I7m04
= PFM_CLK_period (usec) * I7m04

Default: 15
PWM Deadtime = 0.135 usec * 15 = 2.03 usec
PFM Pulse Width = [1 / 9.8304 MHz] * 15 = 1.526 usec (with default I7m03)

I7m04 controls the deadtime period between top and bottom on-times in the automatic PWM generation for machine interface channels on PMAC2-style Servo IC m (m = 0 to 9). In conjunction with I7m03, it also controls the pulse width for PMAC2's automatic pulse-frequency modulation generation for the machine interface channels on Servo IC m.

The PWM deadtime, which is the delay between the top signal turning off and the bottom signal turning on, and vice versa, is specified in units of 16 PWM_CLK cycles. This means that the deadtime can be specified in increments of 0.135 µsec. The equation for I7m04 as a function of PWM deadtime is:

$$I7m04 = \frac{DeadTime(\mu sec)}{0.135\mu sec}$$

The PFM pulse width is specified in PFM_CLK cycles, as defined by I7m03. The equation for I7m04 as a function of PFM pulse width and PFM_CLK frequency is:

$$I7m04 = PFM\_CLK\_Freq(MHz) * PFM\_Pulse\_Width(\mu sec)$$

In PFM pulse generation, the minimum off time between pulses is equal to the pulse width. This means that the maximum PFM output frequency is

$$PFM\_Max\_Freq(MHz) = \frac{PFM\_CLK\_Freq(MHz)}{2 * I7m04}$$

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m54, not I7m04.

Examples:

A PWM deadtime of approximately 1 microsecond is desired:

I7m04 ≅ 1 µsec / 0.135 µsec ≅ 7

With a 2.4576 MHz PFM_CLK frequency, a pulse width of 0.4 usec is desired:

I7m04 ≅ 2.4576 MHz * 0.4 usec ≅ 1

See Also: I7m03, I6804

I7m05 Servo IC m DAC Strobe Word

Range:	$000000 - $FFFFFF
Units:	Serial Data Stream (MSB first, starting on rising edge of phase clock)
Default:	$7FFFC0

I7m05 controls the DAC strobe signal for machine interface channels on Servo IC m. The 24-bit word set by I7m05 is shifted out serially on the DAC_STROB lines, MSB first, one bit per DAC_CLK cycle starting on the rising edge of the phase clock. The value in the LSB is held until the next phase clock cycle.

For a typical *n*-bit DACs, the strobe line is held high for *n*-1 clock cycles. Therefore, the common settings of this variable are:

- 18-bit DACs: $7FFFC0 (high for 17 clock cycles)
- 16-bit DACs: $7FFF00 (high for 15 clock cycles)
- 12-bit DACs: $7FF000 (high for 11 clock cycles)

The default I7m05 value of $7FFFC0 is suitable for the 18-bit DACs on the ACC-8E Analog Interface Board. I7m05 should not be changed from the default unless different DACs are used.

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m55, not I7m05.

I7m06 Servo IC m ADC Strobe Word

Range: $000000 - $FFFFFF

Units: Serial Data Stream (MSB first, starting on rising edge of phase clock)

Default: $FFFFFE

I7m06 controls the ADC strobe signal for machine interface channels on Servo IC m. The 24-bit word set by I7m06 is shifted out serially on the ADC_STROB lines, MSB first, one bit per DAC_CLK cycle starting on the rising edge of the phase clock. The value in the LSB is held until the next phase clock cycle.

The first '1' creates a rising edge on the ADC_STROB output that is typically used as a "start-convert" signal. Some A/D converters just need this rising edge for the conversion; others need the signal to stay high all of the way through the conversion. The LSB of I7m06 should always be set to 0 so that a rising edge is created on the next cycle. The default I7m06 value of $FFFFFE is suitable for virtually all A/D converters.

The A/D converters used on matching Delta Tau products just need the rising edge at the start of a conversion cycle; this permits intermediate bits in the data stream to be used as special control bits. Delta Tau's ACC-8T Supplemental Flag Multiplexer Board uses these bits to control the multiplexing; Delta Tau's ACC-8K1 Fanuc C/S-Series PWM Interface Board uses these bits to control the magnetic contactors on the drives.

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m56, not I7m06.

I7m07 Servo IC m Phase/Servo Clock Direction

Range: 0 - 3

Units: None

Default: I7007 = 0 (non-Ultralite); = 3 (Ultralite)
I7107 – I7907 = 3

I7m07 controls whether Servo IC m uses its own internally generated Phase and Servo clock signals as controlled by I7m00, I7m01, and I7m02, or whether it uses Phase and Servo clock signals from an outside source.

In any Turbo PMAC2 system, there must be either one and only one source of servo and phase clock signals for the system – one of the Servo ICs or MACRO ICs, or a source external to the system. Only in a 3U-format Turbo PMAC2 system (UMAC Turbo or 3U Turbo Stack) can the system clock signals come from an accessory board. In all other Turbo PMAC2 systems, the system clock signals must come from and IC on the base PMAC2 boards, or be brought from an external source through the serial port.

I7m07 is a 2-bit value. Bit 0 is set to 0 for the IC to use its own Phase clock signal and output it; it is set to 1 to use an externally input Phase clock signal. Bit 1 is set to 1 for the IC to use its own Servo clock signal and output it; it is set to 1 to use an externally input Servo clock signal. This yields four possible values for I7m07:

- I7m07 = 0: Internal Phase clock; internal Servo clock
- I7m07 = 1: External Phase clock; internal Servo clock
- I7m07 = 2: Internal Phase clock; external Servo clock
- I7m07 = 3: External Phase clock; external Servo clock

In all normal use, I7m07 is either set to 0 (on at most one IC) or 3 (on all the other ICs).

In general, Servo IC 0 or MACRO IC 0 (on an Ultralite board that has no Servo ICs) will be used to generate Phase and Servo clock signals for the entire PMAC systems, so I7007 is set to 0 (or I6807 on an Ultralite board), and I7107 through I7907 are set to 3.

During re-initialization, Turbo PMAC2 determines which IC it will use as the source of its system Phase and Servo clock signals, setting I19 to the number of the clock-direction I-variable whose IC is selected as the source. This clock-direction I-variable is then automatically set to 0; all other clock-direction I-variables are set to 1 or 3. Most users will never change these settings.

When a clock-direction I-variable is commanded to its default value (e.g. **I7207=***), Turbo PMAC2 looks to the value of I19 to determine whether this I-variable is set to 0 or 3.

On the reset of a 3U-format Turbo PMAC2 system (UMAC Turbo or 3U Turbo Stack), the values set for these I-variables are determined by the saved value of I19, and not by the saved values of these I-variables themselves. On these systems, if you wish to change which IC is the source of the system clocks, change the value of I19, save this setting, and reset the card.

In other Turbo PMAC2 systems, if you wish to change which IC is the source of the system clocks, it is best to change both clock-direction I-variables on a single command line (e.g. **I6807=1 I7007=0**), then **SAVE** these new settings.

If all of the Servo and MACRO ICs in a Turbo PMAC2 system have been set up for external Phase and Servo clocks, but these clock signals are not provided, the Turbo PMAC2 will immediately trip its watchdog timer.

For accessory boards in which alternate addressing of the Servo IC is used (labeled Servo IC m*), this function is controlled by I7m57, not I7m07.

PMAC2-Style Channel-Specific Servo IC I-Variables

(For Servo IC *m* Channel *n*, where *m* = 0 to 9, and *n* = 1 to 4)

I7mn0 Servo IC m Channel n Encoder/Timer Decode Control

Range: 0 - 15

Units: None

Default: 7

I7mn0 controls how the input signal for Encoder n on a PMAC2-style Servo IC m is decoded into counts. As such, this defines the sign and magnitude of a "count". The following settings may be used to decode an input signal.

- I7mn0 = 0: Pulse and direction CW
- I7mn0 = 1: x1 quadrature decode CW
- I7mn0 = 2: x2 quadrature decode CW
- I7mn0 = 3: x4 quadrature decode CW
- I7mn0 = 4: Pulse and direction CCW
- I7mn0 = 5: x1 quadrature decode CCW
- I7mn0 = 6: x2 quadrature decode CCW
- I7mn0 = 7: x4 quadrature decode CCW
- I7mn0 = 8: Internal pulse and direction
- I7mn0 = 9: Not used
- I7mn0 = 10: Not used
- I7mn0 = 11: x6 hall-format decode CW*
- I7mn0 = 12: MLDT pulse timer control
  (internal pulse resets timer; external pulse latches timer)
- I7mn0 = 13: Not used
- I7mn0 = 14: Not used
- I7mn0 = 15: x6 hall-format decode CCW*

*requires version B or newer of the DSPGATE1 Servo IC.

In any of the quadrature decode modes, the Servo IC is expecting two input waveforms on CHAn and CHBn, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. Times-one (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The clockwise (CW) and counterclockwise (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa).

WARNING:

Changing the direction sense of the decode for the feedback encoder of a motor that is operating properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes. The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

In the pulse-and-direction decode modes, the Servo IC is expecting the pulse train on CHAn, and the direction (sign) signal on CHBn. If the signal is unidirectional, the CHBn line can be allowed to pull up to a high state, or it can be hardwired to a high or low state.

If I7mn0 is set to 8, the decoder inputs the pulse and direction signal generated by Channel n's pulse frequency modulator (PFM) output circuitry. This permits the PMAC2 to create a phantom closed loop when driving an open-loop stepper system. No jumpers or cables are needed to do this; the connection is entirely within the Servo IC. The counter polarity automatically matches the PFM output polarity.

If I7mn0 is set to 11 or 15, Channel n is expecting three Hall-sensor format inputs on CHAn, CHBn, and CHCn, each with approximately 50% duty cycle, and approximately one-third (120°e) of a cycle out of phase with each other. The decode circuitry will generate one count on each edge of each signal, yielding 6 counts per signal cycle ("x6 decode"). The difference between 11 and 15 is which direction of signal causes the counter to count up.

If I7mn0 is set to 12, the timer circuitry is set up to read magnetostrictive linear displacement transducers (MLDTs) such as Temposonics™. In this mode, the timer is cleared when the PFM circuitry sends out the excitation pulse to the sensor on PULSEn, and it is latched into the memory-mapped register when the excitation pulse is received on CHAn.

I7mn1 Servo IC m Channel n Position Compare Channel Select

Range: 0 - 1

Units: None

Default: 0

I7mn1 controls which channel's encoder counter is tied to the position compare circuitry for Channel n on a PMAC2-style Servo IC m. It has the following possible settings:

- I7mn1 = 0: Use Channel n encoder counter for position compare function
- I7mn1 = 1: Use Channel 1 encoder counter on IC for position compare function

When I7mn1 is set to 0, Channel n's position compare registers are tied to the channel's own encoder counter, and the position compare signal appears only on the EQU output for that channel.

When I7mn1 is set to 1, the channel's position compare register is tied to the first encoder counter on the Servo IC, and the position compare signal appears both on Channel n's EQU output, and combined into the EQU output for Channel 1 on the Servo IC (EQU1 or EQU5 on the board); executed as a logical OR.

I7m11 performs no effective function, so is always 1. It cannot be set to 0.

I7mn2 Servo IC m Channel n Capture Control

Range: 0 - 15

Units: none

Default: 1

I7mn2 determines which input signal or combination of signals for Channel n of a PMAC2-style Servo IC m, and which polarity, triggers a hardware position capture of the counter for Encoder n. If a flag input (home, limit, or user) is used, I7mn3 determines which flag. Proper setup of this variable is essential for a successful homing search move or other move-until-trigger for the Motor xx using Channel n for its position-loop feedback and flags if the super-accurate hardware position capture function is used. If Ixx97 is at its default value of 0 to select hardware capture and trigger, this variable must be set up properly.

The following settings of I7mn2 may be used:

- I7mn2 = 0: Immediate capture
- I7mn2 = 1: Capture on Index (CHCn) high
- I7mn2 = 2: Capture on Flag n high
- I7mn2 = 3: Capture on (Index high AND Flag n high)
- I7mn2 = 4: Immediate capture
- I7mn2 = 5: Capture on Index (CHCn) low
- I7mn2 = 6: Capture on Flag n high
- I7mn2 = 7: Capture on (Index low AND Flag n high)
- I7mn2 = 8: Immediate capture
- I7mn2 = 9: Capture on Index (CHCn) high
- I7mn2 = 10: Capture on Flag n low
- I7mn2 = 11: Capture on (Index high AND Flag n low)
- I7mn2 = 12: Immediate capture
- I7mn2 = 13: Capture on Index (CHCn) low
- I7mn2 = 14: Capture on Flag n low
- I7mn2 = 15: Capture on (Index low AND Flag n low)

Only flags and index inputs of the same channel number as the encoder may be used for hardware capture of that encoder's position. This means that to use the hardware capture feature for the homing search move, Ixx25 must use flags of the same channel number as the encoder that Ixx03 uses for position-loop feedback.

The trigger is armed when the position capture register is read. After this, as soon as the Servo IC hardware sees that the specified input lines are in the specified states, the trigger will occur -- it is level-triggered, not edge-triggered.

I7mn3 Servo IC m Channel n Capture Flag Select Control

Range: 0 - 3

Units: none

Default: 0

I7mn3 determines which of the "Flag" inputs will be used for hardware position capture (if one is used) of the encoder counter of Channel n on a PMAC2-style Servo IC m. I7mn2 determines whether a flag is used and which polarity of the flag will cause the trigger. The possible values of I7mn3 and the flag each selects is:

- I7mn3 = 0: HOMEn (Home Flag n)

- I7mn3 = 1: PLIMn (Positive End Limit Flag n)
- I7mn3 = 2: MLIMn (Negative End Limit Flag n)
- I7mn3 = 3: USERn (User Flag n)

I7mn3 is typically set to 0 for homing search moves in order to use the home flag for the channel. It is typically set to 3 afterwards to select the User flag if other uses of the hardware position capture function are desired, such as for probing and registration. If you wish to capture on the PLIMn or MLIMn overtravel limit flags, you probably will want to disable their normal functions with Ixx25, or use a channel n where none of the flags is used for the normal axis functions.

I7mn4 Servo IC m Channel n Encoder Gated Index Select

Range: 0 - 1

Units: none

Default: 0

I7mn4 controls whether the raw encoder index channel input or a version of the input gated by the AB-quadrature state is used for position capture of Encoder n on a PMAC2-style Servo IC m. It has the following possible settings:

- I7mn4 = 0: Use ungated index for encoder position capture
- I7mn4 = 1: Use index gated by quadrature channels for position capture

When I7mn4 is set to 0, the encoder index channel input (CHCn) is passed directly into the position capture circuitry.

When I7mn4 is set to 1, the encoder index channel input (CHCn) is logically combined with ("gated by") the quadrature signals of Encoder n before going to the position capture circuitry. The intent is to get a "gated index" signal exactly one quadrature state wide. This provides a more accurate and repeatable capture, and makes the use of the capture function to confirm the proper number of counts per revolution very straightforward.

In order for the gated index capture to work reliably, the index pulse must reliably span one, but only one, "high-high" or "low-low" AB quadrature state of the encoder. I7mn5 allows you to select which of these two possibilities is used.

Note:

If I7mn4 is set to 1, but I7mn2 bit 0 is set to 0, so the index is not used in the position capture, then the encoder position is captured on the first edge of any of the U, V, or W flag inputs for the channel. In this case, bits 0, 1, and 2 of the channel status word tell what hall-state edge caused the capture.

I7mn5 Servo IC m Channel n Encoder Index Gate State/Demux Control

Range: 0 - 3

Units: none

Default: 0

I7mn5 is a 2-bit variable that controls two functions for the index channel of the encoder.

When using the "gated index" feature of a PMAC2-style Servo IC for more accurate position capture (I7mn4=1), bit 0 of I7mn5 specifies whether the raw index-channel signal fed into Encoder n of Servo IC m is passed through to the position capture signal only on the "high-high" quadrature state (bit 0 = 0), or only on the "low-low" quadrature state (bit 0 = 1).

Bit 1 of I7mn5 controls whether the Servo IC "de-multiplexes" the index pulse and the 3 hall-style commutation states from the third channel based on the quadrature state, as with Yaskawa incremental encoders. If bit 1 is set to 0, this de-multiplexing function is not performed, and the signal on the "C" channel of the encoder is used as the index only. If bit 1 is set to 1, the Servo IC breaks out the third-channel signal into four separate values, one for each of the four possible AB-quadrature states. The de-multiplexed hall commutation states can be used to provide power-on phase position using Ixx81 and Ixx91.

The following table shows what hall or index state is broken out for each of the four quadrature states:

A	B	C
1	1	Z
1	0	U
0	0	V
0	1	W

Note:

The "B" revision or newer of the DSPGATE1 Servo IC is required to support this hall de-multiplexing feature.

Note:

Immediately after power-up, the Yaskawa encoder automatically cycles its AB outputs forward and back through a full quadrature cycle to ensure that all of the hall commutation states are available to the controller before any movement is started. However, if the encoder is powered up at the same time as the Turbo PMAC, this will happen before the Servo IC is ready to accept these signals. Bit 2 of the channel's status word, "Invalid De-multiplex", will be set to 1 if the Servo IC has not seen all of these states when it was ready for them. To use this feature, it is recommended that the power to the encoder be provided through a software-controlled relay to ensure that valid readings of all states have been read before using these signals for power-on phasing.

I7mn5 has the following possible settings:

- I7mn5 = 0: Gate index with "high-high" quadrature state (GI = A & B & C), no demux
- I7mn5 = 1: Gate index with "low-low" quadrature state (GI = A/ & B/ & C), no demux
- I7mn5 = 2 or 3: De-multiplex hall and index from third channel, gating irrelevant

I7mn6 Servo IC m Channel n Output Mode Select

Range: 0 - 3

Units: none

Default: 0

I7mn6 controls what output formats are used on the command output signal lines for machine interface channel n of a PMAC2-style Servo IC m. It has the following possible settings:

- I7mn6 = 0: Outputs A & B are PWM; Output C is PWM
- I7mn6 = 1: Outputs A & B are DAC; Output C is PWM
- I7mn6 = 2: Outputs A & B are PWM; Output C is PFM
- I7mn6 = 3: Outputs A & B are DAC; Output C is PFM

If a three-phase direct PWM command format is desired, I7mn6 should be set to 0. If signal outputs for (external) digital-to-analog converters are desired, I7mn6 should be set to 1 or 3. In this case, the C output can be used as a supplemental (non-servo) output in either PWM or PFM form. For example, it can be used to excite an MLDT sensor (e.g. Temposonics™) in PFM form.

I7mn7 Servo IC m Channel n Output Invert Control

Range: 0 - 3

Units: none

Default: 0

I7mn7 controls the high/low polarity of the command output signals for Channel n on a PMAC2-style Servo IC m. It has the following possible settings:

- I7mn7 = 0: Do not invert Outputs A & B; Do not invert Output C
- I7mn7 = 1: Invert Outputs A & B; Do not invert Output C
- I7mn7 = 2: Do not invert Outputs A & B; Invert Output C
- I7mn7 = 3: Invert Outputs A & B; Invert Output C

The default non-inverted outputs are high true. For PWM signals on Outputs A, B, and C, this means that the transistor-on signal is high. Delta Tau PWM-input amplifiers, and most other PWM-input amplifiers, expect this non-inverted output format. For such a 3-phase motor drive, I7mn7 should be set to 0.

Note:

If the high/low polarity of the PWM signals is wrong for a particular amplifier, what was intended to be deadtime between top and bottom on-states as set by I6m04 becomes overlap. If the amplifier-input circuitry does not lock this out properly, this causes an effective momentary short circuit between bus power and ground. This would destroy the power transistors very quickly.

For PFM signals on Output C, non-inverted means that the pulse-on signal is high (direction polarity is controlled by I7mn8). During a change of direction, the direction bit will change synchronously with the leading edge of the pulse, which in the non-inverted form is the rising edge. If the drive requires a set-up time on the direction line before the rising edge of the pulse, the pulse output can be inverted so that the rising edge is the trailing edge, and the pulse width (established by I6m04) is the set-up time.

For DAC signals on Outputs A and B, non-inverted means that a 1 value to the DAC is high. DACs used on Delta Tau accessory boards, as well as all other known DACs always expect non-inverted inputs, so I6mn7 should always be set to 0 or 2 when using DACs on Channel n.

Note:

Changing the high/low polarity of the digital data to the DACs has the effect of inverting the voltage sense of the DACs' analog outputs. This changes the polarity match between output and feedback. If the feedback loop had been stable with negative feedback, this change would create destabilizing positive feedback, resulting in a dangerous runaway condition that would only be stopped when the motor exceeded Ixx11 fatal following error

I7mn8 Servo IC m Channel n PFM Direction Signal Invert Control

Range: 0 - 1

Units: none

Default: 0

I7mn8 controls the polarity of the direction output signal in the pulse-and-direction format for Channel n of a PMAC2-style Servo IC m. It is only active if I7mn6 has been set to 2 or 3 to use Output C as a pulse-frequency-modulated (PFM) output. It has the following possible settings:

- I7mn8 = 0: Do not invert direction signal (+ = low; - = high)
- I7mn8 = 1: Invert direction signal (- = low; + = high)

If I7mn8 is set to the default value of 0, a positive direction command provides a low output; if I7mn8 is set to 1, a positive direction command provides a high output.

I7mn9 Servo IC m Channel n Hardware-1/T Control

Range: 0 – 1

Units: none

Default: 0

I7mn9 controls whether the "hardware-1/T" functionality is enabled for Channel n of a PMAC2-style Servo IC m. If I7mn9 is set to the default value of 0, the hardware-1/T functionality is disabled, permitting the use of the "software-1/T" position extension that is calculated by default with encoder conversion method $0. If I7mn9 is set to 1, the hardware-1/T functionality is enabled (if present on the IC), and the software-1/T cannot be used.

The hardware-1/T functionality is present only on Revision D and newer of the PMAC2-style DSPGATE1 IC, released at the beginning of the year 2002. Setting I7mn9 to 1 on an older revision IC does nothing – software-1/T functions can still be used. However, it is strongly recommended that I7mn9 be left at 0 in this case, to prevent possible problems when copying a configuration to newer hardware.

When the hardware-1/T functionality is enabled, the IC computes a new fractional-count position estimate based on timers every SCLK (encoder sample clock) cycle. This permits the fractional count data to be used for hardware capture and compare functions, enhancing their resolution. The sub-count position-capture data can be used automatically in Turbo PMAC triggered-move functions if bit 12 of Ixx24 is set to 1. This is particularly useful when the IC is used on an ACC-51 high-resolution analog-encoder interpolator board. However, it replaces the timer registers at the first two "Y" addresses for the channel with fractional count position data, so the traditional software-1/T method of the conversion table cannot work if this is enabled.

If you enable the hardware-1/T functionality, and want to be able to use 1/T interpolation in your servo loop, you must use the hardware-1/T extension method ($C method digit with the mode switch bit set to 1) in the encoder conversion table.

PMAC(1)-Style Servo IC Setup I-Variables

I7mn0 Servo IC m Channel n Encoder/Timer Decode Control

Range: 0 - 15

Units: None

Default: 7

I7mn0 controls how the input signal for Encoder n on PMAC(1)-style Servo IC m is decoded into counts. As such, this defines the sign and magnitude of a "count". The following settings may be used to decode an input signal.

- I7mn0 = 0: Pulse and direction CW

- I7mn0 = 1: x1 quadrature decode CW
- I7mn0 = 2: x2 quadrature decode CW
- I7mn0 = 3: x4 quadrature decode CW
- I7mn0 = 4: Pulse and direction CCW
- I7mn0 = 5: x1 quadrature decode CCW
- I7mn0 = 6: x2 quadrature decode CCW
- I7mn0 = 7: x4 quadrature decode CCW

In any of the quadrature decode modes, PMAC is expecting two input waveforms on CHAn and CHBn, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. "Times-one" (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The "clockwise" (CW) and "counterclockwise" (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa).

WARNING:

Changing the direction sense of the encoder decode for a motor that is servoing properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes (for motors not commutated by PMAC from the same encoder). The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

In the pulse-and-direction decode modes, PMAC is expecting the pulse train on CHAn, and the direction (sign) signal on CHBn. If the signal is unidirectional, the CHBn input can be tied high (to +5V) or low (to GND), or, if set up by E18-E21, E24-E27 for single-ended (non-differential) input, left to float high.

Any spare encoder counters may be used as fast and accurate timers by setting this parameter in the 8 to 15 range. In this range, any input signal is ignored. The following settings may be used in timer mode

- I7mn0 = 8: Timer counting up at SCLK/10
- I7mn0 = 9: Timer counting up at SCLK/10
- I7mn0 = 10: Timer counting up at SCLK/5
- I7mn0 = 11: Timer counting up at SCLK/2.5
- I7mn0 = 12: Timer counting down at SCLK/10
- I7mn0 = 13: Timer counting down at SCLK/10
- I7mn0 = 14: Timer counting down at SCLK/5
- I7mn0 = 15: Timer counting down at SCLK/2.5

These timers are particularly useful when the related capture and compare registers are utilized for precise event marking and control, including triggered time base. The SLCK frequency is determined by the crystal clock frequency and E34-E38.

I7mn1 Servo IC m Channel n Encoder Filter Disable

Range: 0 - 1

Units: None

Default: 0

I7mn1 controls whether the Encoder n on PMAC(1)-style Servo IC m enables or disables its digital delay filter. The possible settings of I7mn1 are:

- I7mn1 = 0: Encoder n digital delay filter enabled
- I7mn1 = 1: Encoder n digital delay filter disabled (bypassed)

The filter is a 3-stage digital delay filter with best-2-of-3 voting to help suppress noise spikes on the input lines. It does introduce a small delay into the signal, which can be unacceptable if the motor is using interpolated sub-count parallel data input, because of loss of synchronization between the quadrature and parallel data signals.

Generally, the only people to disable this filter are those using the special interpolated parallel data format. These people should disable the filters both on the encoder for their quadrature signals and the encoder matching their parallel data input.

The sampling frequency for the filter is that of the SCLK signal, which is set by the master clock frequency and jumpers E34-E38. The higher the frequency of SCLK, the higher the possible count rate, but the narrower the pulse that can be filtered out. SCLK should be set to allow the maximum expected encoder frequency, but no faster, in order to provide the maximum noise protection.

I7mn2 Servo IC m Channel n Capture Control

Range: 0 - 15

Units: none

Default: 1

I7mn2 determines which input signal or combination of signals for PMAC(1)-style Servo IC m Channel n, and which polarity, triggers a hardware position capture of the counter for Encoder n. If a flag input (home, limit, or user) is used, I7mn3 determines which flag. Proper setup of this variable is essential for a successful homing search move or other move-until-trigger for the Motor xx using Channel n for its position-loop feedback and flags if the super-accurate hardware position capture function is used. If Ixx97 is at its default value of 0 to select hardware capture and trigger, this variable must be set up properly.

The following settings of I7mn2 may be used:

- I7mn2 = 0: Software control – armed
- I7mn2 = 1: Capture on Index (CHCn) high
- I7mn2 = 2: Capture on Flag n high
- I7mn2 = 3: Capture on (Index high AND Flag n high)
- I7mn2 = 4: Software control – triggered
- I7mn2 = 5: Capture on Index (CHCn) low
- I7mn2 = 6: Capture on Flag n high
- I7mn2 = 7: Capture on (Index low AND Flag n high)
- I7mn2 = 8: Software control – armed
- I7mn2 = 9: Capture on Index (CHCn) high
- I7mn2 = 10: Capture on Flag n low
- I7mn2 = 11: Capture on (Index high AND Flag n low)
- I7mn2 = 12: Software control – triggered

- I7mn2 = 13: Capture on Index (CHCn) low
- I7mn2 = 14: Capture on Flag n low
- I7mn2 = 15: Capture on (Index low AND Flag n low)

Only flags and index inputs of the same channel number as the encoder may be used for hardware capture of that encoder's position. This means that to use the hardware capture feature for the homing search move, Ixx25 must use flags of the same channel number as the encoder that Ixx03 uses for position-loop feedback.

The trigger is armed when the position capture register is read. After this, as soon as the Servo IC hardware sees that the specified input lines change into the specified states, the trigger will occur -- it is edge-triggered, not level-triggered.

Note:

Several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 8; when the parameter is then changed to 4 or 12, the capture is triggered (this is not of much practical use, but can be valuable for testing the capture function).

I7mn3 Servo IC m Channel n Capture Flag Select Control

Range: 0 - 3

Units: none

Default: 0

I7mn3 determines which of the "Flag" inputs will be used for hardware position capture (if one is used) of the encoder counter of Channel n on PMAC(1)-style Servo IC m. I7mn2 determines whether a flag is used and which polarity of the flag will cause the trigger. The possible values of I7mn3 and the flag each selects is:

- I7mn3 = 0: HMFLn (Home Flag n)
- I7mn3 = 1: -LIMn (Positive End Limit Flag n)
- I7mn3 = 2: +LIMn (Negative End Limit Flag n)
- I7mn3 = 3: FAULTn (Amplifier Fault Flag n)

I7mn3 is typically set to 0 for homing search moves in order to use the home flag for the channel. If you wish to capture on the -LIMn or +LIMn overtravel limit flags, or the FAULTn amplifier fault flag, you probably will want to disable their normal functions with Ixx25, or use a channel n where none of the flags is used for the normal axis functions.

Note:

The direction sense of the limit inputs is the opposite of what many people consider intuitive. That is, the +LIMn input, when taken high (opened), stops commanded motion in the negative direction; the -LIMn input, when taken high, stops commanded motion in the positive direction. It is important to confirm the direction sense of your limit inputs in actual operation.

Conversion Table I-Variables

I8000 - I8191 Conversion Table Setup Lines

Range: $000000 - $FFFFFF

Units: Modified Turbo PMAC Addresses

Defaults:

Turbo PMAC(1) Defaults

I-Var.	Setting	Meaning	I-Var.	Setting	Meaning
I8000	$078000	1/T Extension of Encoder 1	I8004	$078100	1/T Extension of Encoder 5
I8001	$078004	1/T Extension of Encoder 2	I8005	$078104	1/T Extension of Encoder 6
I8002	$078008	1/T Extension of Encoder 3	I8006	$078108	1/T Extension of Encoder 7
I8003	$07800C	1/T Extension of Encoder 4	I8007	$07810C	1/T Extension of Encoder 8
Note: I8008 - I8191 = 0					

Turbo PMAC2 Defaults

I-Var.	Setting	Meaning	I-Var.	Setting	Meaning
I8000	$078000	1/T Extension of Encoder 1	I8004	$078100	1/T Extension of Encoder 5
I8001	$078008	1/T Extension of Encoder 2	I8005	$078108	1/T Extension of Encoder 6
I8002	$078010	1/T Extension of Encoder 3	I8006	$078110	1/T Extension of Encoder 7
I8003	$078018	1/T Extension of Encoder 4	I8007	$078118	1/T Extension of Encoder 8
Note: I8008 - I8191 = 0					

Turbo PMAC2 Ultralite Defaults

I-Var.	Setting	Meaning	I-Var.	Setting	Meaning
I8000	$2F8420	MACRO Node 0 Reg. 0 Read	I8008	$2F8430	MACRO Node 8 Reg. 0 Read
I8001	$018000	24 bits, bit 0 LSB	I8009	$018000	24 bits, bit 0 LSB
I8002	$2F8424	MACRO Node 1 Reg. 0 Read	I8010	$2F8434	MACRO Node 9 Reg. 0 Read
I8003	$018000	24 bits, bit 0 LSB	I8011	$018000	24 bits, bit 0 LSB
I8004	$2F8428	MACRO Node 4 Reg. 0 Read	I8012	$2F8438	MACRO Node 12 Reg. 0 Read
I8005	$018000	24 bits, bit 0 LSB	I8013	$018000	24 bits, bit 0 LSB
I8006	$2F842C	MACRO Node 5 Reg. 0 Read	I8014	$2F843C	MACRO Node 13 Reg. 0 Read
I8007	$018000	24 bits, bit 0 LSB	I8015	$018000	24 bits, bit 0 LSB
Note: I8016 - I8191 = 0					

I8000 to I8191 form the 192 setup lines of the Turbo PMAC's Encoder Conversion Table (ECT). The main purpose of the ECT is to provide a pre-processing of feedback and master data to prepare it for use by the servo loop. It can also be used to execute certain simple calculations at the servo update frequency.

Each I-variable occupies a fixed register in the Turbo PMAC's memory map. The register addresses are important, because the results of the ECT are accessed by address.

The ECT has two halves: setup and results. The "setup" half resides in Turbo PMAC's Y-memory, and can be accessed through these 192 I-variables. The "result" half resides in Turbo PMAC's X-memory. Each of the 192 I-variables has a matching result X-register at the same numerical address. If the entry consists of more than one line, the last line has the final result; any previous lines contain intermediate results.

The entries in the ECT are usually set up through the table's configuration menu in the PMAC Executive program.

The following table shows the address of each ECT I-variable.

I-Variable	Address	I-Variable	Address	I-Variable	Address	I-Variable	Address
I8000	$003501	I8048	$003531	I8096	$003561	I8144	$003591
I8001	$003502	I8049	$003532	I8097	$003562	I8145	$003592
I8002	$003503	I8050	$003533	I8098	$003563	I8146	$003593
I8003	$003504	I8051	$003534	I8099	$003564	I8147	$003594
I8004	$003505	I8052	$003535	I8100	$003565	I8148	$003595
I8005	$003506	I8053	$003536	I8101	$003566	I8149	$003596
I8006	$003507	I8054	$003537	I8102	$003567	I8150	$003597
I8007	$003508	I8055	$003538	I8103	$003568	I8151	$003598
I8008	$003509	I8056	$003539	I8104	$003569	I8152	$003599
I8009	$00350A	I8057	$00353A	I8105	$00356A	I8153	$00359A
I8010	$00350B	I8058	$00353B	I8106	$00356B	I8154	$00359B
I8011	$00350C	I8059	$00353C	I8107	$00356C	I8155	$00359C
I8012	$00350D	I8060	$00353D	I8108	$00356D	I8156	$00359D
I8013	$00350E	I8061	$00353E	I8109	$00356E	I8157	$00359E
I8014	$00350F	I8062	$00353F	I8110	$00356F	I8158	$00359F
I8015	$003510	I8063	$003540	I8111	$003570	I8159	$0035A0
I8016	$003511	I8064	$003541	I8112	$003571	I8160	$0035A1
I8017	$003512	I8065	$003542	I8113	$003572	I8161	$0035A2
I8018	$003513	I8066	$003543	I8114	$003573	I8162	$0035A3
I8019	$003514	I8067	$003544	I8115	$003574	I8163	$0035A4
I8020	$003515	I8068	$003545	I8116	$003575	I8164	$0035A5
I8021	$003516	I8069	$003546	I8117	$003576	I8165	$0035A6
I8022	$003517	I8070	$003547	I8118	$003577	I8166	$0035A7
I8023	$003518	I8071	$003548	I8119	$003578	I8167	$0035A8
I8024	$003519	I8072	$003549	I8120	$003579	I8168	$0035A9
I8025	$00351A	I8073	$00354A	I8121	$00357A	I8169	$0035AA
I8026	$00351B	I8074	$00354B	I8122	$00357B	I8170	$0035AB
I8027	$00351C	I8075	$00354C	I8123	$00357C	I8171	$0035AC
I8028	$00351D	I8076	$00354D	I8124	$00357D	I8172	$0035AD
I8029	$00351E	I8077	$00354E	I8125	$00357E	I8173	$0035AE
I8030	$00351F	I8078	$00354F	I8126	$00357F	I8174	$0035AF
I8031	$003520	I8079	$003550	I8127	$003580	I8175	$0035B0
I8032	$003521	I8080	$003551	I8128	$003581	I8176	$0035B1
I8033	$003522	I8081	$003552	I8129	$003582	I8177	$0035B2
I8034	$003523	I8082	$003553	I8130	$003583	I8178	$0035B3
I8035	$003524	I8083	$003554	I8131	$003584	I8179	$0035B4
I8036	$003525	I8084	$003555	I8132	$003585	I8180	$0035B5
I8037	$003526	I8085	$003556	I8133	$003586	I8181	$0035B6
I8038	$003527	I8086	$003557	I8134	$003587	I8182	$0035B7
I8039	$003528	I8087	$003558	I8135	$003588	I8183	$0035B8
I8040	$003529	I8088	$003559	I8136	$003589	I8184	$0035B9
I8041	$00352A	I8089	$00355A	I8137	$00358A	I8185	$0035BA
I8042	$00352B	I8090	$00355B	I8138	$00358B	I8186	$0035BB
I8043	$00352C	I8091	$00355C	I8139	$00358C	I8187	$0035BC
I8044	$00352D	I8092	$00355D	I8140	$00358D	I8188	$0035BD
I8045	$00352E	I8093	$00355E	I8141	$00358E	I8189	$0035BE
I8046	$00352F	I8094	$00355F	I8142	$00358F	I8190	$0035BF
I8047	$003530	I8095	$003560	I8143	$003590	I8191	$0035C0

Table Structure: The ECT consists of a series of "entries", with each entry creating one processed ("converted") feedback value. An entry in the ECT can have 1, 2, or 3 lines, with each line containing a 24-bit setup word (I-variable) in Y-memory, and a 24-bit result register in X-memory. Therefore, each entry contains 1, 2, or 3 of these 24-bit I-variables. The final result is always in the X-memory register matching the *last* I-variable in the entry.

The variables that commonly contain the address of the last line of the entry are Ixx03 Motor xx Position-Loop Feedback Address, Ixx04 Motor xx Velocity-Loop Feedback Address, Ixx05 Motor xx Master Position Address and Isx93 Coordinate System 'x' Time-Base Address.

The addresses for these variables can be specified directly using the above table (e.g. **I103=$3501**) or by reference to the table I-variable with the special on-line command **I{constant}=@I{constant}**, which sets the first I-variable to the address of the second (e.g. **I103=@I8000**).

Entry First Line: The first line's setup register (I-variable) in each entry consists of a source address in the low 19 bits (bits 0 – 18), which contains the Turbo PMAC address of the raw data to be processed, a possible mode switch in bit 19, and a "method" value in the high 4 bits (first hex digit), which specifies how this data is to be processed. If the first line (I-variable) in the entry is $000000, this signifies the end of the active table, regardless of what subsequent entries in the table (higher numbered I-variables) contain.

Entry Additional Lines: Depending on the method, 1 or 2 additional lines (I-variables) may be required in the entry to provide further instructions on processing.

The following table summarizes the content of entries in the Encoder Conversion Table:

Method Digit	# of lines	Process Defined	Mode Switch	1st Additional Line	2nd Additional Line
$0	1	1/T Extension of Incremental Encoder	None	-	-
$1	1	ACC-28 style A/D converter (high 16 bits, no rollover)	0 = signed data 1 = unsigned data	-	-
$2	2	Parallel Y-word data, no filtering	0 = normal shift 1 = unshifted	Width/Offset Word	-
$3	3	Parallel Y-word data, with filtering	0 = normal shift 1 = unshifted	Width/Offset Word	Max Change per Cycle
$4	2	"Time Base" scaled digital differentiation	None	Time Base Scale Factor	-
$5	2	Integrated ACC-28 style A/D converter	0 = signed data 1 = unsigned data	Input Bias	-
$6	2	Parallel Y/X-word data, no filtering	0 = normal shift 1 = unshifted	Width/Offset Word	-
$7	3	Parallel Y/X-word data, with filtering	0 = normal shift 1 = unshifted	Width/Offset Word	Max Change per Cycle
$8	1	Parallel Extension of Incremental Encoder	0 = PMAC(1) IC 1 = PMAC2 IC	-	-
$9	2	Triggered Time Base, frozen	0 = PMAC(1) IC 1 = PMAC2 IC	Time Base Scale Factor	-
$A	2	Triggered Time Base, running	0 = PMAC(1) IC 1 = PMAC2 IC	Time Base Scale Factor	-
$B	2	Triggered Time Base, armed	0 = PMAC(1) IC 1 = PMAC2 IC	Time Base Scale Factor	-
$C	1	Incremental Encoder, no extension	None	-	-
$D	3	Exponential filter of parallel data	None	Max Change per Cycle	Filter Gain (Inverse Time Constant)
$E	1	Sum or difference of entries	None	-	-
$F	-	(Extended entry – type determined by 1st digit of 2nd line)	-	-	-

$F/$0	3	High-Resolution Interpolator	0 = PMAC(1) IC 1 = PMAC2 IC	$0 Method digit & Address of 1st A/D converter	A/D Bias Term
$F/$2	2	Byte-wide parallel Y-word data, no filtering	0 = normal shift 1 = unshifted	$2 and Width/Offset Word	-
$F/$3	3	Byte-wide parallel Y-word data, with filtering	0 = normal shift 1 = unshifted	$3 and Width/Offset Word	Max Change per Cycle

Incremental Encoder Entries ($0, $8, $C): These three conversion table methods utilize the incremental encoder registers in the Servo ICs. Each method provides a processed result with the units of (1/32) count – the low 5 bits of the result are fractional data.

Software 1/T Extension: With the $0 method, the fractional data is computed by dividing the "Time Since Last Count" register by the "Time Between Last 2 Counts" register. This technique is known as "1/T extension", and is the default and most commonly used method. It can be used with a digital incremental encoder connected directly to the Turbo PMAC, through either PMAC(1)-style or PMAC2-style Servo ICs.

Note:

1/T extension with 8 bits of fractional resolution (units of 1/256 count) can be gotten using the intermediate result value of the "triggered time-base" conversion in "running mode". This intermediate result is in the first line of the entry. If used for position data, one true count of the position is considered by Turbo PMAC software to be 8 counts.

Parallel Extension: With the $8 method, the fractional data is computed by reading the 5 inputs at bits 19-23 either of the specified address (USERn, Wn, Vn, Un, and Tn flag inputs, respectively) if the mode switch bit of the setup I-variable is set to 1 for PMAC2-style Servo ICs, or of the specified address plus 4 (CHC[n+1], HMFL[n+1], +LIM[n+1], -LIM[n+1], FAULT[n+1]) if the mode switch bit of the setup I-variable is set to 0 for PMAC(1)-style Servo ICs. This technique is known as "parallel extension", and can be used with an analog incremental encoder processed through an ACC-8D Opt 8 Analog Encoder Interpolator board or its equivalent.

No Extension: In the $C method with the mode switch bit set to 0, the fractional data is always set to zero, which means there is no extension of the incremental encoder count. This setting is used mainly to verify the effect of one of the two extension methods. It is also recommended when feeding back the pulse-and-direction outputs for stepper drives.

Hardware 1/T Extension: In the $C method with the mode switch bit set to 1, the fractional data is read from a special timer-based register in the Servo IC that has already computed the fractional-count data in hardware. This feature is only supported in the D-revision or newer (first shipments around the beginning of 2002) of the PMAC2-style "DSPGATE1" Servo ICs. The alternate timer registers for the encoder channel must be selected by setting I7mn9 for the channel to 1.

Using this mode, permits timer-based sub-count capture and compare features to be used on this encoder channel.

With any of these three conversion methods, the source address in the low 19 bits (bits 0 - 18) is that of the starting register of the machine interface channel.

The first table below shows the entries for PMAC(1)-style encoder channels. The 'm' in the first hex digit (bits 20 - 23) represents the conversion method ($0, $8, or $C). For the PMAC(1)-style channels, the bit 19 mode switch is always 0, so the second hex digit is always '7' for the hardware registers.

Entries for PMAC(1)-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$m78000	$m78004	$m78008	$m7800C	1st IC on board PMAC
1	$m78100	$m78104	$m78108	$m7810C	2nd IC on board PMAC
2	$m78200	$m78204	$m78208	$m7820C	1st IC on 1st ACC-24P/V
3	$m78300	$m78304	$m78308	$m7830C	2nd IC on 1st ACC-24P/V
4	$m79200	$m79204	$m79208	$m7920C	1st IC on 2nd ACC-24P/V
5	$m79300	$m79304	$m79308	$m7930C	2nd IC on 2nd ACC-24P/V
6	$m7A200	$m7A204	$m7A208	$m7A20C	1st IC on 3rd ACC-24P/V
7	$m7A300	$m7A304	$m7A308	$m7A30C	2nd IC on 3rd ACC-24P/V
8	$m7B200	$m7B204	$m7B208	$m7B20C	1st IC on 4th ACC-24P/V
9	$m7B300	$m7B304	$m7B308	$m7B30C	2nd IC on 4th ACC-24P/V

The next table shows the entry values for PMAC2-style encoder channels. The 'm' in the first hex digit (bits 20 – 23) represents the conversion method ($0, $8, or $C). The 'n' in the second hex digit (bits 16 – 19) contains the bit 19 mode switch and the start of the source address. For methods $0 (software 1/T extension) and $C (no extension), the bit 19 mode switch is 0, making the second hex digit '7'. For method $8 (parallel extension) or for method $C for hardware 1/T extension, the bit 19 mode switch is 1, changing the second hex digit from '7' to 'F'.

Entries for PMAC2-Style Servo ICs

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$mn8000	$mn8008	$mn8010	$mn8018	1st IC on board PMAC2, 3U stack
1	$mn8100	$mn8108	$mn8010	$mn8018	2nd IC on board PMAC2, 3U stack
2	$mn8200	$mn8208	$mn8210	$mn8218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$mn8300	$mn8308	$mn8310	$mn8318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$mn9200	$mn9208	$mn9210	$mn9218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$mn9300	$mn9308	$mn9310	$mn9318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$mnA200	$mnA208	$mnA210	$mnA218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$mnA300	$mnA308	$mnA310	$mnA318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$mnB200	$mnB208	$mnB210	$mnB218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$mnB300	$mnB308	$mnB310	$mnB318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

Entries for PMAC2 MACRO IC 0

Handwheel Channel #	PMAC2
Channel 1	$mn8410
Channel 2	$mn8418

These are single-line entries in the table, so the next line (I-Variable) is the start of the next entry.

ACC-28 Style A/D Entries ($1, $5): The "A/D" feedback entries read from the high 16 bits of the specified address and shift the data right three bits so that the least significant bit of the processed result in bit 5. Unlike the "parallel feedback" methods, this method will not "roll over" and extend the result.

The $1 method processes the information directly, essentially a copying with shift. The $5 integrates the input value as it copies and shifts it. That is, it reads the input value, shifts it right three bits, adds the bias term in the second line, and adds this value to the previous processed result.

If the bit 19 mode switch of the entry is '0', the 16-bit source value is treated as a signed quantity; this should be used for the ACC-28A. If bit 19 of the entry is '1', the 16-bit value is treated as an unsigned quantity; this should be used for the ACC-28B or the ACC-28E.

The first two tables show the entry values that should be used for ACC-28 boards interfaced to PMAC(1)-style Servo ICs. The 'm' in the first hex digit refers to the method digit -- $1 for un-integrated; $5 for integrated. Note that setting the bit 19 mode switch bit to 1 for the ACC-28B changes the second hex digit from '7' to 'F'.

Entries for PMAC(1)-Style Servo ICs using ACC-28A

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$m78006	$m78007	$m7800E	$m7800F	1st IC on board PMAC
1	$m78106	$m78107	$m7810E	$m7810F	2nd IC on board PMAC
2	$m78206	$m78207	$m7820E	$m7820F	1st IC on 1st ACC-24P/V
3	$m78306	$m78307	$m7830E	$m7830F	2nd IC on 1st ACC-24P/V
4	$m79206	$m79207	$m7920E	$m7920F	1st IC on 2nd ACC-24P/V
5	$m79306	$m79307	$m7930E	$m7930F	2nd IC on 2nd ACC-24P/V
6	$m7A206	$m7A207	$m7A20E	$m7A20F	1st IC on 3rd ACC-24P/V
7	$m7A306	$m7A307	$m7A30E	$m7A30F	2nd IC on 3rd ACC-24P/V
8	$m7B206	$m7B207	$m7B20E	$m7B20F	1st IC on 4th ACC-24P/V
9	$m7B306	$m7B307	$m7B30E	$m7B30F	2nd IC on 4th ACC-24P/V

Entries for PMAC(1)-Style Servo ICs using ACC-28B

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$mF8006	$mF8007	$mF800E	$mF800F	1st IC on board PMAC
1	$mF8106	$mF8107	$mF810E	$mF810F	2nd IC on board PMAC
2	$mF8206	$mF8207	$mF820E	$mF820F	1st IC on 1st ACC-24P/V
3	$mF8306	$mF8307	$mF830E	$mF830F	2nd IC on 1st ACC-24P/V
4	$mF9206	$mF9207	$mF920E	$mF920F	1st IC on 2nd ACC-24P/V
5	$mF9306	$mF9307	$mF930E	$mF930F	2nd IC on 2nd ACC-24P/V
6	$mFA206	$mFA207	$mFA20E	$mFA20F	1st IC on 3rd ACC-24P/V
7	$mFA306	$mFA307	$mFA30E	$mFA30F	2nd IC on 3rd ACC-24P/V
8	$mFB206	$mFB207	$mFB20E	$mFB20F	1st IC on 4th ACC-24P/V
9	$mFB306	$mFB307	$mFB30E	$mFB30F	2nd IC on 4th ACC-24P/V

The next table shows the entry values that should be used for ACC-28B boards interfaced to PMAC2-style Servo ICs (ACC-28A is not compatible with these ICs). The 'm' in the first hex digit refers to the method digit -- $1 for un-integrated; $5 for integrated. Note that setting the bit 19 mode switch bit to 1 for the ACC-28B changes the second hex digit from '7' to 'F'.

Entries for PMAC2-Style ADC Registers Using ACC-28B

Register	PMAC2	1st ACC-24P/V2	2nd ACC-24P/V2	3rd ACC-24P/V2	4th ACC-24P/V2
ADC 1A	$mF8005	$mF8205	$mF9205	$mFA205	$mFB205
ADC 1B	$mF8006	$mF8206	$mF9206	$mFA206	$mFB206
ADC 2A	$mF800D	$mF820D	$mF920D	$mFA20D	$mFB20D
ADC 2B	$mF800E	$mF820E	$mF920E	$mFA20E	$mFB20E
ADC 3A	$mF8015	$mF8215	$mF9215	$mFA215	$mFB215
ADC 3B	$mF8016	$mF8216	$mF9216	$mFA216	$mFB216
ADC 4A	$mF801D	$mF821D	$mF921D	$mFA21D	$mFB21D
ADC 4B	$mF801E	$mF821E	$mF921E	$mFA21E	$mFB21E
ADC 5A	$mF8105	$mF8305	$mF9305	$mFA305	$mFB305
ADC 5B	$mF8106	$mF8306	$mF9306	$mFA306	$mFB306
ADC 6A	$mF810D	$mF830D	$mF930D	$mFA30D	$mFB30D
ADC 6B	$mF810E	$mF830E	$mF930E	$mFA30E	$mFB30E
ADC 7A	$mF8115	$mF8315	$mF9315	$mFA315	$mFB315
ADC 7B	$mF8116	$mF8316	$mF9316	$mFA316	$mFB316
ADC 8A	$mF811D	$mF831D	$mF931D	$mFA31D	$mFB31D
ADC 8B	$mF811E	$mF831E	$mF931E	$mFA31E	$mFB31E

The next table shows the entry values that should be used for ACC-28E boards in a UMAC Turbo system. The 'm' in the first hex digit refers to the method digit -- $1 for un-integrated; $5 for integrated. Note that setting the bit 19 mode switch bit to 1 for the ACC-28E changes the second hex digit from '7' to 'F'.

Entries for UMAC ACC-28E ADCs

I/O IC #	SW1-1	SW1-2	SW1-3	SW1-4	Chan. 1	Chan. 2	Chan. 3	Chan. 4
0	ON	ON	ON	ON	$mF8C00	$mF8C01	$mF8C02	$mF8C03
1	OFF	ON	ON	ON	$mF8D00	$mF8D01	$mF8D02	$mF8D03
2	ON	OFF	ON	ON	$mF8E00	$mF8E01	$mF8E02	$mF8E03
3	OFF	OFF	ON	ON	$mF8F00	$mF8F01	$mF8F02	$mF8F03
4	ON	ON	OFF	ON	$mF9C00	$mF9C01	$mF9C02	$mF9C03
5	OFF	ON	OFF	ON	$mF9D00	$mF9D01	$mF9D02	$mF9D03
6	ON	OFF	OFF	ON	$mF9E00	$mF9E01	$mF9E02	$mF9E03
7	OFF	OFF	OFF	ON	$mF9F00	$mF9F01	$mF9F02	$mF9F03
8	ON	ON	ON	OFF	$mFAC00	$mFAC01	$mFAC02	$mFAC03
3	OFF	ON	ON	OFF	$mFAD00	$mFAD01	$mFAD02	$mFAD03
4	ON	OFF	ON	OFF	$mFAE00	$mFAE01	$mFAE02	$mFAE03
5	OFF	OFF	ON	OFF	$mFAF00	$mFAF01	$mFAF02	$mFAF03
6	ON	ON	OFF	OFF	$mFBC00	$mFBC01	$mFBC02	$mFBC03
7	OFF	ON	OFF	OFF	$mFBD00	$mFBD01	$mFBD02	$mFBD03
8	ON	OFF	OFF	OFF	$mFBE00	$mFBE01	$mFBE02	$mFBE03
9	OFF	OFF	OFF	OFF	$mFBE00	$mFBE01	$mFBE00	$mFBE03

Integration Bias: The $5 integrated format requires a second line to specify the bias of the A/D converter. This bias term is a signed quantity (even for an unsigned A/D converter), with units of 1/256 of the LSB of the 16-bit A/D converter. This value is *subtracted* from the reading of the ADC before the integration occurs.

For example, if there were an offset in a 16-bit ADC of +5 LSBs, this term would be set to 1280. If no bias is desired, a zero value should be entered here. If the conversion is unsigned, the result after the bias is not permitted to be less than zero. This term permits reasonable integration, even with an analog offset.

Parallel Feedback Entries ($2, $3, $6, $7): The parallel feedback entries read a word from the address specified in the low 19 bits (bits 0 to 18) of the first line. The four methods in this class are:

- $2: Y-word parallel, no filtering (2-line entry)
- $3: Y-word parallel, with filtering (3-line entry)
- $6: Y/X-word parallel, no filtering (2-line entry)
- $7: Y/X-word parallel, with filtering (3-line entry)

The Bit-19 mode switch in the first line controls whether the least significant bit (LSB) of the source register is placed in Bit 5 of the result register ("normal shift"), providing the standard 5 bits of (non-existent) fraction, or the LSB is placed in Bit 0 of the result register ("unshifted"), creating no fractional bits.

Normally, the Bit-19 mode switch is set to 0 to place the source LSB in Bit 5 of the result register. Bit 19 is set to 1 to place to source LSB in Bit 0 of the result register for one of three reasons:

- The data already comes with 5 bits of fraction, as from a Compact MACRO Station.
- The normal shift limits the maximum velocity too much ($V_{max}<2^{18}$ LSBs per servo cycle)
- The normal shift limits the position range too much (Range$<\pm 2^{47}$/Ix08/32 LSBs)

Unless this is done because the data already contains fractional information, the "unshifted" conversion will mean that the motor position loop will consider 1 LSB of the source to be 1/32 of a count, instead of 1 count.

Width/Offset Word: The second setup line (I-variable) of a parallel read entry contains the width of the data to be read, and the location of the LSB. This 24-bit value, usually represented as 6 hexadecimal digits, is split evenly into two halves, each of 3 hex digits. The first half represents

the width of the parallel data in bits, and can range from $001 (1 bit wide – not of much practical use) to $018 (24 bits wide).

The second half of the line contains the bit location of the LSB of the data in the source word, and can range from $000 (Bit 0 of the Y-word at the source address is the LSB), through $017 (Bit 23 of the Y-word at the source address), and $018 (Bit 24, which is Bit 0 of the next word, is the LSB) to $02F (Bit 47, which is Bit 23 of the next word, is the LSB).

If the LSB bit location exceeds 23, or the sum of the LSB bit location and the bit width exceeds 24, the source data extends into the "next word". If the method character is $2 or $3, the next word is the Y-word at the source address + 1. If the method character is $6 or $7, the next word is the X-word at the source address.

For example, to use 20 bits starting at bit 0 (bits 0 – 19) of the Y-word of the source address, this word would be set to $014000. To use all 24 bits of the X-word of the source address, this word would be set to $018018. To use 24 bits starting at bit 12 of the specified address (with the highest 12 bits coming from the X-word or the next higher Y-address, this word would be set to $01800C.

Maximum Change Word: If the method character for a parallel read is $3 or $7, specifying filtered parallel read, there is a third setup line (I-variable) for the entry. This third line contains the maximum change in the source data in a single cycle that will be reflected in the processed result, expressed in LSBs per servo cycle. The filtering that this creates provides an important protection against noise and misreading of data. This number is effectively a velocity value, and should be set slightly greater than the maximum true velocity ever expected.

ACC-14: The Accessory 14 family of boards is often used to bring parallel data feedback to the Turbo PMAC, such as that from parallel absolute encoders, and from interferometers. The following table shows the first line of the entries for ACC-14D/V boards connected to a Turbo PMAC controller over a JEXP expansion port cable:

Entries for ACC-14D/V Registers

Register	First Line Value	Register	First Line Value
1st ACC-14D/V Port A	$m78A00	4th ACC-14D/V Port A	$m78D00
1st ACC-14D/V Port B	$m78A01	4th ACC-14D/V Port B	$m78D01
2nd ACC-14D/V Port A	$m78B00	5th ACC-14D/V Port A	$m78E00
2nd ACC-14D/V Port B	$m78B01	5th ACC-14D/V Port B	$m78E01
3rd ACC-14D/V Port A	$m78C00	6th ACC-14D/V Port A	$m78F00
3rd ACC-14D/V Port B	$m78C01	6th ACC-14D/V Port B	$m78F01

MACRO Position Feedback: When position feedback is received through the MACRO ring, the MACRO input registers are treated as parallel-data feedback. The following table shows the first line of the entries for MACRO position feedback registers.

Entries for Type 1 MACRO Position Feedback Registers

Register	First Line Value	Register	First Line Value
MACRO IC 0 Node 0 Reg. 0	$2F8420	MACRO IC 2 Node 0 Reg. 0	$2FA420
MACRO IC 0 Node 1 Reg. 0	$2F8424	MACRO IC 2 Node 1 Reg. 0	$2FA424
MACRO IC 0 Node 4 Reg. 0	$2F8428	MACRO IC 2 Node 4 Reg. 0	$2FA428
MACRO IC 0 Node 5 Reg. 0	$2F842C	MACRO IC 2 Node 5 Reg. 0	$2FA42C
MACRO IC 0 Node 8 Reg. 0	$2F8430	MACRO IC 2 Node 8 Reg. 0	$2FA430
MACRO IC 0 Node 9 Reg. 0	$2F8434	MACRO IC 2 Node 9 Reg. 0	$2FA434
MACRO IC 0 Node 12 Reg. 0	$2F8438	MACRO IC 2 Node 12 Reg. 0	$2FA438
MACRO IC 0 Node 13 Reg. 0	$2F843C	MACRO IC 2 Node 13 Reg. 0	$2FA43C
MACRO IC 1 Node 0 Reg. 0	$2F9420	MACRO IC 3 Node 0 Reg. 0	$2FB420
MACRO IC 1 Node 1 Reg. 0	$2F9424	MACRO IC 3 Node 1 Reg. 0	$2FB424
MACRO IC 1 Node 4 Reg. 0	$2F9428	MACRO IC 3 Node 4 Reg. 0	$2FB428
MACRO IC 1 Node 5 Reg. 0	$2F942C	MACRO IC 3 Node 5 Reg. 0	$2FB42C
MACRO IC 1 Node 8 Reg. 0	$2F9430	MACRO IC 3 Node 8 Reg. 0	$2FB430
MACRO IC 1 Node 9 Reg. 0	$2F9434	MACRO IC 3 Node 9 Reg. 0	$2FB434
MACRO IC 1 Node 12 Reg. 0	$2F9438	MACRO IC 3 Node 12 Reg. 0	$2FB438
MACRO IC 1 Node 13 Reg. 0	$2F943C	MACRO IC 3 Node 13 Reg. 0	$2FB43C

Note that the bit-19 mode switch has been set to 1 so that the data out of the MACRO node is not shifted. This changes the second hex digit from '7' to 'F'. Type 1 MACRO feedback comes with fractional count information in the low 5 bits, so it does not need to be shifted.

The second line of an entry for MACRO feedback should be $018000 to specify the use of 24 bits ($018) starting at bit 0 ($000).

When performing commutation of motors over the MACRO ring, it is advisable to get servo position feedback data not directly from the MACRO ring registers, as shown above, but from the motor's "previous phase position" register instead. This is where the commutation algorithm has stored the position it read from the ring (with Ixx83) for use in its next cycle.

Using this register prevents the possibility of jitter if the conversion table execution can be pushed too late in the cycle. The following table shows the first line of the conversion table entry for each motor's "previous phase position" register:

Entries for Turbo PMAC Previous Phase Position Registers

Motor #	First Line Value	Motor #	First Line Value	Motor #	First Line Value	Motor #	First Line Value
1	$2800B2	9	$2804B2	17	$2808B2	25	$280CB2
2	$280132	10	$280532	18	$280932	26	$280D32
3	$2801B2	11	$2805B2	19	$2809B2	27	$280DB2
4	$280232	12	$280632	20	$280A32	28	$280E32
5	$2802B2	13	$2806B2	21	$280AB2	29	$280EB2
6	$280332	14	$280732	22	$280B32	30	$280F32
7	$2803B2	15	$2807B2	23	$280BB2	31	$280FB2
8	$280432	16	$280832	24	$280C32	32	$281032

Note

The bit 19 mode switch has been set to 1 so that the data out of the previous phase position register from the MACRO ring is not shifted. This changes the second hex digit from '0' to '8'. Type 1 MACRO feedback comes with fractional count information in the low 5 bits, so it does not need to be shifted.

The second line of an entry for "previous phase position" feedback should be $018000 to specify the use of 24 bits ($018) starting at bit 0 ($000).

MLDT Feedback: PMAC2-style Servo ICs have the ability to interface directly to magnetostrictive linear displacement transducers (MLDTs), outputting the excitation pulse, receiving the echo pulse, and measuring the time between the two. This time is directly proportional to the distance. For this feedback the "time between last two counts" register is used like an absolute encoder. The following table shows the first line of the parallel feedback entry for each channel's timer register:

Entries for PMAC2-Style MLDT Timer Registers

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$378000	$378008	$378010	$378018	1st IC on board PMAC2, 3U stack
1	$378100	$378108	$378010	$378018	2nd IC on board PMAC2, 3U stack
2	$378200	$378208	$378210	$378218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$378300	$378308	$378310	$378318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$379200	$379208	$379210	$379218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$379300	$379308	$379310	$379318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$37A200	$37A208	$37A210	$37A218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$37A300	$37A308	$37A310	$37A318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$37B200	$37B208	$37B210	$37B218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$37B300	$37B308	$37B310	$37B318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

The second line in an MLDT entry should be $013000 to specify the use of 19 bits ($013) starting at bit 0 ($000).

The third line in an MLDT entry should contain a number slightly greater than the maximum velocity ever expected, expressed as timer increments per servo cycle. An increment of the 120 MHz timer represents about 0.024mm (0.0009 in) on a typical MLDT device. This value represents the maximum change in position reading that will be passed through the conversion table in a single servo cycle, and it provides an important protection against missing or spurious echo pulses.

Time-Base Entries ($4, $9, $A, $B): A time-base entry performs a scaled digital differentiation of the value in the source register. It is most often used to perform "electronic cam" functions, slaving a motion sequence to the frequency of a master encoder. There are two types of time-base entries: "untriggered" and "triggered". An untriggered time base does not provide a specific starting point in the master source data. A triggered time base starts the differentiation upon receipt of a hardware trigger on the master encoder's channel, referenced to the position captured by that trigger. This can be used to create an absolute synchronization between the master position and the slave trajectory.

Time-base entries are two-line entries. The first setup line (I-variable) contains the method digit and the address of the source-data register. The second setup line (I-variable) contains the "time-base scale factor". The first result line contains the intermediate result value of the source data, saved for the next cycle to be able to compute the differentiation. The second result line contains the final result, which is the differentiated value. Most commonly this result is used as the time-base source for a coordinate system, so Isx93 for the coordinate system points to this second line.

Untriggered Time Base ($4): In an untriggered time-base entry, the first setup line (I-variable) contains a "4" in the method digit (bits 20 – 23) and the address of the source register in bits 0 – 18. The source register is usually the result register of an incremental encoder entry (e.g. 1/T) higher in the table (addresses $3501 to $35C0). Refer to the table above, which lists the addresses of each line in the encoder conversion table. For example, to use the result of the fourth line of the conversion table as a source, this I-variable would be $403504.

The second setup line (I-variable) is the "time-base scale factor" which multiplies the differentiated source value. The final result value equals *2 * Time-Base-Scale-Factor * (New Source Value - Old Source Value)*.

New Source Value and Old Source Value (stored from the previous servo cycle) are typically in units of 1/32 of a count, the usual scaling of a 1/T encoder conversion result.

When this time base entry is used to calculate a frequency-based time base for a coordinate system, the TBSF should be set to 2^{17}/Real-Time Input Frequency (131,072/RTIF), where the Real-Time Input Frequency (RTIF) in counts per millisecond, is the frequency at which motion trajectories using this time base will execute at the programmed speed or in the programmed time. The motion sequence to be slaved to this frequency should be written assuming that the master is always generating this real-time input frequency (so always moving at the "real-time speed"). The true speed of trajectories using this time base will vary proportionately with the actual input frequency.

Example

The application requires the use of Encoder 4 on board a Turbo PMAC2 as an untriggered time-base master for Coordinate System 1. The real-time input frequency is selected as 256 counts/msec. The conversion table starts with 8 single-line entries in I8000 – I8007, with the 4th line (I8003) doing a 1/T conversion of Encoder 4.

```
; Setup on-line commands
I8003=$078018          ; 1/T conversion of Encoder 4
I8008=$403504          ; Unriggered time base from 1/T encoder
I8009=512              ; TBSF=131072/256
I5193=@I8009           ; C.S.1 use I8009 result for time base
```

Triggered Time Base ($9, $A, $B): A "triggered" time-base entry is like a regular "untriggered" time-base entry, except that it is easy to "freeze" the time base, then start it exactly on receipt of a trigger that captures the "starting" master position or time.

In a triggered time-base entry, the first setup line (I-variable) contains a '9' 'A' or 'B' in the method digit (bits 20 – 23), depending on its present state. It contains the address of the source register in bits 0 – 18. The source register for triggered time base must be the starting (X) address for one of the machine interface channels of a Servo IC. The bit 19 mode switch must be set to 0 if a PMAC(1)-style Servo IC (DSPGATE) is addressed; it must be set to 1 if a PMAC2-style Servo or MACRO IC ("DSPGATE1" or DSPGATE2) is addressed. Note that setting bit 19 to 1 changes the second hex digit of the I-variable typically from '7' to 'F'.

The second setup line (I-variable) is the "time-base scale factor" which multiplies the differentiated source value. The final result value (when running) equals *512 * Time-Base-Scale-Factor * (New Source Count - Old Source Count)*. "New Source Count" and "Old Source Count" are the values of the addressed encoder counter, in counts.

When this time-base entry is used to calculate a frequency-based time base for a coordinate system, the TBSF should be set to 2^{14}/Real-Time Input Frequency (16,384/RTIF), where the Real-Time Input Frequency (RTIF) in counts per millisecond, is the frequency at which motion trajectories using this time base will execute at the programmed speed or in the programmed time. (Note that the TBSF is 1/8 of the value for an untriggered time base, because the triggered time base creates an extra 3 bits [8x] of fractional information with its 1/T extension.) The motion sequence to be slaved to this frequency should be written assuming that the master is always generating this real-time input frequency (so always moving at the "real-time speed"). The true speed of trajectories using this time base will vary proportionately with the actual input frequency.

A triggered time-base entry in Turbo PMAC automatically computes the 1/T count extension of the input frequency itself before the differentiation. It computes this to 1/256 of a count. This is compared to the 1/32 of a count that the separate 1/T encoder extension uses.

The extra fractional information can reduce the quantization noise created by the differentiation and provide smoother operation under external time base.

Note: The intermediate result in the first line of a triggered time-base entry contains the undifferentiated 1/T extension of the source encoder position, in units of 1/256 of a count. This value can be used as feedback data or master position data, with more resolution than the standard 1/T extension.

In use, the method digit (comprising bits 20-23 of the first line) is changed as needed by setting of the I-variable. Triggered time base has three states, "frozen", "armed", and "running", all of which must be used to utilize the triggering feature.

First, the method digit is set to $9 (e.g. I8010=$978008) before the calculations of the triggered move are started, to freeze the time base (and therefore the motion) while the move calculations are done. This is typically done in the user's motion program. When this entry is in the frozen state, the table reads the channel's capture position register each servo cycle to ensure the triggering logic is reset for the next capture. The final result of the entry is always 0 when frozen.

Note:

In a Turbo PMAC application with a light computational load, it is possible that the entry will not be in the "frozen" state during a servo interrupt, and the table will not get a chance to reset the trigger logic. Therefore, it is advisable to reset the triggering logic explicitly in the user program with a "dummy" read of the channel's captured position register, which is the X-register with an address 3 greater than the address specified in the entry (e.g. X:$07800B if the entry specifies $078008). The suggested M-variable for the captured position register is Mxx03.

Next, the method digit is set to $B (e.g. I8010=$B78008) after the calculations of the triggered move are finished, to "arm" the time base for the trigger. This is typically done in a PLC program that simply looks to see if the entry is frozen and changes it to the armed state. The final result of the entry is always 0 when armed.

In the armed state, the Table checks every servo cycle for the channel's trigger bit to be set. When the Table sees the trigger (the capture trigger for the machine interface channel as defined by I7mn2 and I7mn3 for Servo IC m Channel n, or by I68n2 and I68n3 for MACRO IC 0 Channel n), it automatically sets the method digit to $A for "running" time base. It uses the position captured by the trigger as the starting position ("time zero") for the running time base. (Those using this method for the reduced quantization noise may simply leave the method digit at $A.)

The following tables show the possible 1st-line entries for triggered time base (running mode):

Triggered Time-Base Entries for PMAC(1)-Style Servo ICs (Running State)

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$A78000	$A78004	$A78008	$A7800C	1st IC on board PMAC
1	$A78100	$A78104	$A78108	$A7810C	2nd IC on board PMAC
2	$A78200	$A78204	$A78208	$A7820C	1st IC on 1st ACC-24P/V
3	$A78300	$A78304	$A78308	$A7830C	2nd IC on 1st ACC-24P/V
4	$A79200	$A79204	$A79208	$A7920C	1st IC on 2nd ACC-24P/V
5	$A79300	$A79304	$A79308	$A7930C	2nd IC on 2nd ACC-24P/V
6	$A7A200	$A7A204	$A7A208	$A7A20C	1st IC on 3rd ACC-24P/V
7	$A7A300	$A7A304	$A7A308	$A7A30C	2nd IC on 3rd ACC-24P/V
8	$A7B200	$A7B204	$A7B208	$A7B20C	1st IC on 4th ACC-24P/V
9	$A7B300	$A7B304	$A7B308	$A7B30C	2nd IC on 4th ACC-24P/V

Triggered Time-Base Entries for PMAC2-Style Servo ICs (Running State)

Servo IC #	Chan. 1	Chan. 2	Chan. 3	Chan. 4	Notes
0	$AF8000	$AF8008	$AF8010	$AF8018	1st IC on board PMAC2, 3U stack
1	$AF8100	$AF8108	$AF8010	$AF8018	2nd IC on board PMAC2, 3U stack
2	$AF8200	$AF8208	$AF8210	$AF8218	1st ACC-24E2x, 1st IC on 1st ACC-24P/V2
3	$AF8300	$AF8308	$AF8310	$AF8318	2nd ACC-24E2x, 2nd IC on 1st ACC-24P/V2
4	$AF9200	$AF9208	$AF9210	$AF9218	3rd ACC-24E2x, 1st IC on 2nd ACC-24P/V2
5	$AF9300	$AF9308	$AF9310	$AF9318	4th ACC-24E2x, 2nd IC on 2nd ACC-24P/V2
6	$AFA200	$AFA208	$AFA210	$AFA218	5th ACC-24E2x, 1st IC on 3rd ACC-24P/V2
7	$AFA300	$AFA308	$AFA310	$AFA318	6th ACC-24E2x, 2nd IC on 3rd ACC-24P/V2
8	$AFB200	$AFB208	$AFB210	$AFB218	7th ACC-24E2x, 1st IC on 4th ACC-24P/V2
9	$AFB300	$AFB308	$AFB310	$AFB318	8th ACC-24E2x, 2nd IC on 4th ACC-24P/V2

Entries for PMAC2 MACRO IC 0

Handwheel Channel #	PMAC2
Channel 1	$AF8410
Channel 2	$AF8418

Example:

The application requires the use of Encoder 4 on board a Turbo PMAC2 as a triggered time base master for coordinate system 1. It is to be triggered by the rising edge of its index channel. The real-time input frequency is selected as 256 counts/msec. The conversion table starts with 8 single-line entries in I8000 – I8007.

```
; Setup on-line command
I8008=$AF8018                    ; Triggered time base from PMAC2 channel 4
I8009=64                         ; TBSF=16384/256
I7042=1                          ; Servo IC 0 Channel 4 trigger on rising index
I5193=@I8009                     ; C.S.1 use I8009 result for time base
M403->X:$07801B,0,24,S           ; Channels' captured position register

; Motion program segment
DWELL 0                          ; Stop any lookahead
I8008=$9F8018                    ; Freeze the time base
P403=M403                        ; Dummy read to ensure capture logic reset
X10                              ; Calculate first move

; PLC program segment
IF (I8008=$9F8018)               ; If frozen
  I8008=$BF8018                  ; Then arm
ENDIF
```

Exponential-Filter Entries ($D): The $D entry is used to create an exponential filter on a word of input data. This is particularly useful for smoothing master position values in position following (electronic gearing) or external time-base (electronic cam) applications, especially when the slave is "geared up" from the master; i.e. the slave moves more than one count for each count of the master, where it can significantly smooth the motion of the following axis. Exponential filters are seldom used on feedback position values, because the delay introduced by the filter has a destabilizing effect on the servo loop.

The equation of the exponential filter executed every servo cycle *n* is:

$$Out(n) = Out(n-1) + (K/2^{23})*[In(n)-Out(n-1)]$$

If [Out(n) - Out(n-1)] > Max_change, Out(n) = Out(n-1) + Max_change

If [Out(n) - Out(n-1)] < -Max_change, Out(n) = Out(n-1) - Max_change

In, *Out*, and *K* are all signed 24-bit numbers (range -8,388,608 to 8,388,607). The difference *[In(n)-Out(n-1)]* is truncated to 24 bits to handle rollover properly.

The time constant of the filter, in servo cycles, is $(2^{23}/K)-1$. The lower the value of K, the longer the time constant.

No shifting action is performed. Any operations such as 1/T interpolation should have been done on the data already, so the source register for this filter is typically the result register of the previous operation.

Method/Address Word: The first setup line (I-variable) of an exponential filter entry contains a 'D' in the first hex digit (bits 20 – 23) and the address of the source X-register in bits 0 – 18. Bit 19 is not used. If it is desired to execute an exponential filter on the contents of a Y-register, the contents of the Y-register must first be copied to an X-register in the conversion table with a "parallel" entry ($2) higher in the table. The source addresses for exponential filter entries are almost always from the conversion table itself (X:$3501 – X:$35CF). For example, to perform an exponential filter on the result of the fourth line of the table, the first setup line of the filter entry would be $D03504.

Filter Gain Word: The second setup line (I-variable) of an exponential filter entry contains the filter gain value *K*, which sets a filter time constant T_f of $(2^{23}/K)-1$ servo cycles. Therefore, the gain value *K* can be set as $2^{23}/(T_f+1)$. For example, to set a filter time constant of 7 servo cycles, the filter gain word would be 8,388,608/(7+1) = 1,048,576.

Maximum Change Word: The third setup line (I-variable) of an exponential filter entry contains the value "max change" that limits how much the entry can change in one servo cycle. The units of this entry are whatever the units of the input register are, typically 1/32 of a count. For example, to limit the change in one servo cycle to 64 counts with an input register in units of 1/32 count, this third line would be 64*32 = 2048.

Result Word: The output value of the exponential filter is placed in the X register of the third line of the conversion table entry. An operation that uses this value should address this third register; for example Ixx05 for position following, or the source address for a time-base conversion-table entry (to keep position lock in time base, this filter must be executed *before* the time-base differentiation, not afterward).

Addition/Subtraction of Entries ($E): The $E entry is used to add the results of two other entries in the Table, possibly after negating one or both of them (which can effectively create subtraction), with the option of integrating the sum. It is a single-line entry.

Control Digit: The second hex digit of the I-variable consists of four independent control bits (bits 19-16) and determines whether the result is integrated or not, whether a second source entry is used or not, and whether each of the source entries is negated before addition or not.

If the bit 19 mode switch bit is 0, which makes the second hex digit 0, the values in the two specified entries are simply added. If the mode switch bit 19 is 1, the sum of the two entries.

If bit 18 is set to 1, the second entry to be added (as specified by bits 8-15) is not used. This permits easy negation (change in sign) of a single entry. If bit 18 is set to 0, the second entry is used.

If bit 17 is set to 1, the second entry to be added (as specified by bits 8-15) is negated before the addition, which means that it is effectively subtracted from the first entry. If bit 17 is not set to 1, the second entry to be added is not negated.

If bit 16 is set to 1, the first entry to be added (as specified by bits 0-7) is negated before the addition, which means that it is effectively subtracted. If bit 16 is not set to 1, the first entry to be added is not negated.

Second Source Offset: Bits 8-15, which form the third and fourth hex digits of the entry, specify the offset from the beginning of the table to the second entry to be used used, as an unsigned 8-bit quantity. The value in these digits should equal the number of the I-variable matching the second entry minus 8000.

First Source Offset: Bits 0-7, which form the fifth and sixth hex digits of the entry, specify the address offset from the beginning of the table to the first entry to be used, as an unsigned 8-bit quantity. The value in these digits should equal the number of the I-variable matching the first entry minus 8000.

Examples:

To add the results of the first two lines in the table, from I8000 and I8001, the I-variable would be $E00100. The 'E' specifies addition, the '0' specifies no integration, using the second source, and no negation of either source. The '01' specifies the second line of the table (matching I8001) as the second source, and the final '00' specifies the first line of the table (matching I8000) as the first source.

To subtract the result of the second line (from I8001) of the table from that of the first line (from I8000), the I-variable would be $E20100. The 'E' specifies addition, the '2' (0010 binary) specifies no integration, using the second source, negating the second source, but not the first source. The '01' specifies the second line of the table (matching I8001) as the second source, and the final '00' specifies the first line of the table (matching I8000) as the first source.

To invert the 20th line of the table (from I8019), the I-variable would be $E50013. The 'E' specifies "addition", the '5' (0101 binary) specifies no integration, not using the second source, and negating the first source. The '00' is not important, because the second source is not used. The '13' (19 decimal) specifies the result matching I8019 as the first source.

Extended Entries ($F): Encoder conversion table entries in which the first hex digit of the first line is $F are "extended entries". In these entries, the actual method is dependent on the first digit of the second line. Extended entries are a minimum of 2 lines.

High-Resolution Interpolator Entries ($F/$0): An ECT entry in which the first hex digit of the first line is $F and the first hex digit of the second line is $0 processes the result of a high-resolution interpolator for analog "sine-wave" encoders, such as the ACC-51. This entry, when used with a high-resolution interpolator, produces a value with 4096 states per line. The entry must read both an encoder channel for the whole number of lines of the encoder, and a pair of A/D converters to determine the location within the line, mathematically combining the values to produce a single position value.

Encoder Channel Address: The first line of the three-line entry contains $F in the first hex digit and the base address of the encoder channel to be read in the low 19 bits (bits 0 to 18). If the bit-19 mode switch of the line is set to 0, Turbo PMAC expects a PMAC(1)-style Servo IC on the interpolator, as in the ACC-51P. If the bit-19 mode switch bit is set to1, Turbo PMAC expects a PMAC2-style Servo IC on the interpolator, as in the ACC-51E.

The following table shows the possible entries when PMAC(1)-style Servo ICs are used, as in the ACC-51P.

High-Res Interpolator Entry First Lines for PMAC(1)-Style Servo ICs

Servo IC #	Channel 1	Channel 2	Channel 3	Channel 4
2	$F78200	$F78204	$F78208	$F7820C
3	$F78300	$F78304	$F78308	$F7830C
4	$F79200	$F79204	$F79208	$F7920C
5	$F79300	$F79304	$F79308	$F7930C
6	$F7A200	$F7A204	$F7A208	$F7A20C
7	$F7A300	$F7A304	$F7A308	$F7A30C
8	$F7B200	$F7B204	$F7B208	$F7B20C
9	$F7B300	$F7B304	$F7B308	$F7B30C

The following table shows the possible entries when PMAC2-style Servo ICs are used, as in the ACC-51E:

High-Res Interpolator Entry First Lines for PMAC2-Style Servo ICs

Servo IC #	Channel 1	Channel 2	Channel 3	Channel 4
2	$FF8200	$FF8208	$FF8210	$FF8218
3	$FF8300	$FF8308	$FF8310	$FF8318
4	$FF9200	$FF9208	$FF9210	$FF9218
5	$FF9300	$FF9308	$FF9310	$FF9318
6	$FFA200	$FFA208	$FFA210	$FFA218
7	$FFA300	$FFA308	$FFA310	$FFA318
8	$FFB200	$FFB208	$FFB210	$FFB218
9	$FFB300	$FFB308	$FFB310	$FFB318

Note

By setting the bit-19 mode switch to 1, the second hex digit changes from "7" to "F".

A/D Converter Address: The second line of the entry contains $0 in the first hex digit and the base address of the first of two A/D converters to be read in the low 19 bits (bits 0 to 18). The second A/D converter will be read at the next higher address. The following table shows the possible entries when the ACC-51P, with PMAC(1) style Servo ICs, is used:

High-Res Interpolator Entry Second Lines for PMAC(1)-Style Servo ICs

Servo IC #	Channel 1	Channel 2	Channel 3	Channel 4
2	$078202	$078206	$07820A	$07820E
3	$078302	$078306	$07830A	$07830E
4	$079202	$079206	$07920A	$07920E
5	$079302	$079306	$07930A	$07930E
6	$07A202	$07A206	$07A20A	$07A20E
7	$07A302	$07A306	$07A30A	$07A30E
8	$07B202	$07B206	$07B20A	$07B20E
9	$07B302	$07B306	$07B30A	$07B30E

The following table shows the possible entries when PMAC2-style Servo ICs are used, as in the ACC-51E:

High-Res Interpolator Entry First Lines for PMAC2-Style Servo ICs

Servo IC #	Channel 1	Channel 2	Channel 3	Channel 4
2	$078205	$07820D	$078215	$07821D
3	$078305	$07830D	$078315	$07831D
4	$079205	$07920D	$079215	$07921D
5	$079305	$07930D	$079315	$07931D
6	$07A205	$07A20D	$07A215	$07A21D
7	$07A305	$07A30D	$07A315	$07A31D
8	$07B205	$07B20D	$07B215	$07B21D
9	$07B305	$07B30D	$07B315	$07B31D

A/D Bias Term: The third line of the entry contains the bias in the A/D converter values. This line should contain the value that the A/D converters report when they should ideally report zero. Turbo PMAC subtracts this value from both A/D readings before calculating the arctangent. Many users will leave this value at 0, but it is particularly useful to remove the offsets of single-ended analog encoder signals.

This line is scaled so that the maximum A/D converter reading provides the full value of the 24-bit register (+/-2^{23}, or +/-8,388,608). It is generally set by reading the A/D converter values directly as 24-bit values, computing the average value over a cycle or cycles, and entering this value here.

Conversion Result: The result of the conversion is placed in the X-register of the third line of the entry. Careful attention must be paid to the scaling of this 24-bit result. The least significant bit (Bit 0) of the result represents 1/4096 of a line of the sine/cosine encoder.

When Turbo PMAC software reads this data for servo use with Ixx03, Ixx04, Ixx05, or Isx93, it expects to find data in units of 1/32 of a "count". Therefore, PMAC software regards this format as producing 128 "counts" per line. (The fact that the hardware counter used produces 4 counts per line is not relevant to the actual use of this format; this fact would only be used when reading the actual hardware counter for commutation or debugging purposes.)

Example: This format is used to interpolate a linear scale with a 40-micron pitch (40μm/line), producing a resolution of about 10 nanometers (40,000/4096), used as position feedback for a motor. PMAC considers a "count" to be 1/128 of a line, yielding a count length of 40/128 = 0.3125 μm. To set user units of millimeters for the axis, the axis scale factor would be:

$$AxisScaleFactor = \frac{1mm}{UserUnit} * \frac{1000\mu m}{mm} * \frac{count}{0.3125\mu m} = 3200\frac{counts}{UserUnit}$$

Byte-Wide Parallel Feedback Entries ($F/$2, $F/$3): An ECT entry in which the first hex digit of the first line is $F and the first hex digit of the second line is $2 or $3 processes the result of a parallel data feedback source whose data is in byte-wide pieces in consecutive Y-words. This is used to process feedback from 3U-format parallel-data I/O boards: the ACC-3E in stack form, and the ACC-14E in pack (UMAC) form.

Address Word: The first setup line (I-variable) of the entry contains $F in the first hex digit (bits 20-23). The bit-19 mode-switch bit in the first line controls whether the least significant bit (LSB) of the source register is placed in bit 5 of the result register (normal shift), providing the standard 5 bits of (non-existent) fraction, or the LSB is placed in Bit 0 of the result register (unshifted), creating no fractional bits.

Normally, the Bit-19 mode switch is set to 0 to place the source LSB in Bit 5 of the result register. Bit 19 is set to 1 to place to source LSB in Bit 0 of the result register for one of three reasons:

- The data already comes with 5 bits of fraction, as from a Compact MACRO Station.
- The normal shift limits the maximum velocity too much ($V_{max}<2^{18}$ LSBs per servo cycle)
- The normal shift limits the position range too much (Range<$\pm 2^{47}$/Ix08/32 LSBs)

Unless this is done because the data already contains fractional information, the "unshifted" conversion will mean that the motor position loop will consider 1 LSB of the source to be 1/32 of a count, instead of 1 count.

Bits 0 to 18 of the first line contain the base address of the parallel data to be read. This is the address of the least significant byte in the parallel feedback word. The following table shows the possible entries when an ACC-3E stack I/O board is used:

Entry First Lines for ACC-3E 3U-Stack I/O Boards

ACC-3E Address Jumper	E1	E2	E3	E4
First-Line Value	$F7880x	$F7890x	$F78A0x	$F78B0x

The following table shows the possible entries when the ACC-14E UMAC I/O board is used:

Entry First Lines for ACC-14E UMAC I/O Boards

DIP-Switch Setting	**SW1-1 ON (0) SW1-2 ON (0)**	**SW1-1 OFF (1) SW1-2 ON (0)**	**SW1-1 ON (0) SW1-2 OFF (1)**	**SW1-1 OFF (1) SW1-2 OFF (1)**
SW1-3 ON (0) SW1-4 ON (0)	$F78C0x	$F78D0x	$F78E0x	$F78F0x
SW1-3 OFF (1) SW1-4 ON (0)	$F79C0x	$F79D0x	$F79E0x	$F79F0x
SW1-3 ON (0) SW1-4 OFF (1)	$F7AC0x	$F7AD0x	$F7AE0x	$F7AF0x
SW1-3 OFF (1) SW1-4 OFF (1)	$F7BC0x	$F7BD0x	$F7BE0x	$F7BF0x
A switch that is ON is CLOSED; a switch that is OFF is OPEN.				

In both of these tables, the second digit should be changed from a '7' to an 'F' if bit 19 is set to 1 to disable the data shift.

The final digit, represented by an 'x' in both of these tables, can take a value of 0 to 5, depending on which I/O point on the board is used for the LSB:

- x=0: I/O00-07 I/O48-55 I/O96-103
- x=1: I/O08-15 I/O56-63 I/O104-111
- x=2: I/O16-23 I/O64-71 I/O112-119
- x=3: I/O24-31 I/O72-79 I/O120-127
- x=4: I/O32-39 I/O80-87 I/O128-135
- x=5: I/O40-47 I/O88-95 I/O136-143

Width/Offset Word: The second setup line (I-variable) of this parallel read entry contains information about what data is to be read starting at the base address. This 24-bit value, usually represented as 6 hexadecimal digits, is split into four parts, as shown in the following table.

Hex Digit	1	2	3	4	5	6
Contents	2 or 3	Bit Width		Byte	LSB Location	

The first hex digit contains a 2 or a 3. If it has a 2, there is no filtering of the data, and the entry is a 2-line entry. If it has a 3, the input data is filtered to protect against noise or data corruption, and the entry is a 3-line entry, with the third line controlling the filtering.

The second and third digits represent the width of the parallel data in bits, and can range from $01 (1 bit wide – not of much practical use) to $18 (24 bits wide). If the value of these digits is from $01 to $08, only the base address in the first line is used. If the value of these digits is from $09 to $10 (16), the base address and the next higher-numbered address are used. If the value of these digits is from $11 to $18 (17 to 24), three addresses starting at the base address are used.

The fourth digit represents which byte of the source words is used. It has three valid values:

- 0: Low byte (bits 0 – 7)
- 1: Middle byte (bits 8 – 15)
- 2: High byte (bits 16 – 23)

The fifth and sixth digits contain the bit location of the LSB of the data in the source word at the base address, and can range from $00 (Bit 0 of the source address is the LSB), through $07 (Bit 7 of the source address is the LSB). To calculate this value, divide the number of the I/O point used for the LSB by 8 and use the remainder here. For example, if I/O19 is used for the LSB, the remainder of 19/16 is 3.

Maximum Change Word: If the method character for a parallel read is $3 or $7, specifying "filtered" parallel read, there is a third setup line (I-variable) for the entry. This third line contains the maximum change in the source data in a single cycle that will be reflected in the processed result, expressed in LSBs per servo cycle. The filtering that this creates provides an important protection against noise and misreading of data. This number is effectively a velocity value, and should be set slightly greater than the maximum true velocity ever expected.

TURBO PMAC ON-LINE COMMAND SPECIFICATION

<CONTROL-A>

Function: Abort all programs and moves.

Scope: Global

Syntax: **ASCII Value 1D; $01**

This command aborts all motion programs and stops all non-program moves on the card. It also brings any disabled or open loop motors to an enabled zero-velocity closed-loop state. Each motor will decelerate at a rate defined by its own motor I-variable I*xx*15. However, a multi-axis system may not stay on its programmed path during this deceleration.

A **<CTRL-A>** stop to a program is not meant to be recovered from gracefully, because the axes will in general not stop at a programmed point. An on-line **J=** command may be issued to each motor to cause it to move to the end point that was programmed when the abort occurred. Then the program(s) can be resumed with an **R** (run) command.

To stop a motion sequence in a manner that can be recovered from easily, use instead the Quit (**Q** or **<CTRL-Q>**) or the Hold (**H** or **<CTRL-O>**) command.

When Turbo PMAC is set up to power on with all motors killed (Ixx80 = 0), this command can be used to enable all of the motors (provided that they are not synchronous motors commutated by Turbo PMAC -- in that case, the motors should be enabled with the **$** or **$$** command).

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also:
Stop Commands (Making Your Application Safe)
On-line commands **A**, **$**, **$$**, **/**, ****, **J=**, **H**, **<CTRL-O>**, **Q**, **<CTRL-Q>**
I-variables **Ixx15**, **Ixx80**.

<CONTROL-B>

Function Report status word for 8 motors.

Scope Global

Syntax **ASCII Value 2D; $02**

This command causes Turbo PMAC to report the status words for 8 selected motors to the host in hexadecimal ASCII form, 12 characters per motor starting with the lowest-numbered of the selected motors, with the characters for each motor separated by spaces. The characters reported for each motor are the same as if the **?** command had been issued for that motor.

The set of eight motors whose data is reported is selected by the most recent **##{constant}** value for this port:

- **##0**: Motors 1 – 8 (default)
- **##1**: Motors 9 – 16
- **##2**: Motors 17 – 24
- **##3**: Motors 25 – 32

The detailed meanings of the individual status bits are shown under the **?** command description.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (**@n**).

Example:
<CTRL-B>

```
812000804001 812000804001 812000A04001 812000B04001 050000000000
050000000000 050000000000 050000000000<CR>
```

See Also:
On-line commands **<CTRL-C>**, **<CTRL-G>**, **##**, **##{constant}**, **?**, **@n**
Memory-map registers X:$0000B0, X:$000130, etc., Y:$0000C0, Y:$000140, etc.;
Suggested M-Variable definitions Mxx30-Mxx45.

<CONTROL-C>

Function: Report all coordinate system status words

Scope: Global

Syntax: ASCII Value 3D, $03

This command causes Turbo PMAC to report the status words for all 16 of the coordinate systems to the host in hexadecimal ASCII form, 12 characters per coordinate system starting with coordinate system 1, with the characters for each coordinate system separated by spaces. The characters reported for each coordinate system are the same as the first twelve characters reported if the **??** command had been issued for that coordinate system.

The detailed meanings of the individual status bits are shown under the **??** command description.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example:
<CTRL-C>

```
A80020020000 A80020020000 A80020020000 A80020020000 A80020000000
A80020000000 A80020000000 A80020000000 A80020020000 A80020020000
A80020020000 A80020020000 A80020000000 A80020000000 A80020000000
A80020000000<CR>
```

See Also:
On-line commands **<CTRL-B>**, **<CTRL-G>**, **??**;
Memory-map registers X:$002040, X:$0020C0, etc., Y:$00203F, Y:$0020BF, etc.;
Suggested M-variable definitions Msx80-Msx90.

<CONTROL-D>

Function: Disable all PLC programs.

Scope: Global

Syntax: **ASCII Value 4D; $04**

This command causes all PLC programs to be disabled (i.e. stop executing). This is the equivalent of **DISABLE PLC 0..31** and **DISABLE PLCC 0..31**. It is especially useful if a **CMD** or **SEND** statement in a PLC has run amok.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Example:

```
TRIGGER FOUND
TRIGTRIGER FOTRIGGER FOUND
TRTRIGTRIGGER FOUND        (Out-of-control SEND message from PLC)
<CTRL-D>.....              (Command to disable the PLCs)
.....                      (No more messages; can now edit PLC)
```

See Also:
On-line commands **DISABLE PLC**, **ENABLE PLC**, **DISABLE PLCC**, **ENABLE PLCC**, **OPEN PLC**
Program commands **DISABLE PLC**, **ENABLE PLC**, **DISABLE PLCC**, **ENABLE PLCC**, **COMMAND**, **SEND**

<CONTROL-F>

Function: Report following errors for 8 motors.

Scope: Global.

Syntax: **ASCII Value 6D; $06**

This command causes Turbo PMAC to report the following errors of a set of 8 motors to the host. The errors are reported in an ASCII string, each error scaled in counts, rounded to the nearest tenth of a count. A space character is returned between the reported errors for each motor.

The set of eight motors whose data is reported is selected by the most recent **##{constant}** value for this port:

- **##0**: Motors 1 – 8 (default)
- **##1**: Motors 9 – 16
- **##2**: Motors 17 – 24
- **##3**: Motors 25 – 32

Refer to the on-line **F** command for more detail as to how the following error is calculated.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example:
```
<CTRL-F>
0.5 7.2 -38.3 1.7 0 0 0 0<CR>
```

See Also:
I-variables Ixx11, Ixx12
On-line commands **##**, **##{constant}**, **F**, **<CTRL-P>**, **<CTRL-V>**

<CONTROL-G>

Function: Report global status word.

Scope: Global

Syntax: **ASCII Value 7D; $07**

This command causes Turbo PMAC to report the global status words to the host in hexadecimal ASCII form, using 12 characters. The characters sent are the same as if the **???** command had been sent, although no command acknowledgement character (<ACK> or <LF>, depending on I3) is sent at the end of the response.

The detailed meanings of the individual status bits are shown under the **???** command description.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example:
```
<CTRL-G>
003000400000<CR>
```

See Also:
On-line commands **<CTRL-B>**, **<CTRL-C>**, **???**
Memory-map registers X:$000006, Y:$000006.

<CONTROL-H>

Function: Erase last character.

Scope: Port specific

Syntax: **ASCII Value 8D; $08 (<BACKSPACE>).**

This character, usually entered by typing the **<BACKSPACE>** key when talking to Turbo PMAC in terminal mode, causes the most recently entered character in Turbo PMAC's command-line-receive buffer for this port to be erased.

See Also:
Talking to Turbo PMAC
On-line command **<CTRL-O>** (Feed Hold All)

<CONTROL-I>

Function: Repeat last command line.

Scope: Port specific

Syntax: **ASCII Value 9D; $09 (<TAB>).**

This character, sometimes entered by typing the **<TAB>** key, causes the most recently sent alphanumeric command line to Turbo PMAC on this port to be re-commanded. It provides a convenient way to quicken a repetitive task, particularly when working interactively with Turbo PMAC in terminal mode. Other control-character commands cannot be repeated with this command.

Note:

Most versions of the PMAC Executive Program "trap" a **<CTRL-I>** or **<TAB>** for their own purposes, and do not send it on to Turbo PMAC, even when in terminal mode.

Example:
This example shows how the tab key can be used to look for some event:

```
PC<CR>
P1:10<CR>
<TAB>
P1:10<CR>
<TAB>
P1:10<CR>
<TAB>
P1:11<CR>
```

See Also:
On-line command **<CONTROL-Y>**.

<CONTROL-K>

Function: Kill all motors.

Scope: Global

Syntax: **ASCII Value 11D; $0B**

This command kills all motor outputs by opening the servo loop, commanding zero output, and taking the amplifier enable signal (AENA*n*) false (polarity is determined by jumper E17x on Turbo PMAC(1) boards) for all motors on the card. If any motion programs are running, they will be aborted automatically.

(For the motor-specific **K** (kill) command, if the motor is in a coordinate system that is executing a motion program, the program execution must be stopped with either an **A** (abort) or **Q** (quit) command before Turbo PMAC will accept the **K** command.)

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also: On-line commands **K**, **A**, **<CONTROL-A>**.

<CONTROL-M>

Function: Enter command line.

Scope: Port specific

Syntax: **ASCII Value 13D; $0D (<CR>)**

This character, commonly known as **<CR>** (carriage return), causes the alphanumeric characters in the Turbo PMAC's command-line-receive buffer for this port to be interpreted and acted upon. (Control-character commands do not require a **<CR>** character to execute.)

Note that for multiple Turbo PMACs daisy-chained together on a serial interface, this will act on all cards simultaneously, not just the software-addressed card. For simultaneous action on multiple cards, it is best to load up the command-line-receive buffers on all cards before issuing the **<CR>** character.

Example:

```
#1J+<CR>
P1<CR>
@0&1B1R@1&1B7R<CR>      (This causes card 0 on the serial daisychain to
.........               have its CS 1 execute PROG 1 and card 1 to
.........               have its CS 1 execute PROG 7 simultaneously.)
```

See Also: Talking to Turbo PMAC

<CONTROL-N>

Function: Report command line checksum.

Scope: Port specific

Syntax: **ASCII Value 14D; $0E**

This character causes Turbo PMAC to calculate and report the checksum of the alphanumeric characters of the present command line (i.e. since the most recent carriage-return character) for this port.

As typically used, the host computer would send the entire command line up to, but not including, the carriage return. It would then send the **<CTRL-N>** character, and Turbo PMAC would return the checksum value. If this value agreed with the host's internally calculated checksum value, the host would then send the **<CR>** and Turbo PMAC would execute the command line. If the values did not agree, the host would send a **<CTRL-X>** command to erase the command line, then resend the line, repeating the process.

> ***Note:***
>
> The PMAC Executive Program terminal mode will not display the checksum values resulting from a **<CTRL-N>** command.

Example:

With I4=1 and I3=2:

Host sends: .....**J+<CTRL-N>**
Turbo PMAC sends: *<117dec>* (117=74[J] + 43[+]; correct)
Host sends: .....**<CR>**

Turbo PMAC sends:	*<ACK><117dec>*	(handshake & checksum again)
Host sends:	**J/<CTRL-N>**	
Turbo PMAC sends:	*<122dec>*	(122 != 74[J] +47[/]; incorrect)
Host sends:	**<CTRL-X>**	(Erase the incorrect command)
.....	**J/<CTRL-N>**	(Send the command again)
Turbo PMAC sends:	*<121dec>*	(121 = 74[J] + 47[/]; correct)
Host sends:	**<CR>**	
Turbo PMAC sends:	*<ACK><121dec>*	(handshake & checksum again)

See Also:
Communications Checksum (Writing a Host Communications Program)
I-variables I3, I4
On-line commands **<CTRL-M>** (**<CR>**), **<CTRL-X>**

<CONTROL-O>

Function: Feed hold on all coordinate systems.

Scope: Global

Syntax: **ASCII Value 15D; $0F**

This command causes all coordinate systems in Turbo PMAC to undergo a feed hold. It is equivalent to issuing the **H** command to each coordinate system. Refer to the **H** command specification for more detail on the action.

A feed hold is much like a **%0** command where the coordinate system is brought to a stop without deviating from the path it was following, even around curves. However, with a feed hold, the coordinate system slows down at a slew rate determined by Isx95, and can be started up again with an **R** (run)command. The system then speeds up at the rate determined by Isx95, until it reaches the desired **%** value (from internal *or* external timebase). From then on, any timebase changes occur at a rate determined by Isx94.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also:
Resetting Turbo PMAC (Talking to Turbo PMAC)
I-variables Isx94, Isx95
On-line commands **<CTRL-H>** (backspace) **H** (feedhold), **R** (run), **%** (feedrate override).

<CONTROL-P>

Function: Report positions for 8 motors.

Scope: Global

Syntax: **ASCII Value 16D; $10**

This command causes the positions of a selected 8 motors to be reported to the host. The positions are reported as a decimal ASCII string, scaled in counts, rounded to the nearest 1/32 of a count, with a space character in between each motor's position.

The set of eight motors whose data is reported is selected by the most recent **##{constant}** value for this port:

- **##0**: Motors 1 – 8 (default)
- **##1**: Motors 9 – 16
- **##2**: Motors 17 – 24
- **##3**: Motors 25 – 32

The position window in the Turbo PMAC Executive program works by repeatedly sending the **<CTRL-P>** command and rearranging the response into the window.

Turbo PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 1 of Ixx06 is 1 (handwheel offset mode), minus the master position register.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example:

```
<CTRL-P>
9999.5 10001.2 5.7 -2.1 0 0 0 0<CR>
```

See Also:

On-line commands **##**, **##{constant}**, **P**, **<CTRL-V>**, **<CTRL-F>**.

<CONTROL-Q>

Function: Quit all executing motion programs.

Scope: Global

Syntax: **ASCII Value 17D; $11**

This command causes any and all motion programs running in any coordinate system to stop executing either at the end of the currently executing move, or after the moves that have already been calculated are finished, depending on the mode. It is equivalent to issuing the **Q** command to all coordinate systems. Refer to the **Q** command description for more details.

Program execution may be resumed from this point with the **R** (run) or **S** (step) commands.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also:

On-line commands **<CTRL-A>**, **<CTRL-K>**, **<CTRL-O>**, **<CTRL-R>**, **<CTRL-S>**, **Q**
Motion-program command **STOP**.

<CONTROL-R>

Function: Begin execution of motion programs in all coordinate systems.

Scope: Global

Syntax: **ASCII Value 18D; $12**

This command is the equivalent of issuing the **R** (run) command to all coordinate systems in Turbo PMAC. Each active coordinate system (i.e. one that has at least one motor assigned to it) that is to run a program must already be pointing to a motion program (initially this is done with a **B{*prog num*}** command).

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Example:

```
&1B1&2B500<CR>
<CTRL-R>
```

See Also:

Executing a Motion Program (Writing a Motion Program)
Resetting Turbo PMAC (Talking to Turbo PMAC)
On-line commands **R**, **B{constant}**

<CONTROL-S>

Function: Step working motion programs in all coordinate systems.

Scope: Global

Syntax: **ASCII Value 19D; $13**

This command is the equivalent of issuing an **S** (step) command to all of the coordinate systems in Turbo PMAC.

Each active coordinate system (i.e. one that has at least one motor assigned to it) that is to run a program must already be pointing to a motion program (initially this is done with a **B{*prog num*}** command).

A program that is not running will execute all lines down to and including the next motion command (move or dwell), or if it encounters a **BLOCKSTART** command first, all lines down to and including the next **BLOCKSTOP** command.

If a program is already running in continuous execution mode (from an **R** (run) command), an **S** command will put the program in single-step mode, stopping execution after the next motion command). In this situation, it has exactly the same effect as a **Q** (quit) command.

For multiple cards on a single serial daisy chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also:
On-line commands **<CTRL-A>**, **<CTRL-O>**, **<CTRL-Q>**, **<CTRL-R>**, **A**, **H**, **O**, **Q**, **R**, **S**;
Motion-program commands **BLOCKSTART**, **BLOCKSTOP**, **STOP**.
Control-panel port (JPAN) input STEP/.

<CONTROL-T>

Function: Cancel MACRO pass-through mode

Scope: Global

Syntax: **ASCII Value 20D; $14**

This command causes Turbo PMAC to cancel the MACRO pass-through mode it had been put in with the **MACROMSTASCII** or the **MACROSTASCII** command on this port. In the MACRO pass-through mode, any command received on the port is passed on to another master on the ring through the MACRO link, the response is received over the ring from the other master, and this response is reported back to the host over this port.

The **<CONTROL-T>** command ends this mode, and resumes normal communications over this port. Subsequent commands on the port are acted on by this Turbo PMAC, and responses go directly over the communications port to the host computer.

If I63 is set to its default value of 0, Turbo PMAC sends no acknowledgment that it has finished its action on the **<CTRL-T>** command. If I63 is set to 1, Turbo PMAC acknowledges that it has finished its action by returning a **<CTRL-X>** character back to the host.

If the port that receives the **<CONTROL-T>** command is not currently in the MACRO pass-through mode, Turbo PMAC will take no action on receipt of the command.

See Also:
MACRO Master-to-Master Communications
On-line command **MACROMSTASCII**, **MACROSTASCII**

<CONTROL-V>

Function: Report velocity for 8 motors.

Scope: Global

Syntax: **ASCII Value 22D; $16**

This command causes Turbo PMAC to report the velocities of a selected set of 8 motors to the host. The velocity units are typically scaled in encoder counts per servo cycle, rounded to the nearest tenth. The velocity window in the Turbo PMAC Executive program works by repeatedly issuing the **<CTRL-V>** command and displaying the response on the screen.

To scale these values into counts/msec, multiply the response by 8,388,608/I10 (servo cycles/msec).

The set of eight motors whose data is reported is selected by the most recent **##{constant}** value for this port:

- **##0**: Motors 1 – 8 (default)
- **##1**: Motors 9 – 16
- **##2**: Motors 17 – 24
- **##3**: Motors 25 – 32

This command returns filtered velocity values, with the filter time constant controlled by global variables I60 and I61. It does not report the raw velocity register calculated by the servo loop each servo cycle.

For multiple cards on a single serial daisy chain, this command affects only the card currently addressed in software (**@n**).

See Also:
I-variables I10, I59 I60, I61 Ixx60
On-line commands **<CTRL-B>**, **<CTRL-F>**, **<CTRL-P>**, **##**, **##{constant}**, **V**

<CONTROL-X>

Function: Cancel in-process communications.

Scope: Port-specific

Syntax: **ASCII Value 24D; $18**

This command causes the Turbo PMAC to stop sending any messages that it had started to send, even multi-line messages, on the port over which this command is sent. This also causes Turbo PMAC to empty the port's command queue from the host, so it will erase any partially sent commands.

It can be useful to send this before sending a query command for which you are expecting an exact response format, if you are not sure what Turbo PMAC has been doing before, because it makes sure nothing else comes through before the expected response. As such, it is often the first character sent to Turbo PMAC from the host when trying to establish initial communications.

If I63 is set to its default value of 0, Turbo PMAC sends no acknowledgment that it has finished its action on the **<CTRL-X>** command. If I63 is set to 1, Turbo PMAC acknowledges that it has finished its action by echoing the **<CTRL-X>** character back to the host.

This can result in more efficient communications, and is supported in PCOMM32 communications routines in V2.21 and newer (March 1999 and later).

> ***Note:***
>
> This command empties the command queue in Turbo PMAC RAM, but it cannot erase the 1 or 2 characters already in the response port. A robust algorithm for clearing responses would include two-character read commands that can time-out if necessary.

For multiple cards on a single serial daisy chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also:
I-variable I63
On-line command **<CTRL-H>**

!{axis}{constant}[{axis}{constant}…]

Function: Alter destination of RAPID move

Scope: Coordinate-system specific

Syntax: **!{axis}{constant}[{axis}{constant}…]**

where:

- **{axis}** is the letter specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **{constant}** is a numerical value representing the end position;
- **[{axis}{constant}…]** is the optional specification of simultaneous movement for more axes.

or

!{axis}Q{constant}[{axis}Q{constant}…]

where:

- **{axis}** is the letter specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **{constant}** is a numerical value representing the number or the Q-variable whose value specifies the end position;
- **[{axis}Q{constant}…]** is the optional specification of simultaneous movement for more axes.

This command creates a RAPID-mode move of the specified axis or axes to the specified destination(s). If another RAPID-mode move of an axis is in progress, that move is "broken into" and the motion of the axes is blended into the move to this new destination, effectively altering the destination of the move in progress.

Each axis destination can be specified either directly as a numerical constant (e.g. **!X63.72**), or indirectly by specifying the Q-variable whose value represents the axis destination (e.g. **!XQ15**).

In either case, the destination value for each axis is in the scaled engineering units for the axis. The destination value always represents the end position for the axis, relative to "program zero", even if the axis is currently in incremental mode. Execution of this command does not change the mode of the axis. The order in which the axes are specified in this command does not matter.

If a programmed move of a mode other than RAPID is in progress when this command is sent, this command will be rejected with an error.

If no move is in progress when this command is sent, this command will simply execute a RAPID-mode move to the specified destination. In this case, before starting the move, Turbo PMAC will automatically execute the PMATCH position-matching function to make sure motor and axis positions are properly linked in order for the move to execute properly.

Examples:

```
!X5
!X23.762 Y-345.124
!A-90.2 B37.3
!XQ152 YQ154
!XQ30 Y37.936
```

See Also:
Altered Destination Moves
RAPID-Mode Moves
I-Variables Ixx16, Ixx19, Ixx20, Ixx21, Ixx22, Ixx90, Ixx92

@

Function: Report currently addressed card on serial daisy-chain

Scope: Global

Syntax: **@**

This command causes the addressed Turbo PMAC on a serial daisy-chain to report its number to the host. The number is set by variable I0 on the board, and can range from 0 to 15. If all cards are addressed, card @0 will return an **@** character.

I1 must be set to 2 or 3 for this command to be accepted. Otherwise, ERR003 is reported.

Example:

@ ..... ; Ask Turbo PMAC chain which card is addressed
4 ..... ; Turbo PMAC @4 reports that it is addressed

See Also:
Addressing Commands (Talking to Turbo PMAC)
Multiple-Card Applications (Synchronizing Turbo PMAC to External Events)
I-variables I0, I1
On-line commands **#**, **#{constant}**, **&**, **&{constant}**, **@{constant}**

@{card}

Function: Address a card on the serial daisychain.

Scope: Global

Syntax: **@{card}**

where:

- **{card}** is a hexadecimal digit (0 to 9, A to F), representing the number of the card on the serial daisychain to be addressed; or the @ character, denoting that all cards are to be addressed simultaneously.

This command makes the Turbo PMAC board specified by **{card}** the addressed board on the serial daisychain. (the one on which subsequent commands will act). The number for each board is set by variable I0 on the board. The addressing is modal, so all further commands will affect this board until a different board is addressed. At power-up/reset, Board @0 is addressed.

I1 must be set to 2 or 3 for this command to be accepted. Otherwise, ERR003 is reported.

To address all cards simultaneously, use the **@@** command. Query commands (those requiring a data response) will be rejected in this mode.

It is best to send a **<CR>** carriage return character immediately after the **@{card}** command before any other command is sent, to give the card that had been addressed time to tri-state its serial port outputs so that it will not interfere with the response of the newly addressed card.

This command should only be used when multiple Turbo PMAC cards are connected on a single serial cable. In this case, I-variable I1 should be set to 2 or 3 on all boards.

Example:

```
I1=2@0.....     ; This sequence can be used the first time talking to
.............     multiple cards on a chain to put them in the proper
.............     configuration.
@0#1J+.....     ; Jog motor 1 of Card 0.
@5P20.....      ; Request the value of P20 on card @5
@@R.....        ; All cards, addressed C.S. run active program
```

See Also:

Addressing Commands (Talking to Turbo PMAC)
Multiple-Card Applications (Synchronizing Turbo PMAC to External Events)
I-variables I0, I1
On-line commands **#**, **&**, **&{constant}**, **@**

#

Function: Report port's currently addressed motor

Scope: Port specific

Syntax: **#**

This command causes Turbo PMAC to return the number of the motor currently addressed for the communications port over which this command is sent. This is the motor that will act on subsequent motor-specific commands sent over this port until a different motor is addressed with a **#{constant}** command.

Other communications ports may be addressing different motors at the same time, as set by **#{constant}** commands sent over those ports. In addition, each background PLC program can individually modally address a motor using the **ADDRESS** statement for subsequent **COMMAND** statements, and the hardware control panel on a Turbo PMAC(1) can separately select a motor for its hardware inputs.

Note:

In firmware versions 1.934 and older, all communications ports addressed the same motor, so a **#{constant}** command sent over any port set the addressed motor for all ports.

Example:

```
#  .....     ; Ask Turbo PMAC which motor is addressed
2  .....     ; Turbo PMAC reports that motor 2 is addressed
```

See Also:

Control-Panel Port Inputs (Connecting Turbo PMAC to the Machine)
On-line commands #{constant}, &, &{constant}, @{constant}
Program commands **ADDRESS**, **COMMAND**

#{constant}

Function: Select port's addressed motor

Scope: Port specific

Syntax: **#{constant}**

where:

- **{constant}** is an integer from 1 to 32, representing the number of the motor to be addressed

This command makes the motor specified by **{constant}** the addressed motor for the communications port over which this command is sent. This is the motor that will act on subsequent motor-specific commands sent over this port until a different motor is addressed with another **#{constant}** command.

Other communications ports may be addressing different motors at the same time, as set by **#{constant}** commands sent over those ports. In addition, each background PLC program can individually modally address a motor using the **ADDRESS** statement for subsequent **COMMAND** statements, and the hardware control panel on a Turbo PMAC(1) can separately select a motor for its hardware inputs.

Note:

In firmware versions 1.934 and older, all communications ports addressed the same motor, so a **#{constant}** command sent over any port set the addressed motor for all ports.

Example:

```
#1J+ ..... ; Command Motor 1 to jog positive
J- ..... ; Command Motor 1 to jog negative
#2J+ ..... ; Command Motor 2 to jog positive
J/ ..... ; Command Motor 2 to stop jogging
```

See Also:

Control-Panel Port Inputs (Connecting Turbo PMAC to the Machine)
Addressing commands (Talking to Turbo PMAC)
Program commands **COMMAND**, **ADDRESS**
On-line commands **#**, **&**, **&{constant}**, **@{constant}**

#{constant}->

Function: Report the specified motor's coordinate system axis definition.

Scope: Coordinate-system specific

Syntax: **#{constant}->**

where:

- **{constant}** is an integer from 1 to 32 representing the number of the motor whose axis definition is requested

Note:

No spaces are allowed in this command.

This command causes Turbo PMAC to report the current axis definition of the specified motor in the currently addressed coordinate system. If the motor has not been defined to an axis in the currently addressed system, Turbo PMAC will return a **0** (even if the motor has been assigned to an axis in another coordinate system). A motor can have an axis definition in only one coordinate system at a time.

Example:

```
&1 ..... ; Address Coordinate System 1
#1-> ..... ; Request Motor 1 axis definition in C.S. 1
10000X..... ; Turbo PMAC responds with axis definition
&2 ..... ; Address Coordinate System 2
#1-> ..... ; Request Motor 1 axis definition in C.S. 2
0 ..... ; Turbo PMAC shows no definition in this C.S.
```

See Also:
Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->0**, **#{constant}->{axis definition}**, **UNDEFINE**, **UNDEFINE ALL**.

#{constant}->0

Function: Clear axis definition for specified motor.

Scope: Coordinate-system specific

Syntax: **#{constant}->0**

where:

- **{constant}** is an integer from 1 to 32 representing the number of the motor whose axis definition is to be cleared

Note:

No spaces are allowed in this command.

This command clears the axis definition for the specified motor *if* the motor has been defined to an axis in the currently addressed coordinate system. If the motor is defined to an axis in another coordinate system, this command will not be effective. This allows the motor to be redefined to another axis in this coordinate system or a different coordinate system.

Compare this command to **UNDEFINE**, which erases all the axis definitions in the addressed coordinate system, and to **UNDEFINE ALL**, which erases all the axis definitions in all coordinate systems.

Example:
This example shows how the command can be used to move a motor from one coordinate system to another:

```
&1 ..................  ; Address C.S. 1
#4-> ................  ; Request definition of #4
5000A................  ; Turbo PMAC responds
#4->0................  ; Clear definition
&2 ..................  ; Address C.S. 2
#4->5000A ..... ; Make new definition in C.S. 2
```

See Also:
Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **UNDEFINE**, **UNDEFINE ALL**, **#{constant}->{axis definition}**.

#{constant}->{axis definition}

Function: Assign an axis definition for the specified motor.

Scope: Coordinate-system specific

Syntax: **#{constant}->{axis definition}**

where:

- **{constant}** is an integer from 1 to 32 representing the number of the motor whose axis definition is to be made;
- **{axis definition}** consists of 1 to 3 sets of **[{scale factor}]{axis}**, separated by the **+** character, in which:
  - the optional **{scale factor}** is a floating-point constant representing the number of motor counts per axis unit (engineering unit); if none is specified, Turbo PMAC assumes a value of 1.0;

- **{axis}** is a letter (X, Y, Z, A, B, C, U, V, W) representing the axis to which the motor is to be matched;
- **[+{offset}]** (optional) is a floating-point constant representing the difference between axis zero position and motor zero (home) position, in motor counts; if none is specified, Turbo PMAC assumes a value of 0.0

> ***Note:***
>
> No space is allowed between the motor number and the "arrow" double character, or between the scale factor and the axis letter.

This command assigns the specified motor to a set of axes in the addressed coordinate system. It also defines the scaling and starting offset for the axis or axes.

In the vast majority of cases, there is a one-to-one matching between Turbo PMAC motors and axes, so this axis definition statement only uses one axis name for the motor.

A scale factor is typically used with the axis character, so that axis moves can be specified in standard units (e.g. millimeters, inches, degrees). This number is what defines what the user units will be for the axis. If no scale factor is specified, a user unit for the axis is one motor count.

Occasionally an offset parameter is used to allow the axis zero position to be different from the motor home position. (This is the starting offset; it can later be changed in several ways, including the **PSET**, **{axis}=**, **ADIS**, and **IDIS** commands).

If the specified motor is currently assigned to an axis in a different coordinate system, Turbo PMAC will reject this command (reporting an ERR003 if I6=1 or 3). If the specified motor is currently assigned to an axis in the addressed coordinate system, the old definition will be overwritten by this new one.

To undo a motor's axis definition, address the coordinate system in which it has been defined, and use the command **#{constant}->0**. To clear all of the axis definitions within a coordinate system, address the coordinate system and use the **UNDEFINE** command. To clear all axis definitions in *all* coordinate systems, use **UNDEFINE ALL**.

For more sophisticated systems, two or three cartesian axes may be defined as a linear combination of the same number of motors. This allows coordinate system rotations and orthogonality corrections, among other things. One to three axes may be specified (if only one, it amounts to the simpler definition above). All axes specified in one definition must be from the same triplet set of cartesian axes: XYZ or UVW. If this multi-axis definition is used, a command to move an axis will result in multiple motors moving.

Example:

```
#1->X.......          ; User units = counts
#4->2000 A..          ; 2000 counts/user unit
#9->3333.333Z-666.667 ; Non-integers OK

#3->Y.....            ; 2 motors may be assigned to the same axis;
#2->Y.....            ; both motors move when a Y move is given

#1->8660X-5000Y       ;This provides a 30o rotation of X and Y...
#2->5000X+8660Y       ;with 10000 cts/unit -- this rotation does
#3->2000Z-6000        ;not involve Z, but it could have
```

This example corrects for a Y axis 1 arc minute out of square:

```
#5->100000X           ;100000 cts/in
#6->-29.1X+100000Y    ;sin and cos of 1/60
```

See Also:

Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->**, **#{constant}->0**, **UNDEFINE**, **UNDEFINE ALL**.

#{constant}->I

Function: Assign inverse-kinematic definition for specified motor

Scope: Coordinate-system specific

Syntax: **#{constant}->I[+{offset}]**

where:

- **{constant}** is an integer from 1 to 32 representing the number of the motor whose axis definition is to be made;
- **[+{offset}]** (optional) is a floating-point constant representing the difference between axis zero position and motor zero (home) position, in motor counts; if none is specified, Turbo PMAC assumes a value of 0.0

> ***Note:***
>
> No space is allowed between the motor number and the "arrow" double character.

This command assigns the specified motor to an inverse-kinematic axis in the addressed coordinate system. It also defines the offset for the axis. A motor assigned in this way must get its commanded positions each programmed move or segment from the inverse-kinematic program for the coordinate system. This program, created with an **OPEN INVERSE** command, is executed automatically each programmed move or segment if Isx50 for the coordinate system is set to 1.

At the end of each execution of the inverse-kinematic program for the coordinate system, Turbo PMAC expects to find the motor position calculated by the program for each Motor xx in the coordinate system defined as an inverse-kinematic axis in variable Pxx (e.g. P13 for Motor 13).

See Also:
Inverse Kinematics
I-variable Isx50
On-line commands **OPEN FORWARD**, **OPEN INVERSE**

##

Function: Report port's motor group

Scope: Port specific

Syntax: **##**

This command causes Turbo PMAC to return the number of the motor group currently selected on this port for on-line commands **<CTRL-B>**, **<CTRL-F>**, **<CTRL-P>**, and **<CTRL-V>**. This value can be set for the port by the **##{constant}** command, and defaults to 0 on power-up/reset. Each communications port can have a different value.

> ***Note:***
>
> This is not related to the individual motor addressed with the # command, and reported with the **#{constant}** command

The possible values returned and the motors they represent are:

- 0: Motors 1 – 8
- 1: Motors 9 – 16
- 2: Motors 17 – 24
- 3: Motors 25 – 32

Note:

In Turbo PMAC firmware versions 1.934 and older, this function was controlled commonly for all ports by global I-variable I59.

See Also:
I-variable I59

On-line commands **<CTRL-B>**, **<CTRL-F>**, **<CTRL-P>**, **<CTRL-V>**, **##{constant}**

##{constant}

Function: Select port's motor group

Scope: Port specific

Syntax: **##{constant}**

where:

- **{constant}** is an integer from 0 to 3 representing the motor group

This command selects the group of eight motors whose data will be supplied in response to subsequent **<CTRL-B>** (report motor status words), **<CTRL-F>** (report motor following errors), **<CTRL-P>** (report motor positions), and **<CTRL-V>** (report motor velocities) commands issued on this same port. It does not affect the behavior of these commands issued on any other port.

Note:

This is not related to the individual motor addressed with the # command, and reported with the **#{constant}** command

The possible versions of the **##{constant}** command and the motors they select are:

- **##0**: Motors 1 – 8
- **##1**: Motors 9 – 16
- **##2**: Motors 17 – 24
- **##3**: Motors 25 – 32

Note:

In Turbo PMAC firmware versions 1.934 and older, this function was controlled commonly for all ports by global I-variable I59.

See Also:
I-variable I59
On-line commands **<CTRL-B>**, **<CTRL-F>**, **<CTRL-P>**, **<CTRL-V>**, **##**

$

Function: Establish phase reference for motor

Scope: Motor specific

Syntax: **$**

This command causes Turbo PMAC to attempt to establish the phase reference and close the servo loop for a PMAC-commutated (Ixx01 bit 0 = 1) synchronous (Ixx78 = 0) motor. On other types of motors, where there is no need to establish a phase reference, the **$** command will simply close the servo loop for the motor (a **J/** command is also suitable for these motors).

The phase reference for a synchronous PMAC-commutated can be established either by a phasing search move if Ixx74 > 0, or by an absolute position read if Ixx81 > 0. If both of these variables are set to 0, Turbo PMAC will set the "phase reference error" status bit for the motor on a **$** command, leaving the motor in the "killed" state, and not permitting the servo loop to be closed until the error status bit is cleared.

If Ixx80 bit 0 is saved as 0, no phase reference is performed automatically at power-up or reset of the full board, and the "phase reference error" bit is set, prohibiting the closing of the servo loop. A subsequent **$** command, successfully executed, is required to establish the phase reference for synchronous, PMAC-commutated motor.

If Ixx80 bit 0 is saved as 1, the phase reference operation is performed automatically at power-up or reset of the full board. In this case it is possible, but not required, to re-establish the phase reference with a subsequent **$** command.

A phasing search move checks for any of the following error conditions both before and after the search:

- Hardware overtravel limits
- Amplifier fault
- I^2T overcurrent fault
- Fatal following error fault
- Integrated following error fault

If any of these error conditions is present, the phase reference is considered to have failed and the "phase reference error" status bit is set. Also, if no movement is detected during the search, the error bit is set

An absolute phase position read checks for any of the above fault conditions shortly after the read. If any of these is found, the read is presumed to have failed and the error bit is set. Also, if an illegal value is read from the sensor (e.g. all 3 hall sensors at 0 or 1), the error bit is set.

If the **$** command is issued while the motor is executing a move, the command will be rejected, with Turbo PMAC reporting `ERR018` if I6 is set to 1 or 3.

If another command to move the motor is issued while the phase reference is still in progress, that command will be rejected, with Turbo PMAC reporting `ERR018` if I6 is set to 1 or 3. The phase reference in progress status bit is set to 1 while the reference is being done.

Example:

```
I180                ; Request value of #1 power-on mode variable

0 .....             ; Turbo PMAC responds with 0
.....               ; powers on unphased and killed
$$$.....            ; Reset card; motor is left in killed state
#1$.....            ; Initialize motor, phasing and reading as necessary
```

See Also:

Absolute Sensors (Setting Up a Motor)
Power-on Phasing (Setting Up Turbo PMAC Commutation)
I-variables Ixx10, Ixx73, Ixx74, Ixx75, Ixx80, Ixx81
On-line commands **$***, **$$**, **$$***, **$$$**, **J/**

$$

Function: Establish phase reference for motors in coordinate system

Scope: Coordinate system specific

Syntax: **$$**

This command causes Turbo PMAC to attempt to establish the phase references and close the servo loops for all of the motors in the addressed coordinate system.

For PMAC-commutated (Ixx01 bit 0 = 1) synchronous (Ixx78 = 0) motors, a phasing search move (Ixx74 > 0) or absolute phase position read (Ixx81 > 0) is performed, and the servo loop is closed. For other types of motors, where there is no need to establish a phase reference, the **$$** command will simply close the servo loop for the motor.

The action of the **$$** command is equivalent to that of the **$** command issued to each motor in the coordinate system. For details on the action performed, refer to the specification of the **$** command.

If the **$$** command is issued while any motor is executing a move, the command will be rejected, with Turbo PMAC reporting ERR018 if I6 is set to 1 or 3.

If another command to move a motor is issued while the phase reference for that motor is still in progress, that command will be rejected, with Turbo PMAC reporting ERR018 if I6 is set to 1 or 3. The phase reference in progress status bit for the motor is set to 1 while the reference is being done.

Example:

```
I180 ..................  ; Request value of #1 power-on mode variable
0 ......................  ; Turbo PMAC responds with 0
I280 ..................  ; Request value of #2 power-on mode variable
0 ......................  ; Turbo PMAC responds with 0
.......................  ; powers on unphased and killed
$$$.....................  ; Reset card; motors are left in killed state
M100=1 M200=1          ; Manually supply power to drives
&1$$ ..................  ; Initialize motors, phasing and reading as necessary
```

See Also:

Absolute Sensors (Setting Up a Motor)
Power-on Phasing (Setting Up Turbo PMAC Commutation)
I-variables Ixx10, Ixx73, Ixx74, Ixx75, Ixx80, Ixx81
On-line commands **$**, **$***, **$$***, **$$$**, **J/**

$$$

Function: Full card reset.

Scope: Global

Syntax: **$$$**

This command causes Turbo PMAC to do a full card reset. The effect of **$$$** is equivalent to that of cycling power on Turbo PMAC, or taking the INIT/ line low, then high.

With the re-initialization jumper (E51 on a Turbo PMAC(1), E3 on a Turbo PMAC2) OFF, this command does a standard reset of the Turbo PMAC. Turbo PMAC copies the contents of the flash memory into active main memory during a normal reset cycle, overwriting any current contents. This means that anything changed in Turbo PMAC's active main memory that was not saved to flash memory will be lost. Contents of the Option 16 supplemental battery-backed parameter memory are not changed by the **$$$** command.

With the re-initialization jumper ON, this command does a reset and re-initialization of the Turbo PMAC. Instead of copying the last saved I-variable values from flash memory into active memory, Turbo PMAC copies the factory default I-variable values into active memory.

Note:

Because this command immediately causes Turbo PMAC to enter its power-up/rest cycle, there is no acknowledging character (**<ACK>** or **<LF>**) returned to the host.

Example:

```
I130=60000 ....  ; Change #1 proportional gain
SAVE ..........  ; Copy active memory to non-volatile flash memory
I130=80000 ....  ; Change gain again
$$$.............  ; Reset card
I130 ..........  ; Request value of parameter
```

```
60000.....          ; Turbo PMAC reports current value, which is
.....               ; SAVEd value
(Put E51 {E3} on)
$$$.....            ; Reset card
I130 .....          ; Request value of parameter
2000 .....          ; Turbo PMAC reports current value, which is default
```

See Also:
Resetting Turbo PMAC (Talking to Turbo PMAC)
Control-Panel Port INIT/ Input (Connecting Turbo PMAC to the Machine)
On-line command **$$$*****
I-variables I5, Ixx80
JPAN Connector Pin 15
Jumpers E3, E51.

$$$***

Function: Global card reset and reinitialization.

Scope: Global

Syntax: **$$$*****

This command performs a full reset of the card and re-initializes the memory. All programs and other buffers are erased in active memory. All I-variables are returned to their factory defaults. (Previously **SAVE**d states for these programs, buffers, and variables are still held in flash memory, and can be brought into active memory with a subsequent **$$$**command). The **$$$*** ** command will also recalculate the firmware checksum reference value and eliminate any password that might have been entered.

M-variable definitions, P-variable values, Q-variable values, and axis definitions are not affected by this command. They can be cleared by separate commands (e.g. **M0..8191->***, **P0..8191=0**, **Q0..8191=0**, **UNDEFINE ALL**).

This command is particularly useful if the program buffers have become corrupted. It clears the buffers and buffer pointers so the files can be re-sent to Turbo PMAC. Regular backup of parameters and programs to the disk of a host computer is strongly encouraged so this type of recovery is possible. The PMAC Executive program has "Save Full Turbo PMAC Configuration" and "Restore Full Turbo PMAC Configuration" functions to make this process easy.

Example:

```
I130=60000....      ; Set #1 proportional gain
SAVE .....          ; Save to non-volatile memory
$$$*** .....        ; Reset and re-initialize card
I130 .....          ; Request value of I130
2000 .....          ; Turbo PMAC reports current value, which is default
$$$.....            ; Normal reset of card
I130 .....          ; Request value of I130
60000.....          ; Turbo PMAC reports last SAVEd value
```

See Also:
On-line command **$$$**, **PASSWORD={string};**
Jumper E3 (PMAC2), E51 (PMAC(1))
PMAC Executive Program Save/Restore Full Configuration.

$$*

Function: Read motor absolute positions

Scope: Coordinate system specific

Syntax: **$$***

The **$$*** command causes PMAC to perform a read of the absolute positions for all motors in the addressed coordinate system that require an absolute position read (Ixx10 > 0), as defined by Ixx10 and Ixx95 for the motor. This command performs the same actions in reading the absolute position data that are normally performed during the board's power-up/reset cycle if Ixx80 bit 2 is set to the default of 0.

The action of this command is equivalent to that of a motor-specific **$*** command to each motor in the coordinate system. Refer to the **$*** command description for the exact actions of this command.

$*

Function: Read motor absolute position

Scope: Motor specific

Syntax: **$***

The **$*** command causes PMAC to perform a read of the absolute position for the addressed motor, as defined by Ixx10 and Ixx95 for the motor. It performs the same actions that are normally performed during the board's power-up/reset cycle.

The **$*** command performs the following actions on the addressed motor:

- The motor is killed (servo loop open, zero command, amplifier disabled).
- If the motor is set up for local hardware encoder position capture by input flags, with bit 0 of Ixx97 set to 0 to specify hardware capture, and bit 18 of Ixx24 set to 0 to specify local, not MACRO, flag operation (these are default values), the hardware encoder counter for the same channel as the flag register specified by Ixx25 is set to 0 (e.g. if Ixx25 specifies flags from channel 3, then encoder counter 3 is cleared).
- The motor "home complete" status bit is cleared.
- The motor "position bias" register, which contains the difference between motor and axis zero positions, is set to 0.
- If Ixx10 for the motor is greater than 0, specifying an absolute position read, the sensor is read as specified by Ixx10 and Ixx95 to set the motor actual position. The actual position value is set to the sum of the sensor value and the Ixx26 "home offset" parameter. Unless the read is determined to be unsuccessful, the motor "home complete" status bit is set to 1.
- If Ixx10 for the motor is set to 0, specifying no absolute position read, the motor actual position register is set to 0.
- Because the motor is "killed" the actual position value is automatically copied into the command position register for the motor.

There are several things to note with regard to this command:

- The motor is left in the "killed" state at the end of execution of this command. To enable the motor, a **$** command should be used if this is a PMAC-commutated motor and a phase reference must be established; otherwise a **J/**, **A**, or **<CTRL-A>** command should be used to enable the motor and close the loop.
- If bit 2 of Ixx80 is set to 1, PMAC will not attempt an absolute position read at the board power-on/reset; in this case, the **$*** command must be used to establish the absolute sensor. If bit 2 of Ixx80 is set to 0 (the default), PMAC will attempt an absolute position read at the board power-on/reset.

- With Ixx10 set to 0, the action of **$*** is very similar to that of the **HOMEZ** command. There are a few significant differences, however:
  - **$*** always kills the motor; **HOMEZ** leaves the servo in its existing state.
  - **$*** sets the present actual position to be zero; **HOMEZ** sets the present commanded position to be zero.
  - **$*** zeros the hardware encoder counter in most cases; **HOMEZ** does not change the hardware encoder counter.
  - All of the motors in a single coordinate system that require an absolute position read can be commanded at once with the coordinate-system specific **$$*** command.

See Also:
I-variables Ixx03, Ixx10, Ixx24, Ixx25, Ixx80, Ixx81
On-line commands **$**, **$$$**, **$$***, **HOMEZ**

%

Function: Report the addressed coordinate system's feedrate override value.

Scope: Coordinate-system specific

Syntax: **%**

This command causes Turbo PMAC to report the present feedrate-override (time-base) value for the currently addressed coordinate system. A value of 100 indicates "'real time"'; i.e. move speeds and times occur as specified.

Turbo PMAC will report the value in response to this command, regardless of the source of the value (even if the source is not the **%{constant}** command).

Example:

```
%  .....; Request feedrate-override value
100.....; Turbo PMAC responds: 100 means real time
H  .....; Command a feed hold
%  .....; Request feedrate-override value
0  .....; Turbo PMAC responds: 0 means all movement frozen
```

See Also:
Time-Base Control (Synchronizing Turbo PMAC to External Events)
I-Variables I10, Isx93, Isx94, Isx95
On-line commands **%{constant}**, **H**

%{constant}

Function: Set the addressed coordinate system's feedrate override value.

Scope: Coordinate-system specific

Syntax: **%{constant}**

where:

- **{constant}** is a non-negative floating point value specifying the desired feedrate override (time-base) value (100 represents real-time)

This command specifies the feedrate override value for the currently addressed coordinate system. The rate of change to this newly specified value is determined by coordinate system I-variable Isx94.

I-variable Isx93 for this coordinate system must be set to its default value (which tells to coordinate system to take its time-base value from the % -command register) in order for this command to have any effect.

The maximum % value that Turbo PMAC can implement is equal to (2^{23}/I10)*100 or the (servo update rate in kHz)*100. If you specify a value greater than this, Turbo PMAC will saturate at this value instead.

If you want to control the time base based on a variable value, you should assign an M-variable (suggested Msx97) to the commanded time base register (X:$002000, X:$002100, etc.), then assign a variable value to the M-variable. The value assigned here should be equal to the desired % value times (I10/100).

Example:

```
%0 ..................... ; Command value of 0, stopping motion
%33.333 ..... ; Command 1/3 of real-time speed
%100 ..... ; Command real-time speed
%500 ..... ; Command too high a value
% ..... ; Request current value
225.88230574 ; Turbo PMAC responds; this is max allowed value
M5197->X:$002000,24,S ; Assign variable to C.S. 1 % command reg.
M5197=P1*I10/100 ; Equivalent to &1%(P1)
```

See Also:

Time-Base Control (Synchronizing Turbo PMAC to External Events)
I-Variables I10, Isx93, Isx94, Isx95
On-line commands **%**, **H**
Memory map registers X:$002000, X:$002100, etc.

&

Function: Report port's currently addressed coordinate system.

Scope: Port specific

Syntax: **&**

This command causes Turbo PMAC to return the number of the coordinate system currently addressed for the communications port over which this command is sent. This is the coordinate system that will act on subsequent coordinate-system-specific commands sent over this port until a different coordinate system is addressed with an **&{constant}** command.

Other communications ports may be addressing different coordinate systems at the same time, as set by **&{constant}** commands sent over those ports. In addition, each background PLC program can individually modally address a coordinate system using the **ADDRESS** statement for subsequent **COMMAND** statements, and the hardware control panel on a Turbo PMAC(1) can separately select a coordinate system for its hardware inputs.

> ***Note:***
>
> In firmware versions 1.934 and older, all communications ports addressed the same coordinate system, so an **&{constant}** command sent over any port set the addressed coordinate system for all ports.

Example:

```
& ..... ; Ask Turbo PMAC which C.S. is addressed
4 ..... ; Turbo PMAC reports that C.S. 4 is addressed
```

See Also:

I-variable I2
On-line commands **#, #{constant},&{constant}**;
Program commands **ADDRESS, COMMAND**;

&{constant}

Function: Select port's addressed coordinate system.

Scope: Port specific

Syntax: **&{constant}**

where:

- **{constant}** is an integer from 1 to 16, representing the number of the coordinate system to be addressed on this port

This command makes the coordinate system specified by **{constant}** the addressed coordinate system for the communications port over which this command is sent. This is the coordinate system that will act on subsequent coordinate-system -specific commands sent over this port until a different coordinate system is addressed with another **&{constant}** command.

Other communications ports may be addressing different coordinate systems at the same time, as set by **&{constant}** commands sent over those ports. In addition, each background PLC program can individually modally address a coordinate system using the **ADDRESS** statement for subsequent **COMMAND** statements, and the hardware control panel on a Turbo PMAC(1) can separately select a coordinate system for its hardware inputs.

> ***Note:***
>
> In firmware versions 1.934 and older, all communications ports addressed the same coordinate system, so an **&{constant}** command sent over any port set the addressed coordinate system for all ports.

Example:

```
&1B4R.....      ; C.S.1 point to Beginning of Prog 4 and Run
Q .....         ; C.S.1 Quit running program
&3B6R.....      ; C.S.3 point to Beginning of Prog 5 and Run
A .....         ; C.S.3 Abort program
```

See Also:

I-variable I2

On-line commands **#**, **#{constant}**, **&**

Program commands **ADDRESS**, **COMMAND**

\

Function: Quick Stop in Lookahead / Feed Hold

Scope: Coordinate-system specific

Syntax: \

This command causes the Turbo PMAC to calculate and execute the quickest stop within the lookahead buffer for the addressed coordinate system that does not violate acceleration constraints for any motor within the coordinate system. Motion will continue to a controlled stop along the programmed path, but the stop will not necessarily be at a programmed point.

The \ quick-stop command is generally the best command to stop motion interactively within lookahead. Its function is much like that of a traditional feed-hold command, but unlike the regular **H** feed-hold command in Turbo PMAC, it is guaranteed to observe constraints.

Once stopped, several options are possible:

- Jog axes away with any of the jogging commands. The on-line jog commands can be used to jog any of the motors in the coordinate system away from the stopped point. However, before execution of the programmed path can be resumed, all motors must be returned to the original stopping point with the **J=** command.

- Start reverse execution along the path with the **<** command.
- Resume forward execution with the **>**, **R**, or **S** command.
- End program execution with the **A** command.

This same functionality can be obtained from within a Turbo PMAC program by setting Isx21 to 4, which executes more quickly than **CMD "&n\"**.

If the **** command is given to a coordinate system that is not currently executing moves within the lookahead buffer, Turbo PMAC will execute the **H** "feed-hold" command instead.

See Also:
I-variables Isx13, Isx20, Isx21
On-line commands **<**, **>**, **/**, **A**, **H**, **J=**, **R**, **S**

<

Function: Back up through Lookahead Buffer

Scope: Coordinate-system specific

Syntax: **<**

This command causes the Turbo PMAC to start reverse execution in the lookahead buffer for the addressed coordinate system. If the program is currently executing in the forward direction, it will be brought to a quick stop (the equivalent of the **** command) first.

Execution proceeds backward through points buffered in the lookahead buffer, observing velocity and acceleration constraints just as in the forward direction. This execution continues until one of the following occurs:

- Reverse execution reaches the "beginning" of the lookahead buffer – the oldest stored point still remaining in the lookahead buffer – and it comes to a controlled stop at this point, observing acceleration limits in decelerating to a stop.
- The **** "quick-stop" command is given, which causes Turbo PMAC to come to the quickest possible stop in the lookahead buffer.
- The **>** "resume-forward", **R** "run", or **S** "step" command is given, which causes Turbo PMAC to resume normal forward execution of the program, adding to the lookahead buffer as necessary.
- An error condition occurs, or a non-recoverable stopping command is given.

If any motor has been jogged away from the "quick-stop" point, and not returned with a **J=** command, Turbo PMAC will reject the **<** "back-up" command, reporting ERR017 if I6 is set to 1 or 3.

This same functionality can be obtained from within a Turbo PMAC program by setting Isx21 to 7, which executes more quickly than **CMD "&n<"**.

If the coordinate system is not currently in the middle of a lookahead sequence, Turbo PMAC will treat this command as an **H** "feed-hold" command.

See Also:
I-variables Isx13, Isx20, Isx21
On-line commands ****, **>**, **/**, **A**, **H**, **J=**, **R**, **S**

>

Function: Resume Forward Execution in Lookahead Buffer

Scope: Coordinate-system specific

Syntax: **>**

This command causes the Turbo PMAC to resume forward execution in the lookahead buffer for the addressed coordinate system. It is typically used to resume normal operation after a \ "quick-stop" command, or a **<** "back-up" command. If the program is currently executing in the backward direction, it will be brought to a quick stop (the equivalent of the **** command) first.

If previous forward execution had been in continuous mode (started with the **R** command), the **>** command will resume it in continuous mode. If previous forward execution had been in single-step mode (started with the **S** command), the **>** command will resume it in single-step mode.

The R and S commands can also be used to resume forward execution, but they may change the continuous/single-step mode.

Deceleration from a backward move (if any) and acceleration in the forward direction observe the Ixx17 acceleration limits.

If any motor has been jogged away from the "quick-stop" point, and not returned with a **J=** command, Turbo PMAC will reject the **>** "resume" command, reporting ERR017 if I6 is set to 1 or 3.

This same functionality can be obtained from within a Turbo PMAC program by setting Isx21 to 6, which executes more quickly than **CMD "&n>"**.

If the coordinate system is not currently in the middle of a lookahead sequence, Turbo PMAC will treat this command as an **R** "run" command.

See Also:
I-variables Isx13, Isx20, Isx21
On-line commands \, **<**, **/**, **A**, **H**, **J=**, **R**, **S**

/

Function: Halt Motion at End of Block

Scope: Coordinate-system specific

Syntax: **/**

This command causes PMAC to halt the execution of the motion program running in the currently addressed coordinate system at the end of the currently executing move, provided the coordinate system is in segmentation mode (Isx13 > 0). If the coordinate system is not in segmentation mode (Isx13 = 0), the **/** "end-block" command has the same effect as the **Q** or **S** command. It will halt execution at the end of the latest *calculated* move, which can be 1 or 2 moves past the currently *executing* move.

If the coordinate system is currently executing moves with the special lookahead function, motion will stop at the end of the move currently being *added* to the lookahead buffer. This is not necessarily the move that is currently executing from the lookahead buffer, and there can be a significant delay before motion is halted. Acceleration limits will be observed while ramping down to a stop at the programmed point.

Once halted at the end of the move, program execution can be resumed with the **R** "run" or **S** single-step command. In the meantime, the individual motors may be jogged way from this point, but they must all be returned to this point using the **J=** command before program execution may be resumed.

An attempt to resume program execution from a different point will result in an error (ERR017 reported if I6 = 1 or 3). If resumption of this program from this point is not desired, the **A** (abort) command should be issued before other programs are run.

See Also:
I-variables Isx13, Isx20, Isx21
On-line commands ****, **<**, **>**, **A**, **H**, **J=**, **R**, **S**

?

Function: Report motor status

Scope: Motor specific

Syntax: **?**

This command causes Turbo PMAC to report the motor status bits as an ASCII hexadecimal word. Turbo PMAC returns twelve characters, representing two status words. Each character represents four status bits. The first character represents Bits 20-23 of the first word; the second shows Bits 16-19; and so on, to the sixth character representing Bits 0-3. The seventh character represents Bits 20-23 of the second word; the twelfth character represents Bits 0-3.

If the Turbo PMAC is in "bootstrap mode" (suitable for the downloading of new firmware) instead of the normal operational mode, its response to this command will simply be `BOOTSTRAP PROM`.

The value of a bit is 1 when the condition is true; 0 when it is false. The meaning of the individual bits is:

First Word Returned (X:$0000B0, X:$000130, etc.):

First character returned:
Bit 23 ***Motor Activated*:** This bit is 1 when Ixx00 is 1 and the motor calculations are active; it is 0 when Ixx00 is 0 and motor calculations are deactivated.
Bit 22 ***Negative End Limit Set*:** This bit is 1 when motor actual position is less than the software negative position limit (Ixx14), or when the hardware limit on this end (+LIMn on Turbo PMAC(1) -- note!) has been tripped; it is 0 otherwise. If the motor is deactivated (bit 23 of the first motor status word set to zero) or killed (bit 19 of the first motor status word set to zero), this bit is not updated.
Bit 21 ***Positive End Limit Set*:** This bit is 1 when motor actual position is greater than the software positive position limit (Ixx13), or when the hardware limit on this end (-LIMn -- note!) has been tripped; it is 0 otherwise. If the motor is deactivated (bit 23 of the first motor status word set to zero) or killed (bit 14 of the second motor status word set to zero), this bit is not updated.
Bit 20 ***Extended Servo Algorithm Enabled*:** This bit is 1 when Iyy00/Iyy50 for the motor is set to 1 and the extended servo algorithm for the motor is selected. It is 0 when Iyy00/Iyy50 is 0 and the PID servo algorithm is selected.

Second character returned:
Bit 19 ***Amplifier Enabled*:** This bit is 1 when the outputs for this motor's amplifier are enabled, either in open loop or closed-loop mode (refer to Open-Loop Mode status bit to distinguish between the two cases). It is 0 when the outputs are disabled (killed).
Bit 18 ***Open Loop Mode*:** This bit is 1 when the servo loop for the motor is open, either with outputs enabled or disabled (killed). (Refer to Amplifier Enabled status bit to distinguish between the two cases.) It is 0 when the servo loop is closed (under position control, always with outputs enabled).

Bit 17 ***Move Timer Active*****:** This bit is 1 when the motor is executing any move with a predefined end-point and end-time. This includes any motion program move dwell or delay, any jog-to-position move, and the portion of a homing search move after the trigger has been found. It is 0 otherwise. It changes from 1 to 0 when execution of the *commanded* move finishes.
Bit 16 ***Integration Mode*****:** This bit is 1 when Ixx34 is 1 and the servo loop integrator is only active when desired velocity is zero. It is 0 when Ixx34 is 0 and the servo loop integrator is always active.

Third character returned:
Bit 15 ***Dwell in Progress*****:** This bit is 1 when the motor's coordinate system is executing a **DWELL** instruction. It is 0 otherwise.
Bit 14 ***Data Block Error*****:** This bit is 1 when move execution has been aborted because the data for the next move section was not ready in time. This is due to insufficient calculation time. It is 0 otherwise. It changes from 1 to 0 when another move sequence is started. This is related to the *Run Time Error* Coordinate System status bit.
Bit 13 ***Desired Velocity Zero*****:** This bit is 1 if the motor is in closed-loop control and the commanded velocity is zero (i.e. it is trying to hold position). It is zero either if the motor is in closed-loop mode with non-zero commanded velocity, or if it is in open-loop mode.
Bit 12 ***Abort Deceleration*****:** This bit is 1 if the motor is decelerating due to an Abort command, or due to hitting hardware or software position (overtravel) limits. It is 0 otherwise. It changes from 1 to 0 when the *commanded* deceleration to zero velocity finishes.

Fourth character returned:
Bit 11 ***Block Request*****:** This bit is 1 when the motor has just entered a new move section, and is requesting that the upcoming section be calculated. It is 0 otherwise. It is primarily for internal use.
Bit 10 ***Home Search in Progress*****:** This bit is set to 1 when the motor is in a move searching for a trigger: a homing search move, a jog-until trigger, or a motion program move-until-trigger. It becomes 1 as soon as the calculations for the move have started, and becomes zero again as soon as the trigger has been found, or if the move is stopped by some other means. This is *not* a good bit to observe to see if the full move is complete, because it will be 0 during the post-trigger portion of the move. Use the Home Complete and Desired Velocity Zero bits instead.
Bit 9 ***User-Written Phase Enable*****:** This bit is 1 when Ixx59 bit 1 for the motor is set to 1 and the motor executes the user-written phase routine instead of the normal phase routine. It is 0 when Ixx59 bit 1 is 0 and the motor executes the normal phase routine.
Bit 8 ***User-Written Servo Enable*****:** This bit is 1 when Ixx59 bit 0 for the motor is set to 1 and the motor executes the user-written servo routine instead of the normal servo routine. It is 0 when Ixx59 bit 0 is 0 and the motor executes the normal servo routine.

Fifth character returned:
Bit 7 ***Alternate Source/Destination*****:** This bit is 1 when Ixx01 bit 1 is 1 and an alternate source or destination for the motor algorithms is used. If Ixx01 bit 0 is 0, this means that the motor writes its command to an X-register instead of the standard Y-register. If Ixx01 bit 0 is 1, this means that the motor reads its commutation feedback from a Y-register instead of the standard X-register. This bit is 0 when Ixx01 bit 1 is 0, and the standard source or destination is used for the motor.
Bit 6 ***Phased Motor*****:** This bit is 1 when Ixx01 bit 0 is 1 and this motor is being commutated by Turbo PMAC; it is 0 when Ixx01 bit 0 is 0 and this motor is not being commutated by Turbo PMAC.
Bit 5 ***Following Offset Mode:*** This bit is 1 when Ixx06 bit 1 is 1 and position following is executed in "offset mode", in which the motor's programming reference position moves with the following. This bit is 0 when Ixx06 bit 1 is 0 and position following is executed in "normal mode", in which the motor's programming reference does not move with the following.

Bit 4 ***Following Enabled*****:** This bit is 1 when Ixx06 bit 0 is 1 and position following for this axis is enabled; it is 0 when Ixx06 bit 0is 0 and position following is disabled.
Sixth character returned:
Bit 3 ***Error Trigger*****:** This bit is 1 when Ixx97 bit 1 is set to 1 and the motor's triggered moves trigger on the warning following error limit being exceeded. Itis 0 when Ixx97 bit 1 is set to 0 and the motor's triggered moves trigger on a specified input flag state.
Bit 2 ***Software Position Capture*****:** This bit is 1 when Ixx97 bit 0 is set to 1 and the motor's triggered moves use a software-captured position as the reference for the post-trigger move. It is 0 when Ixx97 bit 0 is set to 0 and the motor's triggered moves use the hardware-captured counter position as the reference for the post-trigger move.
Bit 1 ***Alternate Command-Output Mode*****:** This bit is 1 when Ixx96 is set to 1 and the motor's commands are output in the alternate mode. If Ixx01 bit 0 is 1, this means that open-loop direct-microstepping commutation is performed instead of the normal closed-loop commutation. If Ixx01 bit 0 is 0, this means that the motor's non-commutated output is formatted as a sign-and-magnitude signal pair, instead of a single bipolar signal output. This bit is 0 when Ixx96 is set to 0 and the motor's commands are output in the standard mode.
Bit 0 ***Maximum Rapid Speed*****:** This bit is 1 when Ixx90 is set to 1 and the motor uses its Ixx16 maximum speed parameter for RAPID moves. It is 0 when Ixx90 is set to 0 and the motor uses its Ixx22 jog speed parameter for RAPID moves.

Second Word Returned (Y:$0000C0, Y:$000140, etc.):

Seventh character returned:
Bits 20-23 ***(C.S. - 1) Number*****:** These three bits together hold a value equal to the (Coordinate System number minus one) to which the motor is assigned. Bit 23 is the MSB, and bit 20 is the LSB. For instance, if the motor is assigned to an axis in C. S. 6, these bits would hold a value of 5: bit 23 = 0, bit 22 = 1, bit 21 = 0, and bit 20 = 1.

Eighth character returned:
Bits 16-19 ***Coordinate Definition*****:** These four bits tell what axis or axes this motor has been assigned to in an axis definition statement. The following values are currently used:

0: No definition
1: Assigned to A-axis
2: Assigned to B-axis
3: Assigned to C-axis
4: Assigned to UVW axes
7: Assigned to XYZ axes

Ninth Character Returned:
Bit 15 ***Assigned to C.S.*****:** This bit is 1 when the motor has been assigned to an axis in any coordinate system through an axis definition statement. It is 0 when the motor is not assigned to an axis in any coordinate system.
Bit 14 ***(Reserved for future use)***
Bit 13 ***Foreground In-Position*****:** This bit is 1 when the foreground in-position checking is enabled with I13=1 and when four conditions are satisfied: the loop is closed, the desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, DWELL, or DELAY), and the magnitude of the following error is smaller than Ixx28. It is 0 otherwise.
Bit 12 ***Stopped on Desired Position Limit*****:** This bit is 1 if the motor has stopped because the desired position has exceeded the software overtravel limit parameters (Ixx24 bit 15 must be 1 to enable this function). It is 0 otherwise.

Tenth Character Returned:

Bit 11 ***Stopped on Position Limit*****:** This bit is 1 if this motor has stopped because of either a software or a hardware position (overtravel) limit, even if the condition that caused the stop has gone away. It is 0 at all other times, even when into a limit but moving out of it.

Bit 10 ***Home Complete*****:** This bit, set to 0 on power-up or reset, becomes 1 when the homing move *successfully* locates the home trigger. Usually, at this point in time the motor is decelerating to a stop or moving to an offset from the trigger determined by Ixx26. If a second homing move is done, this bit is set to 0 at the beginning of the move, and only becomes 1 again if that homing move *successfully* locates the home trigger. Use the Desired Velocity Zero bit and/or the In Position bit to monitor for the end of motor motion.

Bit 9 ***Phasing Search/Read Active*****:** This bit is set to 1 if the phasing search move or phasing absolute position read is currently ongoing for the motor. It is set to 0 otherwise.

Bit 8 ***Phasing Reference Error*****:** This bit is set to 1 on power-up/reset for a PMAC-commutated (Ixx01 bit 0 = 1) synchronous motor. It is also set to 1 at the beginning of a phasing search move or phasing absolute position read for such a motor. It is set to 0 on the successful completion of a phasing search move or phasing absolute position read. If this bit is 1, the position/velocity servo loop cannot be closed for this motor.

This bit is set to 1 if the phasing search move for a Turbo PMAC-commutated motor has failed due to amplifier fault, overtravel limit, or lack of detected motion. It is set to 0 if the phasing search move did not fail by any of these conditions (not an absolute guarantee of a successful phasing search).

Eleventh Character Returned:

Bit 7 ***Trigger Move*****:** This bit is set to 1 at the beginning of a jog-until-trigger or motion program move-until-trigger. It is set to 0 at the end of the move if the trigger has been found, but remains at 1 if the move ends with no trigger found. This bit is useful to determine whether the move was successful in finding the trigger.

Bit 6 ***Integrated Fatal Following Error*****:** This bit is 1 if this motor has been disabled due to an integrated following error fault, as set by Ixx11 and Ixx63. The fatal following error bit (bit 2) will also be set in this case. Bit 6 is zero at all other times, becoming 0 again when the motor is re-enabled.

Bit 5 ***I^2T Amplifier Fault Error*****:** This bit is 1 if this motor has been disabled by an integrated current fault. The amplifier fault bit (bit 3) will also be set in this case. Bit 5 is 0 at all other times, becoming 0 again when the motor is re-enabled.

Bit 4 ***Backlash Direction Flag*****:** This bit is 1 if backlash has been activated in the negative direction. It is 0 otherwise.

Twelfth Character Returned:

Bit 3 ***Amplifier Fault Error*****:** This bit is 1 if this motor has been disabled because of an amplifier fault signal, *even if the amplifier fault signal has gone away*, or if this motor has been disabled due to an I^2T integrated current fault (in which case bit 5 is also set). It is 0 at all other times, becoming 0 again when the motor is re-enabled.

Bit 2 ***Fatal Following Error*****:** This bit is 1 if this motor has been disabled because it exceeded its fatal following error limit (Ixx11) or because it exceeded its integrated following error limit (Ixx63; in which case bit 6 is also set). It is 0 at all other times, becoming 0 again when the motor is re-enabled.

Bit 1 ***Warning Following Error*****:** This bit is 1 if the following error for the motor exceeds its warning following error limit (Ixx12). It stays at 1 if the motor is killed due to fatal following error. It is 0 at all other times, changing from 1 to 0 when the motor's following error reduces to under the limit, or if killed, is re-enabled.

Bit 0 ***In Position*****:** This bit is 1 when five conditions are satisfied: the loop is closed, the desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, **DWELL**, or **DELAY**), the magnitude of the following error is smaller than Ixx28.and the first four conditions have been satisfied for (Ixx88+1) consecutive scans.

Example:

```
#1?.....          ; Request status of Motor 1
81200001C401      ; PMAC responds with 12 hex digits representing 48 bits
.....             ; The following bits are true (all others are false)
.....             ; Word 1 Bit 23: Motor Activated
.....             ;   Bit 16: Integration Mode
.....             ;   Bit 13: Desired Velocity Zero
.....             ; Word 2 (Bits 20-23 all 0 – assigned to C.S.1)
.....             ;   (Bits 16-19 form 1 – assigned to A-axis)
.....             ;   Bit 15: Assigned to Coordinate System
.....             ;   Bit 14: Amplifier Enabled
.....             ;   Bit 10: Home Complete
.....             ;   Bit 0: In Position
```

See Also:

On-line commands **<CTRL-B>**, **??**, **???**

Memory-map registers X:$0000B0, X:$000130, etc., Y:$0000C0, Y:$000140, etc.;
Suggested M-Variable definitions Mxx30-Mxx45.

??

Function: Report the status words of the addressed coordinate system.

Scope: Coordinate-system specific

Syntax: **??**

This command causes Turbo PMAC to report status bits of the addressed coordinate system as an ASCII hexadecimal word. Turbo PMAC returns eighteen characters, representing three 24-bit status words. Each character represents four status bits. The first character represents bits 20-23 of the first word; the second shows bits 16-19; and so on, to the sixth character representing bits 0-3. The seventh character represents bits 20-23 of the second word; the twelfth character represents bits 0-3.

If the Turbo PMAC is in "bootstrap mode" (suitable for the downloading of new firmware) instead of the normal operational mode, its response to this command will simply be `BOOTSTRAP PROM`.

The value of a bit is 1 when the condition is true; 0 when it is false. The meanings of the individual bits are:

First Word Returned (X:$002040, X:$0020C0, etc.)

First character returned:

Bit 23 ***Z-Axis Used in Feedrate Calculations*****:** This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the **FRAX** command.

Bit 22 ***Z-Axis Incremental Mode*****:** This bit is 1 if this axis is in incremental mode -- moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode -- moves specified by end position, not distance. See the **INC** and **ABS** commands.

Bit 21 ***Y-Axis Used in Feedrate Calculations*****:** (See bit 23 description.)

Bit 20 ***Y-Axis Incremental Mode*****:** (See bit 22 description.)

Second character returned:
Bit 19 ***X-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 18 ***X-Axis Incremental Mode*:** (See bit 22 description.)
Bit 17 ***W-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 16 ***W-Axis Incremental Mode*:** (See bit 22 description.)

Third character returned:
Bit 15 ***V-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 14 ***V-Axis Incremental Mode*:** (See bit 22 description.)
Bit 13 ***U-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 12 ***U-Axis Incremental Mode*:** (See bit 22 description.)

Fourth character returned:
Bit 11 ***C-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 10 ***C-Axis Incremental Mode*:** (See bit 22 description.)
Bit 9 ***B-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 8 ***B-Axis Incremental Mode*:** (See bit 22 description.)

Fifth character returned:
Bit 7 ***A-Axis Used in Feedrate Calculations*:** (See bit 23 description.)
Bit 6 ***A-Axis Incremental Mode*:** (See bit 22 description.)
Bit 5 ***Radius Vector Incremental Mode*:** This bit is 1 if circle move radius vectors are specified incrementally (i.e. from the move start point to the arc center). It is 0 if circle move radius vectors are specified absolutely (i.e. from the XYZ origin to the arc center). See the **INC (R)** and **ABS (R)** commands.
Bit 4 ***Continuous Motion Request*:** This bit is 1 if the coordinate system has requested of it a continuous set of moves (e.g. with an **R** command). It is 0 if this is not the case (e.g. not running program, Isx92=1, or running under an **S** command).

Sixth character returned:
Bit 3 ***Move-Specified-by-Time Mode*:** This bit is 1 if programmed moves in this coordinate system are currently specified by time (TM or TA), and the move speed is derived. It is 0 if programmed moves in this coordinate system are currently specified by feedrate (speed; F) and the move time is derived.
Bit 2 ***Continuous Motion Mode*:** This bit is 1 if the coordinate system is in a sequence of moves that it is blending together without stops in between. It is 0 if it is not currently in such a sequence, for whatever reason.
Bit 1 ***Single-Step Mode*:** This bit is 1 if the motion program currently executing in this coordinate system has been told to "step" one move or block of moves, or if it has been given a Q (Quit) command. It is 0 if the motion program is executing a program by a R (run) command, or if it is not executing a motion program at all.
Bit 0 ***Running Program*:** This bit is 1 if the coordinate system is currently executing a motion program. It is 0 if the C.S. is not currently executing a motion program. Note that it becomes 0 as soon as it has *calculated* the last move and reached the final **RETURN** statement in the program, even if the motors are still *executing* the last move or two that have been calculated. Compare to the motor *Running Program* status bit.

Second Word Returned (Y:$00203F, Y:$0020BF, etc.)

Seventh character returned:
Bit 23 ***Lookahead in Progress*:** This bit is 1 when the coordinate system is actively computing and/or executing a move sequence using the multi-block lookahead function. It is 0 otherwise.

Bit 22 ***Run-Time Error*****:** This bit is 1 when the coordinate system has stopped a motion program due to an error encountered while executing the program (e.g. jump to non-existent label, insufficient calculation time, etc.) It is 0 otherwise. The run-time error code word (Y:$002x14) shows the cause of a run-time error.
Bit 21 ***Move In Stack*****:** (For internal use)
Bit 20 ***Amplifier Fault Error*****:** This bit is 1 when any motor in the coordinate system has been killed due to receiving an amplifier fault signal. It is 0 at other times, changing from 1 to 0 when the offending motor is re-enabled.

Eighth character returned:
Bit 19 ***Fatal Following Error*****:** This bit is 1 when any motor in the coordinate system has been killed due to exceeding its fatal following error limit (Ixx11). It is 0 at other times. The change from 1 to 0 occurs when the offending motor is re-enabled.
Bit 18 ***Warning Following Error*****:** This bit is 1 when any motor in the coordinate system has exceeded its warning following error limit (Ixx12). It stays at 1 if a motor has been killed due to fatal following error limit. It is 0 at all other times. The change from 1 to 0 occurs when the offending motor's following error is reduced to under the limit, or if killed on fatal following error as well, when it is re-enabled.
Bit 17 ***In Position*****:** This bit is 1 when *all* motors in the coordinate system are "in position". Five conditions must apply for all of these motors for this to be true:, the loops must be closed, the desired velocity must be zero for all motors, the coordinate system cannot be in any timed move (even zero distance) or DWELL, all motors must have a following error smaller than their respective Ixx28 in-position bands, and the above conditions must have been satisfied for (Ixx88+1) consecutive scans.
Bit 16 ***Rotary Buffer Request*****:** This bit is 1 when a rotary buffer exists for the coordinate system and enough program lines have been sent to it so that the buffer contains at least I17 lines ahead of what has been calculated. Once this bit has been set to 1 it will not be set to 0 until there are less than I16 program lines ahead of what has been calculated. The 'PR' command may be used to find the current number of program lines ahead of what has been calculated.

Ninth character returned:
Bit 15 ***Delayed Calculation Flag*****: (for internal use)**
Bit 14 ***End of Block Stop*****:** This bit is 1 when a motion program running in the currently addressed Coordinate System is stopped using the ' / ' command from a segmented move (Linear or Circular mode with Isx13 > 0).
Bit 13 ***Synchronous M-variable One-Shot*****:** (for internal use)
Bit 12 ***Dwell Move Buffered*****:** (for internal use)

Tenth character returned:
Bit 11 ***Cutter Comp Outside Corner*****:** This bit is 1 when the coordinate system is executing an added outside corner move with cutter compensation on. It is 0 otherwise.
Bit 10 ***Cutter Comp Move Stop Request*****:** This bit is 1 when the coordinate system is executing moves with cutter compensation enabled, and has been asked to stop move execution. This is primarily for internal use.
Bit 9 ***Cutter Comp Move Buffered*****:** This bit is 1 when the coordinate system is executing moves with cutter compensation enabled, and the next move has been calculated and buffered. This is primarily for internal use.
Bit 8 ***Pre-jog Move Flag*****:** This bit is 1 when any motor in the coordinate system is executing a jog move to "pre-jog" position (**J=** command). It is 0 otherwise.

Eleventh character returned:
Bit 7 ***Segmented Move in Progress*****:** This bit is 1 when the coordinate system is executing motion program moves in segmentation mode (Isx13>0). It is 0 otherwise. This is primarily for internal use.

Bit 6 ***Segmented Move Acceleration*****:** This bit is 1 when the coordinate system is executing motion program moves in segmentation mode (Isx13>0) and accelerating from a stop. It is 0 otherwise. This is primarily for internal use.
Bit 5 ***Segmented Move Stop Request*****:** This bit is 1 when the coordinate system is executing motion program move in segmentation mode (Isx13>0) and it is decelerating to a stop. It is 0 otherwise. This is primarily for internal use.
Bit 4 ***PVT/SPLINE Move Mode*****:** This bit is 1 if this coordinate system is in either PVT move mode or SPLINE move mode. (If bit 0 of this word is 0, this means PVT mode; if bit 0 is 1, this means SPLINE mode.) This bit is 0 if the coordinate system is in a different move mode (LINEAR, CIRCLE, or RAPID). See the table below.

Twelfth character returned:
Bit 3 ***2D Cutter Comp Left/3D Cutter Comp On***: With bit 2 equal to 1, this bit is 1 if the coordinate system has 2D cutter compensation on, compensating to the left when looking in the direction of motion. It is 0 if 2D compensation is to the right. With bit 2 equal to 0, this bit is 1 if the coordinate system has 3D cutter compensation on. It is 0 if no cutter compensation is on.
Bit 2 ***2D Cutter Comp On***: This bit is 1 if the coordinate system has 2D cutter compensation on. It is 0 if 2D cutter compensation is off (but 3D cutter compensation may be on if bit 3 is 1).
Bit 1 ***CCW Circle\Rapid Mode*****:** When bit 0 is 1 and bit 4 is 0, this bit is set to 0 if the coordinate system is in CIRCLE1 (clockwise arc) move mode and 1 if the coordinate system is in CIRCLE2 (counterclockwise arc) move mode. If both bits 0 and 4 are 0, this bit is set to 1 if the coordinate system is in RAPID move mode. Otherwise this bit is 0. See the table below.
Bit 0 ***CIRCLE/SPLINE Move Mode*****:** This bit is 1 if the coordinate system is in either CIRCLE or SPLINE move mode. (If bit 4 of this word is 0, this means CIRCLE mode; if bit 4 is 1, this means SPLINE mode.) This bit is 0 if the coordinate system is in a different move mode (LINEAR, PVT, or RAPID.). See the table below.

The states of bits 4, 1, and 0 in the different move modes are summarized in the following table:

Mode	Bit 4	Bit 1	Bit 0
LINEAR	0	0	0
RAPID	0	1	0
SPLINE	1	0	1
CIRCLE1	0	0	1
CIRCLE2	0	1	1
PVT	1	1	0

Third Word Returned (Y:$002040, Y:$0020C0, etc.)

Thirteenth character returned:
Bit 23 ***Lookahead Buffer Wrap*****:** This bit is 1 when the lookahead buffer for the coordinate system is active and has "wrapped around" since the beginning of the current continuous motion sequence, meaning that retrace back to the beginning of the sequence is no longer possible. It is 0 otherwise.
Bit 22 ***Lookahead Lookback Active:*** (For internal use)
Bit 21 ***Lookahead Buffer End:*** (For internal use)
Bit 20 ***Lookahead Synchronous M-variable:*** (For internal use)

Fourteenth character returned:
Bit 19 ***Lookahead Synchronous M-variable Overflow:*** This bit is 1 if the program has attempted to put more synchronous M-variable assignments into the lookahead buffer than the buffer has room for. If this bit is set, one or more synchronous M-variable assignments have failed to execute or will fail to execute.

Bit 18 ***Lookahead Buffer Direction:*** This bit is 1 if the lookahead buffer is executing in the reverse direction, or has executed a quick stop from the reverse direction. It is 0 if the lookahead buffer is executing in the forward direction, has executed a quick stop for the forward direction, or is not executing.
Bit 17 ***Lookahead Buffer Stop:*** This bit is 1 if the lookahead buffer execution is stopping due to a quick-stop command or request. It is 0 otherwise.
Bit 16 ***Lookahead Buffer Change:*** This bit is 1 if the lookahead buffer is currently changing state between forward and reverse direction, or between executing and stopped. It is 0 otherwise.

Fifteenth character returned:
Bit 15 ***Lookahead Buffer Last Segment:*** This bit is 1 if the lookahead buffer is currently executing the last segment before the end of a sequence. It is 0 otherwise.
Bit 14 ***Lookahead Buffer Recalculate:*** This bit is 1 if the lookahead buffer is recalculating segments already in the buffer due to a change in the state of the buffer. It is 0 otherwise.
Bit 13 ***Lookahead Buffer Flush:*** This bit is 1 if the lookahead buffer is executing segments but not adding any new segments. It is 0 otherwise.
Bit 12 ***Lookahead Buffer Last Move:*** This bit is 1 if the last programmed move in the buffer has reached speed. It is 0 otherwise.

Sixteenth character returned:
(Bits 8 – 11 form variable Isx21.)
Bit 11 ***Lookahead Buffer Single-Segment Request*:** This bit can be set to 1 by the user as part of a request to change the state of the lookahead buffer. It should be set to 1 to request the buffer to move only a single segment from a stopped state (in either direction). It should be set to 0 otherwise. Turbo PMAC leaves this bit in the state of the last request, even after the request has been processed.
Bit 10 ***Lookahead Buffer Change Request:*** This bit can be set to 1 by the user to request a change in the state of the lookahead buffer. It remains at 1 until the Turbo PMAC processes the change, at which time Turbo PMAC changes it to 0.
Bit 9 ***Lookahead Buffer Movement Request:*** This bit can be set by the user as part of a request to change the state of the lookahead buffer. It should be set to 1 to request the buffer to operate (in either the forward or reverse direction); it should be set to 0 to request the buffer to execute a quick stop. Turbo PMAC leaves this bit in the state of the last request, even after the request has been processed.
Bit 8 ***Lookahead Buffer Direction Request:*** This bit can be set by the user as part of a request to change the state of the lookahead buffer. It should be set to 1 to request operation in the reverse direction; it should be set to 0 to request operation in the forward direction. Its state does not matter in a request to execute a quick stop. Turbo PMAC leaves this bit in the state of the last request, even after the request has been processed.

Seventeenth character returned:
Bits 4 – 7 (Reserved for future use)

Eighteenth character returned:
Bit 3 ***Radius Error:*** This bit is 1 when a motion program has been stopped because it was asked to do an arc move whose distance was more than twice the radius (by an amount greater than Ixx96).
Bit 2 ***Program Resume Error:*** This bit is 1 when the user has tried to resume program operation after a feed-hold or quick-stop, but one or more of the motors in the coordinate system are not at the location of the feed-hold or quick-stop. It is 0 otherwise.
Bit 1 ***Desired Position Limit Stop:*** This bit is 1 if the motion program in the coordinate system has stopped due to the desired position of a motor exceeding a limit.

Bit 0 ***In-Program PMATCH:*** This bit is 1 if Turbo PMAC is automatically executing the PMATCH function, as at the end of a move-until-trigger. It is 0 otherwise. This bit is primarily for internal use.

Example:

```
??.....          ; Request coordinate system status words
A8002A020010000000
.....            ; Turbo PMAC responds; the following bits are true:
.....            ; Word 1 Bit 23: Z-axis used in feedrate calcs
.....            ;   Bit 21: Y-axis used in feedrate calcs
.....            ;   Bit 19: X-axis used in feedrate calcs
.....            ;   Bit 5: Radius vector incremental mode
.....            ;   Bit 3: Move specified by time
.....            ;   Bit 1: Single-step mode
.....            ; Word 2 Bit 17: In-position
.....            ;   Bit 4: PVT/Spline mode
.....            ; Word 3 no bits set – no lookahead active
```

See Also:

On-line commands **<CONTROL-C>**, **?**, **???**

Memory-map registers X/Y:$002040, X/Y:$0020C0, etc., Y:$00203F, Y:$0020BF, etc.; Suggested M-variable definitions Msx80-Msx90.

???

Function: Report global status words.

Scope: Global

Syntax: **???**

This command causes Turbo PMAC to return the global status bits in ASCII hexadecimal form. Turbo PMAC returns twelve characters, representing two status words. Each character represents four status bits. The first character represents Bits 20-23 of the first word, the second shows Bits 16-19; and so on, to the sixth character representing Bits 0-3. The seventh character represents Bits 20-23 of the second word; the twelfth character represents Bits 0-3 of the second word.

If the Turbo PMAC is in "bootstrap mode" (suitable for the downloading of new firmware) instead of the normal operational mode, its response to this command will simply be `BOOTSTRAP PROM`.

A bit has a value of 1 when the condition is true; 0 when false. The meaning of the individual status bits is:

First Word Returned (X:$000006):

First character returned:

Bit 23 (Reserved for future use)

Bit 22 ***Real-Time Interrupt Re-entry*:** This bit is 1 if a real-time interrupt task has taken long enough so that it was still executing when the next real-time interrupt came (I8+1 servo cycles later). It stays at 1 until the card is reset, or until this bit is manually changed to 0. If motion program calculations cause this it is not a serious problem. If PLC 0 causes this (no motion programs running) it could be serious.

Bit 21 ***CPU Type Bit 1***: This bit is 1 if the Turbo PMAC has an Option 5Ex DSP56311 or an Option 5Fx DSP56321 processor. It is 0 if it has an Option 5Cx DSP56303 or an Option 5Dx DSP56309 processor. In both cases, bit 21 in the second word returned (Y:$000006) distinguishes between processor types.

Bit 20 ***Servo Error*:** This bit is 1 if Turbo PMAC could not properly complete its servo routines. This is a serious error condition. It is 0 if the servo operations have been completing properly.

Second character returned:
Bit 19 ***Data Gathering Function On*****:** This bit is 1 when the data gathering function is active; it is 0 when the function is not active.
Bit 18 (Reserved for future use)
Bit 17 ***Data Gather to Start on Trigger*****:** This bit is 1 when the data gathering function is set up to start on the rising edge of Machine Input 2. It is 0 otherwise. It changes from 1 to 0 as soon as the gathering function actually starts.
Bit 16 ***Servo Request*****:** (Internal use).

Third character returned:
Bit 15 ***Watchdog Timer*****:** (Internal use)
Bit 14 ***Leadscrew Compensation On*****:** This bit is 1 if leadscrew compensation is currently active in Turbo PMAC. It is 0 if the compensation is not active.
Bit 13 ***Any Memory Checksum Error*****:** This bit is 1 if a checksum error has been detected for either the Turbo PMAC firmware or the user program buffer space. Bit 12 of this word distinguishes between the two cases.
Bit 12 ***PROM Checksum Active*****:** This bit is 1 if Turbo PMAC is currently evaluating a firmware checksum (Bit 13 = 0), or has found a firmware checksum error (Bit 13 = 1). It is 0 if Turbo PMAC is evaluating a user program checksum (Bit 13 = 0), or has found a user program checksum error (Bit 13 = 1).

Fourth character returned:
Bit 11 ***DPRAM Error***: This bit is 1 if Turbo PMAC detected an error in its automatic DPRAM check function at power-up/reset due to missing or defective DPRAM. It is 0 otherwise.
Bit 10 ***Flash Error*****:** This bit is 1 if Turbo PMAC detected a checksum error in reading saved data from the flash memory on board reset. It is 0 otherwise.
Bit 9 ***Real-Time Interrupt Warning*****:** This bit is 1 if a real-time interrupt task (motion program or PLC 0) has taken more than one interrupt period – a *possible* sign of CPU loading problems. It is 0 otherwise.
Bit 8 ***Illegal L-Variable Definition***: This bit is 1 if a compiled PLC has failed because it used an L-variable pointer that accessed an illegal M-variable definition. It is 0 otherwise.

Fifth character returned:
Bit 7 ***Configuration Error*****:** (For internal use)
Bit 6 ***TWS Variable Parity Error*****:** This bit is 1 if the most recent TWS-format M-variable read or write operation with a device supporting parity had a parity error; it is 0 if the operation with such a device had no parity error. The bit status is indeterminate if the operation was with a device that does not support parity.
Bit 5 ***MACRO Auxiliary Communications Error*****:** This bit is 1 if the most recent MACRO auxiliary read or write command has failed. It is set to 0 at the beginning of each MACRO auxiliary read or write command.
Bit 4 ***MACRO Ring Check Error*****:** This bit is 1 if the MACRO ring check function is enabled (I80 > 0) and Turbo PMAC has either detected at least I81 ring communication errors in an I80-servo-cycle period, or has failed to detect the receipt of I82 ring sync packets.

Sixth character returned:
Bit 3 ***Phase Clock Missing*** This bit is set to 1 if the CPU received no hardware-generated phase clock from a source external to it (Servo IC, MACRO IC, or through serial port). If this bit is set, no motor may be enabled (starting in V1.940). This bit is 0 otherwise.
Bit 2 (Reserved for future use)
Bit 1 ***All Cards Addressed*****:** This bit is set to 1 if all cards on a serial daisychain have been addressed simultaneously with the **@@** command. It is 0 otherwise.
Bit 0 ***This Card Addressed*****:** This bit is set to 1 if this card is on a serial daisychain and has been addressed with the **@n** command. It is 0 otherwise.

Second Word Returned (Y:$000006)

Seventh character returned:

Bit 23 ***Turbo Ultralite*****:** This bit is 1 if Turbo PMAC has detected that it is an "Ultralite" PMAC2 with no Servo ICs on board. It is 0 if Turbo PMAC has detected that it has Servo ICs on board.

Bit 22 ***Turbo VME*****:** This bit is 1 if Turbo PMAC has detected that it has a VME bus interface on board. It is 0 otherwise.

Bit 21 ***CPU Type Bit 0***: This bit is 1 if the Turbo PMAC has an Option 5Dx DSP56309 or an Option 5Fx DSP56321 processor. It is 0 if it has an Option 5Cx DSP56303 or an Option 5Dx DSP56311 processor. In both cases, bit 21 in the first word returned (X:$000006) distinguishes between processor types.

Bit 20 ***Binary Rotary Buffers Open*****:** This bit is 1 if the rotary motion program buffers on Turbo PMAC are open for binary-format entry through the DPRAM. It is 0 otherwise.

Eighth character returned:

Bit 19 ***Motion Buffer Open*****:** This bit is 1 if any motion program buffer (PROG or ROT) is open for entry. It is 0 if none of these buffers is open.

Bit 18 ***ASCII Rotary Buffer Open*****:** This bit is 1 if the rotary motion program buffers on Turbo PMAC are open for ASCII-format entry. It is 0 otherwise.

Bit 17 ***PLC Buffer Open*****:** This bit is 1 if a PLC program buffer is open for entry. It is 0 if none of these buffers is open.

Bit 16 ***UMAC System*****:** This bit is 1 if the Turbo PMAC is a 3U Turbo system (UMAC or Stack). It is 0 otherwise.

Ninth character returned:

Bits 14-15 ***Kinematics Active*****:** (For internal use)

Bit 13 ***Ring-Master-to-Master Communications*****:** (For internal use)

Bit 12 ***Master-to-Ring-Master Communications*****:** (For internal use)

Tenth character returned:

Bit 11 ***Fixed Buffer Full*****:** This bit is 1 when no fixed motion (PROG) or PLC buffers are open, or when one is open but there are less than I18 words available. It is 0 when one of these buffers is open and there are more than I18 words available.

Bits 8-10 (For Internal use)

Eleventh and twelfth characters returned:

Bits 0-7 (Reserved for future use)

Example:

```
???.....                ; Ask Turbo PMAC for global status words
003000400000            ; Turbo PMAC returns the global status words
.....                   ; 1st word bit 13 (Any checksum error) is true;
.....                   ; 1st word bit 12 (PROM checksum error) is true;
.....                   ; 2nd word bit 23 (for internal use) is true;
.....                   ; All other bits are false
```

See Also:

On-line commands **?**, **??**, **<CTRL-G>**

Memory-map registers X:$000006, Y:$000006.

A

Function: Abort all programs and moves in the currently addressed coordinate system.

Scope: Coordinate-system specific

Syntax: **A**

This command causes all axes defined in the current coordinate system to begin immediately to decelerate to a stop, aborting the currently running motion program (if any). It also brings any disabled (killed) or open loop motors (defined in the current coordinate system) to an enabled zero-velocity closed-loop state.

If moving, each motor will decelerate its commanded profile at a rate defined by its own motor I-variable Ixx15. If there is significant following error when the **A** command is issued, it may take a long time for the actual motion to stop.

Although the command trajectory is brought to a stop at a definite rate, the actual position will continue to "catch up" to the commanded position for a longer time.

Note that a multi-axis system may not stay on its programmed path during this deceleration.

An **A** (abort) stop to a program is not meant to be recovered from gracefully, because the axes will in general not stop at a programmed point. An on-line **J=** command may be issued to each motor to cause it to move to the end point that was programmed when the abort occurred. Then the program(s) can be resumed with an **R** (run) command.

To stop a motion sequence in a manner that can be recovered from easily, use instead the Quit (**Q** or **<CTRL-Q>**) or the Hold (**H** or **<CTRL-O>**) command.

Example:

```
B1R.....          ; Start Motion Program 1
A .....           ; Abort the program
#1J=#2J=.....     ; Jog motors to original move-end position
R .....           ; Resume program with next move
```

See Also:

Stop Commands (Making Your Application Safe)
Control-Panel Port STOP/ Input (Connecting Turbo PMAC to the Machine)
I-variable Ixx15
On-line commands **<CONTROL-A>**, **H**, **J/**, **K**, **Q**
JPAN connector pin 10

ABR[{constant}]

Function: Abort currently running motion program and start another

Scope: Coordinate-system specific

Syntax: **ABR[{constant}]**

where:

- **{constant}** is a numerical value whose integer part specifies the number of the program to be run, and whose fractional part (if any) specifies the line label of that program where execution is to begin.

The **ABR** command permits a very quick end to the execution of one motion program and starting (or restarting) of another (or the same) motion program. It performs the following tasks for the addressed coordinate system, all in a unitary command:

- It immediately stops execution of the currently running motion program in the addressed coordinate system.

- It brings the commanded trajectories of all motors in the coordinate system to stop at the rate set by Ixx15 for each motor.
- It points the coordinate system's program counter to a specific location in that program or another program. If stopping and resuming the rotary motion program buffer, it immediately clears the rotary buffer of unexecuted lines.
- As soon as the commanded trajectories of all motors in the coordinate system have stopped, it will start execution of the newly addressed program (which could be the same program.

It is essentially a combination of the existing **A** (abort), **B** (point to beginning of program), and **R** (run) commands. By combining these into a single command, all three actions are executed in a single command cycle, speeding the transition.

If the **ABR** command is given without a numerical argument, Turbo PMAC will restart the presently executing program at the top. If an **ABR0** command is given, Turbo PMAC will end execution of the currently executing program – if it is currently executing the rotary program buffer, clear the rotary buffer – and point to the top of the rotary program buffer.

If an **ABR{constant}** command is given with a non-zero constant value, Turbo PMAC will start execution of the program number specified by the integer part of the constant (valid values 1 – 32,767) and at the numeric line label whose value is equal to the fractional part times 100,000 (10^5). If no fractional part is specified, execution will start at the top of this program.

Examples:

```
ABR0          ; Stop execution of rotary buffer, clear, and restart at top
ABR20         ; Stop execution and start at top of program 20
ABR44.37      ; Stop execution and start at N37000 of program 44
```

ABS

Function: Select absolute position mode for axes in addressed coordinate system.

Scope: Coordinate-system specific

Syntax:
```
ABS
ABS ({axis}[,{axis}...])
```

where:

- **{axis}** is a letter (X, Y, Z, A, B, C, U, V, W) representing the axis to be specified, or the character R to specify radial vector mode

No spaces are permitted in this command.

This command, without any arguments, causes all subsequent positions for all axes in the coordinate system in motion commands to be treated as absolute positions (this is the default condition). An **ABS** command with arguments causes the specified axes to be in absolute mode, and all others to remain unchanged.

If R is specified as one of the 'axes', the I, J, and K terms of the circular move radius vector specification will be specified in absolute form (i.e. as a vector from the origin, not from the move start point). An **ABS** command without any arguments does not affect this vector specification. The default radial vector specification is incremental.

If a motion program buffer is open when this command is sent to Turbo PMAC, the command will be entered into the buffer for later execution.

Example:

```
ABS(X,Y).....  ; X & Y made absolute -- other axes and radial vector left unchanged
ABS..........  ; All axes made absolute -- radial vector left unchanged
ABS(R).......  ; Radial vector made absolute -- all axes left unchanged
```

See Also:
Circular Moves (Writing a Motion Program)
On-line command **INC**
Program commands **ABS**, **INC**

{axis}={constant}

Function: Re-define the specified axis position.

Scope: Coordinate-system specific

Syntax: **{axis}={constant}**

where:

- **{axis}** is a letter from the set (X, Y, Z, U, V, W, A, B, C) specifying the axis whose present position is to be re-named;
- **{constant}** is a floating-point value representing the new "name" value for the axis' present position

This command re-defines the current axis position to be the value specified in **{constant}**, in user units (as defined by the scale factor in the axis definition). It can be used to relocate the origin of the coordinate system. This does not cause the specified axis to move; it simply assigns a new value to the position..

Internally, a position bias register is written to which creates this new position offset. **PSET** is the equivalent motion program command.

Example:
X=0 .....; Call axis X's current position zero
Z=5000 .....; Re-define axis Z's position as 5000

See Also:
Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line command **Z**
Program commands **PSET**, **ADIS**, **IDIS**.

B{constant}

Function: Point the addressed coordinate system to a motion program.

Scope: Coordinate-system specific

Syntax: **B{constant}**

where:
{constant} is a floating point value from 0.0 to 32767.99999 representing the program and location to point the coordinate system to; with the integer part representing the program number, and the fractional part multiplied by 100,000 representing the line label (zero fractional part means the top of the program).

This command causes Turbo PMAC to set the program counter of the addressed coordinate system to the specified motion program and location. It is usually used to set the program counter to the *B*eginning of a motion program. The next **R** or **S** command will start execution at this point.

If **{constant}** is an integer, the program counter will point to the beginning of the program whose number matches **{constant}**. Fixed motion program buffers (PROG) can have numbers from 1 to 32,767. The rotary motion program carries program number 0 for the purpose of this command.

If **{constant}** is not an integer, the fractional part of the number represents the line label (**N** or **O**) in the program to which to point. The fractional value multiplied by 100,000 determines the number of the line label to which to point (it fills the fraction to 5 decimal places with zeros).

> ***Note:***
>
> If a motion program buffer (including ROTARY) is open when this command is sent to Turbo PMAC, the command is entered into the buffer for later execution, to be interpreted as a B-axis move command.

Example:

```
B7 ..... ;points to the top of PROG 7
B0 ..... ;points to the top of the rotary buffer
B12.6 ..... ;points to label N60000 of PROG 12
B3.025R ..... ;points to label N2500 of PROG 3 and runs
```

See Also:

On-line commands **DEFINE ROT**, **R**, **S**

Program commands **B{data}**, **N{constant}**, **O{constant}**.

CHECKSUM

Function: Report the firmware checksum value.

Scope: Global

Syntax: **CHECKSUM**
CHKS

This command causes Turbo PMAC to report the reference checksum value of the firmware revision that it is using. The value is reported as a hexadecimal ASCII string. This value was computed during the compilation of the firmware. It is mainly used for troubleshooting purposes.

The comparative checksum value that Turbo PMAC is continually computing by scanning the firmware in active memory is stored in X:$001080. As long as there is no checksum error, this comparative value is continually changing as PMAC continues its calculations. However, if during any pass of the checksum calculations, if the final comparative checksum value does not agree with the reference value, the calculations stop, and the final erroneous value is held in X:$001080.

Example:

```
CHECKSUM        ; Request firmware reference checksum value
9FA263          ; Turbo PMAC returns hex value
```

See Also:

On-line commands **DATE**, **VERSION**

CID

Function: Report card ID or part number.

Scope: Global

Syntax: **CID**

This command causes Turbo PMAC to return the card's part number. This can be used to confirm exactly which type of Turbo PMAC is being used.

The currently existing types of Turbo PMAC and the values they return for **CID** are:

- Turbo PMAC-PC: 602191
- Turbo PMAC-VME: 602199
- Turbo PMAC2-PC: 602404
- Turbo PMAC2-VME: 602413

- Turbo PMAC2-PC Ultralite: 603182
- Turbo PMA2-PCI: 603367
- 3U Turbo PMAC2: 603382

See Also:
I-Variable I4909
On-line commands **CPU**, **TYPE**, **VERSION**, **VID**

CLEAR

Function: Erase currently opened buffer.

Scope: Port specific

Syntax: **CLEAR**
CLR

This command empties the program buffer (PROGRAM, PLC, ROTARY) that is currently opened on the port over which the command is given. If used to empty an open rotary motion program buffer, it only affects the buffer for the addressed coordinate system on that port.

If there is no open program buffer, or if the program buffer that is open was opened from another communications port, the **CLEAR** command will be rejected with an error, reporting ERR007 if I6=1 or 3.

Note:

Prior to V1.936 firmware, an open buffer could be accessed from any port, and the **CLEAR** command could be used on one port to empty a buffer that had been opened on another port. Starting in V1.936, a **CLEAR** command could only be used to empty a buffer opened from the same port.

Typically, as you create a buffer file in your host computer, you will start with the **OPEN {buffer}** and **CLEAR** commands (even though these lines are technically not part of the buffer), and follow with your actual contents. This will allow you to easily edit buffers from your host and repeatedly download the buffers, erasing the old buffer's contents in the process.

Example:

```
OPEN PROG 1..                    ; Open motion program buffer 1
CLEAR.....                       ; Clear out this buffer
F1000.....                       ; Program really starts here!
X2500.....                       ;...and ends on this line!
CLOSE.....                       ; This closes the program buffer
OPEN PLC 3 CLEAR CLOSE           ; This erases PLC 3
```

See Also:
Program Buffers (Talking to Turbo PMAC)
On-line commands **OPEN**, **CLOSE**, **DELETE**.

CLEAR ALL

Function: Erase all fixed motion, kinematic, and uncompiled PLC programs

Scope: Global

Syntax: **CLEAR ALL**
CLR ALL

This command causes Turbo PMAC to erase all fixed (not rotary) motion program buffers (**PROGRAM**), all forward-kinematic program buffers (**FORWARD**), all inverse-kinematic program buffers (**INVERSE**), and all uncompiled PLC program buffers (**PLC**) in the Turbo PMAC buffer space.

This command does not affect rotary motion program buffers (**ROTARY**), compiled PLC program buffers (**PLCC**), or any non-program buffers, such as compensation tables and lookahead buffers.

See Also:
On-line commands **CLEAR**, **CLEAR ALL PLCS**, **OPEN**, **DELETE ALL**, **DELETE ALL TEMP**

CLEAR ALL PLCS

Function: Erase all uncompiled PLC programs

Scope: Global

Syntax: **CLEAR ALL PLCS**
CLR ALL PLCS

This command causes Turbo PMAC to erase all uncompiled PLC program buffers (**PLC**) in the Turbo PMAC buffer space.

This command does not affect fixed motion program buffers (**PROGRAM**), forward-kinematic program buffers (**FORWARD**), inverse-kinematic program buffers (**INVERSE**), rotary motion program buffers (**ROTARY**), compiled PLC program buffers (**PLCC**), or any non-program buffers, such as compensation tables and lookahead buffers.

See Also:
On-line commands **CLEAR**, **CLEAR ALL**, **OPEN**, **DELETE ALL**, **DELETE ALL TEMP**

CLOSE

Function: Close the currently opened buffer.

Scope: Global

Syntax: **CLOSE**
CLS

This command closes the program buffer (PROGRAM, PLC, ROTARY, BINARY ROTARY) that is currently opened on the port over which the command is given. When Turbo PMAC receives a **CLOSE** command, it automatically appends a **RETURN** statement to the end of the open program buffer. If used to close open rotary motion program buffer(s), it closes the rotary program buffers for all coordinate systems simultaneously.

If there is no open program buffer, or if the program buffer that is open was opened from another communications port, the **CLOSE** command will be accepted, but no action will occur.

Note:

Prior to V1.936 firmware, an open buffer could be accessed from any port, and the **CLOSE** command could be used on one port to close a buffer that had been opened on another port. Starting in V1.936, if a **CLOSE** command could only be used to close a buffer opened from the same port.

The **CLOSE** command should be used immediately after the entry of a motion, PLC, rotary, etc. buffer. If the buffer is left open, subsequent statements that are intended as on-line commands (e.g. **P1=0**) will get entered into the buffer instead. It is good practice to have **CLOSE** at the beginning and end of any file to be downloaded to Turbo PMAC.

If the program buffer closed by the **CLOSE** command is improperly structured, structured (e.g. no **ENDIF** or **ENDWHILE** to match an **IF** or **WHILE**), Turbo PMAC will report an error to the **CLOSE** command, returning ERR009 if I6 is 1 or 3. However, the buffer will still be closed.

Example:

```
CLOSE.............      ; This makes sure all buffers are closed
OPEN PROG 1..      ; Open motion program buffer 1
CLEAR.............      ; Clear out this buffer
F1000.............      ; Program actually starts here!...
X2500.......      ;...and ends on this line!
CLOSE ............      ; This closes the program buffer
LIST PROG 1..      ; Request listing of closed program
F1000.............      ; Turbo PMAC starts listing
X2500.............
RETURN............      ; This was appended by the CLOSE command
```

See Also:

Program Buffers (Talking to Turbo PMAC)
On-line commands **OPEN**, **CLEAR**, **<CTRL-L>**, **<CTRL-U>**

CLOSE ALL

Function: Close the currently opened buffer on any port

Scope: Global

Syntax: **CLOSE ALL**
CLS ALL

This command closes the program buffer (PROGRAM, PLC, ROTARY) that is currently opened, regardless of the port over which the buffer was opened. When Turbo PMAC receives a **CLOSE ALL** command, it automatically appends a **RETURN** statement to the end of the open program buffer (except for rotary motion program buffers).

Note:

The similar **CLOSE** command can only affect a buffer that was opened on the same port as which the **CLOSE** command is sent.

See Also:

On-line commands **OPEN{buffer}**, **CLOSE**

{constant}

Function: Assign value to variable P0, or to table entry.

Scope: Global

Syntax: **{constant}**

where:

- **{constant}** is a floating-point value

This command is the equivalent of **P0={constant}**. That is, a value entered by itself on a command line will be assigned to P-variable P0. This allows simple operator entry of numeric values through a dumb terminal interface. Where the value goes is hidden from the operator; the Turbo PMAC user program must take P0 and use it as appropriate.

Note:

If a compensation table on Turbo PMAC (**BLCOMP**, **COMP**, or **TCOMP**) has been defined but not filled, a constant value will be entered into this table, *not* into P0.

Example:
In a motion program:

```
P0=-1.....              ; Set P0 to an "illegal" value
SEND"Enter number of parts in run:"
.....                   ; Prompt operator at dumb terminal
.....                   ; Operator simply needs to type in number
WHILE (P0<1) DWELL10    ; Hold until get legal response
P1=0 .....              ; Initialize part counter
WHILE (P0<P1)           ; Loop once per part
   P1=P1+1
   ...
P0=1........            ; Temporary value for P0
#1DEFINE COMP 5,2000    ; Set up 5-entry table
32 48 -96 64 0 -1 ; Firt 5 numbers into table; sixth into P0
P0..........            ; Query P0 value
-1 .....                ; Turbo PMAC responds
```

See Also:
On-line commands **DEFINE BLCOMP**, **DEFINE COMP**, **DEFINE TCOMP**, **P{constant}={expression}**

CPU

Function: Report the Turbo PMAC CPU type.

Scope: Global

Syntax: **CPU**

This command causes Turbo PMAC to report the part number of the CPU used on the board. It is mainly used for troubleshooting purposes.

Example:

```
CPU             ; Request the CPU part number
56303           ; Turbo PMAC responds
```

See Also:
On-line commands **TODAY**, **VERSION**, **TYPE**

DATE

Function: Report the firmware release date.

Scope: Global

Syntax: **DATE**
DAT

This command causes Turbo PMAC to report the release date of the firmware revision that it is using. The date is reported in the North American format:

`{mm}/{dd}/{yyyy}`

where:

- `{mm}` is the 2-digit representation of the month
- `{dd}` is the 2-digit representation of the day of the month
- `{yyyy}` is the 4-digit representation of the year

The 4-digit representation of the year eliminates possible “Year 2000” problems in user code processing the date information.

The **DATE** command is not to be confused with the **TODAY** command, which causes Turbo PMAC to report the current date.

Example:

```
DATE                    ; Request the firmware release date
07/17/1998              ; Turbo PMAC reports the firmware release date
```

See Also:

On-line commands **CPU**, **TODAY**, **VERSION**, **TYPE**

DEFINE BLCOMP

Function: Define backlash compensation table

Scope: Motor specific

Syntax:
```
DEFINE BLCOMP {entries},{count length}
DEF BLCOMP {entries},{count length}
```

where:

- **{entries}** is a positive integer constant representing the number of values in the table;
- **{count length}** is a positive integer representing the span of the table in encoder counts of the motor.

This command establishes a backlash compensation table for the addressed motor. The next **{entries}** constants sent to Turbo PMAC will be placed into this table. The last item on the command line **{count length},** specifies the span of the backlash table in encoder counts of the motor. In use, if the motor position goes outside of the range 0 to count-length, the position is rolled over to within this range before the compensation is computed. The spacing between entries in the table is **{count length}** divided by **{entries}.**

On succeeding lines will be given the actual entries of the table as constants separated by spaces and or carriage return characters. The units of these entries are 1/16 count, and the entries must be integer values. The first entry is the correction at one spacing from the motor zero position (as determined by the most recent home search move or power-up/reset), the second entry is the correction two spacings away, and so on. The correction from the table at motor zero position is zero by definition.

The correction is the amount *subtracted* (added in the negative direction) from the nominal commanded position when the motor is moving in the negative direction to get the corrected position. The correction from the backlash table is added to the Ixx86 "constant" backlash parameter before adjusting the motor position. Corrections from any leadscrew compensation tables that have this motor as the target motor are always active in both directions.

The last entry in the table represents the correction at **{count length}** distance from the motor's zero position. Since the table has the capability to roll over, this entry also represents the correction at the motor's zero position. If it is set to a non-zero value, the correction at zero will also be zero.

Note:

Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any BLCOMP buffer exists for a lower numbered motor, or if any TBUF, ROTARY, LOOKAHEAD or GATHER buffer exists. Any of these buffers must be deleted first. BLCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

I51 must be set to 1 to enable the table.

Example:

```
#32 DEFINE BLCOMP 100, 100000
```

See Also:
Backlash Compensation (Setting Up a Motor)
Leadscrew Compensation (Setting Up a Motor)
I-variables Ixx85, Ixx86, Ixx87

On-line commands **DEFINE COMP**, **DELETE BLCOMP**

DEFINE CCBUF

Function: Define extended cutter-compensation buffer

Scope: Coordinate-system specific

Syntax:
```
DEFINE CCBUF {constant}
DEF CCBUF {constant}
```

where:

- **{constant}** is a positive integer constant representing the size of the buffer in programmed moves

This command establishes an extended cutter-radius compensation move buffer for the addressed coordinate system. This buffer is required only if it is desired to maintain the compensation through one or more moves perpendicular to the plane of compensation (e.g. Z-axis moves during XY-plane compensation). The **{constant}** value in the command specifies the number of moves that can be stored in this extended buffer. This number must be at least as large as the number of consecutive perpendicular moves that may be executed between two moves with a component in the plane of compensation.

Once this buffer is defined, its use is automatic and invisible to the user. If the number of consecutive moves perpendicular to the plane of compensation exceeds this buffer size, Turbo PMAC will assume that the incoming and outgoing moves to this point in the plane of compensation form an outside corner. If it turns out that they form an inside corner, there will be an overcut, or gouge.

The CCBUF is a temporary buffer. Its contents are never held through a board reset or power cycling. Its structure is only retained through a board reset or power cycling if the latest **SAVE** command was issued with the buffer defined and with I14 = 1.

Turbo PMAC will reject this command, reporting an ERR003 if I6 = 1 or 3, if any CCBUFFER exists for a lower-numbered coordinate system, or if any LOOKAHEAD or GATHER buffer exists on the board. Any of these buffers must be deleted first. CCBUFFERs must be defined from high-numbered coordinate system to low-numbered coordinate system.

Example:
```
DEFINE CCBUF 5
```

See Also:
Cutter Radius Compensation
On-line command **DELETE CCBUF**
Program commands **CC0**, **CC1**, **CC2**, **CCR**

DEFINE COMP (one-dimensional)

Function: Define Leadscrew Compensation Table

Scope: Motor specific

Syntax:
```
DEFINE COMP {entries}, [#{source}[D],[#{target},]]
{count length}
```

where:

- **{entries}** is a positive integer constant representing the number of numbers in the table;

- **{source}** (optional) is a constant from 1 to 32 representing the motor whose position controls which entries in the table are used for the active correction; if none is specified, Turbo PMAC assumes the source is the addressed motor; if a **D** is specified after the source motor number, the desired position of the motor is used to calculate the correction; otherwise the actual position is used;
- **{target}** (optional) is a constant from 1 to 32 representing the motor that receives the correction; if none is specified, Turbo PMAC assumes the target is the addressed motor;
- **{count length}** is a positive integer representing the span of the table in encoder counts of the source motor

This command establishes a leadscrew compensation table assigned to the addressed motor. The next **{entries}** constants sent to Turbo PMAC will be placed into this table. Once defined the tables are enabled and disabled with the variable I51.

The table belongs to the currently addressed motor, and unless otherwise specified in the command line, it will use the addressed motor both for source position data and as the target for its corrections. Each motor can only have one table that "belongs" to it (for a total of 32 tables in one Turbo PMAC), but it can act as a source or a target for multiple motors.

Note:

Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any COMP buffer exists for a lower numbered motor, or if any TCOMP, BLCOMP, TBUF, ROTARY, LOOKAHEAD, or GATHER buffer exists. Any of these buffers must be deleted first. COMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

It is possible to directly specify a source motor (with **#{source}**), or source and target motors (with **#{source},#{target}**), in this command. Either or both of them may be different from the motor to which the table "belongs". (In other words, just because a table belongs to a motor does not necessarily mean that it affects or is affected by that motor's position.)

The table can operate as a function of either the desired (commanded) or actual position of the source motor. If a '**D**' is entered immediately after the source motor number (which must be explicitly declared here), the table operates as a function of the desired position of the source motor; if no '**D**' is entered, the table operates as a function of the actual position of the source motor.

The last item on the command line, **{count length}**, specifies the span of the compensation table in encoder counts of the source motor. In use, if the source motor position goes outside of the range 0 to count-length, the source position is "rolled over" to within this range before the correction is computed. The spacing between entries in the table is **{count length}** divided by **{entries}.**

On succeeding lines will be given the actual entries of the table. The units of these entries are 1/16 count, and the entries must be integer values. The first entry is the correction at one spacing from the motor zero position (as determined by the most recent home search move or power-up/reset), the second entry is the correction two spacings away, and so on. The correction is the amount added to the nominal position to get the corrected position. The correction at the zero position is zero by definition.

The last entry in the table represents the correction at **{count length}** distance from the motor's zero position. Since the table has the capability to roll over, this entry also represents the correction at the motor's zero position. If it is set to a non-zero value, the correction at zero will also be non-zero.

Example:

```
#1 DEFINE COMP 4,2000                  ; Create table for motor 1
ERR003.....                            ; Turbo PMAC rejects this command
DELETE GATHER                          ; Clear other buffers to allow loading
DELETE ROTARY
#8DEFINE COMP 500,20000                ; Uses motor 8 as source and target, 500 entries,
.....                                  ; spacing of 40 counts
#7DEFINE COMP 256,#3D,32768            ; "Belongs" to motor 7, uses #3 desired position as
.....                                  ; source, #7 as target, 256 entries, spacing of 128 counts
#6 DEFINE COMP 400,#5,#4,40000         ; "Belongs" to motor 6,
.....                                  ; uses #5 as source, #4 as target,
.....                                  ; 400 entries, spacing of 100 counts
#5 DEFINE COMP 200,#1D,#1,30000        ; "Belongs" to motor 5, uses #1 desired position as
.....                                  ; source and target, 200 entries, spacing of 150 count
I51=1.......                           ; Enable compensation tables
```

See Also:

Leadscrew compensation (Setting Up a Motor)
I-variable I51
On-line commands **{constant}**, **LIST COMP**, **LIST COMP DEF**, **DELETE COMP**, **DELETE GATHER**, **DELETE ROTARY**, **SIZE**

DEFINE COMP (two-dimensional)

Function: Define two-dimensional position compensation table

Scope: Motor-specific

Syntax:

```
DEFINE COMP {Rows}.{Columns}, #{RowMotor}[D],
[#{ColumnMotor}[D], [#{TargetMotor}]],
{RowSpan},{ColumnSpan}

DEF COMP...
```

where:

- **{Rows}** is a positive integer constant representing the number of rows in the table, where each row represents a fixed location of the second (*column*) source motor;
- **{Columns}** is a positive integer constant representing the number of columns in the table, where each column represents a fixed location of the first (*row*) source motor;
- **{RowMotor}** is an integer constant from 1 to 32 representing the number of the first source motor; defaults to addressed motor; if a **D** is specified after the source motor number, the desired position of the motor is used to calculate the correction; otherwise the actual position is used;
- **{ColumnMotor}** is an integer constant from 1 to 32 representing the number of the second source motor; if a **D** is specified after the source motor number, the desired position of the motor is used to calculate the correction; otherwise the actual position is used;
- **{TargetMotor}** is an integer constant from 1 to 32 representing the number of the target motor; defaults to addressed motor;
- **{RowSpan}** is the span of the table, in counts, along the first (row) source motor's travel;

- **{ColumnSpan}** is the span of the table, in counts, along the second (column) source motor's travel.

This command establishes a two-dimensional position compensation table assigned to the addressed motor. The next *(Rows+1)*(Columns+1)-1* constants sent to Turbo PMAC will be placed into this table. This type of table is usually used to correct a motor position (X, Y, or Z-axis) as a function of the planar position of two motors (e.g. X and Y axes). Once defined, the tables are enabled and disabled with the variable I51.

The table belongs to the currently addressed motor, and unless otherwise specified in the command line, it will use the addressed motor both as the first-motor source position data and as the target for its corrections. Each motor can only have one table that belongs to it (for a total of 32 tables in one PMAC), but it can act as a source and/or a target for multiple tables.

Note:

PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any COMP buffer exists for a lower numbered motor, or if any TCOMP, BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. COMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

The first source motor must be specified in the command line with **#{RowMotor}**. The second source motor may be specified in the command line with **#{ColumnMotor}**; if it is not specified, Turbo PMAC assumes that the second source motor is the currently addressed motor. The target motor may be specified with **#{TargetMotor}**; if it is not specified, Turbo PMAC assumes that the target motor is the currently addressed motor.

In other words, if only one motor is specified in the command line, it is the first ("row") source motor, and the second ("column") source and target motors default to the addressed motor. If two motors are specified in the command line, the first one specified is the first ("row") source motor, the second is the second ("column") source motor, and the target motor defaults to the addressed motor. If three motors are specified, the first is the first ("row") source motor, the second is the second ("column") source motor, and the third is the target motor. None of these motors is required to be the addressed motor.

It is strongly recommended that you explicitly specify both source motors and the target motor in this command to prevent possible confusion.

The table can operate as a function of either the desired (commanded) or actual position of the source motors. If a '**D**' is entered immediately after the source motor number (which must be explicitly declared here), the table operates as a function of the desired position of the source motor; if no '**D**' is entered, the table operates as a function of the actual position of the source motor. If the target motor is also one of the source motors, it is recommended that desired position be used, especially in high-gain systems, to prevent interaction with the servo dynamics.

The last two items on the command line, **{RowSpan}** and **{ColumnSpan}**, specify the span of the compensation table for the two source motors, "row" and "column" respectively, expressed in encoder counts of those motors. In use, if the source motor position goes outside of the range 0 to **{Span}**, the source position is "rolled over" to within this range along this axis before the correction is computed.

The count spacing between columns in the table is **{RowSpan}** divided by **{Columns}**. The count spacing between rows in the table is **{ColumnSpan}** divided by **{Rows}**. Note carefully the interaction between the row parameters and the column parameters.

On succeeding command lines will be given the actual correction entries of the table, given as integer numerical constants in text form. The units of these entries are 1/16 count, and the entries must be integer values. The first entry is the correction at one column spacing from the zero position of the **RowMotor**, and the zero position of the **ColumnMotor**. The second entry is the correction at two column spacings from the zero position of the **RowMotor**, and the zero position of the **ColumnMotor**, and so on. Entry number "**Columns**" is the correction at **RowSpan** counts of the **RowMotor**, and at the zero position of the **ColumnMotor** (this entry should be zero if you wish to use the table along the edge, to match the implied zero correction at the origin). These entries should be considered as constituting "Row 0" of the table.

The next entry (entry **Columns**+1, the first entry of Row 1) is the correction at the zero position of the **RowMotor**, and one row spacing of the **ColumnMotor**. The following entry is the correction at one column spacing of the **RowMotor** and one row spacing of the **ColumnMotor**. The entry after this is the correction at two column spacing of the **RowMotor** and one row spacings of the **ColumnMotor**., and so on. The last entry of Row 1 (entry 2***Columns**+1) is the correction at one row spacing of the **RowMotor**, and **RowSpan** counts of the **ColumnMotor**.

Subsequent rows are added in this fashion, with the corrections of the entries for Row *n* being at *n* row spacings from the zero position of the **ColumnMotor**. The last row (row **Rows**) contains corrections at **ColumnSpan** counts of the **ColumnMotor**.

The size of the table is limited only by available data buffer space in Turbo PMAC's memory.

The following chart shows the order of entries into a 2D table with *r* rows and *c* columns, covering a span along the row motor of *RowSpan*, and along the column motor of *ColSpan*:

	Column Motor Position v	**Col 0**	**Col 1**	**Col 2**	**(Col j)**	**Col *c***
Row Motor Position >		0	RowSpan c	2*RowSpan c		RowSpan
Row 0	0	[0]	E_1	E_2	…	E_c
Row 1	ColSpan r	E_{c+1}	E_{c+2}	E_{c+3}	…	E_{2c+1}
Row 2	2*ColSpan r	E_{2c+2}	E_{2c+3}	E_{2c+4}	…	E_{3c+2}
(Row i)		…	…	…	(E_{ic+I+j})	…
Row *r*	ColSpan	E_{rc+r}	E_{rc+r+1}	E_{rc+r+2}		E_{rc+r+c}

There are several important details to note in the entry of a 2D table:

- The number of rows and number of columns is separated by a period, not a comma.
- The correction to the target motor at the zero position of both source motors is zero by definition. This is an implied entry at the beginning of the table (shown by [0] in the above chart); it should not be explicitly entered.
- Consecutive entries in the table are in the same row (except at row's end) separated by one "column spacing" of the position of the first source ("row") motor.
- Both Row 0 and Row *r* must be entered into the table, so effectively you are entering (*r*+1) rows. If there is any possibility that you may go beyond an edge of the table, matching entries of Row 0 and Row *r* should have the same value to prevent a discontinuity in the correction. Row *r* in the table may simply be an added row beyond your real range of concern used just to prevent possible discontinuities at the edges of your real range of concern.

- Both Column 0 and Column *c* must be entered into the table, so effectively you are entering (*c*+1) columns. If there is any possibility that you may go beyond an edge of the table, matching entries of Column 0 and Column *c* should have the same value to prevent a discontinuity in the correction. Column *c* in the table may simply be an added column beyond your real range of concern used just to prevent possible discontinuities at the edges of your real range of concern.
- Because the outside rows and outside columns must match each other to prevent edge discontinuities, the three explicitly entered corner corrections must be zero to match the implicit zero correction at the first corner of the table.

Examples:

```
#1 DEFINE COMP 40.30,#1,#2,#3,300000,400000
.....                    ; Create table belonging to Motor 1
ERR007.....              ; Turbo PMAC rejects this command
DELETE GATHER            ; Clear other buffers to allow loading
&1 DELETE ROTARY
&2 DELETE ROTARY
#2 DELETE COMP
#3 DELETE COMP
#4 DEFINE COMP 30.40,#1,#2,#3,400000,300000
.....                    ; Create same table, now belonging to Motor 4; #1 & #2 are sources, #3
.....                    ; is target; 30 rows x 40 columns, spacing of 10,000 counts
(1270 entries) .....     ; (30+1)*(40+1)-1 entries of constants
#3 DEFINE COMP 25.20,#2,#3,#1,200000,250000
.....                    ; Create table belonging to Motor 3; #2 and #3 are sources, #1 is target;
.....                    ; 25 rows x 20 columns, spacing of 10,000 counts
(545 entries) .....      ;(25+1)*(20+1)-1 entries of constants
#2 DEFINE COMP 10.10,#1,#4,10000,20000
.....                    ; Create table belonging to Motor 2; #1 and #4 are sources, #2 (default)
.....                    ; is target; 10 rows x10 columns, spacing of 1000 cts between columns;
.....                    ; spacing of 2000 cts between rows;
(120 entries) .....      ; (10+1)*(10+1)-1 entries of constants
#1 DEFINE COMP 12.10,#4,1280,1200
.....                    ; Create table belonging to Motor 1; #4 and #1 (default) are sources, #1
.....                    ; (default) is target;12 rows x 10 columns; spacing of 128 cts between;
.....                    ; columns spacing of 100 cts between rows
(142 entries) .....      ; (12+1)*(10+1)-1 entries of constants
I51=1.......             ; Enable compensation tables
```

DEFINE GATHER

Function: Create a data gathering buffer.

Scope: Global

Syntax:
```
DEFINE GATHER [{constant}]
DEF GAT [{constant}]
```

where:

- **{constant}** is a positive integer representing the number of words of memory to be reserved for the buffer

This command reserves space in Turbo PMAC's memory or in DPRAM depending upon the setting of I5000 for the data gathering buffer and prepares it for collecting data at the beginning of the buffer. If a data gathering buffer already exists, its contents are not erased but the Data Gather Buffer Storage address (`Y:$003120`) is reinitialized to the Data Gather Buffer Start address (`X:$003120`) and the **LIST GATHER** command will no longer function. Data collection will again start at the beginning of the buffer when the command **GATHER** is issued.

If Data Gathering is in progress (an **ENDGATHER** command has not been issued and the gather buffer has not been filled up) Turbo PMAC will report an error on receipt of this command.

The optional **{constant}** specifies the number of long words to be reserved for the data gathering buffer, leaving the remainder of Turbo PMAC's memory available for other buffers such as motion and PLC programs. If **{constant}** is not specified, all of Turbo PMAC's unused buffer memory is reserved for data gathering. Until this buffer is deleted (with the **DELETE GATHER** command), no new motion or PLC programs may be entered into Turbo PMAC.

> ***Note:***
>
> If I5000 = 2 or 3 the **{size}** requested in the **DEFINE GATHER {size}** command refers to a DPRAM long word (32-bits). If the **{size}** is smaller than required to hold an even multiple of the requested data, the actual data storage will go beyond the requested **{size}** to the next higher multiple of memory words required to hold the data. For example, if you are gathering one 24-bit value and one 48-bit value you will need 3 DPRAM long words of memory. If the **{size}** you specify is 4000 words (not a multiple of 3), the actual storage size will be 4002 words (the next higher multiple of 3).

The number of long words of unused buffer memory can be found by issuing the **SIZE** command.

Example:

```
DEFINE GATHER
DEFINE GATHER 1000
```

See Also:

I-variables I5000 – I5051

On-line commands **GATHER**, **DELETE GATHER**, **<CTRL-G>**, **SIZE**

DEFINE LOOKAHEAD

Function: Create a lookahead buffer

Scope: Coordinate-system specific

Syntax:
```
DEFINE LOOKAHEAD {constant},{constant}
DEF LOOK {constant},{constant}
```

where:

- the first **{constant}** is a positive integer representing the number of move segments that can be stored in the buffer;
- the second **{constant}** is a positive integer representing the number of synchronous M-variable assignments that can be stored in the buffer

This command establishes a lookahead buffer for the addressed coordinate system. It reserves memory to buffer both motion equations and "synchronous M-variable" output commands for the lookahead function.

Segment Buffer Size: The first constant value in the command determines the number of motion segments that can be stored in the lookahead buffer. Each motion segment takes Isx13 milliseconds at the motion program speeds and acceleration times (the velocity and acceleration limits may extend these times). However, it is variable Isx20 for the coordinate system that determines how many motion segments the coordinate system will actually look ahead in operation.

The lookahead buffer should be sized large enough to store all of the lookahead segments calculated, which means that this constant value must be greater than or equal to Isx20. If backup capability is desired, the buffer must be sized to be large enough to contain the desired lookahead distance plus the desired backup distance.

The method for calculating the number of segments that must be stored ahead is explained in the Isx20 specification and in the Turbo PMAC User's Guide section on Lookahead. The fundamental equation is:

$$Isx20 = \frac{4}{3} * max\left[\frac{Ixx16}{Ixx17}\right]_{xx} * \frac{1}{2 * Isx13}$$

If backup capability is desired, the buffer must be able to store an additional number of segments for the entire distance to be covered in reversal. This number of segments can be calculated as:

$$BackSegments = \frac{BackupDist(units) * 1000(msec/sec)}{V_{max}(units/sec) * SegTime(msec/seg)}$$

The total number of segments to reserve for the buffer is simply the sum of the forward and back segments you wish to be able to hold:

$$TotalSegments = Isx20 + BackSegments$$

Memory Requirements: For each segment Turbo PMAC is told to reserve space for in the lookahead buffer, Turbo PMAC will reserve (*2x+4*) 48-bit words of memory from the main buffer memory space, where *x* is the number of motors in the coordinate system at the time that the buffer is defined. For example, if there are 5 motors in the coordinate system, a command to reserve space for 50 segments will reserve 50*(2*5 + 4) = 700 words of memory. If a motor is added to the coordinate system, or removed from it, after the lookahead buffer has been defined, the lookahead buffer should be re-defined.

Output Buffer Size: The second constant value in the command determines the number of synchronous M-variable assignments that can be stored in the lookahead buffer. Because these are evaluated at lookahead time, but not actually implemented until move execution time, they must be buffered. This section of the buffer must be large enough to store all of the assignments that could be made in the lookahead distance. Synchronous M-variable assignments are not made during backup, so there is no need to reserve memory to store assignments for the backup distance as well as the lookahead distance.

Memory Requirements: For each synchronous M-variable assignment Turbo PMAC is told to reserve space for in the lookahead buffer, Turbo PMAC will reserve two 48-bit words of memory.

There are no performance penalties for making the lookahead buffer larger than required, but this does limit the amount of Turbo PMAC memory free for other features.

A lookahead buffer is never retained through a power-down or board reset, so this command must be issued after every power-up/reset if the lookahead function is to be used.

Note:

Turbo PMAC will reject the **DEFINE LOOKAHEAD** command, reporting an *ERR003* if I6=1 or 3, if any **LOOKAHEAD** buffer exists for a lower-numbered coordinate system, or if a **GATHER** buffer exists. **LOOKAHEAD** buffers must be defined from high-numbered coordinate system to low-numbered coordinate system, and deleted from low-numbered coordinate system to high-numbered coordinate system.

Example:

```
DELETE GATHER                    ; Ensure open memory
&2DEFINE LOOKAHEAD 1000,200      ; Create buffer for C.S. 2
&1DEFINE LOOKAHEAD 1500,20       ; Create buffer for C.S. 1
```

DEFINE ROTARY

Function: Define a rotary motion program buffer

Scope: Coordinate-system specific

Syntax:
```
DEFINE ROTARY{constant}
DEF ROT{constant}
```

where:

- **{constant}** is a positive integer representing the number of long words of memory to be reserved for the buffer

This command causes Turbo PMAC to create a rotary motion program buffer for the addressed coordinate system, allocating the specified number of long words of memory. Rotary buffers permit the downloading of program lines during the execution of the program.

The buffer should be large enough to allow it to hold safely the number of lines you anticipate downloading to Turbo PMAC ahead of the executing point. Each long word of memory can hold one sub-block of a motion program (i.e. **X1000 Y1000** is considered as two sub-blocks). The allocated memory for this coordinate system's rotary buffer remains resident until the buffer is deleted with **DELETE ROT**.

> ***Note:***
>
> Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any ROTARY buffer exists for a lower numbered coordinate system, or if a LOOKAHEAD or GATHER buffer exists. Any of these buffers must be deleted first. ROTARY buffers must be defined from high-numbered coordinate system to low-numbered coordinate system, and deleted from low-numbered coordinate system to high-numbered coordinate system.

Example:

```
DELETE GATHER                    ; Ensure open memory
&2DEFINE ROT 100                 ; Create buffer for C.S. 2
&1DEFINE ROT 100                 ; Create buffer for C.S. 1
&1B0 &2B0 .....                  ; Point to these buffers
OPEN ROT.........                ; Open these buffers for entry
  ...............               ; enter program lines here
```

See Also:

Rotary Program Buffers (Writing a Motion Program)

On-line commands **OPEN ROTARY**, **DELETE ROTARY**, **DELETE GATHER**

DEFINE TBUF

Function: Create a buffer for axis transformation matrices.

Scope: Global

Syntax:
```
DEFINE TBUF {constant}
DEF TBUF {constant}
```

where:

- **{constant}** is a positive integer representing the number of transformation matrices to create

This command reserves space in Turbo PMAC's memory for one or more axis transformation matrices. These matrices can be used for real-time translation, rotation, scaling, and mirroring of the X, Y, and Z axes of any coordinate system. A coordinate system selects which matrix to use with the **TSELn** command, where n is an integer from 1 to the number of matrices created here.

> ***Note:***
>
> Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any LOOKAHEAD, ROTARY,or GATHER buffer exists. Any of these buffers must be deleted first.

The number of long words of unused buffer memory can be found by issuing the **SIZE** command. Each defined matrix takes 21 words of memory.

Example:

```
DELETE GATHER
DEF TBUF 1
DEFINE TBUF 8
```

See Also:

Axis Transformation Matrices (Writing a Motion Program)
On-line commands **DELETE TBUF**, **DELETE GATHER**, **SIZE**.
Program commands **TSEL**, **ADIS**, **AROT**, **IDIS**, **IROT**, **TINIT**

DEFINE TCOMP

Function: Define torque compensation table

Scope: Motor specific

Syntax:
```
DEFINE TCOMP {entries},{count length}
DEF TCOMP {entries},{count length}
```

where:

- **{entries}** is a positive integer constant representing the number of values in the table;
- **{count length}** is a positive integer representing the span of the table in encoder counts of the motor.

This command establishes a torque compensation table for the addressed motor. The next **{entries}** constants sent to Turbo PMAC will be placed into this table. The last item on the command line **{count length},** specifies the span of the torque compensation table in encoder counts of the motor. In use, if the motor position goes outside of the range 0 to count-length, the position is "rolled over" to within this range before the compensation is computed. The spacing between entries in the table is **{count length}** divided by **{entries}.**

On succeeding lines will be given the actual entries of the table as constants separated by spaces and or carriage return characters. The units of these entries are bits of a 16-bit DAC (even if the actual output device has some other resolution), and the entries must be integer values. The first entry is the correction at one spacing from the motor zero position (as determined by the most recent home search move or power-up/reset), the second entry is the correction two spacings away, and so on. Turbo PMAC computes corrections for positions between the table entries by a first-order interpolation between adjacent entries. The correction from the table at motor zero position is zero by definition.

The correction is the magnitude added to Turbo PMAC's servo loop output at that position. If Turbo PMAC's command is positive, a positive value from the table will increase the magnitude of the output; a negative value will decrease the magnitude of the output. If Turbo PMAC's command is negative, a positive value from the table will increase the magnitude of the output in the negative direction; a negative value will decrease the magnitude of the output.

The last entry in the table represents the correction at **{count length}** distance from the motor's zero position. Since the table has the capability to roll over, this entry also represents the correction at the motor's zero position. If it is set to a non-zero value, the correction at zero will also be zero.

> ***Note:***
>
> Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any TCOMP buffer exists for a lower numbered motor, or if any BLCOMP, TBUF, LOOKAHEAD, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. TCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

I51 must be set to 1 to enable the table.

See Also:
Torque Compensation (Setting Up a Motor)
I-variables I51
On-line command **DELETE TCOMP**

DEFINE UBUFFER [modified description]

Function: Create a buffer for user variable use.

Scope: Global

Syntax:
```
DEFINE UBUFFER {constant}
DEF UBUF {constant}
```

where:

- **{constant}** is a positive integer representing the number of 48-bit words of Turbo PMAC memory to reserve for this purpose

This command reserves space in Turbo PMAC's memory for the user's discretionary use. This memory space will be untouched by any Turbo PMAC automatic functions. User access to this buffer is through M-variables, or possibly through on-line **W** (write) and **R** (read) commands.

The buffer starts at Turbo PMAC data memory end address ($0107FF for the standard data memory, and $03FFFF for the extended data memory) and continues back toward the beginning of memory ($000000) for the number of long (48-bit) words specified by **{constant}**. This memory space can be subdivided any way the user sees fit. These registers are backed up by the flash memory, so the values in the buffer at the last **SAVE** command will be copied from the flash memory into the buffer at power-up or reset.

All other buffers except for fixed motion programs (PROG) and PLC programs must be deleted before Turbo PMAC will accept this command. There can be no rotary motion program, leadscrew compensation table, transformation matrix, data gathering or lookahead buffers in Turbo PMAC memory (any buffer created with a **DEFINE** command) for this command to be accepted. It is usually best to reinitialize the card with a **$$$***** command, or erase all defined buffers with the **DELETE ALL** command, before sending the **DEFINE UBUFFER** command

The address of the end of unreserved memory is held in register X:$0031B2. If no UBUFFER is defined, this register will hold a value of $010800 for the standard data memory configuration, or $040000 for the extended data memory configuration. (Starting with V1.937 firmware, a Turbo PMAC with the extended data memory configuration will at re-initialization have a UBUFFER of 65,536 words defined, causing this register to hold a value of $030000.) Immediately after the user buffer has successfully been defined, this register will hold the address of the start of the buffer (the end of the user buffer is always at the end of data memory).

To free up this memory for other uses, the **DEFINE UBUFFER 0** command should be used (there is no **DELETE UBUFFER** command). It may be more convenient simply to re-initialize the board with a **$$$*** command.

Example:

```
RHX:$0031B2                ; Look for end of unreserved memory
00FC99                     ; Reply indicates some reserved
$$$***                     ; Re-initialize card to clear memory
RHX:$0031B2                ; Check end of unreserved memory
010800                     ; Reply indicates none reserved
DEFINE UBUFFER 2048        ; Reserve memory for buffer
RHX:$0031B2                ; Check for beginning of buffer
010000                     ; Reply confirms 2048 words reserved
M1000->D:$010000           ; Define M-variable to first word
M1010->Y:$010020,12,1      ; Define M-variable to a middle word
M1023->X:$0107FF,24,S      ; Define M-variable to last word
```

See Also:
User Buffer, M-Variables (Computational Features)
I-variable I4908
On-line commands **$$$***, **R[H]{address}**, **W{address}**

DELETE ALL

Function: Erase all defined permanent and temporary buffers

Scope: Global

Syntax: **DELETE ALL**
DEL ALL

This command causes Turbo PMAC to erase all buffers created with a **DEFINE** command, permanent (fixed) and temporary, in Turbo PMAC's memory space. These include:

- User buffer (**UBUFFER**)
- Leadscrew compensation tables (**COMP**)
- Torque compensation tables (**TCOMP**)
- Backlash compensation tables (**BLCOMP**)
- Transformation matrix buffers (**TBUF**)
- Rotary motion program buffers (**ROTARY**)
- Segment lookahead buffers (**LOOKAHEAD**)
- Extended cutter-radius compensation buffers (**CCBUF**)
- Data gathering buffer (**GATHER**)

See Also:
On-line commands **CLEAR**, **CLEAR ALL**, **CLEAR ALL PLCS**, **OPEN**, **DELETE ALL TEMP**

DELETE ALL TEMPS

Function: Erase all defined temporary buffers

Scope: Global

Syntax: **DELETE ALL TEMPS**
DEL ALL TEMP

This command causes Turbo PMAC to erase all temporary buffers created with a **DEFINE** command in Turbo PMAC's memory space.

Temporary buffers are those whose contents are not kept through a power-down or reset, even if the structures for the buffers are (when I14=1). These include:

- Rotary motion program buffers (**ROTARY**)
- Segment lookahead buffers (**LOOKAHEAD**)
- Extended cutter-radius compensation buffers (**CCBUFFER**)
- Data gathering buffer (**GATHER**)

This command does not affect permanent buffers created with a **DEFINE** command. It also does not affect fixed (not rotary) motion program buffers (**PROGRAM**), forward-kinematic program buffers (**FORWARD**), inverse-kinematic program buffers (**INVERSE**), uncompiled PLC program buffers (**PLC**), or compiled PLC program buffers (**PLCC**).

See Also:
I-variable I14
On-line commands **CLEAR**, **CLEAR ALL**, **CLEAR ALL PLCS**, **OPEN**, **DELETE ALL**

DELETE BLCOMP

Function: Erase backlash compensation table

Scope: Motor specific

Syntax: **DELETE BLCOMP**
DEL BLCOMP

This command causes Turbo PMAC to erase the compensation table for the addressed motor, freeing that memory for other use.

> ***Note:***
>
> Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any BLCOMP buffer exists for a lower numbered motor, or if any TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. BLCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

Example:

```
#2 DEL BLCOMP          ; Erase table belonging to Motor 2
ERR003.....            ; Turbo PMAC rejects this command
#1 DEL BLCOMP          ; Erase table belonging to Motor 1
#2 DEL BLCOMP          ; Erase table belonging to Motor 2
```

See Also:
Backlash Compensation (Setting Up a Motor)
I-variables Ixx87, Ixx85, Ixx86
On-line command **DEFINE BLCOMP**

DELETE CCUBUF

Function: Erase extended cutter-compensation buffer

Scope: Motor specific

Syntax: **DELETE CCBUF**
DEL CCBUF

This command causes Turbo PMAC to erase the extended cutter-radius compensation move buffer for the addressed coordinate system, freeing that memory for other use.

Turbo PMAC will reject this command, reporting an ERR003 if I6 = 1 or 3, if any CCBUF exists for a lower-numbered coordinate system, or if any LOOKAHEAD or GATHER buffer exists on the board. Any of these buffers must be deleted first. CCBUFs must be deleted from low-numbered coordinate system to high-numbered coordinate system.

See Also:
Cutter Radius Compensation
On-line command **DEFINE CCBUF**
Program commands **CC0**, **CC1**, **CC2**, **CCR**

DELETE COMP

Function: Erase leadscrew compensation table

Scope: Motor specific

Syntax: **DELETE COMP**
DEL COMP

This command causes Turbo PMAC to erase the compensation table belonging to the addressed motor, freeing that memory for other use.

Note:

Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any COMP buffer exists for a lower numbered motor, or if any TCOMP, BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. COMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

Remember that a compensation table belonging to a motor does not necessarily affect that motor or is not necessarily affected by that motor. The command **LIST COMP DEF** will tell which motors it affects and is affected by.

Example:

```
#2DEL COMP....          ; Erase table belonging to Motor 2
ERR003.....             ; Turbo PMAC rejects this command
#1 DELETE COMP          ; Erase table belonging to Motor 1
#2 DELETE COMP          ; Erase table belonging to Motor 2
```

See Also:
Leadscrew compensation (Setting Up a Motor)
I-variable I51
On-line commands **{constant}**, **LIST COMP**, **LIST COMP DEF**, **DEFINE COMP**

DELETE LOOKAHEAD

Function: Erase the lookahead buffer.

Scope: Coordinate-system specific

Syntax: **DELETE LOOKAHEAD**
DEL LOOK

This command causes Turbo PMAC to erase the lookahead buffer for the addressed coordinate system, freeing that memory for other use.

Note:

Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any LOOKAHEAD buffer exists for a lower numbered coordinate system, or if any ROTARY or GATHER buffer exists. Any of these buffers must be deleted first. LOOKAHEAD buffers must be defined from high-numbered coordinate system to low-numbered coordinate system, and deleted from low-numbered coordinate system to high-numbered coordinate system.

Lookahead buffers are not maintained through a power-down or board reset, even if a **SAVE** command has been done while the buffers exist. Therefore a board reset will automatically delete all lookahead buffers.

DELETE GATHER

Function: Erase the data gather buffer.

Scope: Global

Syntax: **DELETE GATHER**
DEL GAT

This command causes the data gathering buffer to be erased. The memory that was reserved is now de-allocated and is available for other buffers (motion programs, PLC programs, compensation tables, etc.). If Data Gathering is in progress (an **ENDGATHER** command has not been issued and the gather buffer has not been filled up) Turbo PMAC will report an error on receipt of this command.

Turbo PMAC's Executive Program automatically inserts this command at the top of a file when it uploads a buffer from Turbo PMAC into its editor, so the next download will not be hampered by an existing gather buffer. It is strongly recommended that you use this command as well when you create a program file in the editor (see Examples, below).

Note:

When the executive program's data gathering function operates, it automatically reserves the entire open buffer space for gathered data. When this has happened, no additional programs or program lines may be entered into Turbo PMAC's buffer space until the **DELETE GATHER** command has freed this memory.

Example:

```
CLOSE.....          ; Make sure no buffers are open
DELETE GATHER       ; Free memory
OPEN PROG 50        ; Open new buffer for entry
CLEAR.....          ; Erase contents of buffer
.....               ; Enter new contents here
```

See Also:

Buffered Commands (Talking to Turbo PMAC)
On-line commands **GATHER**, **DEFINE GATHER**, **SIZE**

DELETE PLCC

Function: Erase specified compiled PLC program

Scope: Global

Syntax: **DELETE PLCC {constant}**
DEL PLCC {constant}

where:

- **{constant}** is an integer from 0 to 31, representing the program number

This command causes Turbo PMAC to erase the specified compiled PLC program. Remember that because all of the compiled PLC programs must be downloaded to Turbo PMAC together, the only way to restore this PLC is to download the entire set of compiled PLCs.

Only one PLCC program can be deleted in one command. Ranges and lists of PLCC program numbers are not permitted in this command.

To perform the same function for an uncompiled PLC program, the command sequence would be **OPEN PLC n CLEAR CLOSE**.

Example:

```
DELETE PLCC 5          ; Erase compiled PLC program 5
DEL PLCC 0....         ; Erase compiled PLC program 0
```

See Also:

Compiled PLCs (Writing a PLC Program)
I-variable I5
On-line commands **DISABLE PLCC**, **ENABLE PLCC**, **CLEAR**

DELETE ROTARY

Function: Delete rotary motion program buffer of addressed coordinate system

Scope: Coordinate-system specific

Syntax: **DELETE ROTARY**
DEL ROT

This command causes Turbo PMAC to erase the rotary buffer for the currently addressed coordinate system and frees the memory that had been allocated for it.

> ***Note:***
>
> Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if the ROTARY buffer for this coordinate system is open or executing, or if any ROTARY buffer exists for a lower numbered coordinate system, or if a GATHER buffer exists. Any of these buffers must be deleted first. ROTARY buffers must be defined from high-numbered coordinate system to low-numbered coordinate system, and deleted from low-numbered motor to high-numbered motor.

Example:

```
&2 DELETE ROTARY       ; Try to erase C.S. 2's rotary buffer
ERR003.....            ; Turbo PMAC rejects this; C.S. 1 still has
.............          ; a rotary buffer
&1 DELETE ROTARY       ; Erase C.S. 1's rotary buffer
&2 DELETE ROTARY       ; Erase C.S. 2's rotary buffer; OK now
```

See Also:

Rotary Program Buffers (Writing a Motion Program)
On-line commands **DEFINE ROTARY**, **OPEN ROTARY**.

DELETE TBUF

Function: Delete buffer for axis transformation matrices.

Scope: Global

Syntax: **DELETE TBUF**
DEL TBUF

This command frees up the space in Turbo PMAC's memory that was used for axis transformation matrices. These matrices can be used for real-time translation, rotation, scaling, and mirroring of the X, Y, and Z axes of any coordinate system.

Note:

Turbo PMAC will reject this command, reporting an ERR007 if I6=1 or 3, if any ROTARYor GATHER buffer exists. Any of these buffers must be deleted first.

Example:
```
DEL TBUF
DELETE TBUF
```

See Also:
Axis Transformation Matrices (Writing a Motion Program)
On-line commands **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **AROT**, **IDIS**, **IROT**, **TINIT**

DELETE TCOMP

Function: Erase torque compensation table

Scope: Motor specific

Syntax: **DELETE TCOMP**
DEL TCOMP

This command causes Turbo PMAC to erase the torque compensation table for the addressed motor, freeing that memory for other use.

Note:

Turbo PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any TCOMP buffer exists for a lower numbered motor, or if any BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. TCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

Example:
```
#2 DEL TCOMP          ; Erase table belonging to Motor 2
ERR003.....           ; Turbo PMAC rejects this command
#1 DEL TCOMP          ; Erase table belonging to Motor 1
#2 DEL TCOMP          ; Erase table belonging to Motor 2
```

See Also:
Torque Compensation (Setting Up a Motor)
I-variables I51
On-line command **DEFINE TCOMP**

DISABLE PLC

Function: Disable specified PLC program(s).

Scope: Global

Syntax:
```
DISABLE PLC {constant}[,{constant}]
DIS PLC {constant}[,{constant}]
DISABLE PLC {constant}..{constant}
DIS PLC {constant}..{constant}
```

where

- **{constant}** is an integer from 0 to 31, representing the program number

This command causes Turbo PMAC to disable (stop executing) the specified uncompiled PLC program or programs. Execution can subsequently be resumed at the top of the program with the **ENABLE PLC** command. If it is desired to restart execution at the stopped point, execution should be stopped with the **PAUSE PLC** command, and restarted with the **RESUME PLC** command

The on-line **DISABLE PLC** command can only suspend execution of a PLC program at the end of a scan, which is either the end of the program, or after an **ENDWHILE** statement in the program.

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs. PLC programs can be re-enabled by using the **ENABLE PLC** command.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

Example:
```
DISABLE PLC 1
DIS PLC 5
DIS PLC 3,4,7
DISABLE PLC 0..31
```

See Also:

I-variable I5

On-line commands **ENABLE PLC**, **OPEN PLC**, **DISABLE PLCC**, **ENABLE PLCC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **ENABLE PLC**, **DISABLE PLCC**, **ENABLE PLCC**

DISABLE PLCC

Function: Disable compiled PLC program(s).

Scope: Global

Syntax:
```
DISABLE PLCC {constant}[,{constant}]
DIS PLCC {constant}[,{constant}]
PLCC {constant}..{constant}
DIS PLCC {constant}..{constant}
```

where:

- **{constant}** is an integer from 0 to 31, representing the compiled PLC program number

This command causes Turbo PMAC to disable (stop executing) the specified compiled PLC program or programs. Compiled PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs. PLC programs can be re-enabled by using the **ENABLE PLCC** command.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

Example:

```
DISABLE PLCC 1
DIS PLCC 5
DIS PLCC 3,4,7
DISABLE PLCC 0..31
```

See Also:

I-variable I5

On-line commands **DISABLE PLC**, **ENABLE PLC**, **ENABLE PLCC**, **OPEN PLC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **DISABLE PLCC**, **ENABLE PLC**, **ENABLE PLCC**

EAVERSION

Function: Report firmware version information

Scope: Global

Syntax: **EAVERSION**
EAVER

This command causes Turbo PMAC to report information about the firmware version it is using. Turbo PMAC responds with seven decimal digits.

The first 4 digits represent the firmware version number, without the decimal point (e.g. 1934 for version 1.934).

The fifth digit is 0 for a released firmware version. If it has a value 'n' greater than 0, it is reporting the 'nth' test (pre-release) revision of this numerical firmware version.

The sixth digit is reserved for future use. It presently always reports a 0.

The seventh digit is a 0 for a Turbo PMAC1; it is a 1 for a Turbo PMAC2.

Example:

```
EAVERSION .....      ; Ask Turbo PMAC for firmware version
1934201 .........    ; Turbo PMAC responds Version 1.934 2nd test revision
.....                ; Turbo PMAC2 firmware
```

See Also:

Resetting Turbo PMAC (Talking to Turbo PMAC)

On-line command **DATE**, **VERSION**, **TYPE**

ENABLE PLC

Function: Enable specified PLC program(s).

Scope: Global

Syntax:
```
ENABLE PLC {constant}[,{constant}]
ENA PLC {constant}[,{constant}]
ENABLE PLC {constant}..{constant}
ENA PLC {constant}..{constant}
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

This command causes Turbo PMAC to enable (start executing) the specified uncompiled PLC program or programs at the top of the program. Execution of the PLC program may have been stopped with the **DISABLE PLC**, **PAUSE PLC**, or **OPEN PLC** command.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

I-variable I5 must be in the proper state to allow the PLC program(s) specified in this command to execute.

> ***Note:***
>
> This command must be used to start operation of a PLC program after it has been entered or edited, because the **OPEN PLC** command disables the program automatically and **CLOSE** does not re-enable it.

Example:

```
ENABLE PLC 1
ENA PLC 2,7
ENABLE PLC 3,21
ENABLE PLC 0..31
```

This example shows the sequence of commands to download a very simple PLC program and have it enabled automatically on the download:

```
OPEN PLC 7 CLEAR
P1=P1+1
CLOSE
ENABLE PLC 7
```

See Also:

I-variable I5

On-line commands **ENABLE PLC**, **OPEN PLC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **ENABLE PLC**

ENABLE PLCC

Function: Enable specified compiled PLC program(s).

Scope: Global

Syntax: **ENABLE PLCC {constant}[,{constant}]**

```
ENA PLCC {constant}[,{constant}]
 PLCC {constant}..{constant}
ENA PLCC {constant}..{constant}
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

This command causes Turbo PMAC to enable (start executing) the specified compiled PLC program or programs. Compiled PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

I-variable I5 must be in the proper state to allow the compiled PLC program(s) specified in this command to execute.

Example:

```
ENABLE PLCC 1
ENA PLCC 2,7
ENABLE PLCC 3,21
ENABLE PLCC 0..31
```

See Also:

I-variable I5

On-line commands **DISABLE PLC**, **DISABLE PLCC**, **ENABLE PLC**, **OPEN PLC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **DISABLE PLCC**, **ENABLE PLC**, **ENABLE PLCC**

ENDGATHER

Function: Stop data gathering.

Scope: Global

Syntax: **ENDGATHER**
ENDG

This command causes data gathering to cease. Data gathering may start up again (without overwriting old data) with another **GATHER** command.

This command is usually used in conjunction with the data gathering and plotting functions of the Turbo PMAC Executive Program.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

Examples:

```
GAT B1R.....        ; Start gathering and run program 1
ENDG.....           ; Stop gathering -- give this command when moves of interest are done
OPEN PROG2 CLEAR
X10
DWELL1000
CMD"GATHER"..       ; Program issues start command here
X20.....            ; Move of interest
DWELL50
CMD"ENDG" .....     ; Program issues stop command here
CLOSE
```

See Also:

Data Gathering Function (Analysis Features)
I-variables I5000 – I5051
On-line commands **DEFINE GATHER**, **GATHER**, **LIST GATHER**, **DELETE GATHER**
Gathering and Plotting (Turbo PMAC Executive Program Manual)

F

Function: Report motor following error

Scope: Motor specific

Syntax: **F**

This command causes Turbo PMAC to report the present motor following error (in counts, rounded to the nearest tenth of a count) for the addressed motor to the host. Following error is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and Turbo PMAC reports a value of 0.

Example:

```
F .....         ; Ask for following error of addressed motor
12 .....        ; Turbo PMAC responds
#3F.....        ; Ask for following error of Motor 3
-6.7 .....      ; Turbo PMAC responds
```

See Also:

Following Error (Servo Features)
I-variables Ixx11, Ixx12, Ixx67
On-line commands **<CTRL-F>**, **P**, **V**
Suggested M-variable definitions Mxx61, Mxx62
Memory map registers D:$0000DB, D:$00015B, etc.

FRAX

Function: Specify the coordinate system's feedrate axes.

Scope: Coordinate-system specific

Syntax: **FRAX**
FRAX({axis}[,{axis}...])

where:

- **{axis}** (optional) is a character (X, Y, Z, A, B, C, U, V, W) specifying which axis is to be used in the vector feedrate calculations

No spaces are permitted in this command.

This command specifies which axes are to be involved in the vector-feedrate (velocity) calculations for upcoming feedrate-specified (**F**) moves. Turbo PMAC calculates the time for these moves as the vector distance (square root of the sum of the squares of the axis distances) of all the feedrate axes divided by the feedrate. Any non-feedrate axes commanded on the same line will complete in the same amount of time, moving at whatever speed is necessary to cover the distance in that time.

Vector feedrate has obvious geometrical meaning only in a Cartesian system, for which it results in constant tool speed regardless of direction, but it is possible to specify for non-Cartesian systems, and for more than three axes.

Note:

If the move time as calculated for the vector-feedrate axes is less than the time computed as the distance of any non-feedrate axis commanded on the line divided by the Isx86 "alternate feedrate" parameter, this longer time will be used for all axes instead.

If a motion program buffer is open when this command is sent to Turbo PMAC, it will be entered into the buffer for later execution.

For instance, in a Cartesian XYZ system, if you use **FRAX(X,Y)**, all of your feedrate-specified moves will be at the specified vector feedrate in the XY-plane, but not necessarily in XYZ-space. If you use **FRAX(X,Y,Z)** or **FRAX**, your feedrate-specified moves will be at the specified vector feedrate in XYZ-space. Default feedrate axes for a coordinate system are X, Y, and Z.

Example:

```
FRAX ..................  ; Make all axes feedrate axes
FRAX(X,Y) ......        ; Make X and Y axes only the feedrate axes
FRAX(X,Y,Z)..; Make X, Y, and Z axes only the feedrate axes
```

See Also:

Feedrate-Specified Moves (Writing a Motion Program)
Program commands **F{data}**, **FRAX**.

GATHER

Function: Begin data gathering.

Scope: Global

Syntax: **GATHER [TRIGGER]**
GAT [TRIG]

This command causes data gathering to commence according to the configuration defined in I-variables I5000 – I5051. If **TRIGGER** is not used in the command, gathering will start on the next servo cycle. If **TRIGGER** is used, gathering will start on the first servo cycle after machine input MI2 goes true.

Gathering will proceed at the frequency set by I5049 (in number of servo interrupt cycles). If I5049 is 0, only one set of data will be gathered per **GATHER** command. If Turbo PMAC is already gathering data, **GATHER** will cause resynchronization of the gathering cycle to the next servo cycle.

Gathering will continue until Turbo PMAC receives an **ENDGATHER** command, or until the buffer created by the **DEFINE GATHER** command is full.

This command is usually used in conjunction with the data gathering and plotting functions of the Turbo PMAC Executive Program.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

Example:

```
GAT B1R.....          ; Start gathering and run program 1
ENDG.....             ; Stop gathering -- give this command when moves of interest are done
OPEN PROG2 CLEAR
X10
DWELL1000
CMD"GATHER"..         ; Program issues start command here
X20.....              ; Move of interest
DWELL50
CMD"ENDG" .....       ; Program issues stop command here
CLOSE
```

See Also:

Data Gathering Function (Analysis Features)
I-variables I5000 – I5051
On-line commands **DEFINE GATHER**, **GATHER**, **LIST GATHER**, **DELETE GATHER**
Gathering and Plotting (Turbo PMAC Executive Program Manual)

H

Function: Perform a feedhold

Scope: Coordinate-system specific

Syntax: **H**

This causes the currently addressed coordinate system to suspend execution of the program starting immediately by bringing its time base value to zero, decelerating along its path at a rate defined by the coordinate system I-variable Isx95.

The program is suspended while in feedhold mode, but technically still executing. If it is subsequently desired that program execution will not be resumed, program execution should be aborted with the **A** command

The **H** command is very similar in effect to a **%0** command, except that deceleration is controlled by Isx95, not Isx94, and execution can be resumed with an **R** or an **S** command, instead of a **%100** command. In addition, **H** works under external time base, whereas a **%0** command does not.

Jogging moves are permitted while in feedhold mode (unlike in non-Turbo PMACs). However, all motors must be returned to their exact feedhold position (using the **J=** command) before the program may be resumed.

Full speed execution along the path will commence again on an **R** or **S** command. The ramp up to full speed will also take place at a rate determined by Isx95 (full time-base value, either internally or externally set). Once the full speed is reached, Isx94 determines any time-base changes.

If the **R** or **S** command is issued when any motor is not commanding its feedhold position, Turbo PMAC will reject the command with an error, reporting an ERR0017 if I6 has been set to 1 or 3.

See Also:
Stopping Commands (Making Your Application Safe)
Control-Panel Port HOLD/ Input (Connecting Turbo PMAC to the Machine)
Time-Base Control (Synchronizing Turbo PMAC to External Events)
I-variables Isx93, Isx94, Isx95
On-line commands **<CTRL-O>**, **%**, **%{constant}**, **A**, **K**, ****, **Q**
JPAN Connector Pin 12

HOME

Function: Start Homing Search Move

Scope: Motor specific

Syntax: **HOME**
HM

This command causes the addressed motor to perform a homing search routine. The characteristics of the homing search move are controlled by motor I-variables Ixx97 and Ixx19-Ixx26, plus encoder I-variables 2 and 3 for that motor's position encoder.

The on-line home command simply starts the homing search routine. Turbo PMAC provides no automatic indication that the search has completed (although the In-Position interrupt could be used for this purpose) or whether the move completed successfully. Polling, or a combination of polling and interrupts, is generally used to determine completion and success.

By contrast, when a homing search move is given in a motion program (e.g. **HOME1,2**), the motion program will keep track of completion by itself as part of its sequencing algorithms.

Note the difference in syntax between the on-line homing search move command (**#xHOME**) and the buffered motion program homing search move statement (**HOMEn**)

If there is an axis offset in the axis-definition statement for the motor, and/or following error in the motor servo loop, the reported position at the end of the homing search move will be equal to the axis offset minus the following error, not to zero.

Example:

```
HOME ..................  ; Start homing search on the addressed motor
#1HM ..................  ; Start homing search on Motor 1
#3HM#4HM.....  ; Start homing search on Motors 3 and 4
```

See Also:
Control Panel Port HOME/ Input (Connecting Turbo PMAC to the Machine)
Homing Moves (Basic Motor Moves)
I-variables Ixx03, Ixx19-Ixx26, Encoder I-Variables 2 & 3
On-line command **HOMEZ**
Program command **HOME{constant}**, **HOMEZ{constant}**
JPAN Connector Pin 11

HOMEZ

Function: Do a Zero-Move Homing

Scope: Motor specific

Syntax: **HOMEZ**
HMZ

This command causes the addressed motor to perform a zero-move homing search.

Instead of jogging until it finds a pre-defined trigger, and calling its position at the trigger the home position, with this command, the motor calls wherever it is (*commanded* position) at the time of the command the home position.

If there is an axis offset in the axis-definition statement for the motor, and/or following error in the motor servo loop, the reported position at the end of the homing operation will be equal to the axis offset minus the following error, not to zero.

Example:

```
; On-line command examples
HOMEZ.....              ; Do zero-move homing search on the addressed motor
#1HMZ.....              ; Do zero-move homing search on Motor 1
#3HMZ#4HMZ....          ; Do zero-move  homing search on Motors 3 and 4
; Buffered motion program examples
HOMEZ1
HOMEZ2,3
; On-line commands issued from PLC program
IF (P1=1) .....         ;
  CMD"#5HOMEZ"          ; Program issues on-line command
  P1=0.....             ; So command is not repeatedly issued
ENDIF
```

See Also:

Homing Moves (Basic Motor Moves)

On-line command **HOME**

Program command **HOME{constant}**, **HOMEZ{constant}**

I{constant}

Function: Report the current I-variable value(s).

Scope: Global

Syntax: **I{constant}[..{constant}]**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the I-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

I{constant},{constant},{constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first I-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number I-variables whose value is to be reported;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each I-variable whose value is to be reported.

This command causes Turbo PMAC to report the current value of the specified I-variable, or range or set of I-variables.

When I9 is 0 or 2, only the value of the I-variable itself is returned (e.g. *10000*). When I9 is 1 or 3, the entire variable value assignment statement (e.g. *I130=10000*) is returned by Turbo PMAC.

When I9 is 0 or 1, the values of "address" I-variables are reported in decimal form. When I9 is 2 or 3, the values of these variables are reported in hexadecimal form.

Note:

If a motion program buffer (including a rotary buffer) is open, **I{constant}** will be entered into that buffer for later execution, to be interpreted as a full-circle move command with a vector to the center along the X-axis (see Circular Moves in the Writing a Motion Program section).

Example:

```
I5 ..................   ; Request the value of I5
2 ...................   ; Turbo PMAC responds
I130..135 ...........   ; Request the value of I130 through I135
60000.................  ; Turbo PMAC responds with 6 lines
5000
5000
50000
1
20000
I130,4,100..            ; Request the value of I130, I230, I330, I430
60000.......            ; Turbo PMAC responds with 4 lines
55000
60000
65000
```

To see the effect of I9 on the form of the response, observe the following:

```
I9=0  I125
491520.............     ; Short form, decimal
I9=1  I125
I125=491520..; Long form, decimal
I9=2  I125
$78000.............     ; Short form, hexadecimal
I9=3  I125
I125=$78000..; Long form, hexadecimal
```

See Also:

Initialization (I) Variables (Computational Features)
I-Variable Specifications
I-variable I9
On-line commands **I{constant}={expression}**, **M{constant}**, **P{constant}**, **Q{constant}**
Program commands **{axis}{data}{vector}{data}**, **I{data}**

I{data}={expression}

Function: Assign a value to an I-variable.

Scope: Global

Syntax: **I{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the I-variable number;
- **{expression}** contains the value to be given to the specified I-variable

I{constant}..{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first I-variable;

- the second **{constant}** is an integer from 1 to 8191 representing the number of the last I-variable; it must be at least as great as the first **{constant}**
- the final **{constant}** contains the value to be given to the specified range of I-variables

I{constant},{constant},{constant}={constant}
where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first I-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number I-variables whose value is to be set;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each I-variable whose value is to be set;
- the final **{constant}** contains the value to be given to the specified set of I-variables

This command assigns the value on the right side of the equals sign to the specified I-variable, or range or set of I-variables.

If a motion or PLC program buffer is open when the single-variable form of this command is sent to Turbo PMAC, the command will be entered into the buffer for later execution. If a motion or PLC program buffer is open when the multiple-variable form of this command is sent, Turbo PMAC will reject the command with an error, reporting ERR003 if I6 is 1 or 3.

Examples:

```
I5=2
I130=1.25*I130
I(P1*100+30)=300000
I5001..5048=0
I102=$78003
I100,5,100=1        ; Sets I100, I200, I300, I400, I500 to 1
I104=I103
```

See Also:
Initialization (I) Variables (Computational Features)
I-Variable Specifications
On-line commands **I{constant}**, **M{data}={expression}**, **P{data}={expression}**, **Q{data}={expression}**

I{constant}=*

Function: Assign factory default value to an I-variable.

Scope: Global

Syntax: **I{constant}[..{constant}]=***

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the I-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

I{constant},{constant},{constant}
where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first I-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number of I-variables to be set to default values;

- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each I-variable to be set to default values

This command sets the specified I-variable or range of I-variables to the factor default value. Each I-variable has its own factory default; these are shown in the I-Variable Specification section.

Example:

```
I13=*
I100..199=*
I5188,3,100=*     ; Set I5188,I5288,I5388 to default
```

See Also:
Initialization (I) Variables (Computational Features)
I-Variable Specifications -- Default Values
On-line commands **I{constant}**, **I{constant}={expression}**

I{constant}=@I{constant}

Function: Set I-variable to address of another I-variable.

Scope: Global

Syntax: **I{constant}=@I{constant}**

where:

- the first **{constant}** is an integer from 0 to 8191 representing the number of the I-variable whose value is being set
- the second **{constant}** is an integer from 0 to 8191 representing the number of the I-variable whose address is being used to set the value of the first I-variable

This command permits the user to set the value of one I-variable to the address of another I-variable, without having to know that address. The main use of this command is to set the value of "address" I-variables Ixx03, Ixx04, Ixx05, and Isx93 to the address of an entry in the encoder conversion table, without having to know that address explicitly.

Encoder conversion table setup I-variables I8000 – I8191 reside at Turbo PMAC Y-addresses $003501 - $0035C0, respectively. The result of a particular conversion resides in the X-register of the same address as the last setup I-variable for that entry. This command permits the user to specify the address simply by using the number of the last I-variable in the entry.

Examples:

```
I103=@I8000  ; Set I103 to address of first line in ECT
I103         ; Query value of I103
$3501        ; Turbo PMAC reports value
I5193=@I8014 ; Set I5193 to address of 15th line in ECT
I5193        ; Query value of I5193
$350F        ; Turbo PMAC reports value
```

IDC

Function: Force active clock equal to ID-module clock

Scope: Global

Syntax: **IDC**

This command forces the active real-time clock in RAM to have the same time as the non-volatile real-time clock in the Option 18B ID-number and clock/calendar module. It copies the time value from the ID module into the active timer register in RAM, and reports this time back to the host computer. The time is reported in the international 24-hour clock format:

`{hh}:{mm}:{ss}`

where:

- {hh} is the 2-digit representation of the hour (00 <= {hh} <= 23)
- {mm} is the 2-digit representation of the minute (00 <= {mm} <= 59)
- {ss} is the 2-digit representation of the second (00 <= {ss} <= 59)

On power-up/reset, the time value in the Option 18B module is automatically copied into the timer register in RAM. From that point on, however, both timers increment independently, and can drift apart. The **IDC** command can be used periodically to force them back together.

See Also:
On-line commands **TIME**, **TIME={time}**, **TODAY**, **UPDATE**

IDNUMBER

Function: Report electronic identification number

Scope: Global

Syntax: **IDNUMBER**
IDNUM

This command causes Turbo PMAC to report the electronic identification number from the Option 18A ID-number module, or the Option 18B ID-number & clock/calendar module.

The identification number is reported as a hexadecimal 16-digit ASCII string, representing a 64-bit value. The first two hex digits represent the 8-bit checksum value for the module; these should match the checksum digits engraved on the case of the module. The last two hex digits represent the module class; these should match the class digits engraved on the case of the module (currently 01 for Option 18A, and 04 for Option 18B). The middle 12 hex digits represent the unique number for each module and board.

If no ID-number module is present, Turbo PMAC will return a 0.

The electronic identification number has no relationship to the serial number that is engraved on the circuit board.

This command is identical to the **SID** command.

Example:
```
IDNUMBER
374A256E9014D101
```

INC

Function: Specify Incremental Move Mode

Scope: Coordinate-system specific

Syntax: **INC**
INC({axis}[,{axis}...])

where:

- **{axis}** is a letter (X, Y, Z, A, B, C, U, V, W) representing the axis to be specified, or the character R to specify radial vector mode

No spaces are permitted in this command.

The **INC** command without arguments causes all subsequent positions for all axes in position motion commands to be treated as incremental distances. An **INC** statement with arguments causes the specified axes to be in incremental mode, and all others stay the way they were before. The default axis specification is absolute.

If 'R' is specified as one of the "axes", the I, J, and K terms of the circular move radius vector specification will be specified in incremental form (i.e. as a vector from the move start point, not from the origin). An **INC** command without any arguments does not affect this vector specification. The default vector specification is incremental.

If a motion program buffer is open when this command is sent to Turbo PMAC, it will be entered into the buffer as a program statement.

Example:

```
INC(A,B,C)..        ; A, B, and C axes made incremental other axes and rad vector left as is
INC.........        ; All axes made incremental -- radius vector left as is
INC(R).....         ; Radius vector made incremental -- all axes left as is
```

See Also:

Circular Moves (Writing a Motion Program)
On-line command **ABS**
Program commands **ABS**, **INC**

J!

Function: Adjust motor commanded position to nearest integer count

Scope: Motor specific

Syntax: **J!**

This command causes the addressed motor, if the desired velocity is zero, to adjust its commanded position to the nearest integer count value. It can be valuable to stop dithering if the motor is stopped with its commanded position at a fractional value, and integral gain is causing oscillation about the commanded position.

> ***Note:***
>
> A half-count of true deadband created in the servo loop with Ixx64=-16 and Ixx65=8 can serve the same purpose without the need for issuing a command.

Example:

```
OPEN PLC 7 CLEAR
IF (M50=1)..            ; Condition to start branch
    CMD"#1J/"           ; Tell motor to stop
    WHILE (M133=0)      ; Wait for desired velocity to reach zero
    ENDWHILE
    CMD"#1J!"           ; Adjust command position to integer value
    M50=0..             ; To keep from repeated execution
ENDIF
```

See Also:

On-line commands **J/**, **J={constant}**

J+

Function: Jog Positive

Scope: Motor specific

Syntax: **J+**

This command causes the addressed motor to jog in the positive direction indefinitely. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
J+ ..... ; Jog addressed motor positive
#7J+ ..... ; Jog Motor 7 positive
#2J+#3J+..... ; Jog Motors 2 and 3 positive
```

See Also:
Control Panel Port JOG+/ Input (Connecting Turbo PMAC to the Machine)
JPAN Connector Pin 6
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J-**, **J/**, **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**

J-

Function: Jog Negative

Scope: Motor specific

Syntax: **J-**

This command causes the addressed motor to jog in the negative direction indefinitely. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
J- ..... ; Jog addressed motor negative
#5J- ..... ; Jog Motor 5 negative
#3J-#4J-..... ; Jog Motors 3 and 4 negative
```

See Also:
Control Panel Port JOG-/ Input (Connecting Turbo PMAC to the Machine)
JPAN Connector Pin 4
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J+**, **J/**, **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**

J/

Function: Jog Stop

Scope: Motor specific

Syntax: **J/**

This command causes the addressed motor to stop jogging. It also restores position control if the motor's servo loop has been opened (enabled or killed), with the new commanded position set equal to the actual position at the time of the **J/** command. Jogging deceleration is determined by the values of Ixx19-Ixx21 in force at the time of this command.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
#1J+ ..... ; Jog Motor 1 positive
J/ ..... ; Stop jogging Motor 1
O5 ..... ; Open-loop output of 5% on Motor 1
O0 ..... ; Open loop output of 0%
J/ ..... ; Restore closed-loop control
K ..... ; Kill output
J/ ..... ; Restore closed-loop control
```

See Also:
Control Panel Port JOG+/, JOG-/ Inputs (Connecting Turbo PMAC to the Machine)
JPAN Connector Pin 4, 6
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **<CTRL-A>**, **A**, **J+**, **J-**, **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**, **K**, **O{constant}**

J:{constant}

Function: Jog Relative to Commanded Position

Scope: Motor specific

Syntax: **J:{constant}**

where:

- **{constant}** is a floating point value specifying the distance to jog, in counts.

This command causes a motor to jog the distance specified by **{constant}** relative to the present commanded position. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command. Compare to **J^{constant}**, which is a jog relative to the present actual position.

A variable incremental jog command can be executed with the **J:*** command

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
#1HM ..... ; Do homing search move on Motor 1
J:2000..... ; Jog a distance of 2000 counts (to 2000 counts)
J:2000..... ; Jog a distance of 2000 counts (to 4000 counts)
```

See Also:
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J+**, **J-**, **J/**, **J=**, **J={constant}**, **J^{constant}**

J:*

Function: Jog to specified variable distance from present commanded position

Scope: Motor specific

Syntax: **J:***

This command causes the addressed motor to jog the distance specified in the motor's "variable jog position/distance" register relative to the present commanded position. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command. Compare to **J^*** , which is a jog relative to the present actual position.

The variable jog position/distance register is a floating-point register with units of counts. The register is located at Turbo PMAC address L:$0000D7 for motor 1, L:$000157 for motor 2, etc. It is best accessed with a floating-point M-variable; the suggested M-variable is Mxx72. The usual procedure is to write the destination position to this register by assigning a value to the M-variable, then issuing the **J:*** command.

Turbo PMAC will reject this command if the motor is in a coordinate sytem that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
M172->L:$0000D7     ; Define #1 variable jog position/distance reg.
#1HMZ.....          ; Declare present position to be zero
M172=3000 .....     ; Assign distance value to register
#1J:*.....          ; Jog Motor 1 this distance; end cmd. pos. will be 3000
#1J:*.....          ; Jog Motor 1 this distance; end cmd. pos. will be 6000
M172=P1*SIN(P2)     ; Assign new distance value to register
#1J:*.....          ; Jog Motor 1 this distance
#1J=.....           ; Return to prejog target position
```

See Also:

Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
Memory map registers L:$0000D7, L:$000157, etc.
Suggested M-variable definitions M172, M272, etc.
On-line commands **J=**, **J={constant}**, **J=***, **J^***

J=

Function: Jog to Prejog Position

Scope: Motor specific

Syntax: **J=**

This command causes the addressed motor to jog to the last pre-jog and pre-handwheel-move position (the most recent programmed position). Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

If the **H** feedhold command has been used to suspend program execution, and one or more motors jogged away from the stop position, the **J=** command must be used to return the motor(s) back to the stop position before program execution can be resumed.

The **J=** command can also be useful if a program has been aborted in the middle of a move, because it will move the motor to the programmed move end position (provided Isx13=0 so Turbo PMAC is not in segmentation mode), so the program may be resumed properly from that point.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
&1Q.....            ; Stop motion program at end of move
#1J+.....           ; Jog Motor 1 away from this position
J/.....             ; Stop jogging
J=.....             ; Jog back to position where program quit
R.....              ; Resume motion program

&1A.....            ; Stop motion program in middle of move
#1J=#2J=#3J=        ; Move all motors to original move end position
R.....              ; Resume motion program
```

See Also:

Control Panel Port PREJ/ Input (Connecting Turbo PMAC to the Machine)
JPAN Connector Pin 7
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J+**, **J-**, **J/**, **J={constant}**, **J:{constant}**, **J^{constant}**

J={constant}

Function: Jog to specified position

Scope: Motor specific

Syntax: **J={constant}**

where:

- **{constant}** is a floating point value specifying the location to which to jog, in encoder counts.

This command causes the addressed motor to jog to the position specified by **{constant}**. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

A variable jog-to-position can be executed with the **J=*** command

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
J=0.....          ; Jog addressed motor to position 0
#4J=5000.....     ; Jog Motor 4 to 5000 counts
#8J=-32000....    ; Jog Motor 8 to -32000 counts
```

See Also:

Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J+**, **J-**, **J/**, **J=**, **J:{constant}**, **J^{constant}**, **J=***, **J:***, **J^***

J=*

Function: Jog to specified variable position

Scope: Motor specific

Syntax: **J=***

This command causes the addressed motor to jog to the position specified in the motor's "variable jog position/distance" register. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

The variable jog position/distance register is a floating-point register with units of counts. The register is located at Turbo PMAC address L:$0000D7 for motor 1, L:$000157 for motor 2, etc. It is best accessed with a floating-point M-variable; the suggested M-variable is Mxx72. The usual procedure is to write the destination position to this register by assigning a value to the M-variable, then issuing the **J=*** command.

Virtually the same result can be obtained by writing to the motor "target position" register and issuing the **J=** command. However, using the **J=*** command permits you to return to the real target position afterwards without having to restore the target position register. Also, the **J=*** command uses a register whose value is scaled in counts, not fractions of a count.

Turbo PMAC will reject this command if the motor is in a coordinate sytem that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
M172->L:$0000D7     ; Define #1 variable jog position/distance reg.
M172=3000 .....     ; Assign position value to register
#1J=*.............  ; Jog Motor 1 to this position
M172=P1*SIN(P2)     ; Assign new position value to register
#1J=*.............  ; Jog Motor 1 to this position
```

#1J=..... ; Return to prejog target position

See Also:
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
Memory map registers L:$0000D7, L:$0000157, etc.
Suggested M-variable definitions M172, M272, etc.
On-line commands **J=**, **J={constant}**, **J:***, **J^***

J=={constant}

Function: Jog to specified motor position and make that position the "pre-jog" position

Scope: Motor specific

Syntax: **J=={constant}**

where:

- **{constant}** is a floating point value specifying the location to which to jog, in encoder counts

This command causes the addressed motor to jog the position specified by **{constant}**. It also makes this position the "pre-jog" position, so it will be the destination of subsequent **J=** commands. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

#1J==10000.. ; Jog Motor 1 to 10000 counts and make that the pre-jog position.
J+..... ; Jog indefinitely in the positive direction
J=..... ; Return to 10000 counts

See Also:
Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J=**, **J={constant}**, **J=***, **J^***

J^{constant}

Function: Jog Relative to Actual Position

Scope: Motor specific

Syntax: **J^{constant}**

where:

- **{constant}** is a floating point value specifying the distance to jog, in counts.

This causes a motor to jog the distance specified by **{constant}** relative to the present actual position. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command. Compare to **J:{constant}**, which is a jog relative to the present commanded position.

Usually the **J:{constant}** command is more useful, because its destination is not dependent on the following error at the instant of the command. The **J^0** command can be useful for swallowing any existing following error.

A variable incremental jog can be executed with the **J^*** command

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
#1HM.....          ; Do homing search move on Motor 1
J^2000.....        ; Jog a distance of 2000 counts from actual position
.....              ; If actual was -5 cts, new command pos is 1995 cts
J^2000.....        ; Jog a distance of 2000 counts from actual position
.....              ; If actual was 1992 cts, new cmd pos is 3992 cts
```

See Also:

Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
On-line commands **J+**, **J-**, **J/**, **J=**, **J={constant}**, **J:{constant}**, **J=***, **J:***, **J^***

J^*

Function: Jog to specified variable distance from present actual position

Scope: Motor specific

Syntax: **J^***

This command causes the addressed motor to jog the distance specified in the motor's "variable jog position/distance" register relative to the present actual position. Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command. Compare to **J:*** , which is a jog relative to the present commanded position.

The variable jog position/distance register is a floating-point register with units of counts. The register is located at Turbo PMAC address L:$0000D7 for motor 1, L:$000157 for motor 2, etc. It is best accessed with a floating-point M-variable; the suggested M-variable is Mxx72. The usual procedure is to write the destination position to this register by assigning a value to the M-variable, then issuing the **J^*** command.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
M172->L:$0000D7    ; Define #1 variable jog position/distance reg.
#1HMZ.....         ; Declare present position to be zero
M172=3000 .....    ; Assign distance value to register
#1J^*.....         ; Jog Motor 1 this distance;  if following error at command was 3,
.....              ;  end cmd. pos. will be 2997
#1J^*.....         ; Jog Motor 1 this distance; if following error at command was 2, end cmd.
.....              ; pos. will be 5995
M172=P1*SIN(P2)    ; Assign new distance value to register
#1J^*.....         ; Jog Motor 1 this distance
#1J=.....          ; Return to prejog target position
```

See Also:

Jogging Moves (Basic Motor Moves)
I-variables Ixx19-Ixx22
Memory map registers L:$0000D7, L:$000157, etc.
Suggested M-variable definitions M172, M272, etc.
On-line commands **J=**, **J={constant}**, **J=***, **J^***

{jog command}^{constant}

Function: Jog until trigger

Scope: Motor specific

Syntax:
```
J=^{constant}
J={constant}^{constant}
J:{constant}^{constant}
J^{constant}^{constant}
J=*^{constant}
J:*^{constant}
J^*^{constant}
```

where:

- **{constant}** after the ^ is a floating point value specifying the distance from the trigger to which to jog after the trigger is found, in encoder counts

This command format permits a jog-until-trigger function. When the **^{constant}** structure is added to any definite jog command, the jog move can be interrupted by a pre-defined trigger condition, and the motor will move to a point relative to the trigger position as specified by the final value in the command. The "indefinite" jog commands **J+** and **J-** cannot be turned into jog-until-trigger moves.

Jog-until-trigger moves are very similar to homing search moves, except they have a definite end position in the absence of a trigger, and they do not change the motor zero position. In addition, in the absence of a trigger, the move will simply stop at the first destination.

Trigger Condition: The trigger condition for a jog-until-trigger move can either be an input flag, or a warning following error condition for the motor. If bit 1 of Ixx97 is 0 (the default), the trigger is a transition of an input flag and/or encoder index channel from the set defined for the motor by Ixx25. Encoder/flag variables 2 and 3 (e.g. I912 and I913) define which edges of which input signals create the trigger.

If bit 1 of Ixx97 is 1, the trigger is the warning following error status bit of the motor becoming true. Ixx12 for the motor sets the error threshold for this condition.

Trigger Position: The trigger position can either be the hardware-captured position for the, or a software-read position. If bit 0 of Ixx97 is 0 (the default), the encoder position latched by the trigger in Turbo PMAC's DSPGATE hardware is used as the trigger position. This is the most accurate option because it uses the position at the moment of the trigger, but it can only be used with incremental encoder feedback brought in on the same channel number as the triggering flag set. This option cannot be used for other types of feedback, or for triggering on following error.

If bit 0 of Ixx97 is 1, Turbo PMAC reads the present sensor position after it sees the trigger. This can be used with any type of feedback and either trigger condition, but can be less accurate than the hardware capture because of software delays.

Trigger Occurrence: The "trigger move" bit, bit 7 of the second motor status word (Y:$0000C0, Y:$000140, etc.) is set to 1 at the beginning of the pre-trigger move, and cleared to 0 only when the trigger is found. It can be checked at any time, including after the move, to see if the trigger has occurred.

Jogging acceleration and velocity are determined by the values of Ixx19-Ixx22 in force at the time of this command.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
#1J=^1000...        ; Jog to pre-jog position in the absence of a trigger, but if trigger is found,
.....               ; jog to +1000 cts from trigger.
#2J:5000^-100       ; Jog 5000 counts in the positive direction in the absence of a trigger,
.....               ; but if trigger is found, jog to -100 cts from trigger position.
#3J=20000^0..; Jog to 20000 counts in the absence of a trigger, but if  trigger is found,
.....               ; return to trigger position.
```

See Also:

Jogging Moves (Basic Motor Moves)
I-variables Ixx97, Ixx19-Ixx22, Ixx25, I7mn2, I7mn3
On-line commands **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**,
..... **J=***, **J:***, **J^***
Program commands **{axis}{data}^{data}**

K

Function: Kill motor output

Scope: Motor specific

Syntax: **K**

This command causes Turbo PMAC "kill" the outputs for the addressed motor. The servo loop is disabled, the position/velocity servo-loop output is set to zero (Ixx29 and/or Ixx79 offsets are still in effect), and the AENA output for the motor is taken to the disable state (polarity is determined by E17 on PMAC(1)-style channels).

Closed-loop control of this motor can be resumed with a **J/** command. The **A** command will re-establish closed-loop control for all motors in the addressed coordinate system, and the **<CTRL-A>** command will do so for all motors on Turbo PMAC.

The action on a **K** command is equivalent to what Turbo PMAC does automatically to the motor on an amplifier fault or a fatal following error fault.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3). The program must be stopped first, usually with an **A** command. However, the global **<CTRL-K>** command will kill all motors immediately, regardless of whether any are running motion programs.

Example:

```
K ..... ; Kill the addressed motor
#1K..... ; Kill Motor 1
J/ ..... ; Re-establish closed-loop control of Motor 1
```

See Also:

Amplifier Fault, Following Error Limits, Stop Commands (Making Your Application Safe)
I-variables Ixx29, Ixx79
On-line commands **<CTRL-A>**, **<CTRL-K>**, **A**, **Q**, **H**, **J/**
Jumpers E17, E17A-E17H

LEARN

Function: Learn present commanded position

Scope: Coordinate-system specific

Syntax: **LEARN[({axis}[,{axis}...]]**
[({axis}[,{axis}...]]

where:

- **{axis}** (optional) is a character (X, Y, Z, A, B, C, U, V, W) specifying which axis' position is to be learned. If none are listed, the positions for all axes are learned.

No spaces are permitted in this command.

This command causes Turbo PMAC to add a line to the end of the open motion program buffer containing axis position commands equal to the current commanded positions for some or all of the motors defined in the addressed coordinate system. In this way Turbo PMAC can "learn" a sequence of points to be repeated by subsequent execution of the motion program.

If no motion program buffer is open, or if the motion program buffer that is open has been opened from another port, Turbo PMAC will reject this command and report an error (ERR003 if I6=1 or 3).

Turbo PMAC effectively performs a **PMATCH** function, reading motor commanded positions and inverting the axis definition equations to compute axis positions.

If axis names are specified in the **LEARN** command, only position commands for those axes are used in the line added to the motion program. If no axis names are specified in the learn command, position commands for all nine possible axis names are used in the line added to the motion program. The position command for an axis with no motor attached ("phantom" axis) will be zero.

> ***Note:***
>
> If a motor is closed loop, the learned position will differ from the actual position by the amount of the position following error because commanded position is used. If a motor is open-loop or killed, Turbo PMAC automatically sets motor commanded position equal to motor actual position, so the **LEARN** function can be used regardless of the state of the motor.

Example:

```
&1 ........          ; Address coordinate system 1
#1->10000X..         ; Define motor 1 in C.S. 1
#2->10000Y....       ; Define motor 2 in C.S. 1
OPEN PROG 1 CLEAR    Prepare program buffer for entry
F10 TA200 TS50       ; Enter required non-move commands {move motors to a position,
.....                ; e.g. #1 to 13450 commanded, #2 to 29317 commanded}
LEARN(X,Y)....       ; Tell PMAC to learn these positions
X1.345 Y2.9317       ; This is the line that PMAC adds to PROG 1 {move motors to new
.....                ; position, e.g. #1 to 16752 cmd., #2 to 34726 cmd}
LEARN.....           ; Tell PMAC to learn positions
A0 B0 C0 U0 V0 W0 X1.6752 Y3.4726 Z0
.....                ; PMAC adds positions for all axes to PROG 1
```

See Also:

Learning a Motion Program (Writing a Motion Program)
On-line command **PMATCH**

LIST

Function: List the contents of the currently opened buffer.

Scope: Global

Syntax: **LIST**

This command causes Turbo PMAC to report the contents of the currently opened buffer (PLC, PROG, or ROT) to the host. If no buffer is open, or if the buffer that is open has been opened from another port, Turbo PMAC will report an error (ERR003 if I6=1 or 3). Note that what is reported will not include any **OPEN**, **CLEAR**, or **CLOSE** statements (since these are *not* program commands).

You can list an unopened buffer by specifying the buffer name in the list command (e.g. **LIST PROG 1**). See further **LIST** commands, below.

Example:

```
OPEN PROG 1.              ; Open buffer for entry
LIST........              ; Request listing of open buffer
LINEAR......              ; Turbo PMAC reports contents of open buffer
F10
X20 Y20
X0 Y0
RETURN
CLOSE.......              ; Close buffer
LIST........              ; Request listing of open buffer
<BELL>ERR003              ; Turbo PMAC reports error because
...........               ; no open buffer
```

See Also:

On-line commands **OPEN**, **CLOSE**, **LIST PLC**, **LIST PROGRAM**

LIST BLCOMP

Function: List contents of addressed motor's backlash compensation table

Scope: Motor specific

Syntax: **LIST BLCOMP**

This command causes Turbo PMAC to report to the host the contents of the backlash compensation table belonging to the addressed motor. The values are reported in decimal ASCII form, multiple values to a line, with individual values separated by spaces.

The **LIST BLCOMP DEF** command should be used to report the header information for this table.

If there is no table for the addressed motor, Turbo PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example:

```
LIST BLCOMP..                              ; Request contents of backlash comp table
9 17 -3 6 35 87 65 24 18 -9 -16 -34        ; PMAC responds
-7 12 -3 -8 32 44 16 0 -20 -5 0            ; Continued response
```

See Also:

Backlash Compensation Tables (Setting Up a Motor)

On-line commands **DEFINE BLCOMP**, **DELETE BLCOMP**, **LIST BLCOMP DEF**

LIST BLCOMP DEF

Function: List definition of addressed motor's backlash compensation table

Scope: Motor specific

Syntax: **LIST BLCOMP DEF**

This command causes Turbo PMAC to report to the host the definition of the backlash compensation table that belongs to the addressed motor. The definition reported consists of the two items established in the **DEFINE BLCOMP** command that set up the motor:

1. The number of entries in the table;
2. The span of the table in counts of the motor.

If there is no table for the addressed motor, Turbo PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example:

```
LIST BLCOMP DEF          ; Request def of addressed motor backlash
............             ; comp table
100,100000..             ; Turbo PMAC responds; 100 entries in table,
.....                    ; span is 100,000 counts
```

See Also:

Backlash Compensation Tables (Setting Up a Motor)

On-line commands **DEFINE BLCOMP**, **DELETE BLCOMP**, **LIST BLCOMP**

LIST COMP

Function: List contents of addressed motor's compensation table

Scope: Motor specific

Syntax: **LIST COMP**

This command causes Turbo PMAC to report to the host the contents of the compensation table belonging to the addressed motor. The values are reported in decimal ASCII form, multiple values to a line, with individual values separated by spaces.

The **LIST COMP DEF** command should be used to report the header information for this table.

If there is no table for the addressed motor, Turbo PMAC will reject the command (reporting ERR003 if I6=1 or 3).

> ***Note:***
>
> The compensation table "belonging" to this motor may not affect this motor's position or be affected by it.

Example:

```
LIST COMP .....                              ; Request contents of compensation table
9 17 -3 6 35 87 65 24 18 -9 -16 -34 ; PMAC responds
-7 12 -3 -8 32 44 16 0 -20 -5 0      ; Continued response
```

See Also:

Leadscrew Compensation Tables (Setting Up a Motor)

On-line commands **DEFINE COMP**, **DELETE COMP**, **LIST COMP DEF**

LIST COMP DEF

Function: List definition of addressed motor's compensation table

Scope: Motor specific

Syntax: `LIST COMP DEF`

This command causes Turbo PMAC to report to the host the definition of the compensation table that belongs to the addressed motor. The definition reported consists of the four items established in the **`DEFINE COMP`** command that set up the motor (even if some of those items were not specified explicitly):

1. The number of entries in the table (number of rows and number of columns for a two-dimensional table)
2. The number of the motor whose position provides the source data for the table (both source motors for a two-dimensional table)
3. The number of the motor whose position is modified by the table
4. The span of the table in counts of the source motor (in both dimensions for a two-dimensional table)

If there is no table for the addressed motor, Turbo PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Note:

The compensation table "belonging" to this motor may not affect this motor's position or be affected by it.

Example:

```
LIST COMP DEF                          ; Request definition of compensation table
100,#2,#2,100000                       ; Turbo PMAC responds; 100 entries in table, Motor 2 is
.............................          ; source and target, span is 100,000 counts
#3 LIST COMP DEF                       ; Request definition of comp table belonging to Motor 3
10,20,#4,#5,#6,50000,100000            ; Turbo PMAC responds A 2D 10x20 table, span of
.............................          ; 50Kx100K counts #4 & #5 as source, #6 as target
```

See Also:

Leadscrew Compensation Tables (Setting Up a Motor)

On-line commands **`DEFINE COMP`**, **`DELETE COMP`**, **`LIST COMP`**

LIST FORWARD

Function: Report contents of forward-kinematic program buffer.

Scope: Coordinate-system specific

Syntax: **`LIST FORWARD`**
`LIS FWD`

This command causes Turbo PMAC to report the contents of the forward-kinematic program buffer for the addressed coordinate system to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. `ENDW`). If I9 is 1 or 3, the contents are reported in long form (e.g. `ENDWHILE`).

If the forward kinematic program requested by this command does not exist in Turbo PMAC, Turbo PMAC will reject this command (reporting an `ERR003` if I6 is 1 or 3).

Forward kinematic programs can be protected by password. If there is a password for the Turbo PMAC, and the password has not been given, Turbo PMAC will reject this command (reporting an `ERR002` if I6 is 1 or 3).

See Also:
Kinematic Calculations
I-variable Isx50
On-line commands **LIST INVERSE**, **OPEN FORWARD**, **OPEN INVERSE**

LIST GATHER

Function: Report contents of the data gathering buffer.

Scope: Global

Syntax: **LIST GATHER [{start}] [,{length}]**
LIS GAT [{start}] [,{length}]

where:

- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (0 is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

This command causes Turbo PMAC to report the contents of the data gathering buffer to the host. The data is reported as 48-bit long words in hexadecimal format (12 characters per word) separated by spaces, 16 long words per line.

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point. The starting point is referenced to the beginning memory location of the buffer. If gathering has rolled over the buffer, as it can with I5000 = 1 or 3, the reported contents may not come in the same order as they were gathered.

Example:

```
LIST GATHER..          ; reports the whole buffer
LIST GATHER 256        ; skips the first 256 long words
LIST GATHER 0,32       ; reports the first 32 words
LIST GATHER ,32        ; does the same as above
LIST GATHER 64,128     ; skips the first 64 words, reports the next 128
```

See Also:
Data Gathering Function (Analysis Features)
I-variables I5049, I5050 and I5051, I5001-I5048.
On-line commands **GATHER**, **ENDGATHER**, **DEFINE GATHER**
Gathering and Plotting (Turbo PMAC Executive Program Manual)

LIST INVERSE

Function: Report contents of inverse-kinematic program buffer.

Scope: Coordinate-system specific

Syntax: LIST INVERSE

LIS INV

This command causes Turbo PMAC to report the contents of the inverse-kinematic program buffer for the addressed coordinate system to the host.

The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. ENDW). If I9 is 1 or 3, the contents are reported in long form (e.g. ENDWHILE).

If the inverse kinematic program requested by this command does not exist in Turbo PMAC, Turbo PMAC will reject this command (reporting an ERR003 if I6 is 1 or 3).

Inverse kinematic programs can be protected by password. If there is a password for the Turbo PMAC, and the password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6 is 1 or 3).

See Also:
Kinematic Calculations
I-variable Isx50
On-line commands **LIST FORWARD**, **OPEN FORWARD**, **OPEN INVERSE**

LIST LDS

Function: List Linking Addresses of Ladder Functions

Scope: Global

Syntax: **LIST LDS**

This command causes Turbo PMAC to list the addresses of the internal routines that the special ladder-logic PLC cross-compiler needs to properly compile and link programs. This command is used automatically by the cross-compiler; a user only needs it directly for special debugging.

See Also:
On-line commands **LIST LINK**

LIST LINK

Function: List Linking Addresses of Internal Turbo PMAC Routines

Scope: Global

Syntax: **LIST LINK**

This command causes Turbo PMAC to list the addresses of the internal routines that the PLC cross-compiler needs to properly compile and link its programs. This command is used automatically by the PLC cross-compiler in the Executive program.

For the standalone DOS cross-compiler, the ASCII characters of Turbo PMAC's response to this command must be contained in a file named LISTLINK.TXT in the same directory and subdirectory as the cross-compiler. Each separate version of Turbo PMAC's firmware potentially has different addresses for these routines, so a new LISTLINK.TXT file must be created any time the Turbo PMAC firmware is updated, even for a minor change such as from V2.01A to V2.01B.

Example:

```
LIST LINK.....                          ; Request linking addresses
004532 004A97 005619 005F21 0062FE 0063A4
............                            ; Turbo PMAC responds
```

See Also:
Compiled PLCs (Writing a PLC Program)

LIST PC

Function: List Program at Program Counter

Scope: Coordinate-system specific

Syntax: **LIST PC[,[{constant}]]**

where:

- **{constant}** is a positive integer representing the number of words in the program to be listed

This command causes Turbo PMAC to list the program line(s) that it is (are) about to calculate in the addressed coordinate system, with the first line preceded by the program number and each line preceded by the address offset. **LIST PC** just lists the next line to be calculated. **LIST PC,** lists from the next line to be calculated to the end of the program.

LIST PC,{constant} lists the specified address range size starting at the next line to be calculated. To see the current line of execution, use the **LIST PE** command.

Because Turbo PMAC calculates ahead in a continuous sequence of moves, the **LIST PC** (Program Calculation) command will in general return a program line further down in the program than **LIST PE** will.

If the coordinate system is not pointing to any motion program, Turbo PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example:

```
LIST PC.....        ; List next line to be calculated
P1:22:X10Y20        ; Turbo PMAC responds
LIST PC,4 .....     ; List next 4 words of program to be calculated
P1:22:X10Y20        ; Turbo PMAC responds
24:X15Y30
LIST PC,.....       ; List rest of program
P1:22:X10Y20        ; Turbo PMAC responds
24:X15Y30
26:M1=0
28:RETURN
```

See Also:

On-line commands **B{constant}**, **LIST**, **PC**, **LIST PE**, **PE**

LIST PE

Function: List Program at Program Execution

Scope: Coordinate-system specific

Syntax: **LIST PE[,[{constant}]]**

where:

- **{constant}** is a positive integer representing the number of words in the program to be listed

This command causes Turbo PMAC to list the program line(s) starting with the line containing the move that it is currently executing in the addressed coordinate system, with the first line preceded by the program number, and each line preceded by the address offset.

Because Turbo PMAC calculates ahead in a continuous sequence of moves, the **LIST PC** (Program Calculation) command will in general return a program line further down in the program than **LIST PE** will.

LIST PE returns only the currently executing line. **LIST PE,** returns from the currently executing line to the end of the program. **LIST PE,{constant}** returns the specified number of words in the program, starting at the currently executing line.

If the coordinate system is not pointing to any motion program, Turbo PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example:

```
LIST PE.....        ; List presently executing line
P5:35:X5Y30..       ; Turbo PMAC responds
LIST PE,4 .....     ; List 4 program words, starting with executing line
P5:35:X5Y30..       ; Turbo PMAC responds
37:X12Y32
LIST PE,.....       ; List rest of program, starting with executing line
P5:35:X5Y30..       ; Turbo PMAC responds
37:X12Y32
```

```
39:X0 Y10
41:RETURN
```

See Also:
On-line commands **B{constant}**, **LIST**, **LIST PC**, **PC**, **PE**

LIST PLC

Function: List the contents of the specified PLC program.

Scope: Global

Syntax: **LIST PLC{constant} [,[{start}]] [,[{length}]]**

where:

- **{constant}** is an integer from 0 to 31 representing the number of the PLC program
- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (the execution point is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

This command causes Turbo PMAC to report the contents of the specified uncompiled PLC program buffer to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. ENDW). If I9 is 1 or 3, the contents are reported in long form (e.g. ENDWHILE).

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

If the first comma is present, but no start point is specified, the listing will start from the next line to be executed in the PLC program. Because Turbo PMAC can only execute this command between PLC scans, this line will be the first to execute in the next scan. If the second comma is present, but no length is specified, the listing will continue to the end of the program.

If either **{start}**, **{length}**, or both, or just the comma, is included in the command, the listing of the program will include the buffer address offsets with each line.

PLCs 0-15 can be protected by password. If the PLC is protected by password, and the proper password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

Example:

```
LIST PLC 5
P1=0
WHILE (P1<1000)
P1=P1+1
ENDWHILE
RETURN
LIST PLC 5,0
0:P1=0
1:WHILE(P1<1000)
3:P1=P1+1
6:ENDWHILE
7:RETURN
LIST PLC 5,,1
1:WHILE(P1<1000)
LIST PLC 5,,
1:WHILE(P1<1000)
3:P1=P1+1
```

```
6:ENDWHILE
7:RETURN
```

See Also:
PLC Program Features
I-variables I3, I4, I9
On-line commands **LIST**, **LIST PROG**, **PASSWORD={string}**
Program Command Specification

LIST PROGRAM

Function: List the contents of the specified motion program.

Scope: Global

Syntax:
```
LIST PROGRAM {constant} [{start}] [,{length}]
LIST PROG {constant} [{start}] [,{length}]
```

where:

- **{constant}** is an integer from 1 to 32767 specifying the number of the motion program
- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (0 is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

This command causes Turbo PMAC to report the contents of the specified fixed motion program buffer (PROG) to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. *LIN*). If I9 is 1 or 3, the contents are reported in long form (e.g. *LINEAR*).

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported.

If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

If either **{start}**, **{length}**, or both, or just the comma, is included in the command, the listing of the program will include the buffer address offsets with each line. Having a listing with these offsets can be useful in conjunction with later use of the **PC** (Program-Counter) and **LIST PC** commands.

If the motion program requested by this command does not exist in Turbo PMAC, Turbo PMAC will reject this command (reporting an ERR003 if I6=1 or 3).

PROGs 1000-32767 can be protected by password. If the PROG is protected by password, and the proper password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

Example:
```
LIST PROG 9.              ; Request listing of all of motion program 9
LINEAR......              ; Turbo PMAC responds
F10
X10Y10
X0Y0
RETURN
LIST PROG 9,              ; Request listing of program w/ address offsets
0:LINEAR
1:F10
2:X10Y10....              ; Note that a 2-axis command takes 2 addresses
```

```
4:X0Y0
6:RETURN
LIST PROG 9,4              ; Request listing starting at address 4
4:X0Y0
6:RETURN
LIST PROG 9,2,4            ; Request listing starting at 2, 4 words long
2:X10Y10
4:X0Y0
LIST PROG 9,,2             ; Request listing starting at top, 2 words long
0:LINEAR
1:F10
```

See Also:
Writing a Motion Program
I-variables I3, I4, I9
On-line commands **LIST**, **PC**, **LIST PC**., **PASSWORD={string}**.
Program Command Specification

LIST ROTARY

Function: List contents of addressed coordinate system's rotary program buffer

Scope: Coordinate-system specific

Syntax: **LIST ROTARY [{start}] [,{length}]**
LIST ROT [{start}] [,{length}]

where:

- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (0 is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

This command causes Turbo PMAC to report the contents of the rotary motion program buffer for the addressed coordinate system to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. *LIN*). If I9 is 1 or 3, the contents are reported in long form (e.g. *LINEAR*).

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

If either **{start}**, **{length}**, or both, or just the comma, is included in the command, the listing of the program will include the buffer address offsets with each line. Having a listing with these offsets can be useful in conjunction with later use of the **PC** (Program-Counter) and **LIST PC** commands.

If the loading of the rotary buffer has caused the buffer to "wrap around" and re-use the beginning of the buffer, the listing will start relative to this new top of the buffer (even if there are previously loaded, and still unexecuted, lines at the bottom of the buffer.

See Also:
Writing a Motion Program
I-variables I3, I4, I9
On-line commands **LIST**, **PC**, **LIST PE**,
Program Command Specification

LIST TCOMP

Function: List contents of addressed motor's torque compensation table

Scope: Motor specific

Syntax: **LIST TCOMP**

This command causes Turbo PMAC to report to the host the contents of the torque compensation table belonging to the addressed motor. The values are reported in decimal ASCII form, multiple values to a line, with individual values separated by spaces.

The **LIST TCOMP DEF** command should be used to report the header information for this table.

If there is no table for the addressed motor, Turbo PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example:

```
LIST TCOMP....                              ; Request contents of torque comp table
9 17 -3 6 35 87 65 24 18 -9 -16 -34         ; PMAC responds
-7 12 -3 -8 32 44 16 0 -20 -5 0             ; Continued response
```

See Also:

Torque Compensation Tables (Setting Up a Motor)

On-line commands **DEFINE TCOMP**, **DELETE TCOMP**, **LIST TCOMP DEF**

LIST TCOMP DEF

Function: List definition of addressed motor's torque compensation table

Scope: Motor specific

Syntax: **LIST TCOMP DEF**

This command causes Turbo PMAC to report to the host the definition of the torque compensation table that belongs to the addressed motor. The definition reported consists of the two items established in the **DEFINE TCOMP** command that set up the motor:

1. The number of entries in the table;
2. The span of the table in counts of the motor.

If there is no table for the addressed motor, Turbo PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example:

```
LIST TCOMP DEF              ; Request def of addressed motor torque comp table
100,100000..                ; Turbo PMAC responds; 100 entries in table, span is 100,000 counts
```

See Also:

Torque Compensation Tables (Setting Up a Motor)

On-line commands **DEFINE TCOMP**, **DELETE TCOMP**, **LIST TCOMP**

LOCK{constant},P{constant}

Function: Check/set process locking bit

Scope: Global

Syntax: **LOCK{constant},P{constant}**

where:

- the first **{constant}** is an integer from 0 to 7 representing the number of the locking bit
- the second **{constant}** is an integer from 0 to 8191 specifying the number of the P-variable used to report the status of the locking bit

The **LOCK** command permits the user to check and possibly take possession of one of the eight process locking bits in Turbo PMAC. These locking bits can prevent conflicts between tasks of different priorities attempting to manipulate the same register. On-line commands and PLCs 1 – 31 are background tasks; motion programs and PLC 0 are higher-priority foreground tasks.

When the **LOCK** command is invoked, the P-variable specified in the command takes the value of the locking bit immediately before the command is invoked. It takes a value of 0 if the locking bit was not set before the command (meaning the process is available for this task); it takes a value of 1 if the locking bit was set before the command (meaning the process is not available for this task).

The locking bit itself is always set to 1 at the end of a **LOCK** command. It will stay at 1 until cleared by an **UNLOCK** command.

The status of locking bits 0 – 7 is reported as bits 4 – 11, respectively, of I4904.

If a motion program buffer or a PLC program buffer is open when this command is issued, this command will be entered into that buffer as a program command for future execution; it will not be treated as an on-line command.

Example:

```
LOCK4,P10          ; Check status of locking bit 4
P10                ; Ask for result
1                  ; Turbo PMAC reports that process 4 is locked
LOCK4,P10          ; Try again
P10                ; Ask for result
0                  ; Turbo PMAC reports that process 4 is available
M1=M1^1            ; Invert Machine Output 1
UNLOCK4            ; Release process 4 for other tasks
```

M{constant}

Function: Report the current M-variable value(s).

Scope: Global

Syntax: **M{constant}[..{constant}]**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

M{constant},{constant},{constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first M-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number M-variables whose value is to be reported;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each M-variable whose value is to be reported;

This command causes Turbo PMAC to report the current value of the specified M-variable, or range or set of M-Variables. It does not cause Turbo PMAC to report the definition (address) of the M-Variables; that is done with the **M{constant}->** command.

> ***Note:***
>
> If a motion program buffer (including a rotary buffer) is open when this command is sent to Turbo PMAC it will be entered into the buffer for later execution, to be interpreted as an M-code subroutine call.

Example:

```
M0 ..................    ; Host asks for value
3548976 .........        ; Turbo PMAC's response
M165
5.75
M1..3
1
0
1
M103,4,100..             ; Request for values of M103, M203, M303, M403
34221.......             ; Turbo PMAC responds with 4 lines
29726
-38657
47
```

See Also:

M-Variables (Computational Features)
On-line commands **M{data}={expression}**, **M{constant}->**
Program commands **M{data}**, **M{data}={expression}**

M{data}={expression}

Function: Assign value to M-variable(s).

Scope: Global

Syntax: **M{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the M-variable number;
- **{expression}** contains the value to be given to the specified M-variable

M{constant}..{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first M-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number of the last M-variable; it must be at least as great as the first **{constant}**
- the final **{constant}** contains the value to be given to the specified range of M-variables

M{constant},{constant},{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first M-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number M-variables whose value is to be set;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each M-variable whose value is to be set;
- the final **{constant}** contains the value to be given to the specified set of M-variables

This command assigns the value on the right side of the equals sign to the specified M-variable(s). It does not assign a *definition* (address) to the M-variable(s); that is done with the **M{constant}->{definition}** command.

If a motion or PLC program buffer is open when the single-variable form of this command is sent to Turbo PMAC, the command will be entered into the buffer for later execution. If a motion or PLC program buffer is open when the multiple-variable form of this command is sent, Turbo PMAC will reject the command with an error, reporting ERR003 if I6 is 1 or 3.

Example:

```
M1=1
M9=M9 & $20
M102=-16384
M1..8=0
M1,16,2=1          ; Sets M1, M3, M5 … , M31 to 1
```

See Also:

M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->{definition}**
Program commands **M{data}**, **M{data}={expression}**

M{constant}->

Function: Report current M-variable definition(s)

Scope: Global

Syntax: **M{constant}[..{constant}]->**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

This command causes Turbo PMAC to report the definition (address) of the specified M-variable or range of M-variables. It does not cause Turbo PMAC to report the *value* of the M-variable{s); that is done with the **M{constant}** command.

When I9 is 0 or 2, only the definition itself (e.g. `Y:$078802,0`) is returned. When I9 is 1or 3, the entire definition statement (e.g. `M11->Y:$078802,0`) is returned.

Example:

```
M1->.....          ; Host requests definition
Y:$FFC2,8 .....    ; Turbo PMAC's response
M101..103->
X:$C001,24,S
Y:$C003,8,16,S
X:$C003,24,S
```

See Also:

M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->{definition}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->*

Function: Self-Referenced M-Variable Definition

Scope: Global

Syntax: M{constant}[..{constant}]->*

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range.

This command causes Turbo PMAC to reference the specified M-variable or range of M-variables to its own definition word. If you just wish to use an M-variable as a flag, status bit, counter, or other simple variable, there is no need to find an open area of memory, because it is possible to use some of the definition space to hold the value. Simply define this form of the M-variable and you can use this M-variable much as you would a P-variable, except it only takes integer values in the range -2^{35} to $2^{35}-1$. Note, however, that the use of these self-referenced M-variables is less efficient for computation than using P or Q-variables.

When the definition is made, the value is automatically set to 0.

This command is also useful to "erase" an existing M-variable definition.

Example:

```
M100->*
M20..39->*
M0..8191->*.          ; This erases all existing M-variable definitions
.....                 ; It is a good idea to use this before loading new ones
```

See Also:

M-Variables (Computational Features)

On-line commands **M{constant}**, **M{constant}->**, **M{constant}->{definition}**, **M{constant}={expression}**

Program command **M{constant}={expression}**

M{constant}->D:{address}

Function: Long Fixed-Point M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->D[:]{address}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{address}** is an integer constant from $000000 to $FFFFFF (0 to 16,777,215 if specified in decimal).

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to a 48-bit double word (both X and Y memory; X more significant) at the specified location in Turbo PMAC's address space. The data is interpreted as a fixed-point signed (two's complement) integer.

The definition consists of the letter **D**, an optional colon (**:**), and the word address.

Memory locations for which this format is useful are labeled with **D:** in the memory map.

Example:

```
M161->D:$000088     ; Motor 1 desired position register specified in hex
M161->D134          ; Motor 1 desired position register specified in decimal
M162->D$8B          ; Motor 1 actual position register specified in hex
```

See Also:
M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->DP:{address}

Function: Dual-Ported RAM Fixed-Point M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->DP[:]{address}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{address}** is an integer constant from $000000 to $FFFFFF (0 to 16,777,215 if specified in decimal).

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to 32 bits of data in the low 16 bits of both X and Y memory at the specified location in Turbo PMAC's address space. The data is interpreted as a fixed-point signed (two's complement) integer.

The definition consists of the letters **DP**, an optional colon (**:**), and the word address.

This format is really only useful for dual-ported RAM locations $050000 to $05FFFF (Option 2 is required). With this format, the host can read or write to the corresponding location with a standard 32-bit integer data format. The data in the X word is the most significant word, which means on the host side the most significant word is in the higher of two consecutive addresses (standard Intel format).

Example:

```
M150->DP:$052000
M250->DP$052001
```

See Also:
M-Variables (Computational Features)
Dual-Ported RAM (Writing a Host Communications Program)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->F:{address}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->F:{address}

Function: Dual-Ported RAM Floating-Point M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->F[:]{address}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

- **{address}** is an integer constant from $000000 to $FFFFFF (0 to 16,777,215 if specified in decimal).

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to 32 bits of data in the low 16 bits of both X and Y memory at the specified location in Turbo PMAC's address space. The data is interpreted as a floating-point value with the IEEE single-precision (32-bit) format.

The definition consists of the letter **F**, an optional colon (**:**), and the word address.

This format is mainly useful only for dual-ported RAM locations $050000 to $05FFFF (Option 2 required). With this format, the host can read or write to the corresponding location with the standard IEEE 32-bit floating-point data format.

The IEEE 32-bit floating point format has the sign bit in bit 31 (MSB); the biased exponent in bits 30 to 23 (the exponent is this value minus 127), and the fraction in bits 22 to 0 (there is an implied 1 added to the fraction in the mantissa). The words are arranged in the standard Intel format.

Example:
```
M155->F:$054001
M255->F$05402
```

See Also:
M-Variables (Computational Features)
Dual-Ported RAM (Writing a Host Communications Program)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->DP:{address}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->L:{address}

Function: Long Word Floating-Point M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->L[:]{address}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{address}** is an integer constant from $000000 to $FFFFFF (0 to 16,777,215 if specified in decimal).

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to a long word (48 bits) of data -- both X and Y memory -- at the specified location in Turbo PMAC's address space. The data is interpreted as a floating-point value with Turbo PMAC's own 48-bit floating-point format.

The definition consists of the letter **L**, an optional colon (**:**), and the word address.

Memory locations for which this format is useful are labeled with 'L:' in the memory map.

Example:
```
M5147->L:$002047
M5148->L$02048
M5148->L8264
```

See Also:
M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->D:{address}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->TWB:{address}

Function: Binary Thumbwheel-Multiplexer Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->TWB[:]{multiplex address}, {offset},{size},{format}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{multiplex address}** is an integer constant in the range 0 to 255, representing the byte address in the multiplexing scheme on the thumbwheel port of the least significant bit to be used in the M-variable(s);
- **{offset}** is an integer constant from 0 to 7, representing which bit of this byte is the least significant bit to be used in the M-variable;
- **{size}** is an integer constant from 1 to 32, representing the number of consecutive bits to be used in the M-variable(s);
- **{format}** (optional) is either U for unsigned, or S for signed (two's complement). If no format is specified, U (unsigned) is assumed

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to a consecutive of input bits multiplexed on the thumbwheel port with Accessory 18 or compatible hardware, including the Advantage 500 NC control panel.

Example:

```
M0->TWB:0,0,1
M1->TWB:0,1,1
M10->TWB:3,4,4,U
M745->TWB:4,0,16,S
M872->TWB:0,4,1
```

See Also:
M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->TWD:{address}**
Thumbwheel Multiplexer Board (ACC-18) Manual

M{constant}->TWD:{address}

Function: BCD Thumbwheel-Multiplexer M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->TWD[:]{multiplex address}, {offset},{size}[.{dp}],{format}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

- **{multiplex address}** is an integer constant in the range 0 to 255, representing the address in the multiplexing scheme on the thumbwheel port of the most significant digit (lowest address) to be used in the M-variable(s);
- **{offset}** is 0 or 4, representing whether the most significant digit is in the low nibble (left digit of pair) or high nibble (right digit of pair) of the pair of digits at **{multiplex address}**, respectively;
- **{size}** is an integer constant from 1 to 12, representing the number of digits to be used in the M-variable(s);
- **{dp}** (optional) is an integer constant from 0 to 8, representing the number of these digits to be interpreted as being to the right of the decimal point;
- **{format}** (optional) is either U for unsigned, or S for signed. If it is signed, the least significant bit of the most significant digit is taken as the sign bit (the rest of the most significant digit is ignored). If no format is specified, U (unsigned) is assumed.

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to a set of binary-coded-decimal digits multiplexed on the thumbwheel port with Accessory 18 or compatible hardware.

Thumbwheel-multiplexer M-variables are read-only, floating-point variables. Once defined, they are to be used in expressions, and queried. Each time one is used in an expression, the proper addresses on the multiplexer board(s) are read.

Example:
M100->TWD:4,0,8.3,U means the most significant digit is at multiplex address 4, low nibble (left digit); there are 8 digits, 3 of which are fractional; and it is always interpreted as a positive value. This corresponds to eight thumbwheel digits along the bottom row of the lowest-addressed thumbwheel board, with the decimal point 3 digits in from the right.

M99->TWD:0,0,1,U means that a single digit is used, at multiplex address 0, low nibble (left digit). This corresponds to the upper left thumbwheel on the lowest-addressed thumbwheel board.

See Also:
M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->TWB:{address}**
Thumbwheel Multiplexer Board (ACC-18) Manual

M{constant}->TWR:{address}

Function: Resolver Thumbwheel-Multiplexer M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->TWR[:]{multiplex address},{offset}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{multiplex address}** is an integer constant, divisible by 2, in the range 0 to 254, representing the address in the multiplexing scheme of the ACC-8D Option 7 resolver-to-digital converter board on the thumbwheel multiplexer port, as determined by the DIP switch settings on the board

- **{offset}** is an integer constant from 0 to 7, representing the location of the device at the specified multiplexer address, as determined by in the buffer on the ACC-8D Option 7 and the actual pins to which the device was wired.

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to a 12-bit word from a resolver-to-digital (R/D) converter or similar device serially multiplexed on the thumbwheel port on an ACC-8D Option 7 or compatible board.

The address on the multiplex port specified here must match the address set by the DIP switches on board the ACC-8D Opt-7. The ACC-8D Opt-7 manual contains a table listing all of the possibilities.

One of the DIP switches on the ACC-8D Opt-7 board determines whether the R/D converters on board have offset values of 0 to 3 or 4 to 7. The **{offset}** specifier must match this DIP switch setting and the number of the R/D device on the board.

This is a read-only M-variable format. Use of this variable in an on-line query command or a program statement will cause Turbo PMAC to clock in 12 bits of unsigned data (range 0 to 4095) from the specified device through the multiplexer port.

> ***Note:***
>
> It is not necessary to use an M-variable to access an R/D converter for actual servo or phasing feedback purposes. I-variables (Ixx10, Ixx81, Ixx98, Ixx99) are used for that purpose. However, even if this is your only use of the R/D converter, it is usually desirable to assign M-variables to the R/D converters for set-up and diagnostic purposes.

Example:

```
M100->TWR:0,0
M99->TWR:4,5
```

See Also:

M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->TWS:{address}**
Resolver-to-Digital Converter Board (ACC-8D Opt-7) Manual

M{constant}->TWS:{address}

Function: Serial Thumbwheel-Multiplexer M-Variable Definition

Scope: Global

Syntax: **M{constant}[..{constant}]->TWS[:]{multiplex address}**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{multiplex address}** is an integer constant, divisible by 4, in the range 0 to 124, representing the address in the multiplexing scheme of the first of four bytes in the 32-bit input or output word. Adding 1 to the **{multiplex address}** designates it as a read-only variable and adding 2 designates it as a write-only variable.

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to a 32-bit word of input or output serially multiplexed on the "thumbwheel" port on an Accessory 34x board.

Note:

The individual bits of the thumbwheel port on an Accessory 34x board cannot be directly assigned to an M-variable. Only 32-bit words (ports) of input or output can be accessed.

The base address of the ACC-34x on the multiplexer port is set by a DIP switch bank on the board. The base address can take a value from 0 to 248, evenly divisible by 8.

The address of the 32-bit input port on the ACC-34x board is equal to the base address plus 1. (If the base address is 16, the address of the input port is 17.)

The address of the 32-bit output port on the ACC-34x board is equal to the base address plus 6. (If the base address is32, the address of the output port is 38.)

The address of the optional supplemental input port available on some ACC-34x boards is equal to the base address plus 3. (If the base address is 128, the address of the supplemental input port is 131.)

The address on the multiplex port specified here must match the address set by the DIP switch on board the ACC-34x. The ACC-34x manual contains a table listing all of the possibilities.

Because you can not directly access the individual bits of the "thumbwheel" port on an Accessory 34x board and because of the relatively long time it takes to clock the data in or out of Turbo PMAC (A 32-bit Read or a 32-bit Write to an individual port takes approximately 16 microseconds of time in the Turbo PMAC's background time slot) it is best to keep an "image" of each M-variable of this type in internal memory. The image variable would preferably be a 32-bit or 48-bit fixed point M-variable, but it could also be a 48-bit floating point P or Q variable.

The best procedure for using TWS M-variables in a program is as follows. The input word (TWS M-variable) should be copied into its image variable at the beginning of a sequence of operations. The operations can then be done on the image variable without requiring Turbo PMAC to actually read or write to the I/O port for each operation. The output word is first "assembled" into its image variable, then copied to the actual output word once at the end of a sequence of operations. This procedure will allow the most efficient and flexible use of TWS M-variables.

Note:

This type of variable can only be used in background tasks (on-line commands, plus PLCs and PLCCs 1-31). They cannot be used in foreground tasks (motion programs and PLC and PLCC 0).

Example:

For an ACC-34x board with base address 0 on the multiplexer port (all DIP switches ON): the the M-variable for the input port would be:

```
M80->TWS:1          ;Input port at {base+1}
```

The definition for the output port would be:

```
M81->TWS:6          ;Output port at {base+6}
```

See Also:

M-Variables (Computational Features)

On-line commands **M{constant}**, **M{constant}->**, **.......... M{constant}->TWR:{address}**

Serial I/O Multiplexer Board (ACC-34x) Manuals

M{constant}->X/Y:{address}

Function: Short Word M-Variable Definition

Scope: Global

Syntax:
```
M{constant}[..{constant}]->
X[:]{address},{offset}[,{width}[,{format}]]
M{constant}[..{constant}]->
Y[:]{address},{offset}[,{width}[,{format}]]
```

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the M-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;
- **{address}** is an integer constant from $000000 to $FFFFFF (0 to 16,777,215 if specified in decimal).
- **{offset}** is an integer constant from 0 to 23, representing the starting (least significant) bit of the word to be used in the M-variable(s), or 24 to specify the use of all 24 bits;
- **{width}** (optional) is an integer constant from the set {1, 4, 8, 12, 16, 20, 24}, representing the number of bits from the word to be used in the M-variable(s); if **{width}** is not specified, a value of 1 is assumed;
- **{format}** (optional) is a letter from the set [U, S, D, C], specifying how Turbo PMAC is to interpret this value: (U=Unsigned integer, S=Signed integer, D=Binary-coded Decimal, C=Complementary binary-coded decimal); if **{format}** is not specified, U is assumed.

This command causes Turbo PMAC to define the specified M-variable or range of M-variables to point to a location in one of the two halves (X or Y) of Turbo PMAC's data memory. In this form, the variable can have a width of 1 to 24 bits and can be decoded several different ways, so the bit offset, bit width, and decoding format must be specified (the bit width and decoding format do have defaults.

The definition consists of the letter **X** or **Y**, an optional colon (**:**), the word address, the starting bit number (offset), an optional bit width number, and an option format-specifying letter.

Legal values for bit width and bit offset are inter-related. The table below shows the possible values of **{width}**, and the corresponding legal values of **{offset}** for each setting of **{width}**.

{width}	{offset}
1	0 -- 23
4	0,4,8,12,16,20
8	0,4,8,12,16
12	0,4,8,12
16	0,4,8
20	0,4
24	0

The format is irrelevant for 1-bit M-variables, and should not be included for them. If no format is specified, U is assumed.

Examples:

```
; Machine Output 1
M1->Y:$078802,8,1              ; 1-bit (full spec.)
M1->Y$078802,8                 ; 1-bit (short spec.)
; Encoder 1 Capture/Compare Register
M103->X:$078003,0,24,U         ; 24-bit (full spec.)
```

```
M103->X$078003,24          ; 24-bit (short spec.)
; DAC 1 Output Register
M102->Y:$078003,8,16,S     ; 16-bit value
M102->Y491523,8,16,S       ; same, decimal address
```

See Also:
M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
..... **M{constant}->D:{address}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

MACROASCII{master #} [replaced]

Function: Set port in MACRO pass-through mode

Scope: Global (MACRO Ring Master only)

Syntax: **MACROASCII{master #}**

Note:

This command has been replaced starting in V1.936 firmware by the MACROMSTASCII command.

MACROAUX{node #},{param #}

Function: Report MACRO Type 0 auxiliary parameter value from slave node

Scope: Global

Syntax: **MACROAUX{node #},{param #}**
MX{node #},{param #}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
  - **{param #}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node (2 to 253 required for a write operation)

This command causes PMAC to query the MACRO slave station at the specified node number using the MACRO Type 0 master-to-slave auxiliary protocol, and report back the value of the specified slave station parameter to the host computer.

Only one auxiliary access (read or write) of a single node can be done on one command line.

The auxiliary register function for the specified node number must have been enabled by setting the appropriate bit of I70, I72, I74, or I76 to 1. The Type 0 protocol for this node must have been selected by setting the appropriate bit of I71, I73, I75, or I77 to 0.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value.

It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

> ***Note:***
>
> The Delta Tau MACRO Station products use the Type 1 protocol, and so do not use this command.

Example:

```
MACROAUX1,24          ; Request value of Node 1 Parameter 24
2000 .....            ; Turbo PMAC reports value
```

See Also:

On-line commands **MACROAUXREAD, MACROAUXWRITE**

PLC Program commands **MACROAUXREAD, MACROAUXWRITE**

MACROAUX{node #},{param #}={constant}

Function: Set MACRO Type 0 auxiliary parameter value in slave node

Scope: Global

Syntax:
```
MACROAUX{node #},{param #}={constant}
MX{node #},{param #}={constant}
```

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{param #}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node (2 to 253 required for a write operation)
- **{constant}** is an integer constant from -32768 to +32767 representing the value to be written to the specified parameter

This command causes Turbo PMAC to write the specified constant value to the variable of the MACRO slave station at the specified node number using the MACRO Type 0 master-to-slave auxiliary protocol.

Only one auxiliary access (read or write) of a single node can be done on one command line.

The auxiliary register function for the specified node number must have been enabled by setting the appropriate bit of I70, I72, I74, or I76 to 1. The Type 0 protocol for this node must have been selected by setting the appropriate bit of I71, I73, I75, or I77 to 0.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Note:

Delta Tau MACRO Station products use the Type 1 protocol, and so do not use this command.

Example:

```
MACROAUX1,24=2000  ; Set Node 1 Parameter 24 to 2000
```

See Also:

On-line commands **MACROAUXREAD, MACROAUXWRITE**

PLC Program commands **MACROAUXREAD, MACROAUXWRITE**

MACROAUXREAD

Function: Read (copy) Type 0 MACRO auxiliary parameter value from slave node

Scope: Global

Syntax: **MACROAUXREAD{node #},{param #},{variable}**
MXR{node #},{param #},{variable}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{param #}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node
- **{variable}** is the name of the Turbo PMAC variable (I, P, Q, or M) into which the parameter value is to be copied

This command causes Turbo PMAC to copy the value of the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC to the specified Turbo PMAC variable, using the MACRO Type 0 master-to-slave auxiliary protocol.

Only one auxiliary access (read or write) of a single node can be done on one command line.

The auxiliary register function for the specified node number must have been enabled by setting the appropriate bit of I70, I72, I74, or I76 to 1. The Type 0 protocol for this node must have been selected by setting the appropriate bit of I71, I73, I75, or I77 to 0.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Note:

The Delta Tau MACRO Station products use the Type 1 protocol, and so do not use this command.

Example:

```
MACROAUXREAD1,24,P1      ; Read Node 1 Parameter 24 into P1
MXR5,128,M100            ; Read Node 5 Parameter 128 into M100
```

See Also:

On-line commands **MACROAUX, MACROAUXWRITE**
PLC Program commands **MACROAUXREAD, MACROAUXWRITE**

MACROAUXWRITE

Function: Write (copy) Type 0 MACRO auxiliary parameter value to slave node

Scope: Global

Syntax: **MACROAUXWRITE{node #},{param #},{variable}**
MXW{node #},{param #},{variable}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{param #}** is an integer constant from 2 to 253 specifying the auxiliary parameter number for this node
- **{variable}** is the name of the Turbo PMAC variable (I, P, Q, or M) from which the parameter value is to be copied

This command causes Turbo PMAC to copy the value of the specified variable on Turbo PMAC to the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC, using the MACRO Type 0 master-to-slave auxiliary protocol.

Only one auxiliary access (read or write) of a single node can be done on one command line.

The auxiliary register function for the specified node number must have been enabled by setting the appropriate bit of I70, I72, I74, or I76 to 1. The Type 0 protocol for this node must have been selected by setting the appropriate bit of I71, I73, I75, or I77 to 0.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Note:

Delta Tau MACRO Station products use the Type 1 protocol, and so do not use this command.

Example:

```
MACROAUXWRITE1,24,P1      ; Write value of P1 to Node 1 Parameter 24
MXW5,128,M100             ; Write value of M100 to Node 5 Parameter 128
```

See Also:
On-line commands **MACROAUX, MACROAUXREAD**
PLC Program commands **MACROAUXREAD, MACROAUXWRITE**

MACROMST{master#},{master variable}

Function: Report Type 1 MACRO master variable value

Scope: Global

Syntax: **MACROMST{master #},{master variable}**
MM{node #},{master variable}

where:

- **{master #}** is a constant (1-15) representing the number of the remote master whose variable is to be read;
- **{master variable}** is the name of the variable on the remote master station whose value is to be reported

This command causes the Turbo PMAC to query another master station on the ring for a variable value using the MACRO Type 1 master-to-master auxiliary communications protocol, and report back the value of the specified variable to the host computer.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- MACRO IC 0 on this Turbo PMAC must be set up as the synchronizing ring master (I6840 = $xx30);
- MACRO IC 0 on this Turbo PMAC must be set up as Master 0 on the ring (I6841 = $0xxxxx);
- MACRO IC 0 node 14 must be in "broadcast mode" (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

In order for another Turbo PMAC to be able to respond to the communications that this command creates on the MACRO ring (to be the "remote master") the following conditions must be true:

- MACRO IC 0 on the Turbo PMAC must be set up as a master, but not the synchronizing ring master (I6480 = $xx90);
- MACRO IC 0 on the Turbo PMAC must be set up with a non-zero master number on the ring (I6841 = $nxxxxx, where n > 0);
- MACRO IC 0 node 14 must normally be disabled (I6841 bit 14 = 1);
- MACRO IC 0 node 14 must be in "broadcast mode" (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

If the remote master returns an error message or it does not respond within I79 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MM4,I10        ; Causes remote master 4 to report value of variable I10
3713707        ; Turbo PMAC reports this value back to host
MM1,P1         ; Causes remote master 1 to report value of variable P1
3.14159 .....; Turbo PMAC reports this value back to host
```

MACROMST{master#},{master variable}={constant}

Function: Set Type 1 MACRO master auxiliary parameter value

Scope: Global

Syntax: **MACROMST{master #},{master variable}={constant}**
MM{node #},{master variable}={constant}

where:

- **{master #}** is a constant (1-15) representing the number of the remote master whose variable is to be read;
- **{master variable}** is the name of the variable on the remote master station whose value is to be set
- **{constant}** is a number representing the value to be written to the specified variable on the remote master station

This command causes the Turbo PMAC to set a variable value on another master station on the ring using the MACRO Type 1 master-to-master auxiliary communications protocol.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- MACRO IC 0 on this Turbo PMAC must be set up as the synchronizing ring master (I6840 = $xx30);
- MACRO IC 0 on this Turbo PMAC must be set up as Master 0 on the ring (I6841 = $0xxxxx);
- MACRO IC 0 node 14 must be in broadcast mode (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

In order for another Turbo PMAC to be able to respond to the communications that this command creates on the MACRO ring (to be the "remote master") the following conditions must be true:

- MACRO IC 0 on the Turbo PMAC must be set up as a master, but not the synchronizing ring master (I6480 = $xx90);
- MACRO IC 0 on the Turbo PMAC must be set up with a non-zero master number on the ring (I6841 = $nxxxxx, where n > 0);
- MACRO IC 0 node 14 normally must be disabled (I6841 bit 14 = 1);
- MACRO IC 0 node 14 must be in broadcast mode (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

If the remote master returns an error message or it does not respond within I79 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MM4,I10=3713707          ; Causes remote master 4 to set value of variable I10 to 3,713,707
MM1,P1=3.14159           ; Causes remote master 1 to set value of variable P1 to 3.14159
```

MACROMSTASCII{master #}

Function: Set port in MACRO pass-through mode

Scope: Global (MACRO Ring Master only)

Syntax: **MACROMSTASCII{master #}**
MACMA{master #}

where:

- **{master #}** is a constant in the range 1 to 15 representing the master number of the Turbo PMAC to whom the communications will be passed

This command causes Turbo PMAC to put the port on which it receives this command in a MACRO "master-to-master pass-through" mode. In this mode, commands received over this port are not acted on by this Turbo PMAC. Instead, they are passed on over the MACRO ring to the Turbo PMAC with the specified master number. Responses are received from the other Turbo PMAC over the MACRO ring and passed back to the host computer over this port.

This mode of communications, which uses the "Type 1" auxiliary communications protocol, requires V1.936 or newer firmware on all Turbo PMACs on the ring. It permits the host computer to communicate with remote Turbo PMACs as if they were directly connected to the host. Non-Turbo PMACs on the ring will not respond to commands passed over the ring in this protocol.

If used to set up broadcast to multiple masters on the ring (**MACROMSTACII0**), no handshaking between boards is possible on broadcast commands. In this mode, it is strongly suggested that only commands of 6 ASCII characters or less (including the terminating **<CR>** character) be used. This mode is intended mainly to start or stop programs together on multiple cards.

This command can only be given to a Turbo PMAC that is the synchronizing "ring master" ("ring controller") set up for Type 1 auxiliary communications with Node 14 in broadcast mode. This requires that:

- I79 > 0 (enable master-to-master, recommended I79 = 32)
- I6840 = $4030 (ring controller with Node 14 in broadcast mode)*
- I6841 bit 14 set to 1 (enable Node 14 communications)*
- These values be saved, and the card reset, before this mode can be enabled

* These I-variables are used if MACRO IC 0 is used for this communications, the most common configuration. If MACRO IC 1 is used, I6890 and I6891 must take these values; for MACRO IC 2, I6940 and I6941; for MACRO IC 3, I6990 and I6991.

On this Turbo PMAC, the commands will be sent out over the MACRO ring through the MACRO IC (0 – 3) whose number is specified by I84. The base address of MACRO IC *n* is specified by variable I2*n*.

Only one communications port of this ring controller Turbo PMAC may be in "pass-through" mode at any time. The other ports are not in "pass-through" mode and may be used simultaneously for communications with the ring controller Turbo PMAC itself.

Turbo PMACs on the ring to which these commands are passed through must be set as masters but not ring controllers, with Node 14 enabled but not in broadcast mode. On these boards:

- I6480 = $10 (master, but not ring controller; Node 14 not in broadcast mode)
- I6841 bit 14 = 1 (Node 14 enabled)
- These values be saved, and the card reset, before this mode can be enabled.

This port is taken out of MACRO pass-through mode if it is given the **<CTRL-T>** command. To change which other master to which the commands are passed through, you must first take the port out of pass-through mode with a **<CTRL-T>**, then enable the pass-through to another master. If another **MACROMSTASCII** command is sent while the port is in pass-through mode, this command will be passed through and rejected by the other master, returning an ERR008.

In pass-through mode, command errors are always reported in the form **<BELL>ERRnnn<CR>**, regardless of the setting of I6.

See Also:
On-line commands **<CTRL-T>**, **MACROSTASCII{station #}**

MACROMSTREAD

Function: Read (copy) Type 1 MACRO master auxiliary parameter value

Scope: Global

Syntax:
```
MACROMSTREAD{master #},{master variable},
{ring-master variable}
MMR{master #},{master variable},{ring-master variable}
```

where:

- **{master #}** is a constant (1-15) representing the number of the remote master whose variable is to be read;
- **{master variable}** is the name of the variable on the remote master station whose value is to be reported
- **{ring-master variable}** is the name of the variable on the Turbo PMAC executing the command into which the value of the remote master variable is to be copied

This command causes Turbo PMAC to copy the value of the specified variable on the remote master station into the specified variable on the Turbo PMAC executing the command, using the MACRO Type 1 master-to-master auxiliary protocol.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- MACRO IC 0 on this Turbo PMAC must be set up as the synchronizing ring master (I6840 = $xx30);
- MACRO IC 0 on this Turbo PMAC must be set up as Master 0 on the ring (I6841 = $0xxxxx);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

In order for another Turbo PMAC to be able to respond to the communications that this command creates on the MACRO ring (to be the “remote master”) the following conditions must be true:

- MACRO IC 0 on the Turbo PMAC must be set up as a master, but not the synchronizing ring master (I6480 = $xx90);
- MACRO IC 0 on the Turbo PMAC must be set up with a non-zero master number on the ring (I6841 = $nxxxxx, where n > 0);
- MACRO IC 0 node 14 must normally be disabled (I6841 bit 14 = 1);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

If the remote master returns an error message or it does not respond within I79 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

If this command is issued to a Turbo PMAC while a PLC buffer is open, it will be stored in the buffer as a PLC command, not executed as an on-line command.

Examples:

```
MMR4,I10,P10      ; Copies value of remote master 4 variable I10 into Turbo PMAC variable P10
MMR1,P1,P1        ; Copies value of remote master 1 variable P1 into Turbo PMAC variable P1
```

MACROMSTWRITE

Function: Write (copy) Type 1 MACRO master auxiliary parameter value

Scope: Global

Syntax:
```
MACROMSTWRITE{master #},{master variable},
{ring-master variable}
MMW{master #},{master variable},{ring-master variable}
```

where:

- **{master #}** is a constant (1-15) representing the number of the remote master whose variable is to be read;
- **{master variable}** is the name of the variable on the remote master station whose value is to be set;
- **{ring-master variable}** is the name of the variable on the Turbo PMAC executing the command from which the value of the remote master variable is to be copied.

This command causes Turbo PMAC to copy the value of the specified variable on the remote master station from the specified variable on the Turbo PMAC executing the command, using the MACRO Type 1 master-to-master auxiliary protocol.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- MACRO IC 0 on this Turbo PMAC must be set up as the synchronizing ring master (I6840 = $xx30);
- MACRO IC 0 on this Turbo PMAC must be set up as Master 0 on the ring (I6841 = $0xxxxx);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

In order for another Turbo PMAC to be able to respond to the communications that this command creates on the MACRO ring (to be the “remote master”) the following conditions must be true:

- MACRO IC 0 on the Turbo PMAC must be set up as a master, but not the synchronizing ring master (I6480 = $xx90);
- MACRO IC 0 on the Turbo PMAC must be set up with a non-zero master number on the ring (I6841 = $nxxxxx, where n > 0);
- MACRO IC 0 node 14 must normally be disabled (I6841 bit 14 = 1);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

If the remote master returns an error message or it does not respond within I79 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

If this command is issued to a Turbo PMAC while a PLC buffer is open, it will be stored in the buffer as a PLC command, not executed as an on-line command.

Examples:

```
MMW4,I10,P10      ; Copies value of Turbo PMAC variable P10 into remote master 4 variable I10
MMW1,P1,P1        ; Copies value of Turbo PMAC variable P1 into remote master 1 variable P1
```

MACROSLV{command} {node#}

Function: Send command to Type 1 MACRO slave

Scope: Global

Syntax: **MACROSLAVE{command}{node #}**
MS{command}{node #}

where:

- **{command}** is one of the following text strings:
- **$$$** normal station reset
- **$$$*** ** station reset and re-initialize
- **CLRF** station fault clear for
- **CONFIG** report station configuration value
- **DATE** report station firmware date
- **SAVE** save station setup
- **VER** report station firmware version
- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;

This command causes Turbo PMAC to issue the specified command to a MACRO slave station using the Type 1 auxiliary master-to-slave protocol. **{node #}** can be the number of any active node on the slave station. If **{node #}** is set to the highest node number for the MACRO IC (15, 31, 47, or 63), the action automatically applies to all slave stations commanded from that MACRO IC.

The **MS CONFIG** command allows the user to set and report a user-specified configuration value. This provides any easy way for the user to see if the MACRO station has already been configured to the user's specifications.

The factory default configuration value is 0. It is recommended that after the user finishes the software configuration of the station, a special number be given to the configuration value with the **MS CONFIG{node #}={constant}** command. This number will be saved to the non-volatile memory with the **MS SAVE** command.

Subsequently, when the system is powered up, the station can be polled with the **MS CONFIG {node #}** command. If the expected value is returned, the station can be assumed to have the proper software setup. If the expected value is not returned (for instance, when a replacement station has just been installed) then the setup will have to be transmitted to the station.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MS $$$0              ; Resets MACRO station which has active node 0
MS $$$***4           ; Reinitializes MACRO station which has active node 4
MS CLRF8             ; Clears fault on Node 8 of MACRO station
MS CONFIG12          ; Causes MACRO station to report its configuration #
37                   ; PMAC reports MACRO station configuration # to host
MS CONFIG12=37       ; Sets MACRO station configuration number
MS DATE 0            ; Causes MACRO station to report its firmware date
03/27/97             ; PMAC reports MACRO station firmware date to host
MS SAVE 4            ; Causes MACRO station to save setup variables
MS VER 8             ; Causes MACRO station to report its firmware version
1.104                ; PMAC reports MACRO station firmware version to host
```

MACROSLV{node#},{slave variable}

Function: Report Type 1 MACRO auxiliary parameter value

Scope: Global

Syntax:
```
MACROSLAVE{node #},{slave variable}
MS{node #},{slave variable}
```

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;

  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{slave variable}** is the name of the variable on the slave station whose value is to be reported

This command causes Turbo PMAC to query the MACRO slave station at the specified node number using the MACRO Type 1 master-to-slave auxiliary protocol, and report back the value of the specified slave station variable to the host computer. If the variable is not node-specific, **{node #}** can represent the number of any active node on the slave station.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MS0,MI910          ; Causes slave to report value of Node 0 variable MI910
7                  ; PMAC reports this value back to host
MS1,MI997          ; Causes slave to report value global variable MI997
6258               ; PMAC reports this value back to host
```

MACROSLV{node#},{slave variable}={constant}

Function: Set Type 1 MACRO auxiliary parameter value

Scope: Global

Syntax: **MACROSLAVE{node #},{slave variable}={constant}**
MS{node #},{slave variable}={constant}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{slave variable}** is the name of the MI-variable or C-command on the slave station whose value is to be set;
- **{constant}** is a number representing the value to be written to the specified MI-variable

This command causes Turbo PMAC to write the specified constant value to the variable of the MACRO slave station at the specified node number using the MACRO Type 1 master-to-slave auxiliary protocol. If the variable is not node-specific, **{node #}** can represent the number of any active node on the slave station.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MS0,MI910=7          ; Causes slave to set value of Node 0 variable MI910 to 7
MS1,MI997=6528       ; Causes slave to set value global variable MI997 to 6528
MS8,C2=0             ; Causes slave at node 8 to reset
```

MACROSLVREAD

Function: Read (copy) Type 1 MACRO auxiliary parameter value

Scope: Global

Syntax: **MACROSLVREAD{node #},{slave variable},{PMAC variable}**
MSR{node #},{slave variable},{PMAC variable}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{slave variable}** is the name of the variable on the slave station whose value is to be reported
- **{PMAC variable}** is the name of the variable on the Turbo PMAC into which the value of the slave station variable is to be copied

This command causes Turbo PMAC to copy the value of the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC to the specified Turbo PMAC variable, using the MACRO Type 1 master-to-slave auxiliary protocol. If the slave station variable is not node-specific, **{node #}** can represent the number of any active node on the slave station.

The variable on the Turbo PMAC can be any of the I, P, Q, or M-variable on the card.

If this command is issued to the Turbo PMAC while a PLC buffer is open, it will be stored in the buffer as a PLC command, not executed as an on-line command.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MSR0,MI910,P1      ; Copies value of slave Node 0 variable MI910 into PMAC variable P1
MSR1,MI997,M10     ; Copies value of slave Node 1 variable MI997 into PMAC variable M10
```

MACROSLVWRITE

Function: Write (copy) Type 1 MACRO auxiliary parameter value

Scope: Global

Syntax: **`MACROSLVWRITE{node #},{slave variable},{PMAC variable}`**
`MSW{node #},{slave variable},{PMAC variable}`

where:

- **`{node #}`** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **`{node #}`** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **`{node #}`** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **`{node #}`** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **`{node #}`** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **`{slave variable}`** is the name of the MI-variable or C-command on the slave station whose value is to be set;
- **`{PMAC variable}`** is the name of the variable on the PMAC from which the value of the slave station variable is to be copied

This command causes Turbo PMAC to copy the value of the specified variable on Turbo PMAC to the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC, using the MACRO Type 1 master-to-slave auxiliary protocol. If the slave station variable is not node-specific, **`{node #}`** can represent the number of any active node on the slave station.

The variable on the Turbo PMAC can be any of the I, P, Q, or M-variables on the card.

If this command is issued to the Turbo PMAC while a PLC buffer is open, it will be stored in the buffer as a PLC command, not executed as an on-line command.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Examples:

```
MSW0,MI910,P35     ; Copies value of PMAC P35 into MACRO station node 0 variable MI910
MSW4,C4,P0         ; Causes MACRO station with active node 4 to execute Command #4, saving its
                   ; setup variable values to non-volatile memory (P0 is a dummy variable here)
```

MACROSTASCII {station #}

Function: Set port in MACRO pass-through mode

Scope: Global (MACRO Ring Master only)

Syntax: **MACROSTASCII{master #}**
MACSTA{master #}

where:

- **{station #}** is a constant in the range 0 to 255 representing the order number of the station on the ring to whom the communications will be passed. If a 0 is used, subsequent commands will be broadcast to all masters on the ring. If a 255 is used, subsequent commands will be sent to the next "unordered" (station number 0) station on the ring.

This command causes Turbo PMAC to put the port on which it receives this command in a MACRO "master-to-station pass-through" mode. In this mode, commands received over this port are not acted on by this Turbo PMAC. Instead, they are passed on over the MACRO ring to the master or slave station on the ring with the specified station-order number. Responses are received from the other station over the MACRO ring and passed back to the host computer over this port.

The main purpose of this mode of communications is to be able to go around the MACRO ring, station by station, and communicate with each station, even if the normal ring addressing is not set up, not set up properly, or unknown to the system. This mode of communications, which uses the Type 1 auxiliary communications protocol, requires V1.936 or newer firmware on all Turbo PMACs on the ring, and V1.1114 or newer firmware on all MACRO Stations on the ring. Non-Turbo PMACs on the ring will not respond to commands passed over the ring in this protocol.

A **MACROSTASCII255** command will set up communications with the first "unordered" station (station order number of 0) on the ring. Query commands can detect information about this station, then a station order number assigned to this station with the **STN={constant}** or **I85={constant}** command. (Typically, ring-order numbers are given sequentially to stations along the ring, but this is not required.) Once a station order number is assigned to a station, it will no longer respond to commands given in **MACROSTASCII255** mode, and the next unordered (STN=0) station will respond to these commands instead.

A **MACROSTASCII0** command will broadcast subsequent commands to all stations on the ring. In this mode, a **STN=0** or **I85=0** command will "unorder" all stations on the ring so that they can be isolated one by one using the above technique. In this mode, no handshaking between boards is possible on broadcast commands. In this mode, it is strongly suggested that only commands of 6 ASCII characters or less (including the terminating **<CR>** character) be used. This mode is intended mainly to re-initialize the ordering on the ring.

This command can only be given to a Turbo PMAC that is the synchronizing ring master (ring controller) set up for Type 1 auxiliary communications with Node 14 in broadcast mode. This requires that:

- I79 > 0 (enable master-to-master, recommended I79 = 32)
- I6840 = $4030 (ring controller with Node 14 in broadcast mode)*
- I6841 bit 14 set to 1 (enable Node 14 communications)*
- These values be saved, and the card reset, before this mode can be enabled

* These I-variables are used if MACRO IC 0 is used for this communications, the most common configuration. If MACRO IC 1 is used, I6890 and I6891 must take these values; for MACRO IC 2, I6940 and I6941; for MACRO IC 3, I6990 and I6991.

On this Turbo PMAC, the commands will be sent out over the MACRO ring through the MACRO IC (0 – 3) whose number is specified by I84. The base address of MACRO IC *n* is specified by variable I2*n*.

Only one communications port of this ring controller Turbo PMAC may be in "pass-through" mode at any time. The other ports are not in "pass-through" mode and may be used simultaneously for communications with the ring controller Turbo PMAC itself.

This port is taken out of MACRO "pass-through" mode if it is given the **<CTRL-T>** command. If another **MACROSTASCII** command is sent while the port is in pass-through mode, this command will be passed through and rejected by the other station, returning an ERR008.

In pass-through mode, command errors are always reported in the form **<BELL>ERRnnn<CR>**, regardless of the setting of I6.

See Also:
On-line commands **<CTRL-T>**, **MACROMSTASCII{station #}**

MFLUSH

Function: Clear pending synchronous M-variable assignments

Scope: Coordinate-system specific

Syntax: **MFLUSH**

This command permits the user to clear synchronous M-variable assignment commands that have been put on the stack for intended execution with a subsequent move (without executing the commands). As an on-line command, it is useful for making sure pending outputs are not executed after a program has been stopped.

Examples:

```
Q..........      ; Stop execution of a program
MFLUSH.....      ; Clear M-variable stack
B1R........      ; Start another program; formerly pending M-variables will not execute
```

See Also:
Program commands **M{data}=={expression}**,
M{data}&={expression},
M{data}|={expression},
M{data}^={expression}

MOVETIME

Function: Report time left in presently executing move

Scope: Coordinate-system specific

Syntax: **MOVETIME**
MVTM

The **MOVETIME** command causes Turbo PMAC to report the time left in the currently executing move in the addressed coordinate system, scaled in milliseconds.

This time remaining function is not active if the addressed coordinate system is executing moves in the special lookahead buffer.

NOFRAX

Function: Remove all axes from list of vector feedrate axes

Scope: Coordinate-system specific

Syntax: **NOFRAX**

This command causes Turbo PMAC to remove all axes from the list of vector feedrate axes for the addressed coordinate system. In this mode, any feedrate-specified move in the coordinate system will yield a vector distance of 0, forcing the use of the Isx86 alternate feedrate. This can be useful to create a "dry run" of a motion program, overriding the feedrates specified in the motion-program **F** commands.

Axes can be restored to the vector feedrate list with the **FRAX** command.

See Also:
I-variables Isx86, Isx89, Isx90, Isx98
On-line command **FRAX**
Program commands **F**, **FRAX**, **NOFRAX**

NORMAL

Function: Report circle-plane unit normal vector

Scope: Coordinate system specific

Syntax: **NORMAL**
NRM

This command causes Turbo PMAC to report the unit normal vector for the addressed coordinate system. This normal vector defines the plane for circular interpolation and cutter radius compensation.

Turbo PMAC reports the vector by displaying its I, J, and K components, parallel to the X, Y, and Z axes, respectively. If a component is zero, it will not be reported. The vector sum of the components is 1.0 because this is a vector of unit magnitude.

The default normal vector is K-1.0, which specifies the XY plane. This can be modified by the motion program **NORMAL I{data} J{data} K{data}** statement. Note that the vector magnitude of the commanded normal vector does not need to be equal to 1.0 – Turbo PMAC will store a scaled version of it.

Syntax:

```
NORMAL
K-1
NORMAL
I0.7071 J0.7071
```

O{constant}

Function: Open loop output

Scope: Motor specific

Syntax: **O{constant}**

where:

- **{constant}** is a floating-point value representing the magnitude of the output as a percentage of Ixx69 for the motor, with a range of +/-100

This command causes Turbo PMAC to put the motor in open-loop mode and force an output of the specified magnitude, expressed as a percentage of the maximum output parameter for the motor (Ixx69). This command is commonly used for set-up and diagnostic purposes (for instance, a positive **O** command must cause position to count in the positive direction, or closed-loop control cannot be established), but it can also be used in actual applications.

If the motor is *not* Turbo PMAC-commutated, this command will create a DC output voltage on the single DAC for the motor. If the motor is commutated by Turbo PMAC, the commutation algorithm is still active, and the specified magnitude of output is apportioned between the two DAC outputs or the three PWM outputs for the motor according to the instantaneous commutation phase angle.

If the value specified is outside the range +/-100, the output will saturate at +/-100% of Ixx69.

Closed-loop control for the motor can be re-established with the **J/** command. It is a good idea to stop the motor first with an **O0** command if it has been moving in open-loop mode.

To do a variable O-command, define an M-variable to the *filter result* register (X:$0000AE, etc., suggested M-variable Mxx79), command an **O0** to the motor to put it in open-loop mode, then assign a variable value to the M-variable. This technique will even work on Turbo PMAC-commutated motors.

Turbo PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example:

```
O50.....          ; Open-loop output 50% of Ixx69 for addressed motor
#2033.333 .....   ; Open-loop output 1/3 of Ixx69 for Motor 2
O0 .....          ; Open-loop output of zero magnitude
J/ .....          ; Re-establish closed-loop control
```

See Also:

On-line commands **J/**, **K**

Memory-map registers X:$0000BF, X:$00013F, etc.

Suggested M-variable definitions Mxx68.

OPEN BINARY ROTARY

Function: Open all existing rotary buffers for binary DPRAM entry

Scope: Port specific

Syntax: **OPEN BINARY ROTARY**
OPEN BIN ROT

This command causes Turbo PMAC to open all existing rotary motion program buffers for entry of binary-formatted program lines through the DPRAM.

In order to be able to send binary-formatted program lines to a coordinate system's rotary buffer, three steps must first have been performed:

1. Room must have been reserved for the coordinate system's rotary buffer in internal PMAC memory since the last power-up/reset with the **DEFINE ROTARY** command.
2. Room must have been reserved for data transfer for the coordinate system in the DPRAM by setting up pointers in the DPRAM (usually done by PCOMM32 subroutines).
3. The buffers must have been enabled for entry with the **OPEN BIN ROT** command.

No other program buffers (PLC, fixed motion program, or rotary buffers with ASCII input) may be open when this command is sent (Turbo PMAC will report ERR007 if I6 = 1 or 3). It is a good idea always to preced an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

The actual downloading of binary-formatted program lines is almost always done with PCOMM32 software routines.

ASCII text commands sent through any port to the Turbo PMAC while the binary rotary program buffers are open are treated as on-line commands. ASCII text commands sent at this time that can only be interpreted as buffered program commands will be rejected (Turbo PMAC will report ERR005 if I6 is 1 or 3).

The **CLOSE** command issued on this same port will close all open binary rotary buffers.

This function is controlled by variable I57 on non-Turbo PMACs.

OPEN FORWARD

Function: Open a forward-kinematic program buffer for entry

Scope: Port, coordinate-system specific

Syntax: **OPEN FORWARD**
OPEN FWD

This command causes Turbo PMAC to open the forward-kinematic program buffer for the addressed coordinate system for entry or editing through this port. Subsequent program commands valid for these programs sent from this port will be entered into this buffer. Kinematic programs can accept all commands that are valid for PLC programs except **ADDRESS**, **CMD**, and **SEND** commands. When entry of the program is finished, the **CLOSE** command should be used to prevent further lines from being put in this buffer.

If the kinematic-enable variable Isx50 for the coordinate system is set to 1, Turbo PMAC will automatically execute the forward-kinematic program for the coordinate system any time an **R** (run), **S** (step), or **PMATCH** command is given to the coordinate system. Before running the program PMAC will place the commanded position value (in counts) for each Motor *xx* in the coordinate system into global variable P*xx*.

After the program is run, Turbo PMAC will take the values in Q1 – Q9 for the coordinate system and use them as the starting positions for the A, B, C, U, V, W, X, Y, and Z axes, respectively (in engineering units).

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (Turbo PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

No motion programs may be running in any coordinate system when this command is sent (Turbo PMAC will report *ERR001* if I6=1 or 3). As long as a forward kinematic program buffer is open, no motion program may be run in any coordinate system (Turbo PMAC will report *ERR015* if I6=1 or 3).

Forward kinematic programs can be protected by password. If there is a password for the Turbo PMAC, and the password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6 is 1 or 3).

See Also:
Kinematic Calculations
I-variable Isx50
On-line commands **LIST FORWARD**, **LIST INVERSE**, **OPEN INVERSE**

OPEN INVERSE

Function: Open an inverse-kinematic program buffer for entry

Scope: Port, coordinate-system specific

Syntax: **OPEN INVERSE**
OPEN INV

This command causes Turbo PMAC to open the inverse-kinematic program buffer for the addressed coordinate system for entry or editing through this port. Subsequent program commands valid for these programs sent on this port will be entered into this buffer. Kinematic programs can accept all commands that are valid for PLC programs except **ADDRESS**, **CMD**, and **SEND** commands. When entry of the program is finished, the **CLOSE** command should be used to prevent further lines from being put in this buffer.

If the kinematic-enable variable Isx50 for the coordinate system is set to 1, Turbo PMAC will automatically execute the inverse-kinematic program for the coordinate system any time axis positions are calculated during motion program execution. This is either the end of the programmed move for non-segmented moves, or the end of each move segment for segmented **LINEAR** and **CIRCLE**-mode moves with Isx13 greater than 0.

Before each instance of running the inverse-kinematic program, Turbo PMAC will automatically place the commanded position values (in engineering units) for the A, B, C, U, V, W, X, Y, and Z axes into Q1 – Q9, respectively, for the coordinate system. After the program is run, for each Motor *xx* in the coordinate system whose axis definition statement is ***#xx->I***, Turbo PMAC automatically places the value of P*xx* into the target position register for the motor. If executing a PVT-mode move, Turbo PMAC will also place the commanded velocity value (in engineering units) for the axes into Q11 – Q19 before, and place the value of P1*xx* into the motor target velocity register after.

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (Turbo PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

No motion programs may be running in any coordinate system when this command is sent (Turbo PMAC will report *ERR001* if I6=1 or 3). As long as a forward kinematic program buffer is open, no motion program may be run in any coordinate system (Turbo PMAC will report *ERR015* if I6=1 or 3).

Inverse kinematic programs can be protected by password. If there is a password for the Turbo PMAC, and the password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6 is 1 or 3).

See Also:
Kinematic Calculations
I-variable Isx50
On-line commands **LIST FORWARD**, **LIST INVERSE**, **OPEN FORWARD**

OPEN PLC

Function: Open a PLC program buffer for entry

Scope: Port specific

Syntax: **OPEN PLC {constant}**

where:

- **{constant}** is an integer from 0 to 31 representing the PLC program to be opened

This command causes Turbo PMAC to open the specified PLC program buffer for entry and editing on this port. This permits subsequent program lines that are valid for a PLC to be entered into this buffer from this port only. When entry of the program is finished, the **CLOSE** command should be sent on this port to prevent further lines from being put in the buffer.

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (Turbo PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** or **CLOSE ALL** command to make sure no other buffers have been left open.

PLCs 0-15 can be protected by password. If the PLC is protected by password, and the proper password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

Opening a PLC program buffer automatically disables that PLC program. Other PLC programs and motion programs will keep executing. Closing the PLC program buffer after entry does not re-enable the program. To re-enable the program, the **ENABLE PLC** command must be used, or Turbo PMAC must be reset (with a saved value of I5 permitting this PLC program to execute).

Example:

```
CLOSE.....              ; Make sure other buffers are closed
DELETE GATHER           ; Make sure memory is free
OPEN PLC 7....          ; Open buffer for entry, disabling program
CLEAR.....              ; Erase existing contents
IF (M11=1)....          ; Enter new version of program...
  ...
CLOSE.....              ; Close buffer at end of program
ENABLE PLC 7            ; Re-enable program
```

See Also:

PLC Program Features
I-variable I5
On-line commands **CLOSE**, **DELETE GATHER**, **ENABLE PLC**

OPEN PROGRAM

Function: Open a fixed motion program buffer for entry

Scope: Port specific

Syntax: **OPEN PROGRAM {constant}**
OPEN PROG {constant}

where:

- **{constant}** is an integer from 1 to 32767 representing the motion program to be opened

This command causes Turbo PMAC to open the specified fixed (non-rotary) motion program buffer for entry or editing on this port. Subsequent program commands valid for motion programs sent from this port will be entered into this buffer. When entry of the program is finished, the **CLOSE** or **CLOSE ALL** command should be used to prevent further lines from being put in the buffer.

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (Turbo PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

No motion programs may be running in any coordinate system when this command is sent (Turbo PMAC will report *ERR001* if I6=1 or 3). As long as a fixed motion program buffer is open, no motion program may be run in any coordinate system (Turbo PMAC will report *ERR015* if I6=1 or 3).

PROGs 1000-32767 can be protected by password. If the PROG is protected by password, and the proper password has not been given, Turbo PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

After any fixed motion program buffer has been opened, each coordinate system must be commanded to point to a motion program with the **B{constant}** command before it can run a motion command (otherwise Turbo PMAC will report *ERR015* if I6=1 or 3)

Example:

```
CLOSE.....              ; Make sure other buffers are closed
DELETE GATHER           ; Make sure memory is free
OPEN PROG 255           ; Open buffer for entry, disabling program
CLEAR.....              ; Erase existing contents
X10 Y20 F5....          ; Enter new version of program...
  ...
CLOSE.....              ; Close buffer at end of program
&1B255R.....            ; Point to this program and run it
```

See Also:

Writing a Motion Program
On-line commands **CLEAR**, **CLOSE**, **DELETE GATHER**
Program Command Specification

OPEN ROTARY

Function: Open all existing rotary motion program buffers for text entry

Scope: Port specific

Syntax: **OPEN ROTARY**
OPEN ROT

This command causes Turbo PMAC to open all existing rotary motion program buffers (created with the **DEFINE ROTARY** command) for entry with ASCII text program lines from this port. Subsequent program commands valid for rotary motion programs sent on this port are entered into the rotary program buffer of the coordinate system addressed at the time of that command.

No other program buffers (PLC, or fixed motion programs) may be open when this command is sent (Turbo PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** or **CLOSE ALL** command to make sure no other buffers have been left open.

Note:

The **B0** command that points the coordinate system to the rotary buffer cannot be given while the rotary buffers are open, because Turbo PMAC will interpret the command as a B-axis move command.

Example:

```
&2 DEFINE ROT 100  ; Create C.S. 2 rotary buffer
&1 DEFINE ROT 100  ; Create C.S. 1 rotary buffer
&1 B0 &2 B0..      ; Point both C.S.s to rotary buffers
```

```
OPEN ROT.....          ; Open buffers for entry
&1 X10 Y10 F5          ; Write to C.S. 1's buffer
&2 X30 Y30 F10         ; Write to C.S. 2's buffer
&1R &2R.....           ; Start executing both buffers
```

See Also:
Rotary Motion Programs (Writing a Motion Program)
On-line commands **CLOSE**, **DEFINE ROT**, **B{constant}**, **R**

P

Function: Report motor position

Scope: Motor specific

Syntax: **P**

This command causes Turbo PMAC to report the present actual position for the addressed motor to the host as a decimal ASCII string, scaled in counts, rounded to the nearest 1/32 of a count.

Turbo PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 1 of Ixx06 is 1 (handwheel offset mode), minus the master position register.

Example:

```
P .....          ; Request the position of the addressed motor
1995 .....       ; Turbo PMAC responds
#1P.....         ; Request position of Motor 1
-0.5 .....       ; Turbo PMAC responds
#3P.....         ; Request position of Motor 3
2.03125 .....    ; Turbo PMAC responds
#2P#4P.....      ; Request positions of Motors 2 and 4
9998 .....       ; Turbo PMAC responds with Motor 2 position first
10002.....       ; Turbo PMAC responds with Motor 4 position next
```

See Also:
On-line commands **<CTRL-P>**, **F**, **V**
Suggested M-variable definitions Mxx62, Mxx64, Mxx67, Mxx69

P{constant}

Function: Report the current P-variable value(s).

Scope: Global

Syntax: **P{constant}[..{constant}]**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the P-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

P{constant},{constant},{constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first P-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number P-variables whose value is to be reported;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each P-variable whose value is to be reported;

This command causes Turbo PMAC to report the current value of the specified P-variable, or range or set of P-variables.

Example:

```
P1 ..................    ; Host asks for value
25 ..................    ; Turbo PMAC responds
P1005
3.444444444
P100..102
17.5
-373
0.0005
P100,3,100..             ; Request for value of P100, P200, P300
33.7
92.13
8.05
```

See Also:

P-Variables (Computational Features)

On-line commands **I{constant}**, **M{constant}**, **Q{constant}**, **P{data}={expression}**

P{data}={expression}

Function: Assign a value to a P-variable.

Scope: Global

Syntax: **P{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the P-variable number;

{expression} contains the value to be given to the specified P-variable

P{constant}..{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first P-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number of the last P-variable; it must be at least as great as the first **{constant}**

the final **{constant}** contains the value to be given to the specified range of P-variables

P{constant},{constant},{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first P-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number P-variables whose value is to be set;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each P-variable whose value is to be set;
- the final **{constant}** contains the value to be given to the specified set of P-variables

This command causes Turbo PMAC to set the specified P-variable or range of P-variables equal to the value on the right side of the equals sign.

If a motion or PLC program buffer is open when the single-variable form of this command is sent to Turbo PMAC, the command will be entered into the buffer for later execution. If a motion or PLC program buffer is open when the multiple-variable form of this command is sent, Turbo PMAC will reject the command with an error, reporting ERR003 if I6 is 1 or 3.

Example:

```
P1=1
P75=P32+P10
P100..199=0
P10=$2000
P832=SIN(3.14159*Q10)
P(10*Q5)=72
P101,16,100=50..          ; Set P101, P201, P301, … P1601 to 50
```

See Also:

P-Variables (Computational Features)

On-line commands **I{data}={expression}**, **M{data}={expression}**, **Q{data}={expression}**, **P{constant}**

Program command **P{data}={expression}**

PASSWORD={string}

Function: Enter/Set Program Password

Scope: Global

Syntax: **PASSWORD={string}**

where:

- **{string}** is a series of non-control ASCII characters (values from 32 decimal to 255 decimal). The password string is case sensitive.

This command permits the user to enter the card's password, or once entered properly, to change it. Without a properly entered password, Turbo PMAC will not open or list the contents of any motion program numbered 1000 or greater, or of PLC programs 0-15. If asked to do so, it will return an error (ERR002 reported if I6 is set to 1 or 3).

The default password is the null password (which means no password is needed to list the programs). This is how the card is shipped from the factory, and also after a **$$$***** re-initialization command. When there is a null password, you are automatically considered to have entered the correct password on power-up/reset.

If you have entered the correct password (which is always the case for the null password), Turbo PMAC interprets the **PASSWORD={string}** command as changing the password, and you can change it to anything you want. When the password is changed, it has automatically been matched and the host computer has access to the protected programs.

> ***Note:***
>
> The password does not require quote marks. If you use quote marks when you enter the password string for the first time, you must use them every time you match this password string.

If you have not yet entered the correct password since the latest power-up/reset, Turbo PMAC interprets the **PASSWORD={string}** command as an attempt to match the existing password. If the command matches the existing password correctly, Turbo PMAC accepts it as a valid command, and the host computer has access to the protected programs until the Turbo PMAC is reset or has its power cycled.

If the command does not match the existing password correctly, Turbo PMAC returns an error (reporting ERR002 if I6=1 or 3), and the host computer does not have access to the protected programs. The host computer is free to attempt to match the existing password.

There is no way to read the current password. If the password is forgotten and access to the protected programs is required, the card must be re-initialized with the **$$$***** command, which clears all program buffers as well as the password. Then the programs must be reloaded, and a new password entered.

Example:

```
{Starting from power-up/reset with a null password}
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because there is no password
RETURN
PASSWORD=Bush             ; This sets the password to "Bush"
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because password has been
RETURN                    ; matched by changing it.
$$$                       ; Reset the card
LIST PLC 1                ; Request listing of protected program
ERR002                    ; PMAC rejects because password not entered
PASSWORD=Reagan           ; Attempt to enter password
ERR002                    ; PMAC rejects as incorrect password
PASSWORD=BUSH             ; Attempt to enter password
ERR002                    ; PMAC rejects as incorrect (wrong case)
PASSWORD=Bush             ; Attempt to enter password; PMAC accepts as correct password
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because password matched
RETURN
PASSWORD=Clinton          ; This changes password to "Clinton"
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because password has been
RETURN                    ; matched by changing it.
$$$                       ; Reset the card
PASSWORD=Clinton          ; Attempt to enter password
                          ; PMAC accepts as correct password
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because password matched
RETURN
```

See Also:

On-line commands **LIST**, **LIST PC**, **LIST PE**, **OPEN**

PAUSE PLC

Function: Pause specified PLC program(s).

Scope: Global

Syntax:

```
PAUSE PLC {constant}[,{constant}...]
PAU PLC {constant}[,{constant}...]
PAUSE PLC {constant}[..{constant}]
PLC {constant}[..{constant}]
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

This command causes Turbo PMAC to stop execution of the specified uncompiled PLC program or programs, with the capability to restart execution at this point (not necessarily at the top) with a **RESUME PLC** command. Execution can also be restarted at the top of the program with the **ENABLE PLC** command.

The on-line **PAUSE PLC** command can only suspend execution of a PLC program at the end of a scan, which is either the end of the program, or at an **ENDWHILE** statement in the program.

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

Example:

```
PAUSE PLC 1
PAU PLC 5
PAU PLC 3,4,7
PAUSE PLC 0..31
```

See Also:

I-variable I5

On-line commands **DISABLE PLC**, **ENABLE PLC**, **OPEN PLC**, **RESUME PLC**, **LIST PLC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **ENABLE PLC**, **PAUSE PLC**, **RESUME PLC**

PC

Function: Report Program Counter

Scope: Coordinate-system specific

Syntax: **PC**

This command causes Turbo PMAC to report the motion program number and address offset of the line in that program that it will next calculate (in the addressed coordinate system). It will also report the program number and address offset of any lines it must **RETURN** to if it is inside a **GOSUB** or **CALL** jump (up to 15 deep).

The number reported after the colon is not a line number; as an address offset, it is the number of words of memory from the top of the program. The **LIST PROGRAM** command, when used with comma delimiters, shows the program or section of the program with address offsets for each line. The **LIST PC** command can show lines of the program with address offsets from the point of calculation.

Because Turbo PMAC calculates ahead in a continuous sequence of moves, the **PC** (Program Calculation) command will in general return a program line further down in the program than **PE** will.

If the coordinate system is not pointing to any motion program, Turbo PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example:

```
PC
P1:0 .....              ; Ready to execute at the top of PROG 1
PC
P76:22.....             ; Ready to execute at 22nd word of PROG 76
LIST PC
P76:22:X10Y20           ; Program line at 22nd word of PROG 76
PC
P1001:35>P3.12          ; Execution will return to PROG 3, address 12
```

See Also:
On-line commands **B{constant}**, **LIST**, **LIST PC**, **LIST PE**, **LIST PROGRAM**, **PE**

PE

Function: Report Program Execution Pointer

Scope: Coordinate-system specific

Syntax: **PE**

This command causes Turbo PMAC to report the motion program number and address offset of the currently executing programmed move in the addressed coordinate system. This is similar to the **PC** command, which reports the program number and address offset of the next move to be calculated. Since Turbo PMAC is calculating ahead in a continuous sequence of moves, **PC** will in general report a move line several moves ahead of **PE**.

If the coordinate system is not pointing to any motion program, Turbo PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example:
```
PE
P1:2
PE
P1:5
```

See Also:
On-line commands **B{constant}**, **LIST**, **LIST PC**, **LIST PE**, **PC**

PMATCH

Function: Re-match Axis Positions to Motor Positions

Scope: Coordinate-system specific

Syntax: **PMATCH**

This command causes Turbo PMAC to recalculate the *axis* starting positions for the coordinate system to match the current *motor* commanded positions (by inverting the axis definition statement equations and solving for the axis position).

This function is executed automatically by Turbo PMAC each time an **R** (run) or **S** (step) command is given, to make sure the first move is executed properly.

The **PMATCH** command does not need to be executed under normal circumstances. However, if something has changed the relationship between motor and axis in the middle of a motion program, this command should be issued – usually with **CMD "PMATCH"** surrounded by **DWELL**s from within the motion program – before the next move command in the program.

These changes include changing the Ixx06 position following mode between normal and offset mode, bringing a new axis into the coordinate system, and changing an axis definition in the coordinate system.

If an axis move is then attempted without the use of the **PMATCH** command, Turbo PMAC will use the wrong *axis* starting point in its calculations, resulting in a jump at the beginning of the move.

If more than one motor is defined to a given axis (as in a gantry system), the commanded position of the lower-numbered motor is used in the Turbo PMAC calculations.

Example:
```
OPEN PROG 10 CLEAR
...
CMD"&1#4->100C"     ; Bring C-axis into coordinate system
DWELL20
```

```
CMD"PMATCH"..; Issue PMATCH so C-axis has proper start position
DWELL20
C90
...
```

See Also:
Further Position Processing (Setting Up a Motor)
Axes, Coordinate Systems (Setting Up a Coordinate System)
I-variable Ixx06

PR

Function: Report Rotary Program Remaining

Scope: Coordinate-system specific

Syntax: **PR**

This command causes Turbo PMAC to report the number of program lines that have been entered in the rotary buffer for the addressed coordinate system but have not yet been executed (program remaining). This command can be useful for finding out if it is time to send new lines to the buffer. The value returned is accurate only if the rotary program buffer is open.

Example:

```
B0 ..... ; Point to rotary buffer
OPEN ROT ..... ; Open rotary buffer
X10F10 ..... ; Enter 1st line
X20 ..... ; Enter 2nd line
X30 ..... ; Enter 3rd line
X40 ..... ; Enter 4th line
PR ..... ; Ask for "program remaining"
4 ..... ; Turbo PMAC responds that 4 lines remain
R ..... ; Start running the program
PR ..... ; Ask for "program remaining"
2 ..... ; Turbo PMAC responds that 2 lines remain
```

See Also:
Rotary Program Buffers (Writing a Motion Program)
BREQ interrupt (Using Interrupts -- Writing a Host Communications Program)
I-variables I16, I17

Q

Function: Quit Program at End of Move

Scope: Coordinate-system specific

Syntax: **Q**

This causes the currently addressed coordinate system to cease execution of the program either at the end of the currently executing move, or at the end of the last move that has been calculated.

If the coordinate system is in segmentation mode (Isx13 > 0) without buffered lookahead (Isx20 = 0), execution will stop at the end of the currently executing move, even if the next move has already been calculated. In this mode, if axes are jogged away, they must be returned to the quit point with the **J=** command before program execution is resumed. Otherwise, Turbo PMAC will reject the resumption command, reporting ERR017 if I6 = 1 or 3.

If the coordinate system is executing with buffered lookahead (Isx13 > 0, Isx20 > 0), no more moves will be added to the lookahead buffer, but all moves in the lookahead buffer will be executed.

If the coordinate system is not in segmentation mode (Isx13 = 0), execution will stop either at the end of the currently executing move, or if one more move has already been calculated, at the end of that move.

The program counter is set to the next line in the program, so execution may be resumed at that point with an **R** or **S** command.

Example:

```
B10R ..... ; Point to beginning of PROG 10 and run
Q ..... ; Quit execution
R ..... ; Resume execution
Q ..... ; Quit execution again
S ..... ; Resume execution for a single move
```

See Also:

Stopping Commands (Making Your Application Safe)
Control-Panel Port STEP/ Input (Connecting Turbo PMAC to the Machine)
JPAN Connector Pin 9
On-line commands **<CTRL-Q>**, **A**, **H**, **K**, **/**

Q{constant}

Function: Report Q-Variable Value

Scope: Coordinate-system specific

Syntax: **Q{constant}[..{constant}]**

where:

- **{constant}** is an integer from 0 to 8191 representing the number of the Q-variable;
- the optional second**{constant}** must be at least as great as the first **{constant}** -- it represents the number of the end of the range;

Q{constant},{constant},{constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first Q-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number Q-variables whose value is to be reported;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each Q-variable whose value is to be reported;

This command causes Turbo PMAC to report back the present value of the specified Q-variable, or range or set of Q-variables for the addressed coordinate system.

Example:

```
Q10
35
Q255
-3.4578
Q101..103
0
98.5
-0.333333333
```

See Also:

Q-Variables (Computational Features)
On-line commands **I{constant}**, **M{constant}**, **P{constant}**, **Q{data}={expression}**

Q{data}={expression}

Function: Q-Variable Value Assignment

Scope: Coordinate-system specific

Syntax: **Q{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the Q-variable number;
- **{expression}** contains the value to be given to the specified Q-variable

Q{constant}..{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first Q-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number of the last Q-variable; it must be at least as great as the first **{constant}**
- the final **{constant}** contains the value to be given to the specified range of Q-variables

Q{constant},{constant},{constant}={constant}

where:

- the first **{constant}** is an integer from 0 to 8190 representing the number of the first Q-variable;
- the second **{constant}** is an integer from 1 to 8191 representing the number Q-variables whose value is to be set;
- the third **{constant}** is an integer from 1 to 8191 representing the numerical spacing between each Q-variable whose value is to be set;
- the final **{constant}** contains the value to be given to the specified set of Q-variables

This command causes Turbo PMAC to assign the value of the expression to the specified Q-variable or range of Q-variables for the addressed coordinate system.

If a motion or PLC program buffer is open when the single-variable form of this command is sent to Turbo PMAC, the command will be entered into the buffer for later execution. If a motion or PLC program buffer is open when the multiple-variable form of this command is sent, Turbo PMAC will reject the command with an error, reporting ERR003 if I6 is 1 or 3.

Example:

```
Q100=2.5
Q20= Q18*SIN(Q19)
Q(P1+P2)=2.71828
Q(200+Q1)=P1*LN(7.5)
Q1..10=0
Q20,5,10=7.5          ; Sets Q20, Q40, Q60, Q80, & Q100 to 7.5
```

See Also:

Q-Variables (Computational Features)

On-line commands **I{data}={expression}**, **M{data}={expression}**, **P{data}={expression}**, **Q{constant}**

Program command **Q{data}={expression}**

R

Function: Run Motion Program

Scope: Coordinate-system specific

Syntax: **R**

This command causes the addressed Turbo PMAC coordinate system to start continuous execution of the motion program addressed by the coordinate system's program counter from the location of the program counter. Alternately, it will restore operation after a '****' or '**H**' command has been issued (even if a program was or is not running). Addressing of the program counter is done initially using the **B{constant}** command.

The coordinate system must be in a proper condition in order for Turbo PMAC to accept this command. Otherwise Turbo PMAC will reject this command with an error; if I6 is 1 or 3, it will report the error number. The following conditions can cause Turbo PMAC to reject this command (also listed are the remedies):

- Both limits set for a motor in coordinate system (ERR010); clear limits
- Another move in progress (ERR011); stop move (e.g. with **J/**)
- Open-loop motor in coordinate system (ERR012); close loop with **J/** or **A**
- Unactivated motor in coordinate system (ERR013); change Ixx00 to 1 or remove motor from coordinate system
- No motors in the coordinate system (ERR014); put at least 1 motor in C.S.
- Fixed motion program buffer open (ERR015); close buffer and point to program
- No program pointed to (ERR015); point to program with **B** command
- Program structured improperly (ERR016); correct program structure
- Motor(s) not at same position as stopped with **H** or **Q** command (ERR017); move back to stopped position with **J=**

Example:

```
&1B1R.....      ; C.S.1 point to PROG 1 and run
&2B200.06 ..... ; C.S.2 point to N6000 of PROG 200 and run
Q ...........   ; Quit this program
R ...........   ; Resume running from point where stopped
H ...........   ; Do a feed hold on this program
R ...........   ; Resume running from point where stopped
```

See Also:

Control Panel Port START/ Input (Connecting Turbo PMAC to the Machine)
Running a Motion Program (Writing a Motion Program)
I-variable I6
On-line commands **<CTRL-R>**, **A**, **H**, **Q**, **S**
JPAN Connector Pin 8

R[H]{address}

Function: Report the contents of specified memory address[es]

Scope: Global

Syntax: **R[H]{address} [,{constant}[,{constant}]]**

where:

- **{address}** consists of a letter **X**, **Y**, or **L**; an optional colon (:); and an integer value from $000000 to $FFFFFF (0 to 16,777,215); specifying the starting Turbo PMAC memory or I/O address to be read;

- **{constant}** (optional) is an integer from 1 to 65,535 specifying the number of consecutive memory addresses to be read; if this is not specified, Turbo PMAC assumes a value of 1;
- the second optional **{constant}** is an integer from 1 to 16 specifying the number of values to be reported on a single line; if this is not specified, Turbo PMAC will fit as many as it can on one line.

This command causes Turbo PMAC to report the contents of the specified memory word address or range of addresses to the host (it is essentially a PEEK command). The command can specify either short (24-bit) word(s) in Turbo PMAC's X-memory, short (24-bit) word(s) in Turbo PMAC's Y-memory, or long (48-bit) words covering both X and Y memory (X-word more significant). This choice is controlled by the use of the **X**, **Y**, or **L** address prefix in the command, respectively.

If the letter **H** is used after the **R** in the command, Turbo PMAC reports back the register contents in unsigned hexadecimal form, with 6 digits for a short word, and 12 digits for a long word. If the letter **H** is not used, Turbo PMAC reports the register contents in signed decimal form.

Example:

```
RHX:$078000..; Request contents of X-reg $078000 (491520) in hex
8F4017.....                ; Turbo PMAC responds in unsigned hex
RHX:491520....             ; Request contents of X-reg 491520 ($078000) in hex
8F4017.....                ; Turbo PMAC responds in unsigned hex (note no '$')
RX:$078000....             ; Request contents of same register in decimal
-7389161.....              ; Turbo PMAC responds in signed decimal
RX:491520 .....            ; Request contents of same register in decimal
-7389161.....              ; Turbo PMAC responds in signed decimal
RX0.....                   ; Request contents of servo cycle counter in decimal
2953211 .....              ; Turbo PMAC responds in signed decimal
RL$000088 .....            ; Request contents of  #1 cmd. pos. reg in decimal
3072000 .....              ; Turbo PMAC responds (=1000 counts)
RHY:$3501,4..              ; Request contents of 1st 4 lines of conversion table
078000 078004 078008 07800C          ; Turbo PMAC responds
RHY:$3501,4,1              ; Request contents of 1st 4 lines of conversion table
078000 .....               ; Turbo PMAC responds, 1 address per line
078004
078008
07800C
```

See Also:

Turbo PMAC Memory Mapping (Computational Features)
On-line command **W{address}**
Memory and I/O Map Description.

RESUME PLC

Function: Resume execution of specified PLC program(s).

Scope: Global

Syntax:

```
RESUME PLC {constant}[,{constant}...]
RES PLC {constant}[,{constant}...]
RESUME PLC {constant}[..{constant}]
RES PLC {constant}[..{constant}]
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

This command causes Turbo PMAC to resume execution of the specified uncompiled PLC program or programs at the point where execution was suspended with the **PAUSE PLC** command. This can either be at the top of the program, or at a point inside the program.

The **RESUME PLC** command cannot be used to restart execution of a PLC program that has been stopped with a **DISABLE PLC** command. However, after a PLC has been stopped with a **DISABLE PLC** command, if a **PAUSE PLC** command is then given for that PLC, then a **RESUME PLC** command can be given to start operation at the point at which it has been stopped.

Note:

RESUME PLC 0..31 will restart all PLCs that have been paused, but not any that have been disabled.

The line of the PLC at which execution will be resumed can be read with the **LIST PLC,,1** command

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If a motion or PLC program buffer is open when this command is sent to Turbo PMAC, the command will be entered into that buffer for later execution.

I-variable I5 must be in the proper state to allow the PLC program(s) specified in this command to execute.

Example:

```
RESUME PLC 1
RES PLC 2,7
RESUME PLC 3,21
RESUME PLC 0..31
```

Note:

If the **RESUME** command refers to multiple PLCs, only those PLCs that have been stopped with the PAUSE command will be resumed.

See Also:

I-variable I5

On-line commands **DISABLE PLC**, **ENABLE PLC**, **OPEN PLC**, **PAUSE PLC**, **LIST PLC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **ENABLE PLC**, **PAUSE PLC, RESUME PLC**

S

Function: Execute One Move (“Step”) of Motion Program

Scope: Coordinate-system specific

Syntax: **S**

This command causes the addressed Turbo PMAC coordinate system to start single-step execution of the motion program addressed by the coordinate system's program counter from the location of the program counter. Addressing of the program counter is done initially using the **B{constant}** command.

At the default Isx53 value of zero, a Step command causes program execution through the next move or **DWELL** command in the program, even if this takes multiple program lines.

When Isx53 is set to 1, a Step command causes program execution of only a single program line, even if there is no move or **DWELL** command on that line. If there is more than one **DWELL** or **DELAY** command on a program line, a single Step command will only execute one of the **DWELL** or **DELAY** commands.

Regardless of the setting of Isx53, if program execution on a Step command encounters a **BLOCKSTART** statement in the program, execution will continue until a **BLOCKSTOP** statement is encountered.

If the coordinate system is already executing a motion program when this command is sent, the command puts the program in single-step mode, so execution will stop at the end of the latest calculated move. In this case, its action is the equivalent of the **Q** command.

The coordinate system must be in a proper condition in order for Turbo PMAC to accept this command. Otherwise Turbo PMAC will reject this command with an error; if I6 is 1 or 3, it will report the error number. The same conditions that cause Turbo PMAC to reject an **R** command will cause it to reject an **S** command; refer to those conditions under the **R** command specification.

Example:

```
&3B20S.....     ; C.S.3 point to beginning of PROG 20 and step
P1 .....        ; Ask for value of P1
1 .....         ; Turbo PMAC responds
S .....         ; Do next step in program
P1 .....        ; Ask for value of P1 again
-3472563.....   ; Turbo PMAC responds --probable problem
```

See Also:

Control Panel Port STEP/ Input (Connecting Turbo PMAC to the Machine)
Running a Motion Program (Writing a Motion Program)
I-variable I6, Isx53
On-line commands **<CTRL-S>**, **A**, **H**, **Q**, **R**, **/**, ****
Program commands **{axis}{data}**, **BLOCKSTART**, **BLOCKSTOP**, **DWELL**, **DELAY**

SAVE

Function: Copy setup parameters to non-volatile memory.

Scope: Global

Syntax: **SAVE**

This command causes Turbo PMAC to copy setup information from active memory to non-volatile memory, so this information can be retained through power-down or reset.

All user setup information, including programs, buffers, and definitions, is copied to flash memory with the **SAVE** command. This information is copied back from flash to active memory during a normal power-up/reset operation. This means that anything changed in Turbo PMAC's active memory that is not saved to flash memory will be lost in a power-on/reset cycle.

The retrieval of information from non-volatile memory on power-up/reset can be inhibited by a jumper on E51 for a Turbo PMAC(1), or on E3 for a Turbo PMAC2.

Note:

Turbo PMAC does not provide the acknowledging handshake character to the **SAVE** command until it has finished the saving operation, about 1 to 2 seconds. The host program should be prepared to wait much longer for this character than is necessary on most commands. For this reason, it is usually not a good idea to include the **SAVE** command as part of a "dump" download of a large file.

Note:

During execution of the **SAVE** command, Turbo PMAC will execute no other background tasks, including user PLCs and automatic safety checks, such as following error and overtravel limits. You must make sure the system is not depending on these tasks for safety when the **SAVE** command is issued.

Example:

```
I130=60000....   ; Set Motor 1 proportional gain
SAVE.............   ; Save to non-volatile memory
I130=80000....   ; Set new value
$$$..............   ; Reset card
I130.............   ; Request value of I130
60000............   ; Turbo PMAC responds with saved value
```

See Also:

On-line commands **$$$**, **$$$*****

Jumpers E51 (PMAC(1)), E3 (PMAC2).

SETPHASE

Function: Set commutation phase position value

Scope: Global

Syntax:
```
SETPHASE {constant} [,{constant}…]
SETPHASE {constant}..{constant}
[,{constant}..{constant}]
```

where:

- **{constant}** is an integer from 1 to 32 representing a motor number

The **SETPHASE** command causes Turbo PMAC to immediately copy the value of Ixx75 for the specified motor or motors into the active phase position register for that motor or motors. This command is typically used to correct the phasing of a motor at a known angle (such as the index pulse of the encoder) after an initial rough phasing (such as from Hall commutation registers).

To determine the value of Ixx75 to be used, first force an unloaded motor to the zero position in its phasing cycle. Next, manually set the phase position register (suggested M-variable Mxx71) to zero. Finally, move the motor to the "known position", usually with a homing search move to the index pulse or other trigger. Read the phase position register at this point and set Ixx75 to this value. For more details, see the Ixx75 description and the Commutation section of the User's Manual.

If a motion program buffer or a PLC program buffer is open when this command is issued, this command will be entered into that buffer as a program command for future execution; it will not be treated as an on-line command.

Examples:

```
SETPHASE1
SETPHASE1,2,3
SETPHASE1..3,5..7
```

SID

Function: Report serial electronic identification number

Scope: Global

Syntax: **SID**

This command causes Turbo PMAC to report the electronic identification number from the Option 18A ID-number module, or the Option 18B ID-number & clock/calendar module.

The identification number is reported as a hexadecimal 16-digit ASCII string, representing a 64-bit value. The first two hex digits represent the 8-bit checksum value for the module; these should match the checksum digits engraved on the case of the module. The last two hex digits represent the module class; these should match the class digits engraved on the case of the module (currently 01 for Option 18A, and 04 for Option 18B). The middle 12 hex digits represent the unique number for each module and board.

If no ID-number module is present, Turbo PMAC will return a 0.

The electronic identification number has no relationship to the serial number that is engraved on the circuit board.

This command is identical to the **IDNUMBER** command.

Example:

```
IDNUMBER
374A256E9014D101
```

SIZE

Function: Report the amount of unused buffer memory in Turbo PMAC.

Scope: Global

Syntax: **SIZE**

This command causes Turbo PMAC to report to the host the amount of unused long words of memory available for buffers. If no program buffer (motion, PLC or rotary buffer) is open, this value is reported as a positive number. If a buffer is currently open, the value is reported as a negative number.

Example:

```
DEFINE GATHER          ; Reserve all remaining memory for gathering
SIZE.....              ; Ask for amount of open memory
0 .....                ; PMAC reports none available
DELETE GATHER          ; Free up memory from gathering buffer
SIZE.....              ; Ask for amount of open memory
25232.....             ; PMAC reports number of words available
OPEN PROG 10           ; Open a motion program buffer
SIZE.....              ; Ask for amount of open memory
-25232.....            ; The negative sign shows a buffer is open
```

See Also:

I-Variable I18

On-line commands **DELETE GATHER**

STN

Function: Report MACRO station order number

Scope: Global

Syntax: **STN**

This command causes Turbo PMAC to return its station order number, intended to represent the order of the Turbo PMAC in the MACRO ring. This value is stored in variable I85, and can take a value of 0 to 254. When the Turbo PMAC has a station order number of 1 to 254, and receives a **MACROSTASCII{constant}** command whose **{constant}** value matches the station order number, it will respond to the command.

This station order number is set with the **STN={constant}** or the **I85={constant}** command. If no station order number has been assigned to the Turbo PMAC, it will return a value of 0 to this command. The first device in the ring with a station order number of 0 will respond to a **MACROSTASCII255** command.

See Also:
I-variable I85
Commands **STN={constant}**, **MACROSTASCII**

STN={constant}

Function: Set MACRO station order number

Scope: Global

Syntax: **STN={constant}**

where:

- **{constant}** is a value in the range 0 to 254 representing the order of the Turbo PMAC in the MACRO ring

This command causes Turbo PMAC to set its station order number, intended to be the order of the Turbo PMAC in the MACRO ring. When the Turbo PMAC has a station order number of 1 to 254, and receives a **MACROSTASCII{constant}** command whose **{constant}** value matches the station order number, it will respond to the command. The first device in the ring with a station order number of 0 (no order established) will respond to a **MACROSTASCII255** command.

See Also:
I-variable I85
Commands **STN**, **MACROSTASCII**

TIME

Function: Report present time

Scope: Global

Syntax: **TIME**

This command causes Turbo PMAC to report the current time of day. The time is reported in the international 24-hour clock format:

`{hh}:{mm}:{ss}`

where:

- `{hh}` is the 2-digit representation of the hour (00 <= `{hh}` <= 23)
- `{mm}` is the 2-digit representation of the minute (00 <= `{mm}` <= 59)
- `{ss}` is the 2-digit representation of the second (00 <= `{ss}` <= 59)

The time must originally have been set with the **TIME={time}** command for the time reported here to be valid. If the Option 18B non-volatile clock/calendar IC is present, this IC will retain and advance the time information even while no external power is applied. On power-up or reset of the Turbo PMAC, the present date and time are copied from the module into active memory, and the time information is then advanced in active memory. The **TIME** command reports from active memory.

If the Option 18B is not present, Turbo PMAC will default to Jan. 1, 2000, at 00:00:00, on power-up/reset, and date and time will advance in active memory from this point. The actual date may be loaded into active memory after power-up/reset, and the date and time will advance from this point.

Examples:

```
TIME
00:01:42          ; 0 hours, 1 minute, 42 seconds
TIME
13:47:22          ; 13 hours, 47 minutes, 22 seconds (1:47:22 pm)
```

TIME={time}

Function: Set the present time

Scope: Global

Syntax: **TIME={hh}:{mm}[:{ss}]**

where:

- **{hh}** is the 2-digit representation of the hour (00 <= **{hh}** <= 23)
- **{mm}** is the 2-digit representation of the minute (00 <= **{mm}** <= 59)
- **{ss}** is the 2-digit representation of the second (00 <= **{ss}** <= 59)

This command sets the current time in Turbo PMAC's active (volatile) memory. The time is entered in the international 24-hour clock format, as *hour/minute/second*. Entering of the seconds is optional; if the seconds are not entered, Turbo PMAC will use 0 seconds. Date and time automatically advance in Turbo PMAC's active memory, and the present time can be queried at any time with the **TIME** command.

This time information is not retained through a power-down or board reset unless the Option 18B non-volatile clock/calendar IC is present *and* the active date and time information has been copied into the non-volatile IC with the **UPDATE** command.

The present date can be set with the **TODAY={date}** command.

Examples:

```
TIME=08:30          ; Set time to 8:30:00 am
TIME=17:45:24       ; Set time to 5:45:24 pm
```

TODAY

Function: Report present date

Scope: Global

Syntax: **TODAY**

This command causes Turbo PMAC to report the current date. The date is reported in the North American format:

{day} {mm}/{dd}/{yyyy}

where:

- **{day}** is the 3-letter abbreviation for the day of the week
- **{mm}** is the 2-digit representation of the month

- **{dd}** is the 2-digit representation of the day of the month
- **{yyyy}** is the 4-digit representation of the year

The date must originally have been set with the **TODAY={date}** command for an accurate date to be reported here. If the Option 18B non-volatile clock/calendar IC is present, this IC will retain and advance the date information even while no external power is applied. On power-up or reset of the Turbo PMAC, the present date and time are copied into active memory, and then they advance automatically in the active memory. The **TODAY** command reports from active memory.

If the Option 18B is not present, Turbo PMAC will default to Jan. 1, 2000, at 00:00:00, on power-up/reset, and date and time will advance in active memory from this point. The actual date may be loaded into active memory after power-up/reset, and the date and time will advance from this point.

Note:

The 4-digit representation of the year eliminates possible "Year 2000" problems in user code processing the date information.

Examples:

```
TODAY
WED 09/09/1998
TODAY
FRI 04/22/2000
```

TODAY={date}

Function: Set the present date

Scope: Global

Syntax: **TODAY={mm}/{dd}/{yyyy}**

where:

- **{mm}** is the 2-digit representation of the month
- **{dd}** is the 2-digit representation of the day of the month
- **{yyyy}** is the 4-digit representation of the year

This command sets the current date in Turbo PMAC's active (volatile) memory. The date is entered in the North American style, as *month*/*day*/*year*. Date and time automatically advance in Turbo PMAC's active memory, and the present date can be queried at any time with the **TODAY** command.

Note:

The 4-digit representation of the year eliminates possible "Year 2000" problems in user code processing the date information.

This date information is not retained through a power-down or board reset unless the Option 18B non-volatile clock/calendar IC is present *and* the active date and time information have been copied into the non-volatile IC with the **UPDATE** command.

The present time can be set with the **TIME={time}** command.

Examples:

```
TODAY=10/30/1998        ; Set date to October 30, 1998
TODAY=01/02/1999        ; Set date to January 2, 1999
TODAY=05/09/2001        ; Set date to May 9, 2001
```

TYPE

Function: Report type of Turbo PMAC

Scope: Global

Syntax: **TYPE**

This command causes Turbo PMAC to return a string reporting the configuration of the card. It will report the configuration as a text string in the format:

{Turbo PMAC type},{Clock Multiplier}

where:

{Turbo PMAC type} can take the following values:

TURBO1	Turbo PMAC (1)
TURBO2	Turbo PMAC2 (non-Ultralite)
TURBOU	Turbo PMAC2 Ultralite (MACRO only)

{Clock multiplier} is reported in the format

Xn

where n is the multiplication of crystal frequency to CPU frequency.

Since the crystal frequency is always 20 MHz, an "X4" value reports 80 MHz operation. The reported clock multiplier value shows at what frequency the CPU is actually operating, as controlled by I52, not necessarily the maximum frequency rating of the CPU hardware.

Examples:

```
TYPE
TURBO1, X4
TYPE
TURBO2, X4
TYPE
TURBOU, X5
```

See Also:

On-line commands **VERSION, DATE**

UNDEFINE

Function: Erase Coordinate System Definition

Scope: Coordinate-system specific

Syntax: **UNDEFINE**
UNDEF

This command causes Turbo PMAC to erase all of the axis definition statements in the addressed coordinate system. It does not affect the axis definition statements in any other coordinate systems. It can be useful to use before making new axis definitions.

To erase the axis definition statement of a single motor only, use the **#{constant}->0** command; to erase all the axis definition statements in every coordinate system, use the **UNDEFINE ALL** command.

Example:

```
&1 ..................  ; Address C.S.1
#1-> ................  ; Ask for axis definition of Motor 1
10000X..............  ; Turbo PMAC responds
#2-> ................  ; Ask for axis definition of Motor 2
10000Y..............  ; Turbo PMAC responds
UNDEFINE.........  ; Erase axis definitions
&2 ..................  ; Address C.S.2
```

```
#1->10000X....      ; Redefine Motor 1 as X-axis in C.S.2
#2->10000Y....      ; Redefine Motor 2 as Y-axis in C.S.2
```

See Also:
Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->**, **#{constant}->0**, **UNDEFINE ALL**

UNDEFINE ALL

Function: Erase coordinate definitions in all coordinate systems

Scope: Global

Syntax: **UNDEFINE ALL**
UNDEF ALL

This command causes all of the axis definition statements in all coordinate systems to be cleared. It is a useful way of starting over on a reload of Turbo PMAC's coordinate system definitions.

Example:

```
&1#1->.............  ; Request axis definition of Motor 1 in C.S. 1
1000X..............  ; Turbo PMAC responds
&2#5->.............  ; Request axis definition of Motor 5 in C.S. 2
1000X..............  ; Turbo PMAC responds
UNDEFINE ALL       ; Erase all axis definitions
&1#1->.............  ; Request axis definition of Motor 1 in C.S. 1
0 ..................  ; Turbo PMAC responds that there is no definition
&2#5->.............  ; Request axis definition of Motor 5 in C.S. 2
0 ..................  ; Turbo PMAC responds that there is no definition
```

See Also:
Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->0**, **#{constant}->**, **UNDEFINE**.

UNLOCK{constant}

Function: Clear process locking bit

Scope: Global

Syntax: **UNLOCK{constant}**

where:

- **{constant}** is an integer from 0 to 7 representing the number of the locking bit

The **UNLOCK** command permits the user to clear one of the 8 process locking bits in Turbo PMAC, releasing the process for other tasks. These locking bits can prevent conflicts between tasks of different priorities attempting to manipulate the same register. On-line commands and PLCs 1 –31 are background tasks; motion programs and PLC 0 are higher-priority foreground tasks.

The user can check the status of a locking bit with the **LOCK** command.

The status of locking bits 0 – 7 is reported as bits 4 – 11, respectively, of I4904.

If a motion program buffer or a PLC program buffer is open when this command is issued, this command will be entered into that buffer as a program command for future execution; it will not be treated as an on-line command.

UPDATE

Function: Copy present date and time to non-volatile storage

Scope: Global

Syntax: **UPDATE**

This command causes Turbo PMAC to copy the present date and time information from active volatile memory to the Option 18B non-volatile clock and calendar IC. Date and time information that has been entered into active memory is not retained through a power-down or board reset unless the **UPDATE** command has been used once to put this information in the non-volatile IC as well.

If there is no Option 18B non-volatile clock and calendar IC present on the Turbo PMAC, this command is rejected with an error, with the Turbo PMAC reporting an ERR003 if I6 is set to 1 or 3.

Example:

```
TODAY=06/15/1998          ; Set present date in active memory
UPDATE                    ; Copy to non-volatile IC
$$$                       ; Reset Turbo PMAC
TODAY                     ; Request present date
06/15/1998                ; Turbo PMAC reports date
TODAY=11/23/2000          ; Set new present date
$$$                       ; Reset board (without updating)
TODAY                     ; Request present date
06/15/1998                ; Turbo PMAC reports last UPDATEd date
```

V

Function: Report motor velocity

Scope: Motor specific

Syntax: **V**

This command causes Turbo PMAC to report the velocities of the addressed motor to the host. The velocity units are typically scaled in encoder counts per servo cycle, rounded to the nearest tenth.

To scale these values into counts/msec, multiply the response by 8,388,608/I10 (servo cycles/msec).

This command returns filtered velocity values, with the filter time constant controlled by global variables I60 and I61. It does not report the raw velocity register calculated by the servo loop each servo cycle.

Example:

```
V ..................... ; Request actual velocity of addressed motor
21.9 .................. ; Turbo PMAC responds with 21.9 cts/cycle
..................... ; (*8,388,608/3,713,707 = 49.5 cts/msec)
#6V..................... ; Request velocity of Motor 6
-4.2 .................. ; Turbo PMAC responds
#5V#2V..... ; Request velocities of Motors 5 and 2
0 ..................... ; Turbo PMAC responds with Motor 5 first
7.6..... ; Turbo PMAC responds with Motor 2 second
```

See Also:

I-variables I10, Ixx09, Ixx60

On-line commands **<CTRL-V>**, **F**, **P**

VERSION

Function: Report PROM firmware version number

Scope: Global

Syntax: **VERSION**
VER

This command causes Turbo PMAC to report the firmware version it is using.

When a Turbo PMAC is in "bootstrap mode" (powering up with bootstrap jumper on) for re-load of the operational firmware, Turbo PMAC will report the version of the bootstrap firmware, not the operational firmware. Otherwise, it will report the operational firmware version.

Example:

```
VERSION ..... ; Ask Turbo PMAC for firmware version
1.934 ..... ; Turbo PMAC responds
```

See Also:

Resetting Turbo PMAC (Talking to Turbo PMAC)
On-line command **DATE**, **EAVERSION**, **TYPE**

VID

Function: Report vendor identification number

Scope: Global

Syntax: **VID**

This command causes Turbo PMAC to return the vendor ID number. For a Delta Tau product such as the Turbo PMAC, the vendor ID number is 1. This command is part of a multi-vendor identification architecture for the MACRO ring.

See Also:

Commands **CID**, **IDNUMBER**, **SID**, **VID**

W{address}

Function: Write value(s) to a specified address(es).

Scope: Global

Syntax: **W{address},{value} [,{value}...]**

where:

- **{address}** consists of a letter **X**, **Y**, or **L**; an option colon (:); and an integer value from , $000000 to $FFFFFF (0 to 16,777,215); specifying the starting Turbo PMAC memory or I/O address to be read;
- **{constant}** is an integer, specified in decimal or hexadecimal, specifying the value to be written to the specified address;
- further **{constants}** specify integer values to be written into subsequent consecutive higher addresses

This command causes Turbo PMAC to write the specified **{constant}** value to the specified memory word address, or if a series of **{constant}** values is specified, to write them to consecutive memory locations starting at the specified address (it is essentially a memory POKE command). The command can specify either short (24-bit) word(s) in Turbo PMAC's X-memory, short (24-bit) word(s) in Turbo PMAC's Y-memory, or long (48-bit) words covering both X and Y memory (X-word more significant). This choice is controlled by the use of the **X**, **Y**, or **L** address prefix in the command, respectively.

Example:

```
WY:$078002,4194304        ; This should put 5V on DAC2
.........                 ; (provided I200=0 so servo does not overwrite)
WY$003501,$078000,$078004,$078008,$07800C
.........                 ; This writes the first four entries to the
.........                 ; encoder conversion table
```

See Also:

On-line command **R[H]{address}**

Memory and I/O Map.

Z

Function: Make commanded axis positions zero.

Scope: Coordinate-system specific

Syntax: **Z**

This command causes Turbo PMAC to re-label the current commanded axis positions for all axes in the coordinate system as zero. It does not cause any movement; it merely re-names the current position.

This command is simply a short way of executing **{axis}=0** for all axes in the coordinate system. **PSET X0 Y0** (etc.) is the equivalent motion program command.

This does not set the *motor* position registers to zero; it changes motor position bias registers to reflect the new offset between motor zero positions and axis zero positions. However, the motor reported positions will reflect the new bias, and report positions of zero (+/- the following error)

Example:

```
<CTRL-P>....                ; Ask for reported motor positions
2001 5002 3000 0 0 0 0 0    ; Turbo PMAC reports positions
Z ..........                ; Zero axis positions
<CTRL-P>.....               ; Ask for reported motor positions again
1 2 -1 0 0 0 0 0            ; Turbo PMAC responds
```

See Also:

On-line commands **{axis}={constant}**

Memory map registers D:$0000CC, D:$00014C, etc.

Suggested M-variable definitions Mxx64.

TURBO PMAC PROGRAM COMMAND SPECIFICATION

{axis}{data}[{axis}{data}...]

Function: Position-Only Move Specification

Type: Motion program (PROG and ROT)

Syntax: **{axis}{data}[{axis}{data}...]**

where:

- **{axis}** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **{data}** is a constant (no parentheses) or an expression (in parentheses) representing the end position or distance;
- **[{axis}{data}...]** is the optional specification of simultaneous movement for more axes.

This is the basic Turbo PMAC move specification statement. It consists of one or more groupings of an axis label and its associated value. The value for an axis is scaled (units determined by the axis definition statement); it represents a position if the axis is in absolute (ABS) mode, or a distance if the axis is in incremental (INC) mode. The order in which the axes are specified does not matter.

This command tells the axes *where* to move; it does not tell them *how* to move there. Other program commands and parameters define how; these must be set up ahead of time.

The type of motion a given motion command causes are dependent on the mode of motion and the state of the system at the beginning of the move.

Examples:

```
X1000
X(P1+P2)
Y(Q100+500) Z35 C(P100)
X1000 Y1000
A(P1) B(P2) C(P3)
X(Q1*SIN(Q2/Q3)) U500
```

See Also:

Axis Definition Statements (Setting Up a Coordinate System)
Motion program commands **LINEAR**, **CIRCLEn**, **RAPID**, **SPLINE1**, **PVT**
Motion program commands **TA**, **TS**, **TM**, **F**, **ABS**, **INC**
I-variables Isx87, Isx88, Isx89, Isx90.

{axis}{data}:{data} [{axis}{data}:{data}...]

Function: Position and Velocity Move Specification

Type: Motion program (PROG and ROT)

Syntax: **{axis}{data}:{data} [{axis}{data}:{data}...]**

where:

- **{axis}** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **{data}** is a constant (no parentheses) or an expression (in parentheses) representing the end position or distance;
- **:{data}** represents the ending velocity
- **[{axis}{data}:{data}...]** is the optional specification of simultaneous movement for more axes.

In the case of PVT (position, velocity, time) motion mode, both the ending position and velocity are specified for each segment of each axis. The command consists of one or more groupings of axis labels with two data items separated by a colon character.

The first data item for each axis is the scaled ending position or distance (depending on whether the axis is in absolute (**ABS**) or incremental (**INC**) mode; position scaling is determined by the axis definition statement), and the second data item (after the colon) is the ending velocity.

The velocity units are the scaled position units as established by the axis definition statements divided by the time units as set by Isx90 for Coordinate System x. The velocity here is a signed quantity, not just a magnitude. See the examples in the PVT mode description of the Writing a Motion Program section.

The time for the segment is the argument for the most recently executed **PVT** or **TM** command.

In PVT mode, if no velocity is given for the segment, Turbo PMAC assumes an ending velocity of zero for the segment.

Examples:

```
X1000:50
Y500:-32 Z737.2:68.93
A(P1+P2):(P3) B(SIN(Q1)):0
```

See Also:

PVT Mode Moves (Writing a Motion Program)
Axis Definition Statements (Setting Up a Coordinate System)
I-variables Isx89, Isx90
Motion program commands **PVT**, **TM**

{axis}{data}^{data}[{axis}{data}^{data}...]

Function: Move Until Trigger

Type: Motion program

Syntax: **{axis}{data}^{data}[{axis}{data}^{data}...]**

where:

- **{axis}** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W);
- the first **{data}** is a constant (no parentheses) or expression (in parentheses) representing the end position or distance in the absence of a trigger;
- the second **{data}** (after the ^ arrow) is a constant (no parentheses) or expression (in parentheses) representing the distance from the trigger position;
- **[{axis}{data}^{data}...]** is the optional specification of simultaneous movement for more axes

In the **RAPID** move mode, this move specification permits a move-until-trigger function. The first part of the move description for an axis -- before the ^ sign -- specifies where to move in the absence of a trigger. It is a position if the axis is in absolute mode; it is a distance if the axis is in incremental mode. In both cases the units are the scaled axis user units. If no trigger is found before this destination is reached, the move is a standard **RAPID** move.

The second part of the move description for an axis -- after the ^ sign -- specifies the distance from the trigger position to end the post-trigger move if a trigger is found. The distance is expressed in the scaled axis user units.

Each motor assigned to an axis specified in the command executes a separate move-until-trigger. All the assigned motors will start together, but each can have its own trigger condition. If a common trigger is required, the trigger signal must be wired into all motor interfaces. Each motor can finish at a separate time; the next line in the program will not start to execute until all motors have finished their moves. No blending into the next move is possible.

The trigger for a motor can be either a hardware input trigger if bit 1 of Ixx97 is 0, or the motor warning following error status bit if bit 1 of Ixx97 is 1 (bit 0 of Ixx97 should also be set to 1 in this case). If a hardware input trigger is used, I-variables I7mn2 and I7mn3 (e.g. I7012 and I7013) for the flag channel specified by Ixx25 determine which edge(s) of which flag(s) cause the trigger. If the warning following error bit is used for "torque-limited" triggering, then Ixx12 sets the size of the warning following error.

The speed of the move, both before the trigger and after, is set by Ixx22 if Ixx90=0 or by Ixx16 if Ixx90=1. The acceleration is set by Ixx19 to Ixx21.

On the same line, some axes may be specified for normal untriggered **RAPID** moves that will execute starting simultaneously.

If the move ends for a motor without a trigger being found, the "trigger move" status bit (bit 7 of the second motor status word returned on a **?** command) is left set after the end of the move. If the trigger has been found, this bit is cleared to 0 at the end of the move.

Example:

```
X1000^0
X10^-0.01 Y5.43^0.05
A(P1)^(P2) B10^200 C(P3)^0 X10
```

See Also:

Move-Until-Trigger (Writing a Motion Program)
Torque-Limited Triggering (Setting Up a Motor)
RAPID-mode moves (Writing a Motion Program)

{axis}{data}[{axis}{data}...] {vector}{data} [{vector}{data}...]

Function: Circular Arc Move Specification

Type: Motion program (PROG and ROT)

Syntax: **{axis}{data} [{axis}{data}...] {vector}{data} [{vector}{data}...]**

where:

- **{axis}** is a character specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **{data}** is a constant (no parentheses) or an expression (in parentheses) representing the end position or distance;
- **[{axis}{data}...]** is the optional specification of simultaneous movement for more axes;
- **{vector}** is a character (I, J, or K) specifying a vector component (parallel to the X, Y, or Z axis, respectively) to the center of the arc; or the character R specifying the magnitude of the vector;
- **{data}** specifies the magnitude of the vector component;
- **[{vector}{data}...]** is the optional specification of more vector components.

For a blended circular mode move, both the move endpoint and the vector to the arc center are specified. The endpoint is specified just as in a **LINEAR** mode move, either by position (referenced to the coordinate system origin), or distance (referenced to the starting position).

The center of the arc for a circular move must also be specified in the move command. Usually this is done by defining the vector to the center. This vector can either be referenced to the starting point of the move (incremental radial vector mode -- the default, or if an **INC (R)** command has been given), or it can be referenced to the coordinate system origin (absolute radial vector mode -- if an **ABS (R)** command has been given).

Alternatively, just the magnitude of the vector to the center can be specified with **R{data}** on the command line. If this is the case, Turbo PMAC will calculate the location of the center itself. If the value specified by **{data}** is positive, Turbo PMAC will compute the short arc path to the destination (<= 180^{o}); if it is negative, Turbo PMAC will compute the long arc path (>= 180^{o}). It is not possible to specify a full circle in one command with the R vector specifier.

The plane for the circular arc must have been defined by the **NORMAL** command (the default -- **NORMAL K-1** -- defines the XY plane). This command can only define planes in XYZ-space, which means that only the X, Y, and Z axes can be used for circular interpolation. Other axes specified in the same move command will be interpolated linearly to finish in the same time.

The direction of the arc to the destination point -- clockwise or counterclockwise -- is controlled by whether the card is in **CIRCLE1** (clockwise) or **CIRCLE2** (counterclockwise) mode. The sense of clockwise in the plane is determined by the direction of the **NORMAL** vector to the plane.

If the destination point is a different distance from the center point than is the starting point, the radius is changed smoothly through the course of the move, creating a spiral. This is useful in compensating for any round off errors in the specifications. However, if the distance from either the starting point or the destination point to the center point is zero, and error condition will be generated and the program will stop.

If the vector from the starting point to the center point does not lie in the circular interpolation plane, the projection of that vector into the plane is used. If the destination point does not lie in the same circular interpolation plane as the starting point, a helical move is done to the destination point.

If the destination point (or its projection into the circular interpolation plane containing the starting point) is the same as the starting point, a full 360^{o} arc is made in the specified direction (provided that IJK vector specification is used). In this case, only the vector needs to be specified in the move command, because for any axis whose destination is not specified, the destination point is automatically taken to be the same as the starting point.

If no vector, and no radial magnitude, is specified in the move command, a linear move will be done to the destination point, even if the program is in circular mode.

Note:

Turbo PMAC performs arc moves by segmenting the arc and performing the best cubic fit on each segment. I-variable Isx13 determines the time for each segment. Isx 13 *must* be set greater than zero to put the coordinate system into this segmentation mode in order for arc moves to be done. If Isx13 is set to zero, circular arc moves will be done in linear fashion.

Examples:

```
X5000 Y3000 I1000 J1000
X(P101) Z(P102) I(P201) K(P202)
X10 I5
X10 Y20 C5 I5 J5
Y5 Z3 R2
J10                    ; Specifies a full circle of 10 unit radius
```

See Also:
Circular Moves (Writing a Motion Program)
I-variables Isx13, Isx87, Isx88, Isx89, Isx90
Program commands **NORMAL**, **ABS**, **INC**, **CIRCLE1**, **CIRCLE2**, **TA**, **TS**, **TM**, **F**

A{data}

Function: A-Axis Move

Type: Motion program (PROG or ROT)

Syntax: **A{data}**

where:

- **{data}** is a floating-point constant or expression representing the position or distance in user units for the A-axis

This command causes a move of the A-axis. (See **{axis}{data}** descriptions, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:
```
A10
A(P23)
A25 B10 Z35
A(20*SIN(Q5))
CALL50 A-10
```

See Also:
Program commands **{axis}{data}**, **B**, **C**, **U**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**

ABS

Function: Absolute Move Mode

Type: Motion program (PROG and ROT)

Syntax: **ABS [({axis}[,{axis}...])]**

where:

- **{axis}** is a character (X,Y,Z,A,B,C,U,V,W) representing the axis to be specified, or the character R to specify radial vector mode

> ***Note:***
>
> No spaces are permitted in this command.

The **ABS** command without arguments causes all subsequent positions in motion commands for all axes in the coordinate system running the motion program to be treated as absolute positions. This is known as absolute mode, and it is the power-on default condition.

An **ABS** statement with arguments causes the specified axes in the coordinate system running the program to be in absolute mode, and all others stay the way they were before.

If **R** is specified as one of the 'axes', the I, J, and K terms of the circular move radius vector specification will be specified in absolute form (i.e. as a vector from the origin, not from the move start point). An **ABS** command without any arguments does not affect this vector specification. The default radial vector specification is incremental.

If no motion program buffer is open when this command is sent to Turbo PMAC, it will be executed as an on-line coordinate system command.

Examples:
```
ABS(X,Y)
ABS
ABS(V)
ABS(R)
```

See Also:
Circular Moves (Writing a Motion Program)
On-line commands **ABS**, **INC**
Program commands **{axis}{data}**, **{axis}{data}{vector}{data}**, **INC**

ADDRESS

Function: Motor/Coordinate System Modal Addressing

Type: PLC programs 1 to 31 **only**

Syntax:
```
ADDRESS [#{constant}][&{constant}]
ADR [#{constant}][&{constant}]
```

where:

- **{constant}** following the '#' character is an integer constant from 1 to 32 representing the motor number to be addressed;
- **{constant}** following the '&' character is an integer constant from 1 to 16 representing the coordinate system number to be addressed

This statement, when executed, sets the motor and/or coordinate system that will be addressed by this particular PLC program when it **COMMAND**s motor- or coordinate-system-specific commands with no addressing in those commands. The addressed coordinate system also controls which set of Q-variables is accessed, even for ATAN2 functions, which automatically use Q0.

This command does not affect host addressing, the addressing of other PLC programs, or the selection of the control panel inputs. The addressing stays in effect until another **ADDRESS** statement supersedes it. Default addressing at power-on/reset is #1 and &1.

In motion programs, there is no modal addressing for **COMMAND** statements; each **COMMAND** statement must contain the motor or coordinate system specifier within its quotation marks. A motion program automatically operates on the Q-variables of the coordinate system executing the program.

Examples:
```
ADDRESS &4
ADR #2
ADDRESS &2#2
```

```
ADR#1.....          ; Modally address Motor 1
CMD"J+" .....       ; This will start Motor 1 jogging
CMD"#2J+" .....     ; This will start Motor 2 jogging
CMD"J/" .....       ; This will stop Motor 1
```

See Also:
Addressing Modes (Talking To Turbo PMAC)
Q-Variables (Program Computational Features)
Program commands **ADDRESS#P**, **ADDRESS&P**, **COMMAND**, **Q{constant}={expression}**

ADDRESS#P{constant}

Function: Select program's addressed motor

Type: PLC programs 1 to 31 **only**

Syntax: **ADDRESS#P{constant}**
ADR#P{constant}

where:

- **{constant}** is an integer from 0 to 8191 representing the P-variable number whose value determines the motor to be addressed

This statement, when executed, selects the motor that will be addressed by this particular PLC program when it issues motor-specific commands with subsequent **COMMAND** statements. The statement specifies the number of a P-variable whose value determines which motor is addressed. For example, if variable P7 had a value of 10, the statement **ADDRESS#P7**, when executed, would select Motor 10.

This statement permits the user to mathematically determine which motor will be addressed. The value of the floating-point P-variable is rounded to the nearest integer, and if this integer value is out of the range 1 – 32, the remainder of the number when divided by 32 is used. For example, if the P-variable had a value of 34.3, Motor 2 would be addressed (remainder of 34/32 is 2).

This statement does not affect subsequent **COMMAND** statements in this PLC that explicitly address a motor. It does not affect **COMMAND** statements in any other PLC program. It does not affect any motor-specific commands from a host computer issued over any of the communications ports

Examples:

```
P4=12               ; Set value of P4 to 12
ADDRESS#P4          ; Address Motor 12
COMMAND "J+"        ; Jog addressed motor (#12) in
                    ; positive direction
COMMAND "#11J-"     ; Jog motor 11 in negative direction
COMMAND "J/"        ; Stop jogging addressed motor
                    ; (#12 still!)

; Routine to home all odd-numbered motors
P100=1
WHILE (P100<32)           ; Loop thru motors
    ADDRESS#P100          ; Address motor using P100 value
    COMMAND "HM"          ; Start homing addressed motor
    P100=P100+2           ; Increment to next odd-numbered
                          ; motor
ENDWHILE
```

ADDRESS&P{constant}

Function: Select program's addressed coordinate system

Type: PLC programs 1 to 31 **only**

Syntax: **ADDRESS&P{constant}**
ADR&P{constant}

where:

- **{constant}** is an integer from 0 to 8191 representing the P-variable number whose value determines the coordinate to be addressed

This statement, when executed, selects the coordinate system that will be addressed by this particular PLC program when it issues coordinate-system-specific commands with subsequent

COMMAND statements and Q-variable assignments. The statement specifies the number of a P-variable whose value determines which coordinate system is addressed. For example, if variable P4 had a value of 12, the statement **ADDRESS&P4**, when executed, would select Motor 12.

This statement permits the user to mathematically determine which coordinate system will be addressed. The value of the floating-point P-variable is rounded to the nearest integer, and if this integer value is out of the range 1 – 16, the remainder of the number when divided by 16 is used. For example, if the P-variable had a value of 19.2, Motor 3 would be addressed (remainder of 19/16 is 3).

This statement does not affect subsequent **COMMAND** statements in this PLC that explicitly address a coordinate system. It does not affect **COMMAND** statements in any other PLC program. It does not affect any coordinate-system-specific commands from a host computer issued over any of the communications ports

Examples:

```
P9=11                       ; Set value of P9 to 11
ADDRESS&P9                  ; Address C.S. 11
COMMAND "R"                 ; Run program in addressed C.S.
COMMAND "&2S"               ; Single-step program in C.S. 2
Q10=3                       ; Set value of Q10 in addressed C.S. (&11 still!)
```

```
; Routine to hold first 8 coordinate systems
P10=1
WHILE (P10<9)               ; Loop thru motors
    ADDRESS&P10             ; Address C.S. using P10 value
    COMMAND "H"             ; Feed hold addressed C.S.
    P10=P10+1               ; Increment to next C.S.
ENDWHILE
```

ADIS{constant}

Function: Absolute displacement of X, Y, and Z axes

Type: Motion program (PROG and ROT)

Syntax: **ADIS{constant}**

where:

- **{constant}** is an integer constant representing the number of the first of three consecutive Q-variables to be used in the displacement vector

This command loads the currently selected (with **TSEL**) transformation matrix for the coordinate system with offset values contained in the three Q-variables starting with the specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) to the values of these variables (X=Q{data}, Y=Q({data}+1), Z=Q({data}+2)).

This command does not cause any movement of any axes; it simply renames the present positions.

This command is almost equivalent to a **PSET X(Q{data}) Y(Q({data}+1)) Z(Q({data}+2))** command, except that **ADIS** does not force a stop between moves, as **PSET** does.

Examples:

```
Q20=7.5
Q21=12.5
Q22=25
ADIS 20.....                ; This makes the current X position 7.5, Y 12.5, Z25
```

See Also:
Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **AROT**, **IDIS**, **IROT**, **TINIT**

AND ({condition})

Function: Conditional AND

Type: PLC program only

Syntax: **AND ({condition})**

where:

- **{condition}** is a simple or compound condition

This statement forms part of an extended compound **IF** or **WHILE** condition to be evaluated in a PLC program. It must immediately follow an **IF**, **WHILE**, **AND**, or **OR** statement. This **AND** is a Boolean operator logically combining the full conditions on its line and the program line immediately above. It takes lower precedence than **AND** or **OR** operators within a compound condition on a single line (the parentheses cause those to be executed first), but it takes higher precedence than an **OR** operator that starts a line.

In motion programs, there can be compound conditions within one program line, but not across multiple program lines, so this statement is not permitted in motion programs.

> ***Note:***
>
> This logical **AND** command, which acts on conditions, should not be confused with the bit-by-bit **&** (ampersand) operator that acts on values.

Examples:

```
IF (M11=1)....          ; This branch will start a motion program running
AND (M12=1).. ; on a cycle where inputs M11 and M12 are 1 and
AND (M21=0)..; M21 is still zero.  Note that M21 is immediately
CMD"R"......            ; set to one so the run command will not be given
M21=1.....              ; again in the next cycle.
ENDIF
```

See Also:
Writing a PLC Program
Conditions (Program Computational Features)
Program Commands **IF**, **WHILE**, **OR**

AROT{constant}

Function: Absolute rotation/scaling of X, Y, and Z axes

Type: Motion program (PROG and ROT)

Syntax: **AROT{constant}**

where:

- **{constant}** is an integer representing the number of the first of nine consecutive Q-variables to be used in the rotation/scaling matrix

This command loads the currently selected (with **TSEL**) transformation matrix for the coordinate system with rotation/scaling values contained in the nine Q-variables starting with the specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) by multiplying the XYZ vector by this matrix.

The rotation and scaling is done relative to the base XYZ coordinate system, defined by the axis definition statements. The math performed is:

$[Xrot\ Yrot\ Zrot]^T = [Rot\ Matrix]\ [Xbase\ Ybase\ Zbase]^T$

This command does not cause any movement of axes; it simply renames the present positions.

> ***Note:***
>
> When using this command to scale the coordinate system, do not use the radius center specification for circle commands. The radius does not get scaled. Use the **I, J, K** vector specification instead.

Examples:

Create a 3x3 matrix to rotate the XY plane by 30 degrees about the origin

```
Q40=COS(30).Q41=SIN(30) Q42=0
Q43=-SIN(30)       Q44=COS(30) Q45=0
Q46=0.......       Q47=0               Q48=1
AROT 40 .....      ; Implement the change
```

Create a 3x3 matrix to scale the XYZ space by a factor of 3

```
Q50=3.......       Q51=0               Q52=0
Q53=0.......       Q54=3               Q55=0
Q56=0.......       Q57=0               Q58=3
AROT 50 .....      ; Implement the change
```

See Also:

Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **IDIS**, **IROT**, **TINIT**

B{data}

Function: B-Axis Move

Type: Motion program (PROG and ROT)

Syntax: **B{data}**

where:

- **{data}** is a floating-point constant or expression representing the position or distance in user units for the B-axis

This command causes a move of the B-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
B20
B(Q15)
A25 B10 Z35
B(20*COS(Q5))
CALL20 B-7.701
```

See Also:

Program commands **{axis}{data}**, **B**, **C**, **U**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**
Program commands **{axis}{data}**, **A**, **C**, **U**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **READ**

BLOCKSTART

Function: Mark Start of Stepping Block

Type: Motion program (PROG and ROT)

Syntax: **BLOCKSTART**
BSTART

This statement allows for multiple moves to be done on a single 'step' command. Execution on a 'step' command will proceed until the next **BLOCKSTOP** statement in the program (without **BLOCKSTART**, only a single servo command is executed on a 'step' command). Also, if Isx92=1 (move blending disabled), all moves between **BLOCKSTART** and **BLOCKSTOP** will be blended together. This does not affect how a program is executed from a 'run' command if Isx92=0.

This structure is particularly useful for executing a single sequence of PVT mode moves, because the individual segments do not end at zero velocity, making normal stepping very difficult.

Examples:

For the program segment:

```
BLOCKSTART
INC
X10:100
X20:100
X20:100
X10:0
BLOCKSTOP
```

All four move segments will be executed on a single **S** command.

See Also:

I-variable Isx92

On-line commands **<CONTROL-S>**, **R**, **S**.

Program commands **BLOCKSTOP**, **STOP**.

BLOCKSTOP

Function: Mark End of Stepping Block

Type: Motion program (PROG and ROT)

Syntax: **BLOCKSTOP**
BSTOP

This statement marks the end of the block of statements, begun with a **BLOCKSTART**, to be done on a single 'step' command, or to be blended together even if Isx92=1 (move blending disabled). This does not affect how a program is executed from a 'run' command if Isx92=1.

Examples:

See example under **BLOCKSTART**, above.

See Also:

I-variable Isx92

On-line commands **<CONTROL-S>**, **R**, **S**.

Program commands **BLOCKSTART**, **STOP**.

C{data}

Function: C-Axis Move

Type: Motion program (PROG and ROT)

Syntax: **C{data}**

where:

- **{data}** is a floating-point constant or expression representing the position or distance in user units for the C-axis

This command causes a move of the C-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
C30
C(-P71)
B25 C10 Z35
C(20*ATAN(Q80))
CALL5000 C0.210
```

See Also:

Program commands **{axis}{data}**, **A**, **B**, **U**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**

CALL

Function: Jump to Subprogram With Return

Type: Motion program (PROG and ROT)

Syntax: **CALL{data} [{letter}{data}...]**

where:

- the first **{data}** is a floating-point constant or expression from 1.00000 to 32767.99999, with the integer part representing the motion program number to be called, and the fractional part representing the line label (**N** or **O**) within the program to be called (the line label number is equal to the fractional part multiplied by 100,000; every motion program has an implicit **N0** at the top);
- **{letter}** is any letter of the English alphabet, except N or O, representing the variable into which the value following it will be placed (Q101 to Q126 for A to Z respectively);
- following **{data}** is a floating-point constant or expression representing the value to be put into the variable

This command allows the program to execute a subprogram and then return execution to the next line in the program. A subprogram is entered into Turbo PMAC the same as a program, and is labeled as PROGn (so one program can call another as a subprogram). The number n of the PROG heading is the one to which the value after **CALL** refers: **CALL7** would execute **PROG7** and return. Commanding execution of a non-existent subprogram will cause program execution to stop in an error condition.

The value immediately following **CALL** can take fractional values. If there is no fractional component, the called program starts at the beginning. If there is a fractional component, the called program is entered at a line label specified by the fractional component (if this label does not exist, Turbo PMAC will generate an error and stop execution). Turbo PMAC works with five fractional digits to specify the line label; if you use fewer, it automatically fills out the rest with zeros. For instance, **CALL 35.1** is interpreted as **CALL 35.10000**, which causes a jump to label **N10000** of program 35. **CALL 47.123** causes a jump to label **N12300** of program 47.

If letters and data (e.g. **X1000**) follow the **CALL{data}**, these can be arguments to be passed to the subprogram. If arguments are to be passed, the first line executed in the subroutine should be a **READ** statement. This statement will take the values associated with the specified letters and place them in the appropriate Q-variable. For instance, the data following **A** is placed in variable Q101 for the coordinate system executing the program; that following **B** is placed in Q102; and so on, to the data following **Z** being placed in Q126. The subprogram can then use these variables. If the subprogram calls another subprogram with arguments, the same Q-variables are used. Refer to **READ** for more details.

If there is no **READ** statement in the subroutine, or if not all the letter values in the **CALL** line are "read" (the **READ** statement stops as soon as it sees a letter in the calling line that is not in its list of letters to read), the remaining letter commands are executed upon return from the subroutine according to their normal function. For example, **G01 X10 Y10** is equivalent to a **CALL 1000.01 X10 Y10**. To implement the normal function for **G01** (linear move mode), there would be the following subroutine in PROG 1000:

```
N1000 LINEAR RETURN
```

Upon the return, **X10 Y10** would be executed as a move according to the move mode in force, which is **LINEAR**.

If the specified program and line label do not exist, the **CALL** command is ignored, and the program continues as if it were not there.

Examples:

```
CALL500 ..... ; to Prog 500 at the top (N0)
CALL500.1 ..... ; to Prog 500 label N10000
CALL500.12 .... ; to Prog 500 label N12000
CALL500.123 .. ; to Prog 500 label N12300
CALL500.1234 ; to Prog 500 label N12340
CALL500.12345 ; to Prog 500 label N12345
CALL700 D10 E20 ; to Prog 700 passing D and E
```

See Also:

On-line command **B{constant}**

Program commands **GOTO**, **GOSUB**, **READ**, **RETURN**, **G{data}**, **M{data}**, **T{data}**, **D{data}**, **N{constant}**, **O{constant}**, **PRELUDE**

CC0

Function: Turn Off Cutter Radius Compensation

Type: Motion program (PROG and ROT)

Syntax: **CC0**

This turns off the cutter radius compensation mode, reducing it gradually through the next move. This is equivalent to the **G40** command of the machine-tool standard RS-274 language.

Examples:

```
CCR0.5..... ; 1/2 unit cutter radius
CC1..... ; Cutter compensation on to the left
X10 Y10 ..... ; Compensation introduced during this move
X10 Y20
X20 Y20
X20 Y10
X10 Y10
CC0..... ; Cutter compensation off
X0 Y0..... ; Compensation eliminated during this move
```

```
OPEN PROG 1000      ; G-Code Subprogram
  ...
N40000 CC0 RETURN   ; To implement G40 directly in Turbo PMAC
```

See Also:
Cutter (Tool) Radius Compensation
Program commands **CC1**, **CC2**, **CCR{data}**.

CC1

Function: Turn On 2D Cutter Radius Compensation Left

Type: Motion program (PROG and ROT)

Syntax: **CC1**

This turns on the cutter radius compensation mode, introducing the compensation gradually through the next move. The cutter is offset to the left of the programmed tool path, looking in the direction of cutter movement. The plane of the compensation is determined by the **NORMAL** command. This is equivalent to the **G41** command of the machine-tool standard RS-274 language.

Note:

The coordinate system must be in move segmentation mode (Isx13>0) in order to perform cutter radius compensation. If Isx13=0 (no move segmentation), the moves will be executed without compensation.

Examples:

```
CCR0.25 .....   ; 1/4 unit cutter radius
CC1 .....       ; Cutter compensation on to the left
X10 Y10 .....   ; Compensation introduced during this move
X10 Y20
X20 Y20
X20 Y10
X10 Y10
CC0 .....       ; Cutter compensation off
X0 Y0 .....     ; Compensation eliminated during this move

OPEN PROG 1000      ; G-Code Subprogram
  ...
N41000 CC1 RETURN   ; To implement G41 directly in Turbo PMAC
```

See Also:
Cutter (Tool) Radius Compensation
Program commands **CC2**, **CC0**, **CCR{data}**, **NORMAL**

CC2

Function: Turn On 2D Cutter Radius Compensation Right

Type: Motion program (PROG and ROT)

Syntax: **CC2**

This turns on the cutter radius compensation mode, introducing the compensation gradually through the next move. The cutter is offset to the right of the programmed tool path, looking in the direction of cutter movement. The plane of the compensation is determined by the **NORMAL** command. This is equivalent to the **G42** command of the machine-tool standard RS-274 language.

Note:

The coordinate system must be in move segmentation mode (Isx13>0) in order to perform cutter radius compensation. If Isx13=0 (no move segmentation), the moves will be executed without compensation.

Examples:

```
CCR1.5.....          ; 1-1/2 unit cutter radius
CC2.....             ; Cutter compensation on to the right
X10 Y10.....         ; Compensation introduced during this move
X10 Y20
X20 Y20
X20 Y10
X10 Y10
CC0.....             ; Cutter compensation off
X0 Y0.....           ; Compensation eliminated during this move

OPEN PROG 1000       ; G-Code Subprogram
  ...
N42000 CC2 RETURN    ; To implement G42 directly in Turbo PMAC
```

See Also:

Tool Radius Compensation

Program commands **CC1**, **CC0**, **CCR**, **NORMAL**

CC3

Function: Turn On 3D Cutter Radius Compensation

Type: Motion program (PROG and ROT)

Syntax: **CC3**

This statement turns on three-dimensional cutter-radius compensation mode, introducing the compensation gradually through the next move. Because the offset vectors are explicitly specified in 3D compensation, there is no need to declare a left or right mode, as there is in 2D compensation.

When the **CC3** statement is executed, both the tool-orientation and the surface-normal vector are automatically set to the null vector. The tool-orientation vector can subsequently be modified by **TX**, **TY**, and **TZ** values. A non-zero surface-normal vector must be declared with **NX**, **NY**, and **NZ** values before any compensation will actually occur.

3D cutter-radius compensation remains active until the **CC0** statement is executed.

See Also:

Three-Dimensional Compensation, Program commands **CC0**, **CCR{data}**, **NX{data}**, **NY{data}**, **NZ{data}**, **TR{data}**, **TX{data}**, **TY{data}**, **TZ{data}**

CCR{data}

Function: Set Cutter Compensation Radius (2D & 3D)

Type: Motion program (PROG and ROT)

Syntax: **CCR{data}**

where:

- **{data}** is a floating-point constant or expression representing the magnitude of the cutter's end radius

This statement sets the magnitude of the radius of the cutter for two-dimensional and three-dimensional cutter-radius compensation, expressed in the user units of the X, Y, and Z axes. This function is often part of the **D** tool data used in the machine-tool standard RS-274 (G) code.

In 2D compensation, an offset of this magnitude is made in the plane defined by the **NORMAL** vector, perpendicular to the programmed path direction.

In 3D compensation, there are two radius values. The first is the radius of the cutter's end, defined by this **CCR** command. The second is the radius of the cutter's shaft, defined by the **TR** command. In operation, an offset is first made in the direction of the surface-normal vector of a magnitude defined by the **CCR** cutter-end radius command. Then an offset is made of a magnitude equal to the **TR** tool radius minus the **CCR** cutter-end radius. This offset is made in the plane of the surface-normal vector and the tool-orientation vector, perpendicular to the tool-orientation vector.

The **CCR** cutter-end radius should be zero for a flat-end cutter; it should be equal to the **TR** tool-shaft radius for a ball-end cutter; it should be somewhere in between for a partially rounded end. If the **CCR** value is set greater than the **TR** value, the **TR** value is effectively increased to be equal to the **CCR** value

The value declared here affects the compensation of all subsequent moves until another value is declared.

See Also:
Cutter-Radius Compensation, Three-Dimensional Compensation
Program commands **CC1**, **CC2**, **CC3**, **CC0**, **CCR{data}**, **NORMAL**, **NX{data}**, **NY{data}**, **NZ{data}**, **TR{data}**, **TX{data}**, **TY{data}**, **TZ{data}**

CIRCLE1

Function: Set Blended Clockwise Circular Move Mode

Type: Motion program (PROG and ROT)

Syntax: **CIRCLE1**
CIR1

This command puts the program into clockwise circular move mode. The plane for the circular interpolation is defined by the most recent **NORMAL** command, which has also defined the sense of clockwise and counterclockwise in the plane.

The program is taken out of this circular move mode by another move mode command: the other **CIRCLE** mode, **LINEAR**, **PVT**, **RAPID** etc. Any circular move command must have either an **R** or an **IJK** vector specification; otherwise it will be performed as a linear move even when in **CIRCLE** mode.

> ***Note:***
>
> The coordinate system must be in move segmentation mode (Isx13>0) in order to perform circular interpolation. If Isx13=0 (no move segmentation), the moves will be linearly interpolated.

Examples:

```
LINEAR ..... ; Linear interpolation mode
X10Y10 F2 ..... ; Linear move
CIRCLE1 ..... ; Clockwise circular interpolation mode
X20 Y20 I10..; Arc of 10-unit radius
X25 Y15 J-5..; Arc of 5-unit radius
LINEAR..... ; Go back to linear mode
X25 Y5..... ; Linear move
```

See Also:
Circular Moves (Writing a Motion Program)
I-variable Isx13
Program commands **NORMAL**, **CIRCLE2**, **LINEAR**, **PVT**, **RAPID**, **SPLINE1**,
{axis}{data}{vector}{data}

CIRCLE2

Function: Set Blended Counterclockwise Circular Move Mode

Type: Motion program (PROG and ROT)

Syntax: **CIRCLE2**
CIR2

The **CIRCLE2** command puts the program into counterclockwise circular move mode. The plane for the circular interpolation is defined by the most recent **NORMAL** command, which has also defined the sense of clockwise and counterclockwise in the plane.

The program is taken out of this circular move mode by another move mode command: the other **CIRCLE** mode, **LINEAR**, **PVT**, **RAPID** etc. Any circular move command must have either an **R** or an **IJK** vector specification; otherwise it will be performed as a linear move even when in **CIRCLE** mode.

Note:

The coordinate system must be in move segmentation mode (Isx13>0) in order to perform circular interpolation. If Isx13=0 (no move segmentation), the moves will be linearly interpolated.

Examples:

```
LINEAR ..... ; Linear interpolation mode
X10Y0 F2..... ; Linear move
CIRCLE2 ..... ; Counterclockwise circular interpolation mode
X20 Y10 J10..; Arc of 10-unit radius
X15 Y15 I-5..; Arc of 5-unit radius
CIRCLE1 ..... ; Clockwise circle mode
X5 Y25 J10.... ; Arc move of 10-unit radius
```

See Also:
Circular Moves (Writing a Motion Program)
I-variable Isx13
Program commands **NORMAL**, **CIRCLE1**, **LINEAR**, **PVT**, **RAPID**, **SPLINE1**,
{axis}{data}{vector}{data}

COMMANDx"{command}"

Function: Command issuance from internal program

Type: Motion program (PROG and ROT); PLC program

Syntax:
```
COMMAND "{command}"
CMD "{command}"
COMMANDS "{command}"
CMDS "{command}"
COMMANDP "{command}"
CMDP "{command}"
COMMANDR "{command}"
CMDR "{command}"
COMMANDA "{command}"
CMDA"{command}"
```

This statement causes the program to issue a command to Turbo PMAC as if it came from the host (except for addressing modes and response direction).

If there is a motor- or coordinate-system-specifier (**#n** or **&n**) within the quoted string, a motor- or coordinate-system-specific command will be directed to that motor or coordinate system. If there is no specifier, a motor- or coordinate-system-specific command will be directed to the first motor or coordinate system. Any specifier within a **COMMAND** statement is not modal; it does not affect the host addressing specifications or the modal addressing of any program, including its own.

Any data or error response for the command given inside the quote marks is directed to the port specified by the letter at the end of **COMMAND**:

- **COMMAND** provides no response to any port
- **COMMANDS** provides a response to the main serial port
- **COMMANDP** provides a response to the parallel bus port
- **COMMANDR** provides a response to the DPRAM ASCII response buffer
- **COMMANDA** provides a response to the Option 9T auxiliary serial port

If I62=0, Turbo PMAC automatically issues a carriage-return (**<CR>**) character at the end of any data response to the command. If I62=1, Turbo PMAC does not issue a **<CR>** character at the end of the data response; a **SEND^M** must be used to issue an **<CR>** in this case.

Each PLC program has its own addressing mode for both motors and coordinate systems, independent of each other and independent of the host addressing modes. These are controlled by the PLC program **ADDRESS** command. This modal addressing affects commands issued from within a PLC program that do not have motor or coordinate-system specifiers. At power-up/reset, all PLC programs are addressing Motor 1 and C.S.1.

There is no modal **ADDRESS** command in motion programs. Any motor-specific or coordinate-system-specific command issued from within a motion program without a specifier is automatically addressed to Motor 1 or C.S.1, respectively.

Commands issued from within a program are placed in the command queue, to be parsed and acted upon at the appropriate time by Turbo PMAC's command interpreter, which operates in background, between other background tasks. If issued from a motion program, the command will not be interpreted before the next move or dwell command in the motion program is calculated. If issued from a PLC program, the command will not be interpreted before the end of the current scan of the PLC. This delay can make the action appear to execute out of sequence.

Because of the queuing of commands and the fact that command interpretation is a lower priority than command issuing, it is possible to overflow the queue. If there is no room for a new command, program execution is temporarily halted until the new command can be placed on the queue.

Also, commands that generate a response to the host (including errors if I6 is not equal to 2) potentially can fill up the response queue if there is no host or the host is not prepared to read the responses. This will temporarily halt program execution until the response queue is emptied. In standalone applications, it is a good idea to set I1 to 1, disabling the serial handshake, so any responses can be sent out the serial port (the default response port) at any time, even if there is no host to receive it.

In a PLC program, it is a good idea to have at least one of the conditions that caused the command issuance to occur set false immediately. This will prevent the same command from being issued again on succeeding scans of the PLC, overflowing the command and/or response queues. Typically in a motion program, the time between moves prevents this overflow unless there are a lot of commands and the moves take a very short time.

Turbo PMAC will not issue an acknowledging character (**<ACK>** or **<LF>**) to a valid command issued from an internal program. It will issue a **<BELL>** character for an invalid command issued from a program unless I6 is set to 2. It is a good idea to have I6 not set to 2 in early development so you will know when Turbo PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hang-up from a full response queue, or from disturbing the normal host communications protocol.

Many otherwise valid commands will be rejected when issued from a motion program. For instance, you cannot jog any motor in the coordinate system executing the program, because all these motors are considered to be running in the program, even if the program is not requesting a move of the motors at that time.

When issuing commands from a program, be sure to include all the necessary syntax (motor and/or coordinate system specifiers) in the command statement or use the **ADDRESS** command. For example, use **CMD"#4HM"** and **CMD"&1A"** instead of **CMD"HM"** and **CMD"A"**. Otherwise, motor and coordinate system commands will be sent to the most recently addressed motor and coordinate system which may not always be as you intended.

Examples:

```
COMMAND"#1J+"
CMD"#4HM"
CMD"&1B5R"
CMDP"P1"
47.5
ADDRESS#3
COMMAND"J-"
IF(M40=1 AND M41=1)
      CMD"&4R"
      M41=0
ENDIF
```

See Also:

Addressing Modes, On-Line Commands (Talking To Turbo PMAC)
I-variables I1, I3, I6.
Program commands **ADDRESS**, **COMMAND^{letter}**
Writing A PLC Program

COMMANDx^{letter}

Function:	Control-Character Command Issuance from Internal Program
Type:	Motion program (PROG or ROT), PLC program
Syntax:	**COMMAND^{letter}** **CMD^{letter}** **COMMANDS^{letter}** **CMDS^{letter}** **COMMANDP^{letter}** **CMDP^{letter}** **COMMANDR^{letter}** **CMDR^{letter}** **COMMANDA^{letter}** **CMDA^{letter}**

where:

- **{letter}** is a letter character from A to Z (upper or lowercase) representing the corresponding control character

This statement causes the motion program to issue a control-character command as if it came from the host, except for the direction of the response. All control-character commands are global, so there are no addressing concerns.

Any data or error response for the control-character command is directed to the port specified by the letter at the end of **COMMAND**:

- **COMMAND** provides no response to any port
- **COMMANDS** provides a response to the main serial port
- **COMMANDP** provides a response to the parallel bus port
- **COMMANDR** provides a response to the DPRAM ASCII response buffer
- **COMMANDA** provides a response to the Option 9T auxiliary serial port

> ***Note:***
>
> Do not put the up-arrow character and the letter in quotes (do not use **COMMANDx"^A"**) or Turbo PMAC will attempt to issue a command with the two non-control characters, **^** and **A** in this example, instead of the control character.

Commands issued from within a program are placed in the command queue, to be parsed and acted upon at the appropriate time by Turbo PMAC's command interpreter, which operates in background, between other background tasks. If issued from a motion program, the command will not be interpreted before the next move or dwell command in the motion program is calculated. If issued from a PLC program, the command will not be interpreted before the end of the current scan of the PLC. This delay can make the action appear to execute out of sequence.

Because of the queuing of commands and the fact that command interpretation is a lower priority than command issuing, it is possible to overflow the queue. If there is no room for a new command, program execution is temporarily halted until the new command can be placed on the queue.

Also, commands that generate a response to the host (including errors if I6 is not equal to 2) potentially can fill up the response queue if there is no host or the host is not prepared to read the responses. This will temporarily halt program execution until the response queue is emptied. In standalone applications, it is a good idea to set I1 to 1, disabling the serial handshake, so any responses can be sent out the serial port (the default response port) at any time, even if there is no host to receive it.

In a PLC program, it is a good idea to have at least one of the conditions that caused the command issuance to occur set false immediately. This will prevent the same command from being issued again on succeeding scans of the PLC, overflowing the command and/or response queues. Typically in a motion program, the time between moves prevents this overflow unless there are a lot of commands and the moves take a very short time.

Turbo PMAC will not issue an acknowledging character (**<ACK>** or **<LF>**) to a valid command issued from a program. It will issue a **<BELL>** character for an invalid command issued from a program unless I6 is set to 2. It is a good idea to have I6 not set to 2 in early development so you will know when Turbo PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hang-up from a full response queue, or from disturbing the normal host communications protocol

Examples:
CMD^D would disable all PLC programs (equivalent to issuing a **<CONTROL-D>** from the host).
CMD^K would kill (disable) all motors on Turbo PMAC
CMD^A would stop all programs and moves on Turbo PMAC, also closing any loops that were open.
CMDS^P would cause Turbo PMAC to report the positions for 8 motors to the main serial port

See Also:
I-variables I1, I6
On-line commands **<CONTROL-A>** to **<CONTROL-Z>**
Program command **COMMANDx"{command}"**

D{data}

Function: Tool Data (D-Code)

Type: Motion program

Syntax: **D{data}**

where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to

Turbo PMAC interprets this statement as a **CALL 10n3.({data'}*1000)** command, where n is the hundreds'digit of **{data},** and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n3, and the specified line label. (Programs 10n3 are usually used to implement the tool data operations as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

This structure permits the implementation of customizable D-Code routines for machine-tool style applications by the writing of subroutines in motion programs 10n3. Arguments can be passed to these subroutines by following the D-Code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have D-codes only in the range 0-99, which permits the use of PROG 1003 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example:
D01 jumps to **N1000** of PROG 1003
D12 jumps to **N12000** of PROG 1003
D115 jumps to **N15000** of PROG 1013

See Also:
Program commands **CALL{data}**, **G{data}**, **M{data}**, **T{data}**, **RETURN**

DELAY{data}

Function: Delay for Specified Time

Type: Motion program

Syntax: **DELAY{data}**
DLY{data}

where:

- **{data}** is a floating-point constant or expression, specifying the delay time in milliseconds

This command causes Turbo PMAC to keep the command positions of all axes in the coordinate system constant (no movement) for the time specified in **{data}**.

There are three differences between **DELAY** and **DWELL**. First, if **DELAY** comes after a blended move, the **TA** deceleration time from the move occurs within the **DELAY** time, not before it.

Second, the actual time for **DELAY** does varies with a changing time base (current % value, from whatever source), whereas **DWELL** always uses the fixed time base (%100). Third, Turbo PMAC precomputes upcoming moves (and the lines preceding them) during a **DELAY**, but it does not do so during a **DWELL**.

A **DELAY** command is equivalent to a zero-distance move of the time specified in milliseconds. As for a move, if the specified **DELAY** time is less than the acceleration time currently in force (**TA** or 2***TS**), the delay will be for the acceleration time, not the specified **DELAY** time.

Examples:
```
DELAY750
DELAY(Q1+100)
```

See Also:
Time-Base Control (Synchronizing Turbo PMAC to External Events)
I-variables I10, Isx87, Isx88
On-line command **%{constant}**
Program commands **DWELL**, **TA**, **TS**

DISABLE PLC {constant}[,{constant}...]

Function: Disable PLC Program(s)

Type: Motion program (PROG or ROT), PLC program

Syntax:
```
DISABLE PLC {constant}[,{constant}...]
DISABLE PLC {constant}[..{constant}]
DIS PLC {constant}[,{constant}...]
DIS PLC {constant}[..{constant}]
```

where:

- each **{constant}** is an integer from 0 to 31 representing the PLC number

This command causes Turbo PMAC to disable (stop executing) the specified uncompiled PLC program or programs. Execution can subsequently be resumed at the top of the program with the **ENABLE PLC** command. If it is desired to restart execution at the stopped point, execution should be stopped with the **PAUSE PLC** command, and restarted with the **RESUME PLC** command

Execution of a PLC program can only be disabled at the end of a scan, which is either the end of the program, or after executing an **ENDWHILE** statement in the program. (A PLC program can be paused in the middle of a scan, however.)

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to PMAC, it will be executed immediately as an on-line command.

Examples:
```
DISABLE PLC 1
DISABLE PLC 4,5
DISABLE PLC 7..20
DIS PLC 3,8,11
DIS PLC 0..31
```

See Also:
I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**, **DISABLE PLCC**, **PAUSE PLC**, **RESUME PLC**, **<CONTROL-D>**
Program command **ENABLE PLC**, **DISABLE PLCC**, **ENABLE PLCC**, **PAUSE PLC**, **RESUME PLC**

DISABLE PLCC {constant}[,{constant}...]

Function: Disable Compiled PLC Program(s)

Type: Motion program (PROG or ROT), PLC program (uncompiled or compiled), except for PLC 0 and PLCC 0

Syntax:
```
DISABLE PLCC {constant}[,{constant}...]
DISABLE PLCC {constant}[..{constant}]
DIS PLCC {constant}[,{constant}...]
DIS PLCC {constant}[..{constant}]
```

where:

- each **{**constant**}** is an integer from 0 to 31 representing the compiled PLC number

This command disables the operation of the specified compiled PLC (PLCC) program[s]. The programs are specified by number, and can be used singly, in a list separated by commas, or in a continuous range.

Note:

This command should not be used in a foreground PLC -- either uncompiled PLC 0 or compiled PLCC 0 -- as its operation cannot be guaranteed in these programs.

I-variable I5 is a separate master control of PLC program operation. Think of the two bits of I5 as two master circuit breakers for a house, and the individual PLC and PLCC enable/disable bits as separate light switches within the house. Both the master breaker and the switch must be on for the PLC to operate. The breakers and the switches can be operated independently without affecting the setting of the others.

Examples:
```
DISABLE PLCC 1
DISABLE PLCC 4,5
DISABLE PLCC 7..20
DIS PLCC 3,8,11
DIS PLC 0..31
```

See Also:
I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**, **DISABLE PLCC**, **<CONTROL-D>**
Program command **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**

DISPLAY [{constant}] "{message}"

Function: Display Text to Display Port

Type: Motion program (PROG and ROT), PLC program

Syntax:
```
DISPLAY [{constant}] "{message}"
DISP [{constant}] "{message}"
```

where:

- **{constant}** is an integer value between 0 and 79 specifying the starting character number on the display; if no value is specified, 0 is used
- **{message}** is the ASCII text string to be displayed

This command causes Turbo PMAC to send the string contained in **{message}** to the display port for the liquid-crystal or vacuum-fluorescent display (Accessory 12 or equivalent).

The optional constant value specifies the starting point for the string on the display; it has a range of 0 to 79, where 0 is upper left, 39 is upper right, 40 is lower left, and 79 is lower right.

Examples:
```
DISPLAY 10"Hello World"
DISP "VALUE OF P1 IS"
DISP 15, 8.3, P1
```

See Also:
Display Port (Connecting Turbo PMAC to the Machine);
Accessory 12 (Basic Specifications)
Program commands **DISPLAY {variable}**, **SENDx"{message}"**

DISPLAY ... {variable}

Function: Formatted Display of Variable Value

Type: Motion program (PROG and ROT), PLC program

Syntax:
```
DISPLAY {constant}, {constant}.{constant}, {variable}
DISP {constant}, {constant}.{constant}, {variable}
```

where:

- the first **{constant}** is an integer from 0 to 79 representing the starting location (character number) on the display;
- the second **{constant}** is an integer from 2 to 16 representing the total number of characters to be used to display the value (integer digits, decimal point, and fractional digits);
- the third **{constant}** is an integer from 0 to 9 (and at least two less than the second **{constant}**) representing the number of fractional digits to be displayed;
- **{variable}** is the name of the variable to be displayed.

This command causes Turbo PMAC to send a formatted string containing the value of the specified variable to the display port. The value of any I, P, Q, or M variable may be displayed with this command.

The first constant value specifies the starting point for the string on the display; it has a range of 0 to 79, where 0 is upper left, 39 is upper right, 40 is lower left, and 79 is lower right. The second constant specifies the number of characters to be used in displaying the value; it has a range of 2 to 16. The third constant specifies the number of places to the right of the decimal point; it has a range of 0 to 9, and must be at least 2 less than the number of characters. The last thing specified in the statement is the name of the variable -- I, P, Q, or M.

Examples:
```
DISPLAY 0, 8.0, P50
DISPLAY 24, 2.0, M1
DISPLAY 40, 12.4, Q100
```

See Also:
Display Port (Connecting Turbo PMAC to the Machine);
Accessory 12 (Basic Specifications)
Program commands **DISPLAY"{message}"**, **COMMAND"{command}"**

DWELL

Function: Dwell for Specified Time

Type: Motion program (PROG and ROT)

Syntax: **DWELL{data}**
DWE{data}

where:

- **{data}** is a non-negative floating point constant or expression representing the dwell time in milliseconds

This command causes the card to keep the commanded positions of all axes in the coordinate system constant for the time specified in **{data}**. The maximum dwell time is 8,388,607 msec (about 2 hours and 20 minutes). If a longer time is specified in the command, the actual dwell time will be limited to this amount.

There are three differences between **DWELL** and the similar **DELAY** command. First, if the previous servo command was a blended move, there will be a **TA** time deceleration to a stop before the dwell time starts. Second, **DWELL** is not sensitive to a varying time base -- it always operates in 'real time' (as defined by I10). Third, Turbo PMAC does not pre-compute upcoming moves (and the program lines before them during the **DWELL**; it waits until after it is done to start further calculations, which it performs in the time specified by I11 or Ixx12.

Use of any **DWELL** command, even a **DWELL0**, while in external time base will cause a loss of synchronicity with the master signal.

Examples:
```
DWELL250
DWELL(P1+P2)
DWE0
```

See Also:
Dwell and Delay (Writing a Motion Program)
I-variables I10, I11, I12.
Program command **DELAY**

ELSE

Function: Start False Condition Branch

Type: Motion program (PROG only), PLC program

Syntax: **ELSE** (Motion or PLC Program)
ELSE {action} (Motion Program only)

where:

- **{action}** is a program command

This statement must be matched with an **IF** statement (**ELSE** requires a preceding **IF**, but **IF** does not require a following **ELSE**). It follows the statements executed upon a true **IF** condition. It is followed by the statements to be executed upon a false **IF** condition.

Note:

With nested **IF** branches, you must be careful to match the **ELSE** statements to the proper **IF** statement. In a motion program, it is possible to have a single-line **IF** statement (**IF({condition}) {action}**). An **ELSE** statement on the next program line is automatically matched to this **IF** statement, even if you wanted to match a previous IF statement. You must put a non-**ELSE** statement in between to make the next **ELSE** statement match a previous **IF** statement.

ELSE lines can take two forms (only the first of which is valid in a PLC program):

With no statement following on that line, all subsequent statements down to the next **ENDIF** statement will be executed provided that the preceding IF condition is false.

```
ELSE
     {statement}
     [{statement}
     ...]
ENDIF
```

With a statement or statements following on that line, the single statement will be executed provided that the preceding **IF** condition is false. No **ENDIF** statement should be used in this case

```
ELSE {statement} [{statement}...]
```

Note:

(This single-line **ELSE** branch form is valid only in motion programs. If you try this in a PLC program, Turbo PMAC will put the statement(s) on the next program line and expect an **ENDIF** to close the branch. Your logic will not be as you expect it.)

Examples:

This first example has multi-line true and false branches. It could be used in either a motion program or a PLC program.

```
IF (M11=1)
      P1=17
      P2=13
ELSE
      P1=13
      P2=17
ENDIF
```

This second example has a multi-line true branch, and a single-line false branch. This structure could only be used in a motion program.

```
IF (M11=0)
      X(P1)
      DWELL 1000
ELSE DWELL 500
```

This example has a single-line true branch, and a multi-line false branch. This structure could only be used in a motion program.

```
IF (SIN(P1)>0.5) Y(1000*SIN(P1))
ELSE
      P1=P1+5
      Y(1100*SIN(P1))
ENDIF
```

This example has single-line true and false branches. This structure could only be used in a motion program.

```
IF (P1 !< 5) X10
ELSE X-10
```

See Also:
Program commands **IF**, **ENDIF**.

ENABLE PLC

Function: Enable PLC Buffer(s)

Type: Motion program (PROG and ROT), PLC program

Syntax:
```
ENABLE PLC {constant}[,{constant}...]
ENABLE PLC {constant}[..{constant}]
ENA PLC {constant}[,{constant}...]
ENA PLC {constant}[..{constant}]
```

where:

- each **{constant}** is an integer from 0 to 31 representing the PLC number

This command causes Turbo PMAC to enable (start executing) the specified uncompiled PLC program or programs at the top of the program. Execution of the PLC program may have been stopped with the **DISABLE PLC**, **PAUSE PLC**, or **OPEN PLC** command.

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to PMAC, it will be executed immediately as an on-line command.

Examples:
```
ENABLE PLC 0
ENABLE PLC 1,2,5
ENABLE PLC 1..16
ENA PLC 7
```

See Also:
I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **<CONTROL-D>**
Program command **DISABLE PLC**, **PAUSE PLC**, **RESUME PLC**

ENABLE PLCC

Function: Enable Compiled PLC Program(s)

Type: Motion program (PROG and ROT), PLC program (uncompiled and compiled)

Syntax:
```
ENABLE PLCC {constant}[,{constant}...]
ENABLE PLCC {constant}[..{constant}]
ENA PLCC {constant}[,{constant}...]
ENA PLCC {constant}[..{constant}]
```

where:

- each **{constant}** is an integer from 0 to 31 representing the compiled PLC number

This command enables the operation of the specified compiled PLC (PLCC) buffer[s], provided I5 is set properly to allow their operation. The programs are specified by number, and can be used singly, in a list separated by commas, or in a continuous range.

I-variable I5 is a separate master control of PLC program operation. Think of the two bits of I5 as two master circuit breakers for a house, and the individual PLC and PLCC enable/disable bits as separate light switches within the house. Both the master breaker and the switch must be on for the PLC to operate. The breakers and the switches can be operated independently without affecting the setting of the others.

Examples:

```
ENABLE PLCC 0
ENABLE PLCC 1,2,5
ENABLE PLCC 1..16
ENA PLCC 7
```

See Also:

I-variable I5

On-line commands **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**, **DISABLE PLCC**, **<CONTROL-D>**

Program command **ENABLE PLC**, **DISABLE PLC**, **DISABLE PLCC**

ENDIF

Function: Mark End of Conditional Block

Type: Motion program (PROG only), PLC program

Syntax: **ENDIF**
ENDI

This statement marks the end of a conditional block of statements begun by an **IF** statement. It can close out the "true" branch, following the **IF** statement, in which case there is no "false" branch, or it can close out the "false" branch, following the **ELSE** statement.

When nesting conditions, it is important to match this **ENDIF** with the proper **IF** or **ELSE** statement. In a PLC program, every **IF** or **IF/ELSE** pair must take an **ENDIF**, so the **ENDIF** always matches the most recent **IF** statement that does not already have a matching **ENDIF**. In a motion program an **IF** or **ELSE** statement with action on the same line does not require an **ENDIF**, so the **ENDIF** would be matched with a previous **IF** statement.

Examples:

```
IF (P1>0)
      X1000
ENDIF

IF (P5=7)
      X1000
ELSE
      X2000
ENDIF
```

See Also:

Logical Structures (Writing a Motion Program)

Conditional Statements (Writing a PLC Program)

Program commands **IF**, **ELSE**

ENDWHILE

Function: Mark End of Conditional Loop

Type: Motion program (PROG only), PLC program

Syntax: **ENDWHILE**
ENDW

This statement marks the end of a conditional loop of statements begun by a **WHILE** statement. **WHILE** loops can be nested, so an **ENDWHILE** statement matches the most recent **WHILE** statement not already matched by a previous **ENDWHILE** statement.

In a motion program a **WHILE** statement with an action on the same line does not require a matching **ENDWHILE**.

In the execution of a PLC program, when an **ENDWHILE** statement is encountered, that scan of the PLC is ended, and Turbo PMAC goes onto other tasks (communications, other PLCs). The next scan of this PLC will start at the matching **WHILE** statement.

In the execution of a motion program, if Turbo PMAC finds two jumps backward (toward the top) in the program while looking for the next move command, Turbo PMAC will pause execution of the program and not try to blend the moves together. It will go on to other tasks and resume execution of the motion program on a later scan. Two statements can cause such a jump back: **ENDWHILE** and **GOTO** (**RETURN** does not count).

The pertinent result is that Turbo PMAC will not blend moves when it hits two **ENDWHILE** statements (or the same **ENDWHILE** twice) between execution of move commands.

Examples:

```
WHILE (Q10<10)
      Q10=Q10+1
ENDWHILE
```

See Also:
Program commands **WHILE**, **ENDIF**

F{data}

Function: Set Move Feedrate (Velocity)

Type: Motion program (PROG and ROT)

Syntax: **F{data}**

where:

- **{data}** is a positive floating-point constant or expression representing the vector velocity in user length units per user time units

This statement sets the commanded velocity for upcoming **LINEAR** and **CIRCLE** mode blended moves. It will be ignored in other types of moves (**SPLINE**, **PVT**, and **RAPID**). It overrides any previous **TM** or **F** statement, and is overridden by any following **TM** or **F** statement.

The units of velocity specified in an **F** command are scaled position units (as set by the axis definition statements) per time unit (defined by the Isx90 "Feedrate Time Unit" I-variable for the coordinate system).

The velocity specified here is the vector velocity of all of the vector-feedrate axes of the coordinate system. That is, the move time is calculated as the vector distance of the feedrate axes (square root of the sum of the squares of the individual axes), divided by the feedrate value specified here. Any non-feedrate axes commanded to move on the same move-command line will move at the speed necessary to finish in this same amount of time.

Axes are designated as vector-feedrate axes with the **FRAX** command. If no **FRAX** command is used, the default feedrate axes are the X, Y, and Z axes. Any axis involved in circular interpolation is automatically a feedrate axis, regardless of whether it was specified in the latest **FRAX** command. In multi-axis systems, feedrate specification of moves is really only useful for systems with Cartesian geometries, for which these moves give a constant velocity in the plane or in 3D space, regardless of movement direction.

There are several cases in which Turbo PMAC will not use the feedrate commanded with the **F** statement:

- If the specified feedrate causes a move time of over 2^{23} msec (about 2 hours 20 minutes) to be calculated, the move will be executed in 2^{23} msec, at a *higher* speed than what was programmed.
- If the feedrate programmed with the **F** statement exceeds the Isx98 maximum feedrate parameter for the coordinate system, the Isx98 value will be used instead.
- If the feedrate programmed with the **F** statement causes any motor in the coordinate system to exceed its Ixx16 velocity limit (when active), all axes will be slowed so that no motor exceeds its limit.
- If the vector distance of a feedrate-specified move is so short that the computed move time (vector distance divided by feedrate) would be less than the acceleration time currently in force (**TA** or 2***TS** or the time set by the Ixx17 limit when active), the move will take the full acceleration time instead, and the axes will move more slowly than specified by the **F** command. If the acceleration time is 0, the minimum permitted move time is 0.5 msec.
- If vector-feedrate axes and non-feedrate axes are commanded together on the same program line, and the time for any non-feedrate axis, computed as the axis distance divided by the alternate feedrate parameter Isx86, is greater than the move time calculated for the vector-feedrate axes, then Turbo PMAC will use this longer time for the move, resulting in a lower vector feedrate.
- If only non-feedrate axes are commanded to move in a feedrate-specified move, Turbo PMAC will compute the move time as the longest distance commanded for any axis divided by the Isx86 alternate feedrate parameter. If Isx86 is set to zero, it will compute the move time as the longest distance divided by the programmed feedrate. Use of the Isx86 parameter for all moves can be forced by using the **NOFRAX** command, which makes all axes non-feedrate axes.
- If the % override value for the coordinate system is at other than %100 when the move is executed, the move will not execute at the specified speed. The actual speed varies in direct proportion to the % value.

Examples:

```
F100
F31.25
F(Q10)
F(SIN(P8*P9))
```

See Also:

I-variables Isx86, Isx87, Isx88, Isx89, Isx90, Isx98
On-line commands **#{constant}->{axis definition}**, **FRAX**
Program commands **FRAX**, **LINEAR**, **NOFRAX**, **CIRCLE**, **TM**, **TA**, **TS**

FRAX

Function: Specify Feedrate Axes

Type: Motion program (PROG and ROT)

Syntax: **FRAX [({axis}[,{axis}...])]**

where:

- **{axis}** is a character (X, Y, Z, A, B, C, U, V, W) specifying which axis is to be used in the vector feedrate calculations.

This command specifies which axes are to be involved in the vector-feedrate (velocity) calculations for upcoming feedrate-specified (**F**) moves. Turbo PMAC calculates the time for these moves as the vector distance (square root of the sum of the squares of the axis distances) of all the feedrate axes divided by the feedrate. Any non-feedrate axes commanded on the same line will complete in the same amount of time, moving at whatever speed is necessary to cover the distance in that time.

Vector feedrate has obvious geometrical meaning only in a Cartesian system, for which it results in constant tool speed regardless of direction, but it is possible to specify for non-Cartesian systems, and for more than three axes.

> ***Note:***
>
> In a feedrate-specified move, if the move time for any non-feedrate axis, computed as axis distance divided by Isx86, is greater than the move time for the feedrate axes, computed as the vector distance divided by the feedrate, Turbo PMAC will use the move time for the non-feedrate axis instead.

The **FRAX** command without arguments causes all axes in the coordinate system to be feedrate axes in subsequent move commands. The **FRAX** command with arguments causes the specified axes to be feedrate axes, and all axes not specified to be non-feedrate axes, in subsequent move commands.

If no motion program buffer is open when this command is sent to Turbo PMAC, it will be executed as an on-line coordinate system command.

Examples:

For a three-axis cartesian system scaled in millimeters:

```
FRAX(X,Y)
INC
X30 Y40 Z10 F100
```

Vector distance is SQRT($30^2 + 40^2$) = 50 mm. At a speed of 100 mm/sec, move time (unblended) is 0.5 sec. X-axis speed is 30/0.5 = 60 mm/sec; Y-axis speed is 40/0.5 = 80 mm/sec; Z-axis speed is 10/0.5 = 20 mm/sec.

```
Z20
```

Vector distance is SQRT($0^2 + 0^2$) = 0 mm. With Isx86 = 50 (mm/sec), Z-axis speed is 50 mm/sec, move time (unblended) is 0.4 sec.

```
FRAX(X,Y,Z)
INC
X-30 Y-40 Z120 F65
```

Vector distance is SQRT($-30^2 + -40^2 + 120^2$) = 130 mm. Move time is 130/65 = 2.0 sec. X-axis speed is 30/2.0 = 15 mm/sec; Y-axis speed is 40/2.0 = 20 mm/sec; Z-axis speed is 120/2.0 = 60 mm/sec.

See Also:
I-variables Isx86 Isx87, Isx88, Isx89, Isx90
On-line command **FRAX**, **NOFRAX**
Program commands **F**, **NOFRAX**, **LINEAR**, **CIRCLE**, **{axis}{data}**.

G{data}

Function: Preparatory Code (G-Code)

Type: Motion program

Syntax: **G{data}**

where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to

Turbo PMAC interprets this statement as a **CALL 10n0.({data'}*1000)** command, where n is the hundreds'digit of **{data},** and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n0, and the specified line label. (Programs 10n0 are usually used to implement the preparatory codes as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

This structure permits the implementation of customizable G-Code routines for machine-tool style applications by the writing of subroutines in motion programs 10n0. Arguments can be passed to these subroutines by following the G-Code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have G-codes only in the range 0-99, which permits the use of PROG 1000 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example:
G01 jumps to **N1000** of PROG 1000
G12 jumps to **N12000** of PROG 1000
G115 jumps to **N15000** of PROG 1010

See Also:
Program commands **CALL{data}, D{data}, M{data}, T{data}, RETURN**

GOSUB

Function: Unconditional Jump With Return

Type: Motion program (PROG only)

Syntax: **GOSUB{data}**

where:

- **{data}** is a constant or expression representing the line label to jump to

This command causes the motion program execution to jump to the line label (**N** or **O**) of the same motion program specified in **{data}**, with a jump back to the commands immediately following the **GOSUB** upon encountering the next **RETURN** command.

If **{data}** is a constant, the path to the subroutine will have been linked before program run time, so the jump is very quick. If **{data}** is a variable expression, it must be evaluated at run time, and the appropriate label then searched for. The search starts downward in the program to the end, then continues (if necessary) from the top of the program down.

A variable **GOSUB** command permits the equivalent structure to the CASE statement found in many high-level languages.

If the specified line label is not found, the **GOSUB** command will be ignored, and the program will continue as if the command had not occurred.

The **CALL** command is similar, except that it can jump to another motion program.

Examples:
GOSUB300 jumps to **N300** of this program, to jump back on **RETURN**
GOSUB8743 jumps to **N8743** of this program, to jump back on **RETURN**
GOSUB(P17) jumps to the line label of this program whose number matches the current value of P17, to jump back on **RETURN**

See Also:
Writing a Motion Program
Program commands **CALL**, **GOTO**, **N**, **O**, **RETURN**

GOTO

Function: Unconditional Jump Without Return

Type: Motion program (PROG only)

Syntax: **GOTO{data}**

where:

- **{data}** is an integer constant or expression with a value from 0 to 99,999.

This command causes the motion program execution to jump to the line label (**N** or **O**) specified in **{data}**, with no jump back.

If **{data}** is a constant, the path to the label will have been linked before program run time, so the jump is very quick. If **{data}** is a variable expression, it must be evaluated at run time, and the appropriate label then searched for. The search starts downward in the program to the end, and then continues (if necessary) from the top of the program down.

A variable **GOTO** command permits the equivalent structure to the CASE statement found in many high-level languages (see Examples, below).

If the specified line label is not found, the program will stop, and the coordinate system's Run-Time-Error bit will be set.

Note:

Modern philosophies of the proper structuring of computer code strongly discourage the use of **GOTO**, because of its tendency to make code undecipherable.

Examples:

```
GOTO750
GOTO35000
GOTO1

GOTO(50+P1)
N51 P10=50*SIN(P11)
GOTO60
N52 P10=50*COS(P11)
GOTO60
N53 P10=50*TAN(P11)
N60 X(P10)
```

See Also:
Writing a Motion Program;
Program commands **CALL**, **GOSUB**, **N**, **O**.

HOME

Function: Programmed Homing

Type: Motion program

Syntax:
```
HOME {constant} [,{constant}...]
HOME {constant}..{constant}
[,{constant}..{constant}...]
HM {constant} [,{constant}...]
HM {constant}..{constant} [,{constant}..{constant}...]
```

where:

- **{constant}** is an integer from 1 to 8 representing a motor number.

This causes the specified motor(s) to go through their homing search cycle(s). Note that the motors must be specified directly by number, not the matching axis letters. You must specify which motors are to be homed. All motors specified in a single **HOME** command (e.g. **HOME1,2**) will start their homing cycles simultaneously; if you wish some motors to home sequentially, specify them in consecutive commands (e.g. **HOME1 HOME2**), even if on the same line.

Any previous moves will come to a stop before the home moves start. No other program statement will be executed until all specified motors have finished homing. Homing direction, speed, acceleration, etc. are determined by motor I-variables. If a motor is specified that is not in the coordinate system running the program, the command or portion of the command will be ignored, but an error will not be generated.

The speed of the home search move is determined by Ixx23. If Ixx23=0 then the programmed home command for that axis is ignored.

> ***Note:***
>
> Unlike an on-line homing command, the motor number(s) in a program homing command is (are) specified after the word **HOME** itself, not before. In addition, an on-line homing command simply starts the homing search -- it does not give any indication when the search is complete; but a program homing command automatically recognizes the end of the search, and then continues on in the program. A PLC program can only issue an on-line home command.

Examples:

```
HOME1                ;These are motion program commands
HM1,2,3
HOME1..3,5..7
HM1..8

#1HOME               ;These are on-line commands
#1HM,#2HM,#3HM
```

See Also:

Homing-Search Moves (Basic Motor Moves)
On-line motor commands **HOME**, **HOMEZ**
Program command **HOMEZ**

HOMEZ

Function: Programmed Zero-Move Homing

Type: Motion program

Syntax: **HOMEZ {constant} [,{constant}...]**
HOMEZ {constant}..{constant} [,{constant}..{constant}...]
HMZ {constant} [,{constant}...]
HMZ {constant}..{constant} [,{constant}..{constant}...]

where:

- **{constant}** is an integer from 1 to 8 representing a motor number.

This commands causes the specified motor(s) to go through pseudo-homing search cycle(s). In this operation, the present *commanded* position of the motor is made the zero position for the motor and the new *commanded* position for the motor.

If there is following error and/or an axis definition offset at the time of the **HOMEZ** command, the reported position after the command will be equal to the negative of the following error plus the axis definition offset.

Note that the motors must be specified directly by number, not the matching axis letters. You must specify which motors are to be homed. All motors specified in a single **HOMEZ** command (e.g. **HOMEZ1,2**) will home simultaneously.

> ***Note:***
>
> Unlike an on-line homing command, the motor number(s) in a program homing command is (are) specified *after* the word **HOMEZ** itself, not before.

Examples:

```
HOMEZ1                  ;These are motion program commands
HMZ1,2,3
HOMEZ1..3,5..7
HMZ1..8
#1HOMEZ                 ;These are on-line commands
#1HMZ,#2HMZ,#3HMZ
```

See Also:

Homing-Search Moves (Basic Motor Moves)
On-line motor command **HOME**, **HOMEZ**
Program command **HOME**

I{data}

Function: I-Vector Specification for Circular Moves or Normal Vectors

Type: Motion program (PROG or ROT)

Syntax: **I{data}**

where:

- **{data}** is a floating-point constant or expression representing the magnitude of the I-component of the vector in scaled user axis units.

In circular moves, this specifies the component of the vector to the arc center that is parallel to the X-axis. The starting point of the vector is either the move start point (for INC (R) mode -- default) or the XYZ-origin (for ABS (R) mode).

In a **NORMAL** command, this specifies the component of the normal vector to the plane of circular interpolation and tool radius compensation that is parallel to the X-axis.

Examples:

```
X10 Y20 I5 J5
X(2*P1) I(P1)
I33.333          ; specifies a full circle whose center is 33.333 units in the
                 ; positive X-direction from the start and end point
NORMAL I-1       ; specifies a vector normal to the YZ plane
```

See Also:
Circular Interpolation, Tool Radius Compensation (Writing a Motion Program)
On-line command **I{constant}**
Program Commands **{axis}{data}{vector}{data}**, **ABS**, **INC**, **NORMAL**, **J**, **K**, **I{constant}={expression}**

I{data}={expression}

Function: Set I-Variable Value

Type: Motion program (PROG and ROT), PLC Program

Syntax: **I{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the I-variable number;
- **{expression)** represents the value to be assigned to the specified I-variable.

This command sets the value of the specified I-variable to that of the expression on the right side of the equals sign. The assignment is done as the line is processed, which in a motion program is usually one or two moves ahead of the move actually executing at the time (because of the need to calculate ahead in the program).

> ***Note:***
>
> If you desire the assignment of the I-variable value to be synchronous with the beginning of the next move in the program, you should assign an M-variable to the register of the I-variable, and use a synchronous M-variable assignment statement (**M{data}=={expression}**).

Examples:

```
I130=30000
I902=1
I131=P131+1000
```

See Also:
How Turbo PMAC Executes a Motion Program (Writing a Motion Program)
On-line command **I{constant}={expression}**
Program commands **M{data}={expression)**, **P{data}={expression}**, **Q{data}={expression}**, **M{data}=={expression}**

IDIS{constant}

Function: Incremental displacement of X, Y, and Z axes

Type: Motion program (PROG and ROT)

Syntax: **IDIS{constant}**

where:

- **{constant}** is an integer representing the number of the first of three consecutive Q-variables to be used in the displacement vector

This command adds to the offset values of the currently selected (with **TSEL**) transformation matrix for the coordinate system the values contained in the three Q-variables starting with the

specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) by adding the values of these variables (Xnew=Xold+Q{constant}, Ynew=Yold+Q({constant}+1), Znew=Zold+Q({constant}+2)).

This command does not cause any movement of any axes; it simply renames the present positions.

This command is similar to a **PSET** command, except that **IDIS** is incremental and does not force a stop between moves, as **PSET** does.

Examples:

```
X0 Y0 Z0
Q20=7.5
Q21=12.5
Q22=20
IDIS 20 ..... ; This makes the current position X7.5, Y12.5, Z20
IDIS 20 ..... ; This makes the current position X15 Y25 Z40
```

See Also:

Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **AROT**, **IROT**, **TINIT**

IF ({condition})

Function: Conditional branch

Type: Motion and PLC program

Syntax: **IF ({condition})** (Valid in fixed motion (PROG) or PLC program only)
IF ({condition}) {action} [{action}...] (Valid in rotary or fixed motion program only)

where:

- **{condition}** consists of one or more sets of **{expression} {comparator} {expression}**, joined by logical operators **AND** or **OR**.
- **{action}** is a program command

This command allows conditional branching in the program.

With an action statement or statements following on that line, it will execute those statements provided the condition is true (this syntax is valid in motion programs only). If the condition is false, it will not execute those statements; it will only execute any statements on a false condition if the line immediately following begins with **ELSE**. If the next line does not begin with **ELSE**, there is an implied **ENDIF** at the end of the line.

> ***Note:***
>
> When there is an **ELSE** statement on the motion-program line immediately following an **IF** statement with actions on the same line, that **ELSE** statement is automatically matched to this **IF** statement, not to any preceding **IF** statements under which this **IF** statement may be nested.

With no statement following on that line, if the condition is true, Turbo PMAC will execute all subsequent statements on following lines down to the next **ENDIF** or **ELSE** statement (this syntax is valid in motion and PLC programs). If the condition is false, it will skip to the **ENDIF** or **ELSE** statement and continue execution there.

In a rotary motion program, only the single-line version of the **IF** statement is permitted. No **ELSE** or **ENDIF** statements are allowed.

In a PLC program, compound conditions can be extended onto multiple program lines with subsequent **AND** and **OR** statements.

There is no limit on nesting of **IF** conditions and **WHILE** loops (other than total buffer size) in fixed motion and PLC programs. No nesting is allowed in rotary motion programs.

Examples:

```
IF (P1>10) M1=1

IF (M11=0 AND M12!=0) M2=1 M3=1

IF (M1=0) P1=P1-1
ELSE P1=P1+1

IF (M11=0)
      P1=1000*SIN(P5)
      X(P1)
ENDIF

IF (P1<0 OR P2!<0)
AND (P50=1)
      X(P1)
      DWELL 1000
ELSE
      X(P1*2)
      DWELL 2000
ENDIF
```

See Also:
Conditions (Program Computational Features)
Program commands **ELSE**, **ENDIF**, **WHILE**, **AND**, **OR**

INC

Function: Incremental Move Mode

Type: Motion program

Syntax: **INC [({axis}[,{axis}...])]**

where:

- **{axis}** is a letter specifying a motion axis (X, Y, Z, A, B, C, U, V, W), or the letter R specifying the arc center radial vector.

The **INC** command without arguments causes all subsequent command positions in motion commands for all axes in the coordinate system running the motion program to be treated as incremental distances from the latest command point. This is known as incremental mode, as opposed to the default absolute mode.

An **INC** statement with arguments causes the specified axes to be in incremental mode, and all others stay the way they were before.

If **R** is specified as one of the 'axes', the I, J, and K terms of the circular move radius vector specification will be specified in incremental form (i.e. as a vector from the move start point, not from the origin). An **INC** command without any arguments does not affect this vector specification. The default radial vector specification is incremental.

If no motion program buffer is open when this command is sent to Turbo PMAC, it will be executed as an on-line coordinate system command.

Examples:

```
INC(A,B,C)
INC
INC(U)
INC(R)
```

See Also:

Circular Moves (Writing a Motion Program)
On-line commands **ABS**, **INC**
Program commands **{axis}{data}**, **{axis}{data}{vector}{data}**, **ABS**.

IROT{constant}

Function: Incremental rotation/scaling of X, Y, and Z axes

Type: Motion program (PROG and ROT)

Syntax: **IROT{constant}**

where:

- **{constant}** is an integer representing the number of the first of nine consecutive Q-variables to be used in the rotation/scaling matrix

This command multiplies the currently selected (with **TSEL**) transformation matrix for the coordinate system by the rotation/scaling values contained in the nine Q-variables starting with the specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) by multiplying the existing rotation/scaling matrix by the matrix containing these Q-variables, adding angles of rotation and multiplying scale factors.

The rotation and scaling is done relative to the latest rotation and scaling of the XYZ coordinate system, defined by the most recent **AROT** or **IROT** commands. The math performed is:

[New Rot Matrix] = [Old Rot Matrix] [Incremental Rot Matrix]

$[Xrot\ Yrot\ Zrot]^T = [New\ Rot\ Matrix]\ [Xbase\ Ybase\ Zbase]^T$

This command does not cause any movement of any axes; it simply renames the present positions.

Note:

When using this command to scale the coordinate system, do not use the radius center specification for circle commands. The radius does not get scaled. Use the **I, J, K** vector specification instead.

Examples:

Create a 3x3 matrix to rotate the XY plane by 30 degrees about the origin

```
Q40=COS(30).Q41=SIN(30) Q42=0
Q43=-SIN(30)       Q44=COS(30) Q45=0
Q46=0.......       Q47=0               Q48=1
IROT 40.....       ; Implement change, rotating 30 degrees from current
IROT 40.....       ; This rotates a further 30 degrees
```

Create a 3x3 matrix to scale the XYZ space by a factor of 3

```
Q50=3.......       Q51=0               Q52=0
Q53=0.......       Q54=3               Q55=0
Q56=0.......       Q57=0               Q58=3
IROT 50.....       ; Implement the change, scaling up by a factor of 3
IROT 50.....       ; Scale up by a further factor of 3 (total of 9x)
```

See Also:
Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **IDIS**, **AROT**, **TINIT**

J{data}

Function: J-Vector Specification for Circular Moves

Type: Motion program (PROG and ROT)

Syntax: **J{data}**

where:

- **{data}** is a floating-point constant or expression representing the magnitude of the J-component of the vector in scaled user axis units.

In circular moves, this specifies the component of the vector to the arc center that is parallel to the Y-axis. The starting point of the vector is either the move start point (for **INC (R)** mode -- default) or the XYZ-origin (for **ABS (R)** mode).

In a **NORMAL** command, this specifies the component of the normal vector to the plane of circular interpolation and tool radius compensation that is parallel to the Y-axis.

Examples:

```
X10 Y20 I5 J5
Y(2*P1) J(P1)
J33.333          ; Specifies a full circle whose center is 33.333 units in the
                 ; positive Y-direction from the start and end point
NORMAL J-1       ; Specifies a vector normal to the ZX plane
```

See Also:
Circular Interpolation, Tool Radius Compensation (Writing a Motion Program)
Motion Program Commands **{axis}{data}{vector}{data}**, **ABS**, **INC**, **NORMAL**, **I**, **K**.

K{data}

Function: K-Vector Specification for Circular Moves

Type: Motion program (PROG and ROT)

Syntax: **K{data}**

where:

- **{data}** is a floating-point constant or expression representing the magnitude of the K-component of the vector in scaled user axis units.

In circular moves, this specifies the component of the vector to the arc center that is parallel to the Z-axis. The starting point of the vector is either the move start point (for **INC (R)** mode -- default) or the XYZ-origin (for **ABS (R)** mode).

In a **NORMAL** command, this specifies the component of the normal vector to the plane of circular interpolation and tool radius compensation that is parallel to the Y-axis.

Examples:

```
X10 Z20 I5 K5
Z(2*P1) K(P1)
K33.333          ; Specifies a full circle whose center is 33.333 units in the
                 ; positive Z-direction from the start and end point
NORMAL K-1       ; Specifies a vector normal to the XY plane
```

See Also:
Circular Interpolation, Tool Radius Compensation (Writing a Motion Program)
Motion Program Commands **{axis}{data}{vector}{data}**, **ABS**, **INC**, **NORMAL**, **I**, **J**.

LINEAR

Function: Blended Linear Interpolation Move Mode

Type: Motion program (PROG and ROT)

Syntax: **LINEAR**
LIN

The **LINEAR** command puts the program in blended linear move mode (this is the default condition on power-up/reset). Subsequent move commands in the program will be processed according to the rules of this mode. On each axis, the card attempts to reach a constant velocity that is determined by the most recent feedrate (**F**) or move time (**TM**) command.

The **LINEAR** command takes the program out of any of the other move modes (**CIRCLE**, **PVT**, **RAPID**, **SPLINE**). A command for any of these other move modes takes the program out of **LINEAR** mode.

Examples:

```
LINEAR ABS
CIRCLE1 X10 Y20 I5
LINEAR X10 Y0
OPEN PROG 1000 CLEAR
N1000 LINEAR RETURN
```

See Also:
Linear Blended Moves (Writing a Motion Program);
I-variables Isx87, Isx88, Isx89, Isx90;
Program commands **CIRCLE**, **PVT**, **RAPID**, **SPLINE**, **TA**, **TS**, **TM**, **F**, **{axis}{data}**.

LOCK{constant},P{constant}

Function: Check/set process locking bit

Type: Motion program/PLC program

Syntax: **LOCK{constant},P{constant}**

where:

- the first **{constant}** is an integer from 0 to 7 representing the number of the locking bit;
- the second **{constant}** is an integer from 0 to 8191 specifying the number of the P-variable used to report the status of the locking bit.

The **LOCK** command permits the user to check and possibly "take possession of" one of the 8 process locking bits in Turbo PMAC. These locking bits can prevent conflicts between tasks of different priorities attempting to manipulate the same register. On-line commands and PLCs 1 – 31 are background tasks; motion programs and PLC 0 are higher-priority foreground tasks.

When the **LOCK** command is invoked, the P-variable specified in the command takes the value of the locking bit immediately before the command is invoked. It takes a value of 0 if the locking bit was not set before the command (meaning the process is available for this task); it takes a value of 1 if the locking bit was set before the command (meaning the process is not available for this task).

The locking bit itself is always set to 1 at the end of a **LOCK** command. It will stay at 1 until cleared by an **UNLOCK** command.

The status of locking bits 0 – 7 is reported as bits 4 – 11, respectively, of I4904.

If no motion program buffer or PLC program buffer is open when this command is issued, this command will be executed immediately as an on-line command.

Example:

```
P10=1                ; Assume locked to start
WHILE(P10=1)         ; Loop until unlocked
  LOCK4,P10  ; Check status of locking bit 4
ENDWHILE
M1=M1^1              ; Invert Machine Output 1
UNLOCK4              ; Release process 4 for other tasks
```

M{data}

Function: Machine Code (M-Code)

Type: Motion program

Syntax: **M{data}**

where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to

Turbo PMAC interprets this statement as a **CALL 10n1.({data'}*1000)** command, where n is the hundreds'digit of **{data},** and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n1, and the specified line label. (Programs 10n1 are usually used to implement the machine codes as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

This structure permits the implementation of customizable M-Code routines for machine-tool style applications by the writing of subroutines in motion programs 10n1. Arguments can be passed to these subroutines by following the M-Code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have M-codes only in the range 0-99, which permits the use of PROG 1001 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example:

M01 jumps to **N1000** of PROG 1001
M12 jumps to **N12000** of PROG 1001
M115 jumps to **N15000** of PROG 1011

See Also:

Program commands **CALL{data}, D{data}, M{data}, T{data}, RETURN**

M{data}={expression}

Function: Set M-Variable Value

Type: Motion program (PROG and ROT)

Syntax: **M{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the M-variable number;
- **{expression}** is a mathematical expression representing the value to be assigned to this M-variable.

This command sets the value of the specified M-variable to that of the expression on the right side of the equals sign.

Note:

In a motion program, the assignment is done as the line is processed, not necessarily in order with the actual execution of the move commands on either side of it. If it is in the middle of a continuous move sequence, the assignment occurs one or two moves ahead of its apparent place in the program (because of the need to calculate ahead in the program).

If you wish the actual assignment of the value to the variable to be synchronous with the beginning of the next move, use the synchronous M-variable assignment command **M{constant}=={expression}** instead.

Examples:

```
M1=1
M102=$00FF
M161=P161*I108*32
M20=M20 & $0F
```

See Also:

How Turbo PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)

Program Commands **I{data}=**, **P{data}=**, **Q{data}=**, **M{data}==**.

M{data}=={expression}

Function: Synchronous M-Variable Value Assignment

Type: Motion program

Syntax: **M{data}=={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the M-variable number;
- **{expression}** is a mathematical expression representing the value to be assigned to this M-variable.

This command allows the value of an M-variable to be set synchronously with the start of the next move or dwell. This is especially useful with M-variables assigned to outputs, so the output changes synchronously with beginning or end of the move. Non-synchronous calculations (with the single '=') are fully executed ahead of time, during previous moves.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, TWS).

In this form, the expression on the right side is evaluated just as for a non-synchronous assignment, but the resulting value is not assigned to the specified M-variable until the start of the actual execution of the following motion command.

Note:

If you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Examples:

```
X10
M1==1          ; Set Output 1 at start of actual blending to next move.
X20
M60==P1+P2
M(Q5000)==0
```

See Also:

How Turbo PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)

Program Commands **I{data}=**, **P{data}=**, **Q{data}=**, **M{data}=**, **M{data}&=**, **M{data}|=**, **M{data}^=**.

M{data}&={expression}

Function: M-Variable Synchronous "And-Equals" Assignment

Type: Motion program (PROG and ROT)

Syntax: **M{data}&={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the M-variable number;
- **{expression}** is a mathematical expression representing the value to be "ANDed" with this M-variable.

This command is equivalent to **M{data}=M{data}&{expression}**, except that the bit-by-bit AND and the assignment of the resulting value to the M-variable do not happen until the start of the actual execution of the following motion command. The expression itself is evaluated when the program line is encountered, as in a non-synchronous statement.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, or TWS), or with any of the double-word forms (L, D, or F).

Remember that if you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Examples:

```
M20&=$FE.....      ; Mask out LSB of byte M20
M346&=2 .....      ; Clear all bits except bit 1
M(P3+2)&=$55
```

See Also:

How Turbo PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)

Program Commands **M{data}=**, **M{data}==**, **M{data}|=**, **M{data}^=**

M{data}|={expression}

Function: M Variable Synchronous "Or-Equals" Assignment

Type: Motion program (PROG and ROT)

Syntax: **M{data}|={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the M-variable number;

- **{expression}** is a mathematical expression representing the value to be "ORed" with this M-variable.

This form is equivalent to **M{data}=M{data}|{expression}**, except that the bit-by-bit OR and the assignment of the resulting value to the M-variable do not happen until the start of the following servo command. The expression itself is evaluated when the program line is encountered, as in a non-synchronous statement.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, or TWS), or with any of the double-word forms (L, D, or F).

Remember that if you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Examples:

```
M20|=$01.....        ; Set low bit of byte M20, leave other bits
M875|=$FF00..; Set high byte, leaving low byte as is
M(Q5)|=Q8 & $F
```

See Also:

How Turbo PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)

Program Commands **M{data}=**, **M{data}==**, **M{data}&=**, **M{data}^=**

M{data}^={expression}

Function: M-Variable Synchronous "XOR-Equals" Assignment

Type: Motion program (PROG and ROT)

Syntax: **M{data}^={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the M-variable number;
- **{expression}** is a mathematical expression representing the value to be "XORed" with this M-variable.

This form is equivalent to **M{data}=M{data}^{expression}**, except that the bit-by-bit XOR and the assignment of the resulting value to the M-variable do not happen until the start of the following servo command. The expression itself is evaluated when the program line is encountered, as in a non-synchronous statement.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, or TWS), or with any of the double-word forms (L, D, or F).

Remember that if you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Examples:

```
M20^=$FF.....        ; Toggle all bits of byte M20
M99^=$80.....        ; Toggle bit 7 of M99, leaving other bits as is
M(P1)^=P2
```

See Also:
How Turbo PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)
Program Commands **M{data}=**, **M{data}==**, **M{data}&=**, **M{data}|=**

MACROAUXREAD

Function: Read (copy) Type 0 MACRO auxiliary parameter value from slave node

Type: Uncompiled PLC 1 – 31 only

Syntax: **MACROAUXREAD{node #},{param #},{variable}**
MXR{node #},{param #},{variable}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{param #}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node
- **{variable}** is the name of the Turbo PMAC variable (I, P, Q, or M) into which the parameter value is to be copied

This command causes Turbo PMAC to copy the value of the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC to the specified Turbo PMAC variable, using the MACRO Type 0 master-to-slave auxiliary protocol.

The auxiliary register function for the specified node number must have been enabled by setting the appropriate bit of I70, I72, I74, or I76 to 1. The Type 0 protocol for this node must have been selected by setting the appropriate bit of I71, I73, I75, or I77 to 0.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

When Turbo PMAC executes this command in a PLC, it ends the present scan of that PLC. Execution of the PLC resumes on the next line of the PLC in the next scan.

If this command is issued to a Turbo PMAC when no buffer is open, it will be executed as an on-line command.

Note:

Delta Tau MACRO Station products use the Type 1 protocol, and so do not use this command.

Example:

```
MACROAUXREAD1,24,P1      ; Read Node 1 Parameter 24 into P1
MXR5,128,M100            ; Read Node 5 Parameter 128 into M100
```

See Also:
On-line commands **MACROAUXREAD, MACROAUXWRITE**
PLC Program command **MACROAUXWRITE**

MACROAUXWRITE

Function: Write (copy) Type 0 MACRO auxiliary parameter value to slave node

Type: Uncompiled PLC 1 – 31 only

Syntax: **MACROAUXWRITE{node #},{param #},{variable}**
MXW{node #},{param #},{variable}

where:

- **{node #}** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **{node #}** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **{node #}** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **{node #}** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **{node #}** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **{param #}** is an integer constant from 2 to 253 specifying the auxiliary parameter number for this node
- **{variable}** is the name of the Turbo PMAC variable (I, P, Q, or M) from which the parameter value is to be copied

This command causes Turbo PMAC to copy the value of the specified variable on Turbo PMAC to the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC, using the MACRO Type 0 master-to-slave auxiliary protocol.

Only one auxiliary access (read or write) of a single node can be done on one command line.

The auxiliary register function for the specified node number must have been enabled by setting the appropriate bit of I70, I72, I74, or I76 to 1. The Type 0 protocol for this node must have been selected by setting the appropriate bit of I71, I73, I75, or I77 to 0.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

When Turbo PMAC executes this command in a PLC, it ends the present scan of that PLC. Execution of the PLC resumes on the next line of the PLC in the next scan.

If this command is issued to a Turbo PMAC when no buffer is open, it will be executed as an on-line command.

Note:

Delta Tau MACRO Station products use the Type 1 protocol, and so do not use this command.

Example:

```
MACROAUXWRITE1,24,P1      ; Write value of P1 to Node 1 Parameter 24
MXW5,128,M100             ; Write value of M100 to Node 5 Parameter 128
```

See Also:
On-line commands **MACROAUXREAD, MACROAUXREAD**
PLC Program command **MACROAUXREAD**

MACROMSTREAD

Function: Read (copy) Type 1 MACRO master auxiliary parameter value

Type: Uncompiled PLC 1 – 31 only

Syntax: **MACROMSTREAD{master #},{master variable},**
{ring-master variable}
MMR{master #},{master variable},{ring-master variable}

where:

- **{master #}** is a constant (1-15) representing the number of the remote master whose variable is to be read;
- **{master variable}** is the name of the variable on the remote master station whose value is to be reported
- **{ring-master variable}** is the name of the variable on the Turbo PMAC executing the command into which the value of the remote master variable is to be copied

This command causes Turbo PMAC to copy the value of the specified variable on the remote master station into the specified variable on the Turbo PMAC executing the command, using the MACRO Type 1 master-to-master auxiliary protocol.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- MACRO IC 0 on this Turbo PMAC must be set up as the synchronizing ring master (I6840 = $xx30);
- MACRO IC 0 on this Turbo PMAC must be set up as Master 0 on the ring (I6841 = $0xxxxx);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

In order for another Turbo PMAC to be able to respond to the communications that this command creates on the MACRO ring (to be the “remote master”) the following conditions must be true:

- MACRO IC 0 on the Turbo PMAC must be set up as a master, but not the synchronizing ring master (I6480 = $xx90);
- MACRO IC 0 on the Turbo PMAC must be set up with a non-zero master number on the ring (I6841 = $nxxxxx, where n > 0);
- MACRO IC 0 node 14 normally must be disabled (I6841 bit 14 = 1);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

If the remote master returns an error message or it does not respond within I79 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

When Turbo PMAC executes this command in a PLC, it ends the present scan of that PLC. Execution of the PLC resumes on the next line of the PLC in the next scan.

If this command is issued to a Turbo PMAC when no buffer is open, it will be executed as an on-line command.

Examples:

```
MMR4,I10,P10        ; Copies value of remote master 4 variable I10 into
                    ; Turbo PMAC variable P10
MMR1,P1,P1          ; Copies value of remote master 1 variable P1 into
                    ; Turbo PMAC variable P1
```

MACROMSTWRITE

Function: Write (copy) Type 1 MACRO master auxiliary parameter value

Type: Uncompiled PLC 1 – 31 only

Syntax: **`MACROMSTWRITE{master #},{master variable},{ring-master variable}`**
`MMW{master #},{master variable},{ring-master variable}`

where:

- **`{master #}`** is a constant (1-15) representing the number of the remote master whose variable is to be read;
- **`{master variable}`** is the name of the variable on the remote master station whose value is to be set;
- **`{ring-master variable}`** is the name of the variable on the Turbo PMAC executing the command from which the value of the remote master variable is to be copied.

This command causes Turbo PMAC to copy the value of the specified variable on the remote master station from the specified variable on the Turbo PMAC executing the command, using the MACRO Type 1 master-to-master auxiliary protocol.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- MACRO IC 0 on this Turbo PMAC must be set up as the synchronizing ring master (I6840 = $xx30);
- MACRO IC 0 on this Turbo PMAC must be set up as Master 0 on the ring (I6841 = $0xxxxx);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

In order for another Turbo PMAC to be able to respond to the communications that this command creates on the MACRO ring (to be the “remote master”) the following conditions must be true:

- MACRO IC 0 on the Turbo PMAC must be set up as a master, but not the synchronizing ring master (I6480 = $xx90);
- MACRO IC 0 on the Turbo PMAC must be set up with a non-zero master number on the ring (I6841 = $nxxxxx, where n > 0);
- MACRO IC 0 node 14 must normally be disabled (I6841 bit 14 = 1);
- MACRO IC 0 node 14 must be in “broadcast mode” (I6840 bit 14 = 1);
- MACRO IC 0 node 14 auxiliary register copy function must be disabled (I70 bit 14 = 0);
- MACRO IC 0 node 14 must not be used for any other function.

If the remote master returns an error message or it does not respond within I79 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

When Turbo PMAC executes this command in a PLC, it ends the present scan of that PLC. Execution of the PLC resumes on the next line of the PLC in the next scan.

If this command is issued to a Turbo PMAC when no buffer is open, it will be executed as an on-line command.

Examples:

```
MMW4,I10,P10        ; Copies value of Turbo PMAC variable P10 into remote master 4 variable I10
MMW1,P1,P1          ; Copies value of Turbo PMAC variable P1 into remote master 1 variable P1
```

MACROSLVREAD

Function: Read (copy) Type 1 MACRO auxiliary parameter value

Type: Uncompiled PLC 1 – 31 only

Syntax: **`MACROSLVREAD{node #},{slave variable},{PMAC variable}`**
`MSR{node #},{slave variable},{PMAC variable}`

where:

- **`{node #}`** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **`{node #}`** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **`{node #}`** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **`{node #}`** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **`{node #}`** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **`{slave variable}`** is the name of the variable on the slave station whose value is to be reported.
- **`{PMAC variable}`** is the name of the variable on the Turbo PMAC into which the value of the slave station variable is to be copied.

This command causes Turbo PMAC to copy the value of the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC to the specified Turbo PMAC variable, using the MACRO Type 1 master-to-slave auxiliary protocol.

The variable on the Turbo PMAC can be any of the I, P, Q, or M-variable on the card.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

When Turbo PMAC executes this command in a PLC, it ends the present scan of that PLC. Execution of the PLC resumes on the next line of the PLC in the next scan.

If this command is issued to a Turbo PMAC when no buffer is open, it will be executed as an on-line command.

Examples:

```
MSR0,MI910,P1          ; Copies value of slave Node 0 variable MI910 into PMAC variable P1
MSR1,MI997,M10         ; Copies value of slave Node 1 variable MI997 into PMAC variable M10
```

MACROSLVWRITE

Function: Write (copy) Type 1 MACRO auxiliary parameter value

Type: Uncompiled PLC 1 – 31 only

Syntax: **`MACROSLVWRITE{node #},{slave variable},{PMAC variable}`**
`MSW{node #},{slave variable},{PMAC variable}`

where:

- **`{node #}`** is a constant in the range 0 to 63 representing the number of the node on the Turbo PMAC matching the slave node to be accessed, where the node number specification is:
  - **`{node #}`** = 0 – 15 specifies MACRO IC 0 nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 0;
  - **`{node #}`** = 16 – 31 specifies MACRO IC 1 (Ultralites with Option 1A only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 1;
  - **`{node #}`** = 32 – 47 specifies MACRO IC 2 (Ultralites with Option 1B only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 2;
  - **`{node #}`** = 48 – 63 specifies MACRO IC 3 (Ultralites with Option 1C only) nodes 0 – 15, which correspond to slave nodes 0 – 15 of the same master number as MACRO IC 3;
- **`{slave variable}`** is the name of the MI-variable or C-command on the slave station whose value is to be set;
- **`{PMAC variable}`** is the name of the variable on the PMAC from which the value of the slave station variable is to be copied

This command causes Turbo PMAC to copy the value of the specified variable on Turbo PMAC to the specified variable of the MACRO slave station matching the specified node number on the Turbo PMAC, using the MACRO Type 1 master-to-slave auxiliary protocol.

The variable on the Turbo PMAC can be any of the I, P, Q, or M-variables on the card.

In order for the Turbo PMAC to be able to execute this command, the following conditions must be true:

- The MACRO IC used must be set up as a master or the synchronizing ring master (I6840/6890/6940/6990 = $xx90 or $xx30);
- MACRO IC 0 node 15 auxiliary register copy function must be disabled (I70 bit 15 = 0);
- MACRO IC 0 node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I78 servo cycles, Turbo PMAC will report ERR008. Bit 5 of global status register X:$000006 is set to report such a MACRO auxiliary communications error. Register X:$0031EE holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

When Turbo PMAC executes this command in a PLC, it ends the present scan of that PLC. Execution of the PLC resumes on the next line of the PLC in the next scan.

If this command is issued to a Turbo PMAC when no buffer is open, it will be executed as an on-line command.

Examples:

```
MSW0,MI910,P35      ; Copies value of PMAC P35 into MACRO station node 0 variable MI910
MSW4,C4,P0          ; Causes MACRO station with active node 4 to execute Command #4,
                    ; saving its setup variable values to non-volatile memory (P0 is a
                    ;dummy variable here)
```

N{constant}

Function: Program Line Label

Type: Motion program (PROG and ROT)

Syntax: **N{constant}**

where:

- **{constant}** is an integer from 0 to 262,143 (2^{18}-1)

This is a label for a line in the program that allows the flow of execution to jump to that line with a **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, or **D** statement or a **B** command.

A line only needs a label if the user wishes to be able to jump to that line. Line labels do not have to be in any sort of numerical order. The label must be at the beginning of a line. Remember that each location label takes up space in Turbo PMAC memory.

> ***Note:***
>
> There is always an implied **N0** at the beginning of every motion program. Putting an explicit **N0** at the beginning may be useful for people reading the program. Putting an **N0** anywhere else in the program is useless and may confuse people reading the program.

Examples:

```
N1
N65537 X1000
```

See Also:

Subroutines and Subprograms (Writing a Motion Program)

On-line command **B{constant}**

Program commands **O{constant}**, **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, **D**.

NOFRAX

Function: Remove all axes from list of vector feedrate axes

Type: Motion program (PROG and ROT)

Syntax: **NOFRAX**

This command causes Turbo PMAC to remove all axes from the list of vector feedrate axes for the addressed coordinate system. In this mode, any feedrate-specified move in the coordinate system will yield a vector distance of 0, forcing the use of the Isx86 alternate feedrate.

This can be useful to create a dry run of a motion program, overriding the feedrates specified in the motion-program **F** commands.

Axes can be restored to the vector feedrate list with the **FRAX** command.

See Also:
I-variables Isx86, Isx89, Isx90, Isx98
On-line commands **NOFRAX**, **FRAX**
Program commands **F**, **FRAX**

NORMAL

Function: Define Normal Vector to Plane of Circular Interpolation and Cutter Radius Compensation

Type: Motion program (PROG and ROT)

Syntax:
```
NORMAL {vector}{data} [{vector}{data}...]
NRM {vector}{data} [{vector}{data}...]
```

where:

- **{vector}** is one of the letters I, J, and K, representing components of the total vector parallel to the X, Y, and Z axes, respectively
- **{data}** is a constant or expression representing the magnitude of the particular vector component.

This statement defines the orientation of the plane in XYZ-space in which circular interpolation and cutter radius compensation will take place by setting the normal (perpendicular) vector to that plane.

The vector components that can be specified are I (X-axis direction), J (Y-axis direction), and K (Z-axis direction). The ratio of the component magnitudes determines the orientation of the normal vector, and therefore, of the plane. The length of this vector does not matter -- it does not have to be a unit vector.

The direction sense of the vector does matter, because it defines the clockwise sense of an arc move, and the sense of cutter-compensation offset. Turbo PMAC uses a right-hand rule; that is, in a right-handed coordinate system (**I x J = K**), if your right thumb points in the direction of the normal vector specified here, your right fingers will curl in the direction of a clockwise arc in the circular plane, and in the direction of offset-right from direction of movement in the compensation plane. In general, the negative normal vector produces the clockwise/counterclockwise sense expected.

Examples:
The standard settings to produce circles in the principal planes will therefore be:

```
NORMAL K-1....      ; XY plane -- equivalent to G17
NORMAL J-1....      ; ZX plane -- equivalent to G18
NORMAL I-1....      ; YZ plane -- equivalent to G19
```

By using more than one vector component, a circular plane skewed from the principal planes can be defined:

```
NORMAL I0.866 J0.500
NORMAL J25 K-25
NORMAL J(-SIN(Q1)) K(-COS(Q1))
NORMAL I(P101) J(P201) K(301)
```

See Also:
Circular Blended Moves, Cutter Radius Compensation (Writing a Motion Program)
Cartesian Axes (Setting Up a Coordinate System)
Program Commands **CIRCLE1**, **CIRCLE2**, **CC0**, **CC1**, **CC2**

NX{data}

Function: Set 3D-comp surface-normal vector X-component

Type: Motion program (PROG and ROT)

Syntax: **NX{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the X-component of the surface-normal vector

This statement specifies the X-component of the surface-normal vector used for three-dimensional cutter-radius compensation. This value is used along with the Y and Z-components specified by the **NY{data}** and **NZ{data}** statements, respectively, to compute the orientation of the vector.

The total magnitude of the surface-normal vector specified with these three components does not matter, although a unit-magnitude vector is typically specified. The relative magnitudes (including signs) of the three components are what determine the orientation of the vector. The vector must be defined from the surface toward the tool. Generally, all three components be declared together on one line; if only one or two components are declared, the other(s) are left at their old value(s), possibly leading to unpredictable results.

The surface-normal vector affects the compensation for the end-point of the move commanded on the same line as the surface-normal vector. It also affects the compensation for subsequent moves until another surface-normal vector is declared. In typical use, a new surface-normal vector is declared with each move, so the vector only affects the move on the same line in this case.

Example:

```
X3.76 Y8.29 Z-4.83 A34.32 C-29.75 NX0.866 NY0 NZ-0.5
```

See Also:

Three-Dimensional Compensation

Program commands **CC3**, **CCR{data}**, **NY{data}**, **NZ{data}**, **TR{data}**, **TX{data}**, **TY{data}**, **TZ{data}**

NY{data}

Function: Set 3D-comp surface-normal vector Y-component

Type: Motion program (PROG and ROT)

Syntax: **NY{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the Y-component of the surface-normal vector

This statement specifies the Y-component of the surface-normal vector used for three-dimensional cutter-radius compensation. This value is used along with the X and Z-components specified by the **NX{data}** and **NZ{data}** statements, respectively, to compute the orientation of the vector.

The total magnitude of the surface-normal vector specified with these three components does not matter, although a unit-magnitude vector is typically specified. The relative magnitudes (including signs) of the three components are what determine the orientation of the vector. The vector must be defined from the surface toward the tool. Generally, all three components be declared together on one line; if only one or two components are declared, the other(s) are left at their old value(s), possibly leading to unpredictable results.

The surface-normal vector affects the compensation for the end-point of the move commanded on the same line as the surface-normal vector. It also affects the compensation for subsequent moves until another surface-normal vector is declared. In typical use, a new surface-normal vector is declared with each move, so the vector only affects the move on the same line in this case.

Example:
```
X3.76 Y8.29 Z-4.83 A34.32 C-29.75 NX0.866 NY0 NZ-0.5
```

See Also:
Three-Dimensional Compensation
Program commands **CC3**, **CCR{data}**, **NX{data}**, **NZ{data}**, **TR{data}**, **TX{data}**, **TY{data}**, **TZ{data}**

NZ{data}

Function: Set 3D-comp surface-normal vector Z-component

Type: Motion program (PROG and ROT)

Syntax: **NZ{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the Z-component of the surface-normal vector

This statement specifies the Z-component of the surface-normal vector used for three-dimensional cutter-radius compensation. This value is used along with the X and Y-components specified by the **NX{data}** and **NY{data}** statements, respectively, to compute the orientation of the vector.

The total magnitude of the surface-normal vector specified with these three components does not matter, although a unit-magnitude vector is typically specified. The relative magnitudes (including signs) of the three components are what determine the orientation of the vector. The vector must be defined from the surface toward the tool. Generally, all three components be declared together on one line; if only one or two components are declared, the other(s) are left at their old value(s), possibly leading to unpredictable results.

The surface-normal vector affects the compensation for the end-point of the move commanded on the same line as the surface-normal vector. It also affects the compensation for subsequent moves until another surface-normal vector is declared. In typical use, a new surface-normal vector is declared with each move, so the vector only affects the move on the same line in this case.

Example:
```
X3.76 Y8.29 Z-4.83 A34.32 C-29.75 NX0.866 NY0 NZ-0.5
```

See Also:
Three-Dimensional Compensation
Program commands **CC3**, **CCR{data}**, **NX{data}**, **NY{data}**, **TR{data}**, **TX{data}**, **TY{data}**, **TZ{data}**

O{constant}

Function: Alternate Line Label

Type: Motion program (PROG and ROT)

Syntax: **O{constant}**

where:

- **{constant}** is an integer from 0 to 262,143 (2^{18}-1)

This is an alternate form of label in the motion program. It allows the flow of execution to jump to that line with a **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, or **D** statement or a **B** command.

Turbo PMAC will store and report this as an **N{constant}** statement, but **O** labels are legal to send to the program buffer. (**N10** and **O10** are identical labels to Turbo PMAC.)

A line only needs a label if the user wishes to be able to jump to that line. Line labels do not have to be in any sort of numerical order. The label must be at the beginning of a line. Remember that each location label takes up space in Turbo PMAC memory.

Examples:

```
O1
O65537 X1000
```

See Also:
Subroutines and Subprograms (Writing a Motion Program)
On-line command **B{constant}**
Program commands **O{constant}**, **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, **D**.

OR({condition})

Function: Conditional OR

Type: PLC program

Syntax: **OR ({condition})**

This statement forms part of an extended compound condition to be evaluated in a PLC program. It must immediately follow an **IF**, **WHILE**, **AND**, or **OR** statement. This **OR** is a boolean operator logically combining the condition on its line with the condition on the program line above.

It takes lower precedence than operators within a compound condition on a single line (those within parentheses), and also lower precedence than an **AND** operator that starts a line. (**OR**s operate on groups of **AND**ed conditions.)

In motion programs, there can be compound conditions within one program line, but not across multiple program lines, so this statement is not permitted in motion programs.

Note:

This logical **OR**, which acts on conditions, should not be confused with the bit-by-bit **|** (vertical bar) or-operator, which operates on values.

Examples:

```
IF (M11=1)....          ; This branch increments P1 every cycle that
AND (M12=0)..           ; inputs M11 and M12 are different, and decrements
OR (M11=0)....          ; them every cycle that they are the same.
AND (M12=1)
   P1=P1+1
ELSE
   P1=P1-1
ENDIF

IF (M11=1 AND M12=0)    ; This does the same as above
OR (M11=0 AND M12=1)
   P1=P1+1
ELSE
   P1=P1-1
ENDIF
```

See Also:
Conditions (Program Computational Features)
Writing a PLC Program
Program commands **IF**, **WHILE**, **AND**

P{data}={expression}

Function: Set P-Variable Value

Type: Motion program (PROG and ROT)

Syntax: **P{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the P-variable number;
- **{expression}** represents the value to be assigned to this P-variable.

This command sets the value of the specified P-variable to that of the expression on the right side of the equals sign. The assignment is done as the line is processed, which in a motion program is usually one or two moves ahead of the move actually executing at the time (because of the need to calculate ahead in the program).

Examples:

```
P1=0
P746=P20+P40
P893=SIN(Q100)-0.5
```

See Also:

How Turbo PMAC Executes a Motion Program (Writing a Motion Program)
On-line command **P{constant}={expression}**
Program commands **I{data}={expression}**, **M{data}={expression}**, **Q{data}={expression}**.

PAUSE PLC

Function: Pause execution of PLC program(s)

Type: Motion program (PROG or ROT), PLC program

Syntax:

```
PAUSE PLC {constant}[,{constant}...]
PAU PLC {constant}[,{constant}...]
PAUSE PLC {constant}[..{constant}]
PAU PLC {constant}[..{constant}]
```

where:

- each **{constant}** is an integer from 0 to 31 representing the PLC number

This command causes Turbo PMAC to stop execution of the specified uncompiled PLC program or programs, with the capability to restart execution at this point (not necessarily at the top) with a **RESUME PLC** command. Execution can also be restarted at the top of the program with the **ENABLE PLC** command.

If the PLC program is paused from within that PLC, execution is stopped immediately after the **PAUSE PLC** command.

If the PLC program is paused while it is not in the middle of a scan, which is always the case if it is paused from another background PLC, it will obviously be paused at the end of a scan – after an **ENDWHILE** or after the last line.

If the PLC program is paused while it has been interrupted in the middle of a scan (for example, from a motion program), its execution will resume after the interrupt and continue until after it executes any of the following:

- Any **ENABLE PLC**, **DISABLE PLC**, **PAUSE PLC**, or **RESUME PLC** command
- An **ENDWHILE** command
- The last line of the program

Execution will be paused at this point.

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to Turbo PMAC, it will be executed immediately as an on-line command.

Examples:

```
PAUSE PLC 1
PAUSE PLC 4,5
PAUSE PLC 7..20
PAU PLC 3,8,11
PAU PLC 0..31
```

See Also:

I-variable I5

On-line commands **ENABLE PLC**, **DISABLE PLC**, **<CONTROL-D>, PAUSE PLC**, **RESUME PLC, LIST PLC**

Program command **ENABLE PLC**, **DISABLE PLC**, **RESUME PLC**

PRELUDE

Function: Specify automatic subroutine call function

Type: Motion program

Syntax: **PRELUDE1{command}**
PRELUDE0

where:

- **{command}** is a subprogram call from the set **CALL{data}, G{data}, M{data}, T{data}, D{data}**

The **PRELUDE1** command permits automatic insertion of a subprogram call before each subsequent motion command (e.g. **X10Y10**) or other letter-number command (e.g. **L10**) other than a line label in the motion program. The action taken is equivalent to inserting the call into the program text before each subsequent motion command or letter-number command.

The subprogram call to be performed can be specified in the **PRELUDE1** command either as a **CALL** command, or as a **G**, **M**, **T**, or **D** code. The value following the **CALL** or code letter must be a constant; it cannot be a variable or expression. It does not have to be an integer. If the routine called in the subprogram starts with a **READ** statement, the motion or letter-number command itself can become arguments for the subprogram call. Any motion command within a **PRELUDE1** subroutine or subprogram call is executed directly as a motion command, without an automatic subroutine call in front of it..

Turbo PMAC will only execute the **PRELUDE1** function if the motion or letter-number command is found at the beginning of a program line or immediately after the line label. If another type of command occurs earlier on the program line, no **PRELUDE1** function will be executed before the motion or letter-number command. If the command is on a line that is already in a subroutine or subprogram reached by a **CALL** or **GOSUB** command, no **PRELUDE1** function will be executed.

Each **PRELUDE1** command supersedes the previous **PRELUDE1** command. It is not possible to nest automatic **PRELUDE1** calls, but an automatic **PRELUDE1** call can be nested within explicit subroutine and subprogram calls.

PRELUDE0 disables any automatic subroutine call.

Examples:

```
PRELUDE1 CALL10          ; Insert a CALL10 before subsequent moves
X10 Y20 .....            ; Implicit CALL10 before this move
X20 Y30 .....            ; Implicit CALL10 before this move
...
OPEN PROG 10 CLEAR       ; Subprogram
Z-1.....                 ; Move down
DWELL 500 .....          ; Hold position
Z1 .....                 ; Move up
RETURN
...
G71 X7 Y15 P5            ; G71 calls PROG 1000 N71000
X8 Y16 P5 .....          ; With PRELUDE, G71 is implied (modal)
X9 Y15 P8 .....          ; With PRELUDE, G71 is implied (modal)
G70.....                 ; Stop modal canned cycles
...
OPEN PROG 1000
...
N70000.....              ; G70 subroutine
PRELUDE0.....            ; Stop PRELUDE calls
RETURN
N71000.....              ; G71 subroutine
PRELUDE1 G71.1           ; Make G71 modal by using PRELUDE
RETURN
N71100.....              ; G71.1 routine is what executes G71 modally
READ(X,Y,P)..            ; Read values associated with X, Y, and P
{action based on parameters}
RETURN
```

See Also:

Subroutines and Subprograms (Writing a Motion Program)

Program commands **CALL**, **GOSUB**, **READ**, **G**, **M**, **T**, **D**

PSET

Function: Redefine current axis positions (Position SET)

Type: Motion program

Syntax: **PSET{axis}{data} [{axis}{data}...]**

where:

- **{axis}** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **{data}** is a constant or an expression representing the new value for this axis position

This command allows the user to re-define the value of an axis position in the middle of the program. It is equivalent to the RS-274 G-Code **G92**. No move is made on any axis as a result of this command -- the value of the present commanded position for the axis is merely set to the specified value.

Internally, this command changes the value of the position bias register for each motor attached to an axis named in the command. This register holds the difference between the axis zero point and the motor zero (home) point.

This command automatically forces a temporary pause in the motion of the axes; no moves are blended through a **PSET** command. For more powerful and flexible offsets that can be done on the fly (X, Y, and Z axes only), refer to the matrix manipulation commands such as **ADIS** and **IDIS**.

Examples:

```
X10Y20
PSET X0 Y0....          ; Call this position (0,0)
N92000 READ(X,Y,Z)      ; To implement G92 in PROG 1000
PSET X(Q124)Y(Q125)Z(Q126)  ; Equivalent of G92 X..Y..Z..
```

See Also:

Axes (Setting Up a Coordinate System)
On-line command **{axis}={expression}**
Program commands **ADIS**, **AROT**, **IDIS**, **IROT**
Suggested M-variable definitions Mxx64

PVT{data}

Function: Set Position-Velocity-Time mode

Type: Motion program (PROG and ROT)

Syntax: **PVT{data}**

where:

- **{data}** is a positive constant or expression representing the time of a segment in milliseconds

This command puts the motion program into Position-Velocity-Time move mode, and specifies the time for each segment of the move, in milliseconds. In this mode, each move segment in the program must specify the ending position *and* velocity for the axis. Taking the starting position and velocity (from the previous segment), the ending position and velocity, and the segment time, Turbo PMAC computes the unique cubic position profile (parabolic velocity profile) to meet these constraints.

If global I-variable I42 is set to the default value of 0, the specified time uses the "move time" register, and has a range of 0 to 4095.9998 milliseconds, with a resolution of ¼-microsecond. In this case, if the segment time in a sequence of moves is changed on the fly, either another **PVT** command is used, or a **TM** command.

If I42 is set to 1, the specified time uses the "acceleration time" register, and has a range of 0 to 8,388,607 milliseconds(about 2 hours, 20 minutes), with a resolution of 1 millisecond. In this case, if the segment time in a sequence of moves is changed on the fly, either another **PVT** command is used, or a **TA** command.

The segment time in a sequence of moves can be changed on the fly, either with another **PVT** command, or with a **TM** command. **TS**, **TA**, and **F** settings are irrelevant in this mode.

The **PVT** command takes the program out of any of the other move modes (**LINEAR**, **CIRCLE**, **SPLINE**, **RAPID**), and any of the other move mode commands takes the program out of **PVT** move mode.

Refer to the *Writing a Motion Program* section of the manual for more details of this mode.

Examples:

```
INC             ; incremental mode, specify moves by distance
PVT200          ; enter this mode -- move time 200ms
X100:1500       ; cover 100 units ending at 1500 units/sec
X500:3000       ; cover 500 units ending at 3000 units/sec
X500:1500       ; cover 500 units ending at 1500 units/sec
X100:0          ; cover 100 units ending at 0 units/sec
PVT(P37)
```

See Also:
Position-Velocity-Time Mode Moves (Writing a Motion Program)
Program commands **{axis}{data}:{data}...**, **TA**, **LINEAR**, **CIRCLEn**, **RAPID**, **SPLINE1**.

Q{data}={expression}

Function: Set Q-Variable Value

Type: Motion program (PROG and ROT); PLC program

Syntax: **Q{data}={expression}**

where:

- **{data}** is a constant, or an expression in parentheses, for an integer value from 0 to 8191 representing the Q-variable number;
- **{expression}** represents the value to be assigned to the specified Q-variable.

This command sets the value of the specified Q-variable to that of the expression on the right side of the equals sign. The assignment is done as the line is processed, which in a motion program performing a continuous move sequence is usually one or two moves ahead of the move actually executing at the time (because of the need to calculate ahead in the program).

Because each coordinate system has its own set of Q-variables, it is important to know which coordinate system's Q-variable is affected by this command. When executed from inside a motion program, this command affects the specified Q-variable of the coordinate system running the motion program.

When executed from inside a PLC program, this command affects the specified Q-variable of the coordinate system specified by the most recent **ADDRESS** command executed inside *that* PLC program. If there has been no **ADDRESS** command executed since power-on/reset, it affects the Q-variable of Coordinate System 1.

Examples:

```
Q1=3
Q99=2.71828
Q124=P100+ATAN(Q120)
```

See Also:
Q-Variables (Program Computational Features)
On-line command **Q{constant}={expression}**
Program commands **ADDRESS**, **I{data}={expression}**, **M{data}={expression}**, **P{data}={expression}**

R{data}

Function: Set Circle Radius

Type: Motion program (PROG or ROT)

Syntax: **R{data}**

where:

- **{data}** is a constant or expression representing the radius of the arc move specified in user length units.

This partial command defines the magnitude of the radius for the circular move specified on that command line. It does not affect the moves on any other command lines.. (If there is no R radius specification and no IJK vector specification on a move command line, the move will be done linearly, even if the program is in **CIRCLE** mode.)

If the radius value specified in **{data}** is greater than zero, the circular move to the specified end point will describe an arc of less than or equal to 180° with a radial length of the specified value. If the radius value specified in **{data}** is less than zero, the circular move to the specified end point will describe an arc of greater than or equal to 180° with a radial length equal to the absolute value of **{data}**.

Note:

If you use the **AROT** or **IROT** commands to scale the coordinate system, do not use the radius center specification for circle commands. The radius does not get scaled. Use the **I, J, K** vector specification instead.

Note:

If the distance from the start point to the end point is more than twice the magnitude specified in **{data}**, there is no circular arc move possible. If the distance is greater than twice **{data}** by an amount less than Isx96 (expressed in user length units), Turbo PMAC will execute a spiral to the end point. If the distance is greater by more than Isx96, Turbo PMAC will stop the program with a run-time error.

Examples:

```
RAPID X0 Y0..          ; Move to origin
CIRCLE1.....           Clockwise circle mode
X10 Y10 R10..          ; Quarter circle to (10, 10)
X0 Y0 R-10....         Three-quarters circle back to (0, 0)
X(P101) R(P101/2)      ; Half circle to (P101, 0)
```

See Also:

Circular Blended Moves (Writing a Motion Program);
I-variables Isx13, Isx96
Program commands **CIRCLE1**, **CIRCLE2**, **{axis}{data}{vector}{data}...**

RAPID

Function:	Set Rapid Traverse Mode
Type:	Motion program (PROG and ROT)
Syntax:	**RAPID** **RPD**

This command puts the program into a mode in which all motors defined to the commanded axes move to their destination points in jog-style moves. This mode is intended to create the minimum-time move from one point to another. Successive moves are not blended together in this mode, and the different motors do not necessarily all reach their end points at the same time.

The accelerations and decelerations in this mode are controlled by motor jog-acceleration I-variables Ixx19, Ixx20, and Ixx21. If motor I-variable Ixx90 is set to 0, the velocities in this mode are controlled by the motor jog speed I-variables Ixx22. If Ixx90 is set to 1, they are controlled by the motor maximum speed I-variables Ixx16. Only the motor with the greatest distance-to-speed ratio for the move actually moves at this speed; all other motors are slowed from the specified speed to complete the move in approximately the same time, so that the move is nearly linear.

The **RAPID** command takes the program out of any of the other move modes (**LINEAR**, **CIRCLE**, **PVT**, **SPLINE**); any of the other move-mode commands takes the program out of **RAPID** mode.

Examples:

```
RAPID X10 Y20          ; Move quickly to starting cut position
M1=1 .....             ; Turn on cutter
LINEAR X12 Y25 F2      ; Start cutting moves
...
M1=0 .....             ; Turn off cutter
RAPID X0 Y0..          ; Move quickly back to home position
```

See Also:
Rapid Mode Moves (Writing a Motion Program)
I-variables Ixx90, Ixx16, Ixx19, Ixx22
Program commands **LINEAR**, **CIRCLE**, **PVT**, **SPLINE**

READ

Function: Read Arguments for Subroutine

Type: Motion program (PROG only)

Syntax: **READ({letter},[{letter}...])**

where:

- **{letter}** is any letter of the English alphabet, except **N** or **O**, representing the letter on the calling program line whose following value is to be read into a variable

Note:

No space is allowed between **READ** and the left parenthesis.

This statement allows a subprogram or subroutine to take arguments from the calling routine. It looks at the remainder of the line calling this routine (**CALL**, **G**, **M**, **T**, **D**), takes the values following the specified letters and puts them into particular Q-variables for the coordinate system. For the Nth letter of the alphabet, the value is put in Q(100+N).

It scans the calling line until it sees a letter that is not in the list of letters to **READ**, or until the end of the calling line. (Note that if it encounters a letter not to be read, it stops, even if some letters have not yet been read.) Each letter value successfully "read" into a Q-variable causes a bit to be set in Q100, noting that it was read (bit N-1 for the Nth letter of the alphabet). For any letter not successfully read in the most recent **READ** command, the corresponding bit of Q100 is set to zero.

The Q-variable and flag bit of Q100 associated with each letter are shown in the following table:

Letter	Target Variable	Q100 Bit	Bit Value Decimal	Bit Value Hex
A	Q101	0	1	$01
B	Q102	1	2	$02
C	Q103	2	4	$04
D	Q104	3	8	$08
E	Q105	4	16	$10
F	Q106	5	32	$20
G	Q107	6	64	$40
H	Q108	7	128	$80
I	Q109	8	256	$100
J	Q110	9	512	$200
K	Q111	10	1,024	$400
L	Q112	11	2,048	$800
M	Q113	12	4,096	$1000
N*	Q114*	13*	8,192*	$2000*
O*	Q115*	14*	16,384*	$4000*
P	Q116	15	32,768	$8000

Q	Q117	16	65,536	$10000
R	Q118	17	131,072	$20000
S	Q119	18	262,144	$40000
T	Q120	19	524,288	$80000
U	Q121	20	1,048,57	$100000
V	Q122	21	2,097,15	$200000
W	Q123	22	4,194,304	$400000
X	Q124	23	8,388,608	$800000
Y	Q125	24	16,777,216	$1000000
Z	Q126	25	33,554,432	$2000000
*Cannot be used				

Any letter may be **READ** except **N** or **O**, which are reserved for line labels (and should only be at the beginning of a line anyway). If a letter value is read from the calling line, the normal function of the letter (e.g. an axis move) is overridden, so that letter serves merely to pass a parameter to the subroutine. If there are remaining letter values on the calling line that are not read, those will be executed according to their normal function after the return from the subroutine.

Examples:

In standard machine tool code, a two-second **DWELL** would be commanded in the program as a **G04 X2000**, for instance. In Turbo PMAC, a **G04** is interpreted as a **CALL** to label **N04000** of PROG 1000, so to implement this function properly, PROG 1000 would contain the following code:

```
N04000 READ(X)
DWELL (Q124)
RETURN
```

Also, in standard machine tool code, the value assigned to the current position of the axis may be changed with the **G92** code, followed by the letters and the new assigned values of any axes (e.g. **G92 X20 Y30**). It is important only to assign new values to axes specified in this particular **G92** command, so the Turbo PMAC subroutine implementing **G92** with the **PSET** command must check to see if that particular axis is specified:

```
N92000 READ(X,Y,Z)
IF (Q100 & $800000 > 0) PSET X(Q124)
IF (Q100 & $1000000 > 0) PSET Y(Q125)
IF (Q100 & $2000000 > 0) PSET Z(Q126)
RETURN
```

See Also:

Subroutines and Subprograms (Writing a Motion Program)
Program commands **CALL**, **GOSUB**, **G**, **M**, **T**, **D**

RESUME PLC

Function: Resume execution of PLC programs(s)

Type: Motion program (PROG and ROT), PLC program

Syntax:
```
RESUME PLC {constant}[,{constant}...]
RES PLC {constant}[,{constant}...]
RESUME PLC{constant}[..{constant}]
RES PLC {constant}[..{constant}]
```

where:

- each **{constant}** is an integer from 0 to 31 representing the PLC number

This command causes Turbo PMAC to resume execution of the specified uncompiled PLC program or programs at the point where execution was suspended with the **PAUSE PLC** command, which is not necessarily at the top of the program.

The **RESUME PLC** command cannot be used to restart execution of a PLC program that has been stopped with a **DISABLE PLC** command. However, after a PLC has been stopped with a **DISABLE PLC** command, if a **PAUSE PLC** command is then given for that PLC, then a **RESUME PLC** command can be given to start operation at the point at which it has been stopped.

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to Turbo PMAC, it will be executed immediately as an on-line command.

Examples:

```
RESUME PLC 0
RESUME PLC 1,2,5
RESUME PLC 1..16
RES PLC 7
```

See Also:

I-variable I5

On-line commands **ENABLE PLC**, **DISABLE PLC**, **<CONTROL-D>**, **PAUSE PLC**, **RESUME PLC**

Program commands **ENABLE PLC**, **DISABLE PLC**, **PAUSE PLC**

RETURN

Function: Return From Subroutine Jump/End Main Program

Type: Motion program (PROG only)

Syntax: **RETURN**
RET

The **RETURN** command tells the motion program to jump back to the routine that called the execution of this routine. If this routine was started from an on-line command (Run), program execution stops and the program pointer is reset to the top of this motion program -- control is returned to the Turbo PMAC "operating system".

If this routine was started from a **GOSUB**, **CALL**, **G**, **M**, **T**, or **D** command in a motion program, program execution jumps back to the command immediately following the calling command.

When the **CLOSE** command is sent to end the entry into a motion program buffer, Turbo PMAC automatically appends a **RETURN** command to the end of that program. When the **OPEN** command is sent to an existing motion program buffer, the final **RETURN** command is automatically removed.

Examples:

```
OPEN PROG 1 CLEAR
X20 F10
X0
CLOSE.....                ; Turbo PMAC places a RETURN here
OPEN PROG 1000 CLEAR
N0 RAPID RETURN           ; Execution jumps back after one-line routine
N1000 LINEAR RETURN       ; Ditto
N2000 CIRCLE1 RETURN      ; Ditto
...
CLOSE.....                ; Turbo PMAC places a RETURN here
```

See Also:
Subroutines and Subprograms (Writing a Motion Program)
On-line commands **OPEN**, **CLOSE**
Program commands **CALL**, **GOSUB**, **G**, **M**, **T**, **D**

S{data}

Function: Spindle data command

Type: Motion program (PROG and ROT)

Syntax: **S{data}**

where:

- **{data}** is a constant or expression representing the value to be passed to the storage variable for later use

This command causes the value in **{data}** to be loaded in variable Q127 for the coordinate system executing the motion program. It takes no other action. It is intended to pass spindle speed data in machine tool programs. The algorithms that actually control the spindle would then use Q127 in their routines; for instance, to set jog speed, or voltage output.

> ***Note:***
>
> This command is distinct from **S{data}** information passed as part of a subroutine call through a **READ(S)** command; in that form, the value is placed in Q119 for the coordinate system.

Examples:

```
S1800.............  ; This puts a value of 1800 in Q127
S(P1).............  ; This puts the value of P1 in Q127
G96 S50...........  ; Here the S-term is an argument in the G-code call
(PROG 1000).......  ; This is the subroutine that executes G96
N96000 READ(S)      ; This puts a value of 50 in Q119
```

See Also:
Q-variables (Program Computational Features)
Implementing a Machine-Tool Style Program (Writing a Motion Program)
Motion program command **READ**
Example program SPINDLE.PMC

SENDx

Function: Cause Turbo PMAC to Send Message

Type: Motion program (PROG and ROT); PLC program

Syntax:
```
SENDS"{message}"
SENDP"{message}"
SENDR"{message}"
SENDA"{message}"
```

This command causes Turbo PMAC to send the specified message out a specified communications port. This is particularly useful in the debugging of applications. It can also be used to prompt an operator, or to notify the host computer of certain conditions.

If I62=0, Turbo PMAC issues a carriage-return (**<CR>**) character at the end of the message automatically. If I62=1, Turbo PMAC does not issue a **<CR>** character at the end of the message; a **SENDx^M** must be used to issue a **<CR>** in this case.

Note:

If there is no host on the port to which the message is sent, or the host is not ready to read the message, the message is left in the queue. If several messages back up in the queue this way, the program issuing the messages will halt execution until the messages are read. This is a common mistake when the **SENDx** command is used outside of an Edge-Triggered condition in a PLC program. See Writing A PLC Program for more details.

On the serial port, it is possible to send messages to a non-existent host by disabling the port handshaking with I1=1.

SENDS transmits the message to the main serial port.

SENDP transmits the message to the parallel bus port: ISA, VME, or PCI, whichever is present on the board.

SENDR transmits the message through the DPRAM ASCII response buffer.

SENDA transmits the message to the Option 9T auxiliary serial port.

Note:

If a program, particularly a PLC program, sends messages immediately on power-up/reset, it can confuse a host-computer program (such as the Turbo PMAC Executive Program) that is trying to "find" Turbo PMAC by querying it and looking for a particular response.

It is possible, particularly in PLC programs, to order the sending of messages faster than the port can handle them. This will almost always happen if the same **SENDx** command is executed every scan through the PLC. For this reason, it is good practice to have at least one of the conditions that causes the **SENDx** command to execute to be set false immediately to prevent execution of this **SENDx** command on subsequent scans of the PLC.

Note:

To cause Turbo PMAC to send the *value* of a variable, use the **COMMANDx** statement instead, specifying the name of the variable in quotes (e.g. **CMDS"P1"**)

Examples:

```
SENDR"Motion Program Started"
SENDS"DONE"
SENDP"Spindle Command Given"

IF (M188=1)..                       ; C.S.1 Warning Following Error Bit set?
  IF (P188=0)                       ; But not set last scan? (P188 follows M188)
    SENDA"Excessive Following Error"        ; Notify operator
    P188=1....                      ; To prevent repetition of message
  ENDIF
ELSE.................               ; F.E. Bit not set
  P188=0.....                       ; To prepare for next time
ENDIF

SENDS"THE VALUE OF P7 IS:"          ; Turbo PMAC to send message string
CMDS"P7".....                       ; Turbo PMAC to return the value of P7
```

See Also:

I-variables I1, I62

Program commands **COMMANDx**, **DISPLAY**, **SENDx^{letter}**

Writing A PLC Program

SENDx^{letter}

Function: Cause Turbo PMAC to Send Control Character

Type: Motion program (PROG and ROT); PLC program

Syntax:
```
SENDS^{letter}
SENDP^{letter}
SENDR^{letter}
SENDA^{letter}
```

where:

- **{letter}** is one of the characters in the following set: **@ABC...XYZ[\]^_**

This command causes Turbo PMAC to send the specified control character over the specified communications ports. These can be used for printer and terminal control codes, or for special communications to a host computer.

Control characters have ASCII byte values of 0 to 31 ($1F). The specified **{letter}** character determines which control character is sent when the statement is executed. The byte value of the control character sent is 64 ($40) less than the byte value of **{letter}**. The letters that can be used and their corresponding control characters are:

{letter}	Letter Value	Control Character	Value
@	64	NULL	0
A	65	<CTRL-A>	1
B	66	<CTRL-B>	2
C	67	<CTRL-C>	3
...			
X	88	<CTRL-X>	24
Y	89	<CTRL-Y>	25
Z	90	<CTRL-Z>	26
[	91	ESC	27
\	92		28
]	93		29
^	94		30
_	95		31

Note: Do not put the up-arrow character and the letter in quotes (e.g. do not use **SENDx"^A"**) or Turbo PMAC will attempt to send the two non-control characters **^** and **A** for this example, instead of the control character.

SENDS transmits the control character to the main serial port.

SENDP transmits the control character to the parallel bus port: ISA, VME, or PCI, whichever is present on the board.

SENDR transmits the control character through the DPRAM ASCII response buffer.

SENDA transmits the control character to the Option 9T auxiliary serial port.

When Turbo PMAC powers up or resets, the active response port is the serial port. When any command is received over a bus port, the active response port becomes the bus port. Turbo PMAC must then receive a **<CONTROL-Z>** command to cause the response port to revert back to the serial port.

It is possible, particularly in PLC programs, to order the sending of messages faster than the port can handle them. This will almost always happen if the same **SEND** command is executed every scan through the PLC.

For this reason, it is good practice to have at least one of the conditions that causes the **SEND** command to execute to be set false immediately to prevent execution of this **SEND** command on subsequent scans of the PLC.

See Also:
On-line command **<CTRL-Z>**
Program commands **SEND"{message}", COMMAND"{command}", COMMAND^{letter}**

SETPHASE

Function: Set commutation phase position value

Type: Motion programs & PLC programs

Syntax:
```
SETPHASE {constant} [,{constant}…]
SETPHASE {constant}..{constant}
[,{constant}..{constant}]
```

where:

- **{constant}** is an integer from 1 to 32 representing a motor number

The **SETPHASE** command causes Turbo PMAC to immediately copy the value of Ixx75 for the specified motor or motors into the active phase position register for that motor or motors. This command is typically used to correct the phasing of a motor at a known angle (such as the index pulse of the encoder) after an initial rough phasing (such as from Hall commutation registers).

The copying action is done immediately on execution of the command. In a motion program, this is done at program calculation, which can be ahead of execution of the accompanying moves if the command is in the middle of a blended sequence of moves.

To determine the value of Ixx75 to be used, first force an unloaded motor to the zero position in its phasing cycle. Next, manually set the phase position register (suggested M-variable Mxx71) to zero. Finally, move the motor to the "known position", usually with a homing search move to the index pulse or other trigger. Read the phase position register at this point and set Ixx75 to this value. For more details, see the Ixx75 description and the Commutation section of the User's Manual.

If no motion program buffer or PLC program buffer is open when this command is issued, this command will be executed immediately as an on-line command.

Examples:
```
HOME1
WHILE(M140=0)
ENDWHILE
SETPHASE1
```

SPLINE1

Function: Put program in uniform cubic spline motion mode

Type: Motion program (PROG and ROT)

Syntax: **SPLINE1**

This modal command puts the program in cubic B-spline mode. In **SPLINE1** mode, each programmed move takes either the **TM** time (Isx89 is default) if global variable I42 is set to the default value of 0, or the **TA** time (Isx87 is the default) if I42 is set to 1 -- there is no feedrate specification allowed. Each move on each axis is computed as a cubic position trajectory in which the intermediate positions are relaxed somewhat so there are no velocity or acceleration discontinuities in blending the moves together.

If I42 is set to 0, the specified **TM** time has a range of 0 to 4095.9998 milliseconds, with a resolution of ¼-microsecond. If I42 is set to 1, the specified **TA** time has a range of 0 to 8,388,607 milliseconds (about 2 hours, 20 minutes), with a resolution of 1 millisecond.

Before the first move in any series of consecutive moves, a starting move of **TM** or **TA** time is added to blend smoothly from a stop. After the last move in any series of consecutive moves, an ending move of **TM** or **TA** time is added to blend smoothly to a stop.

This command will take the program out of any of the other move modes (**LINEAR**, **CIRCLE**, **PVT**, **RAPID**). The program will stay in this mode until another move mode command is executed.

Examples:

```
RAPID X10 Y10
SPLINE1 TM100
X20 Y15
X32 Y21
X43 Y26
X50 Y30
DWELL100
RAPID X0 Y0
```

See Also:
Cubic Spline Mode (Writing a Motion Program)
Program commands **LINEAR**, **CIRCLE**, **RAPID**, **PVT**, **SPLINE2**, **TA**

SPLINE2

Function: Put program in non-uniform cubic spline motion mode

Type: Motion program (PROG and ROT)

Syntax: **SPLINE2**

This modal command puts the program in non-uniform cubic spline mode. This mode is virtually identical to the **SPLINE1** uniform cubic spline mode described above, except that the **TM** segment time can vary in a continuous spline. This makes **SPLINE2** mode more flexible than **SPLINE1** mode, but it takes slightly more computation time.

Examples:

```
RAPID X10 Y10
SPLINE2
X20 Y15 TA100
X32 Y21 TA120
X43 Y26 TA87
X50 Y30 TA62
DWELL100
RAPID X0 Y0
```

See Also:
Cubic Spline Mode (Writing a Motion Program)
Program commands **LINEAR**, **CIRCLE**, **RAPID**, **PVT**, **SPLINE1**, **TA**

STOP

Function: Stop program execution

Type: Motion program (PROG)

Syntax: **STOP**

This command suspends program execution, whether started by run or step, keeping the program counter pointing to the next line in the program, so that execution may be resumed with a run or step command.

Examples:

```
A10 B10
A20 B0
STOP
A0 B0
```

See Also:

On-line commands **<CONTROL-Q>**, **Q**, **R**, **S**

Program commands **BLOCKSTART**, **BLOCKSTOP**

T{data}

Function: Tool Select Code (T-Code)

Type: Motion program

Syntax: **T{data}**

where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to

Turbo PMAC interprets this statement as a **CALL 10n2.({data'}*1000)** command, where n is the hundreds'digit of **{data},** and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n2, and the specified line label. (Programs 10n2 are usually used to implement the machine codes as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

This structure permits the implementation of customizable T-Code routines for machine-tool style applications by the writing of subroutines in motion programs 10n2. Arguments can be passed to these subroutines by following the T-Code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have T-codes only in the range 0-99, which permits the use of PROG 1002 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Examples:

T01 jumps to **N1000** of PROG 1002
T12 jumps to **N12000** of PROG 1002
T115 jumps to **N15000** of PROG 1012

See Also:

Program commands **CALL{data}, D{data}, M{data}, T{data}, RETURN**

TA{data}

Function: Set Acceleration Time

Type: Motion program (PROG and ROT)

Syntax: **TA{data}**

where:

- **{data}** is a constant or expression representing the acceleration time in milliseconds

This statement specifies the commanded acceleration time between blended moves (**LINEAR** and **CIRCLE** mode), and from and to a stop for these moves. If global variable I42 is set to 1, it also specifies the segment time for **PVT** and **SPLINE** mode moves. The units are milliseconds, and the range is 0 to 8,388,607 msec (about 2 hours, 20 minutes). *Turbo PMAC will round the specified value to the nearest integer number of milliseconds when executing this command* (no rounding is done in storing the value in the buffer).

Note:

If the coordinate system is not in segmentation mode (Isx13 = 0), make sure the specified acceleration time (TA or 2*TS) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters (Ixx17). A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time in this case should be **TA1 TS0**.

If the specified S-curve time (from TS, or Isx88) is greater than half the TA time, the time used for the acceleration for blended moves will be twice the specified S-curve time.

The acceleration time is also the minimum time for a blended move; if the distance on a feedrate-specified (**F**) move is so short that the calculated move time is less than the acceleration time, or the time of a time-specified (**TM**) move is less than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move. If **TA** controls the move time it must be greater than the Isx13 time and the I8 period.

Note:

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration *rate* (Ixx17) for a programmed **LINEAR** mode move with Isx13=0 (no move segmentation).

A move executed in a program before any **TA** statement will use the default acceleration time specified by coordinate system I-variable Isx87.

Examples:

```
TA100
TA(P20)
TA(45.3+SQRT(Q10))
```

See Also:

Linear, Circular Blended Moves, Cubic Spline Moves, PVT Moves (Writing a Motion Program)
I-variables Ixx17, Isx87, Isx88
Program commands **TS**, **TM**

TINIT

Function: Initialize selected transformation matrix

Type: Motion program (PROG and ROT)

Syntax: **TINIT**

This command initializes the currently selected (with **TSEL**) transformation matrix for the coordinate system by setting it to the identity matrix. This makes the rotation angle 0, the scaling 1, and the displacement 0, so the XYZ points for the coordinate system are as the axis definition statements created them. Turbo PMAC will still perform the matrix calculations, even though they have no effect. **TSEL0** should be used to stop the matrix calculations

The matrix can subsequently be changed with the **ADIS**, **IDIS**, **AROT**, and **IROT** commands.

Examples:

```
TSEL 4.............  ; Select transformation matrix 4
TINIT..............  ; Initialize it to the identity matrix
IROT 71............  ; Do incremental rotation/scaling with Q71-Q79
```

See Also:

Axis Matrix Transformations (Writing a Motion Program)

On-line command **DEFINE TBUF**

Program commands **TSEL**, **ADIS**, **IDIS**, **AROT**, **IROT**

TM{data}

Function: Set Move Time

Type: Motion program

Syntax: **TM{data}**

where:

- **{data}** is a floating-point constant or expression representing the move time in milliseconds.

This command establishes the time to be taken by subsequent **LINEAR** and **CIRCLE**, mode moves. If global variable I42 is set to the default value of 0, it also established the time to be taken by subsequent **SPLINE** and **PVT** mode moves. It overrides any previous **TM** or **F** statement, and is overridden by any subsequent **TM** or **F** statement. It is irrelevant in **RAPID** move mode (or in **SPLINE** or **PVT** move mode if I42 is set to 1), but the latest value will stay active through that mode for the next return to **TM**-controlled move modes.

For **LINEAR** and **CIRCLE** mode moves, the effective move time can range from 0.5 msec to 2^{23} msec (about 2 hours 20 minutes), with floating-point resolution. For **SPLINE** and **PVT** mode moves, the effective move time can range from 1 servo cycle to 4096 msec (about 4 seconds), with resolution of ¼-microsecond.

There are several cases in which Turbo PMAC will not use the move time commanded with the **TM** statement:

- If the specified move time is over the maximum -- 2^{23} msec for **LINEAR** or **CIRCLE** moves, 4096 msec for **SPLINE** and **PVT** moves – the move will be executed in the maximum allowed time, at a *higher* speed than what was programmed.
- If the move time programmed with the **TM** statement causes any motor in the coordinate system to exceed its Ixx16 velocity limit (when active), all axes will be slowed so that no motor exceeds its limit.

- If the specified move time is less than the acceleration time currently in force (**TA** or 2***TS** or the time set by the Ixx17 limit when active), the move will take the full acceleration time instead, and the axes will move more slowly than specified by the **TM** command. If the acceleration time is 0, the minimum permitted move time is 0.5 msec.
- If the % override value for the coordinate system is at other than %100 when the move is executed, the move will not execute in the specified time. The actual move time varies in inverse proportion to the % value.

If **TM** controls the move time it must be greater than the Isx13 time and the I8 period. Otherwise, the program can fail on a run-time error because the calculations for the next move may not be completed in time.

Examples:
```
TM30
TM47.635
TM(P1/3)
```

See Also:
Linear and Circular Blended Moves (Writing a Motion Program)
I-variables I8, Ixx16, Ixx17, Isx13, Isx89
Program commands **F**, **TA**, **TS**, **LINEAR**, **CIRCLE**

TR{data}

Function: Set 3D-comp tool-shaft radius magnitude

Type: Motion program (PROG and ROT)

Syntax: **TR{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the magnitude of the tool's shaft radius

This statement sets the magnitude of the radius of the cutting tool's shaft for three-dimensional cutter-radius compensation, expressed in the user units of the X, Y, and Z axes. It is used in conjunction with the cutter's end radius declared by the **CCR** statement. This function is often part of the **D** tool data used in the machine-tool standard RS-274 (G) code.

The default value of the tool-shaft radius at power-up/reset is zero. If the tool-shaft radius declared with the **TR** statement is less than the cutter-end radius set with the **CCR** statement, the **CCR** value will be used instead for the tool-shaft radius.

In operation of 3D compensation, after an offset of the cutter-end radius along the surface-normal vector, there is a second offset of the tool-shaft radius minus the cutter-end radius, perpendicular to the tool-orientation vector.

The tool-shaft value declared with the TR statement affects all subsequent moves with 3D compensation active, until another tool-shaft radius value is declared.

Examples:
```
TR1.356
TR(Q10)
```

See Also:
Three-Dimensional Compensation
Program commands **CC3**, **CCR{data}**, **NX{data}**, **NY{data}**, **NZ{data}**, **TX{data}**, **TY{data}**, **TZ{data}**

TS{data}

Function: Set S-Curve Acceleration Time

Type: Motion program (PROG and ROT)

Syntax: **TS{data}**

where:

- **{data}** is a positive constant or expression representing the S-curve time in milliseconds.

This command specifies the time, at both the beginning and end of the total acceleration time, in **LINEAR** and **CIRCLE** mode blended moves that is spent in S-curve acceleration. The units are milliseconds. Turbo PMAC will round the specified value to the nearest integer number of milliseconds when executing this command (no rounding is done when storing the value in the buffer).

If **TS** is zero, the acceleration is constant throughout the **TA** time and the velocity profile is trapezoidal.

If **TS** is greater than zero, the acceleration will start at zero and linearly increase through **TS** time, then stay constant (for time **TC**) until **TA-TS** time, and linearly decrease to zero at **TA** time (that is, **TA=2TS+TC**).

If **TS** is equal to **TA/2**, the entire acceleration will be spent in S-curve form (**TS** values greater than **TA/2** override the **TA** value; total acceleration time will be **2TS**.

Note:

For **LINEAR** mode moves with Turbo PMAC not in segmentation mode (Isx13=0), if the rate of acceleration for any motor in the coordinate system exceeds that motor's maximum as specified by Ixx17, the acceleration time for all motors is increased so that no motor exceeds its maximum acceleration rate.

TS does not affect **RAPID**, **PVT**, or **SPLINE** mode moves, but it stays valid for the next return to blended moves.

Note:

If the coordinate system is not in segmentation mode (Isx13 = 0), make sure the specified acceleration time (TA or 2*TS) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters (Ixx17). A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time in this case should be **TA1 TS0**.

In executing the **TS** command, Turbo PMAC rounds the specified value to the nearest integer number of milliseconds (there is no rounding done when storing the command in the buffer).

A blended move executed in a program before any **TS** statement will use the default S-curve time specified by coordinate system I-variable Isx88.

Examples:

```
TS20
TS(Q17)
TS(39.32+P43)
```

See Also:

Linear and Circular Blended Moves (Writing a Motion Program)
I-variables Isx13, Ixx17, Ixx21, Isx87, Isx88
Program commands **TA**, **TM**, **F**, **LINEAR**, **CIRCLE**

TSELECT{constant}

Function: Select active transformation matrix for X, Y, and Z axes

Type: Motion program (PROG and ROT)

Syntax: **TSELECT{constant}**
TSEL{constant}

where:

- **{constant}** is an integer representing the number of the matrix to be used

This command selects the specified matrix for use as the active transformation matrix for the X, Y, and Z axes of the coordinate system running the motion program. This matrix can then be modified using the **TINIT**, **ADIS**, **AROT**, **IDIS**, and **IROT** commands to perform translations, rotations, and scaling of the 3 axes. This matrix will be used until another one is selected.

This matrix must already have been created with the on-line **DEFINE TBUF** command. That command specifies the number of matrices to create, and it must have specified a number at least as high as the number used in **TSEL** (you cannot select a matrix that has not been created).

TSEL0 deselects all transformation matrices, saving calculation time.

Examples:

```
DEFINE TBUF 5                    ; Create 5 transformation matrices
OPEN PROG 10 CLEAR
...
TSEL 3.....                      ; Select transformation matrix 3 (of 5)
TINIT.....                       ; Make matrix 3 the identity matrix
```

See Also:

Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **AROT**, **IROT**, **ADIS**, **IDIS**, **TINIT**

TX{data}

Function: Set 3D-comp tool-orientation vector X-component

Type: Motion program (PROG and ROT)

Syntax: **TX{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the X-component of the tool-orientation vector

This statement specifies the X-component of the tool-orientation vector used for three-dimensional cutter-radius compensation. This value is used along with the Y and Z-components specified by the **TY{data}** and **TZ{data}** statements, respectively, to compute the orientation of the vector.

The total magnitude of the tool-orientation vector specified with these three components does not matter, although a unit-magnitude vector is typically specified. The relative magnitudes (including signs) of the three components are what determine the orientation of the vector. The direction sense of the vector does not matter – base to tip, or tip to base. Generally, all three components be declared together on one line; if only one or two components are declared, the other(s) are left at their old value(s), possibly leading to unpredictable results.

The tool-orientation vector affects the compensation for the end-point of the move commanded on the same line as the tool-orientation vector. It also affects the compensation for subsequent moves until another tool-orientation vector is declared. In typical use, a new tool-orientation vector is declared with each move, so the vector only affects the move on the same line in this case.

Examples:

```
X3.76 Y8.29 Z-4.83 A34.32 C-29.75 NX0.866 NY0 NZ-0.5 TX-0.707 TY0.707 TZ0
```

See Also:

Three-Dimensional Compensation

Program commands **CC3**, **CCR{data}**, **NX{data}**, **NY{data}**, **NZ{data}**, **TR{data}**, **TY{data}**, **TZ{data}**

TY{data}

Function: Set 3D-comp tool-orientation vector Y-component

Type: Motion program (PROG and ROT)

Syntax: **TY{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the Y-component of the tool-orientation vector

This statement specifies the Y-component of the tool-orientation vector used for three-dimensional cutter-radius compensation. This value is used along with the X and Z-components specified by the **TX{data}** and **TZ{data}** statements, respectively, to compute the orientation of the vector.

The total magnitude of the tool-orientation vector specified with these three components does not matter, although a unit-magnitude vector is typically specified. The relative magnitudes (including signs) of the three components are what determine the orientation of the vector. The direction sense of the vector does not matter – base to tip, or tip to base. Generally, all three components be declared together on one line; if only one or two components are declared, the other(s) are left at their old value(s), possibly leading to unpredictable results.

The tool-orientation vector affects the compensation for the end-point of the move commanded on the same line as the tool-orientation vector. It also affects the compensation for subsequent moves until another tool-orientation vector is declared. In typical use, a new tool-orientation vector is declared with each move, so the vector only affects the move on the same line in this case.

Examples:

```
X3.76 Y8.29 Z-4.83 A34.32 C-29.75 NX0.866 NY0 NZ-0.5 TX-0.707 TY0.707 TZ0
```

See Also:

Three-Dimensional Compensation

Program commands **CC3**, **CCR{data}**, **NX{data}**, **NY{data}**, **NZ{data}**, **TR{data}**, **TX{data}**, **TZ{data}**

TZ{data}

Function: Set 3D-comp tool-orientation vector Z-component

Type: Motion program (PROG and ROT)

Syntax: **TZ{data}**

where:

- **{data}** is a signed floating-point constant or expression representing the Z-component of the tool-orientation vector

This statement specifies the Z-component of the tool-orientation vector used for three-dimensional cutter-radius compensation. This value is used along with the X and Y-components specified by the **TX{data}** and **TY{data}** statements, respectively, to compute the orientation of the vector.

The total magnitude of the tool-orientation vector specified with these three components does not matter, although a unit-magnitude vector is typically specified. The relative magnitudes (including signs) of the three components are what determine the orientation of the vector. The direction sense of the vector does not matter – base to tip, or tip to base. Generally, all three components be declared together on one line; if only one or two components are declared, the other(s) are left at their old value(s), possibly leading to unpredictable results.

The tool-orientation vector affects the compensation for the end-point of the move commanded on the same line as the tool-orientation vector. It also affects the compensation for subsequent moves until another tool-orientation vector is declared. In typical use, a new tool-orientation vector is declared with each move, so the vector only affects the move on the same line in this case.

Examples:

```
X3.76 Y8.29 Z-4.83 A34.32 C-29.75 NX0.866 NY0 NZ-0.5 TX-0.707 TY0.707 TZ0
```

See Also:

Three-Dimensional Compensation

Program commands **CC3**, **CCR{data}**, **NX{data}**, **NY{data}**, **NZ{data}**, **TR{data}**, **TX{data}**, **TY{data}**

U{data}

Function: U-Axis Move

Type: Motion program

Syntax: **U{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the U-axis.

This command causes a move of the U-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
U10
U(P17+2.345)
X20 U20
U(COS(Q10)) V(SIN(Q10))
G77 U40
```

See Also:

Program commands **{axis}{data}**, **A**, **B**, **C**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**

UNLOCK{constant}

Function: Clear process locking bit

Type: Motion programs & PLC programs

Syntax: **UNLOCK{constant}**

where:

- **{constant}** is an integer from 0 to 7 representing the number of the locking bit

The **UNLOCK** command permits the user to clear one of the 8 process locking bits in Turbo PMAC, releasing the process for other tasks. These locking bits can prevent conflicts between tasks of different priorities attempting to manipulate the same register. On-line commands and PLCs 1 –31 are background tasks; motion programs and PLC 0 are higher-priority foreground tasks.

The user can check the status of a locking bit with the **LOCK** command.

The status of locking bits 0 – 7 is reported as bits 4 – 11, respectively, of I4904.

If no motion program buffer or PLC program buffer is open when this command is issued, this command will be executed immediately as an on-line command.

Examples:

```
P10=1                    ; Assume locked to start
WHILE(P10=1)             ; Loop until unlocked
  LOCK4,P10              ; Check status of locking bit 4
ENDWHILE
M1=M1^1                  ; Invert Machine Output 1
UNLOCK4                  ; Release process 4 for other tasks
```

V{data}

Function: V-Axis Move

Type: Motion program (PROG and ROT)

Syntax: **V{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the V-axis.

This command causes a move of the V-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
V20
U56.5 V(P320)
Y10 V10
V(SQRT(Q20*Q20+Q21*Q21))
CALL300 V(Q400)
```

See Also:

Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **W**, **X**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**

W{data}

Function: W-Axis Move

Type: Motion program

Syntax: **W{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the W-axis.

This command causes a move of the W-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
W5
W(P10+33.5)
Z10 W10
W(ABS(Q22*Q22))
M50 W1
```

See Also:

Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **X**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**

WAIT

Function: Suspend program execution

Type: Motion program (PROG and ROT)

Syntax: **WAIT**

This command may be used on the same line as a **WHILE** condition to hold up execution of the program until the condition goes false. When the condition goes false, program execution resumes on the next line. Turbo PMAC will not blend moves through a true **WHILE…WAIT** loop.

Use of the **WAIT** statement allows indefinite pauses without the need for repeated use of a servo command (e.g. **DWELL** or **DELAY**) to "eat up" the time. However, it is impossible to predict how long the pause will be.

WAIT permits a faster resumption of the program upon the **WHILE** condition going false. Also, the program timer is halted when **WAIT**ing, which allows the "in-position" bit to go true (which can be used to trigger an action, or the next move).

Since Turbo PMAC executes a **WHILE ({condition}) WAIT** statement every Real Time Interrupt until the condition goes false, it is essentially the same as a PLC0. This could use excessive processor time and in severe cases trip the watchdog timer on Turbo PMAC's that simultaneously run several motion programs that use **WAIT** statements and or large PLC0 programs. For example, if the condition only needs to be checked every 20 msec and not every Real Time Interrupt, you should consider using a **DWELL** command to regulate the execution time of the **WHILE** loop.

Examples:

```
WHILE ({condition})
      DWELL20
ENDW

WHILE (M11=0) WAIT          ; Pause here until Machine Input 1 set
WHILE (M187=0) WAIT         ; Pause here until all axes in-position
M1=1 .....                  ; Turn on Output 1 to activate punch
```

See Also:
I-variable Ixx28
Program commands **DWELL**, **DELAY**, **STOP**

WHILE({condition})

Function: Conditional looping

Type: Motion program (PROG only); PLC program

Syntax:
```
WHILE ({condition})
WHILE ({condition}) {action}
```

where:

- **{condition}** consists of one or more sets of **{expression}** **{comparator}** **{expression}**, joined by logical operators **AND** or **OR**.
- **{action}** is a program command

This statement allows repeated execution of a statement or series of statements as long as the condition is true. It is Turbo PMAC's only looping construct. It can take two forms:

(Valid in motion program only) With a statement following on the same line, it will repeatedly execute that statement as long as the condition is true. No **ENDWHILE** is used to terminate the loop.

```
WHILE ({condition}) {action}
```

(Valid in motion and PLC programs) With no statement following on the same line, it will execute statements on subsequent lines down to the next **ENDWHILE** statement.

```
WHILE ({condition})
      {statement}
      [{statement}
      ...]
ENDWHILE
```

If a **WHILE** loop in a motion program has no move, **DWELL**, or **DELAY** inside, Turbo PMAC will attempt to execute the loop twice (while true) each real-time interrupt cycle (stopped from more loops only by the "double-jump-back" rule), much like a PLC0. This can starve the background tasks for time, possibly even tripping the watchdog timer. Turbo PMAC will not attempt to blend moves through such an "empty" **WHILE** loop if it finds the loop condition true twice or more. A **WHILE…WAIT** loop will operate this way on a single jump back.

In PLC programs, extended compound **WHILE** conditions can be formed on multiple program lines through use of **AND** and **OR** commands on the program lines immediately following the **WHILE** command itself (this structure is not available in motion programs). Conditions in each program line can be either simple or compound. **AND** and **OR** operations within a program line take precedence over **AND** and **OR** operations between lines.

Examples:

```
WHILE (P20=0)
   ...
ENDWHILE

WHILE (Q10<5 AND Q11>1)
   ...
ENDWHILE

WHILE (M11=0) WAIT                  ; sit until input goes true
INC
WHILE (M11=0 OR M12=0) X100         ; increment until  2 inputs true
```

To do the equivalent of a For/Next loop:

```
P1=0 .....                    ; Initialize loop counter
WHILE (P1<10)                 ; Loop until counter exceeds limit
    X1000.....                ; Perform action to be repeated
    P1=P1+1.                  ; Increment loop counter
ENDWHILE.....                 ; Loop back
```

To do a timed wait in a PLC program, use the servo cycle counter as timer

```
P90=16777216                  ; Counter rollover value (2^24)
P91=M0.....                   ; Store starting value of M0 (X:$0) counter
P92=0.....                    ; Time elapsed so far
WHILE (P92<P93)               ; Loop until past specified time
   P92=(M0-P91)%P90           ; Calculate time elapsed
.....                         ; Modulo (%) operation to handle rollover
ENDWHILE.....                 ; Loop back
```

To do extended compound conditions in a PLC program

```
WHILE (M11=1 AND M12=1)
OR (M13=1 AND M14=1)
AND (P1>0)
   ...
ENDWHILE
```

See Also:
Program Logic (Writing a Motion Program, Writing a PLC Program)
How Turbo PMAC Executes a Motion Program (Writing a Motion Program)
Program commands **AND**, **OR**, **IF**, **ELSE**, **ENDIF**, **ENDWHILE**

X{data}

Function: X-Axis Move

Type: Motion program

Syntax: **X{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the X-axis.

This command causes a move of the X-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
X10
X15 Y20
X(P1) Y30
X(Q10*COS(Q1)) Y(Q10*SIN(Q1))
X3.76 Z2.92 I0.075 K3.42
CALL100 X5 Y10
```

See Also:
Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **W**, **Y**, **Z**, **CALL**, **PRELUDE**, **READ**

Y{data}

Function: Y-Axis Move

Type: Motion program

Syntax: **Y{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the Y-axis.

This command causes a move of the Y-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
Y50
Y(P100)
X35 Y75
Y-0.221 Z3.475
Y(ABS(P3+P4)) A(INT(P3-P4))
G73 Y20
```

See Also:

Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **W**, **X**, **Z**, **CALL**, **PRELUDE**, **READ**

Z{data}

Function: Z-Axis Move

Type: Motion program

Syntax: **Z{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the Z-axis.

This command causes a move of the Z-axis. (See **{axis}{data}** description, above.) If it follows a subroutine call (explicit or with **PRELUDE**) on a program line, it can be used instead to pass its value to the subroutine through use of the **READ** command.

Examples:

```
Z20
Z(Q25)
X10 Y20 Z30
Z23.4 R10.5
Z(P301+2*P302/P303)
```

See Also:

Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **W**, **X**, **Y**, **CALL**, **PRELUDE**, **READ**

TURBO PMAC MEMORY AND I/O MAP

Program (Machine Code) Memory

P:$000000 - $00FFFF	Firmware
P:$040000 - $0407FF	User-Written Servo
P:$040800 - $040BFF	User-Written Phase
P:$050000 - P$05BFFF	Compiled PLCs (Standard Program Memory Configuration)
P:$050000 - P$0BBFFF	Compiled PLCs (Extended Program Memory Configuration)

Global Servo Registers

X:$000000	Servo interrupt cycle counter (servo cycles)
Y:$000000	Servo count for next real time interrupt
X:$000001	Filtered velocity pointer
Y:$000001	Real time interrupt period minus one (I8)
X:$000002	Data gathering counter (time)
Y:$000002	Data gathering period (I5051)
X:$000003	Phase cycle counter (time)
Y:$000003	Phase cycle period (I7)
X:$000004	DPRAM real-time update counter (time)
Y:$000004	DPRAM real-time update period (I19)
X:$000005	DPRAM background update counter (time)
Y:$000005	DPRAM background update period (from I19)
X:$000006	Global Status Register

(First word returned on **???** command. See **???** in the On-Line Commands section)

0	This Card Addressed Serially
1	All Cards Addressed Serially
2	(Reserved for future use)
3	No hardware clocks found
4	MACRO Ring Check Error
5	MACRO Auxiliary Communications Error
6	TWS Variable Parity Error
7	(Internal use)
8	PLCC L-variable uses illegal M-variable def
9	Real-Time Interrupt Warning
10	Flash Read Error
11	DPRAM Error
12	Firmware Checksum Error
13	Any Memory Checksum Error
14	Compensation On
15	Watchdog Timer
16	(Internal use)
17	Gather on external trigger
18	(Reserved for future use)
19	Data gathering function on
20	Servo Error
21	CPU Type Bit 1
22	Real Time Interrupt Re-entry
23	(Reserved for future use)

Y:$000006 Global Status Register

(Second word returned on **???** command)

0-7	(Reserved for future use)
8-10	(Internal use)
11	Fixed Buffer Full
12-15	(Internal use)
16	3U Turbo System
17	PLC Buffer Open
18	ASCII Rotary Buffer Open
19	Motion Program Buffer Open
20	Binary Rotary Buffer Open
21	CPU Type Bit 0
22	VME Board
23	Ultralite Board

D:$000007 - $00000A	Temporary storage
X:$00000B	Last real-time interrupt execution time (CPU cycles / 2)
Y:$00000B	Longest real-time interrupt execution time (CPU cycles / 2)
D:$00000C - $000016	Temporary storage
D:$000017	Real-time counter (1/1024 sec since power-up/reset, or if Opt 18B is present, since 12am 1/1/2000)
D:$000018 - $000023	Temporary storage
X:$000022	Last background cycle time (CPU cycles/2)
Y:$000022	Longest background cycle time (CPU cycles/2)
X:$000024	P-Buffer Base Address
Y:$000024	Display Buffer Pointer
X:$000025	Minimum watchdog timer count
Y:$000025	Watchdog timer counter value
D:$000026	Global I-variables
X:$000028	Servo interrupt time (I10)
X:$000030	Synchronous M-variable buffer size (minus 1)
X:$00003B	Maximum compensation table number
Y:$00003B	Maximum active coordinate system number
X:$00003C	Maximum commutated motor number
Y:$00003C	Maximum servoed motor number
X:$00003E	PLCC End Address
Y:$00003E	Program Buffer End Address
X:$00003F	Battery-Backed RAM End Address
Y:$00003F	Dual-Ported RAM End Address

Temporary Stack Registers

X/Y:$000040 - $00007F Temporary Stack registers

Motor Registers

Motor #	1	2	3	4	5	6	7	8
Address	$00008x	$00010x	$00018x	$00020x	$00028x	$00030x	$00038x	$00040x
Motor #	9	10	11	12	13	14	15	16
Address	$00048x	$00050x	$00058x	$00060x	$00068x	$00070x	$00078x	$00080x
Motor #	17	18	19	20	21	22	23	24
Address	$00088x	$00090x	$00098x	$000A0x	$000A8x	$000B0x	$000B8x	$000C0x
Motor #	25	26	27	28	29	30	31	32
Address	$000C8x	$000D0x	$000D8x	$000E0x	$000E8x	$000F0x	$000F8x	$00100x

D:$000x80/00 Motor time left in move (X-register units msec*2 at %100)
D:$000x81/01 Motor present desired jerk residual
D:$000x82/02 Motor present desired jerk (dA/dt)
D:$000x83/03 Motor present desired acceleration residual
D:$000x84/04 Motor present desired acceleration (X-register units 6/[Ixx08*32] cts/msec^2 at %100; Y is fractional)
X:$000x85/05 Motor present desired velocity residual
Y:$000x85/05 Motor backlash size
D:$000x86/06 Motor present desired velocity (X-register units 3/[Ixx08*32] cts/msec at %100; Y is fractional)
D:$000x87/07 Motor present desired position residual
D:$000x88/08 Motor present desired position (1/[Ixx08*32] counts)
X:$000x89/09 Motor position feedback pointer (Ixx03)
Y:$000x89/09 Motor position scaling factor (Ixx08)
X:$000x8A/0A Motor master (handwheel) register pointer (Ixx05)
Y:$000x8A/0A Motor previous actual source position value
D:$000x8B/0B Motor present actual position (1/[Ixx08*32] counts)
X:$000x8C/0C Motor master (handwheel) scale factor (Ixx07)
Y:$000x8C/0C Motor previous master (handwheel) source position (1/32 ct)
D:$000x8D/0D Motor present master (handwheel) position (1/[Ixx07*32]cts of the master or 1/[Ixx08*32]cts of the slaved motor).
X:$000x8E/0E Motor feedpot (time base) pointer
Y:$000x8E/0E Motor servo extension timer
X:$000x8F/0F Motor PID Velocity Feedforward gain (Ixx32); Motor ESA S0 gain (Iyy10/60)
Y:$000x8F/0F Motor servo-loop closure extension (Ixx60)

Motor #	1	2	3	4	5	6	7	8
Address	$00009x	$00011x	$00019x	$00021x	$00029x	$00031x	$00039x	$00041x
Motor #	9	10	11	12	13	14	15	16
Address	$00049x	$00051x	$00059x	$00061x	$00069x	$00071x	$00079x	$00081x
Motor #	17	18	19	20	21	22	23	24
Address	$00089x	$00091x	$00099x	$000A1x	$000A9x	$000B1x	$000B9x	$000C1x
Motor #	25	26	27	28	29	30	31	32
Address	$000C9x	$000D1x	$000D9x	$000E1x	$000E9x	$000F1x	$000F9x	$00101x

D:$000x90/10	Motor compensation position correction (1/[Ixx08*32] cts)
D:$000x91/11	Motor following error (1/[Ixx08*32] cts)
X:$000x92/12	Motor PID derivative gain (Ixx31); ESA T0 gain (Iyy20/70)
Y:$000x92/12	Motor PID intermediate value; ESA S1 gain (Iyy11/61)
X:$000x93/13	Motor PID intermediate value; ESA UT1 term
Y:$000x93/13	Motor PID acceleration feedforward gain (Ixx35); Motor ESA T1 gain (Iyy21/71)
X:$000x94/14	Motor ESA UT2 term
Y:$000x94/14	Motor ESA T2 gain (Iyy22/72)
X:$000x95/15	Motor ESA UT3 term
Y:$000x95/15	Motor ESA T3 gain (Iyy23/73)
X:$000x96/16	Motor ESA UT4 term
Y:$000x96/16	Motor ESA T4 gain (Iyy24/74)
X:$000x97/17	Motor ESA UR1 term
Y:$000x97/17	Motor ESA R1 gain (Iyy16/66)
X:$000x98/18	Motor ESA UR2 term
Y:$000x98/18	Motor ESA R2 gain (Iyy17/67)
X:$000x99/19	Motor ESA UR3 term
Y:$000x99/19	Motor PID proportional gain (Ixx30); ESA R3 gain (Iyy18/68)
X:$000x9A/1A	Motor PID previous desired velocity; ESA UR4 term
Y:$000x9A/1A	Motor PID deadband gain (Ixx64); ESA R4 gain (Iyy19/69)
X:$000x9B/1B	Motor velocity-loop position address (Ixx04)
Y:$000x9B/1B	Motor PID position error limit (Ixx67); Motor ESA TS scale gain (Iyy25/75)
X:$000x9C/1C	Motor PID deadband size (Ixx65); ESA F0 gain (Iyy12/62)
Y:$000x9C/1C	Motor velocity-loop scale factor (Ixx09)
X:$000x9D/1D	Motor present actual velocity (unfiltered) (1/[Ixx09*32]cts / [Ixx60+1] servo cycles)
Y:$000x9D/1D	Motor previous velocity-loop source position
D:$000x9E/1E	Motor PID integrated position error; Motor ESA previous velocity position
X:$000x9F/1F	Motor PID integral gain (Ixx33); ESA F1 gain (Iyy13/63)
Y:$000x9F/1F	Motor PID integration limit (Ixx63); ESA G0 gain (Iyy36/86)

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address	$0000Ax	$00012x	$0001Ax	$00022x	$0002Ax	$00032x	$0003Ax	$00042x
Motor #	9	10	11	12	13	14	15	16
Address	$0004Ax	$00052x	$0005Ax	$00062x	$0006Ax	$00072x	$0007Ax	$00082x
Motor #	17	18	19	20	21	22	23	24
Address	$0008Ax	$00092x	$0009Ax	$000A2x	$000AAx	$000B2x	$000BAx	$000C2x
Motor #	25	26	27	28	29	30	31	32
Address	$000CAx	$000D2x	$000DAx	$000E2x	$000EAx	$000F2x	$000FAx	$00102x

X:$000xA0/20	Motor PID integrated error residual; ESA UG1 term
Y:$000xA0/20	Motor PID N1 notch filter gain (Ixx36); Motor ESA G1 gain (Iyy37/87)
X:$000xA1/21	Motor PID intermediate term; ESA UG2 term
Y:$000xA1/21	Motor PID N2 notch filter gain (Ixx37); Motor ESA G2 gain (Iyy38/88)

X:$000xA2/22	Motor PID intermediate term; ESA UD1 term
Y:$000xA2/22	Motor PID D1 notch filter gain (Ixx38); Motor ESA D1 gain (Iyy34/84)
X:$000xA3/23	Motor PID intermediate term; ESA UD2 term
Y:$000xA3/23	Motor PID D2 notch filter gain (Ixx39); Motor ESA D2 gain (Iyy35/85)
X:$000xA4/24	Motor ESA GS scale gain (Iyy39/89)
Y:$000xA4/24	Motor ESA H0 gain (Iyy14/64)
D:$000xA5/25	Motor previous net commanded position (1/[Ixx08*32] cts)
X:$000xA6/26	Motor ESA H1 gain (Iyy15/65)
Y:$000xA6/26	Motor ESA K0 gain (Iyy29/79)
X:$000xA7/27	Motor ESA UK1 term
Y:$000xA7/27	Motor ESA K1 gain (Iyy30/80)
X:$000xA8/28	Motor ESA UK2 term
Y:$000xA8/28	Motor ESA K2 gain (Iyy31/81)
X:$000xA9/29	Motor ESA UK3 term
Y:$000xA9/29	Motor ESA K3 gain (Iyy32/82)
X:$000XAA/2A	Motor ESA UL1 term
Y:$000xAA/2A	Motor ESA L1 gain (Iyy26/76)
X:$000xAB/2B	Motor ESA UL2 term
Y:$000xAB/2B	Motor ESA L2 gain (Iyy27/77)
X:$000xAC/2C	Motor ESA UL3 term
Y:$000xAC/2C	Motor ESA L3 gain (Iyy28/78)
X:$000xAD/2D	Motor ESA KS scale gain (Iyy33/83)
Y:$000xAD/2D	Motor jog move delay time (Ixx92)
X:$000xAE/2E	Motor PID filter result; ESA previous command output
Y:$000xAE/2E	Motor PID command output limit (Ixx69); Motor ESA output scale factor (Ixx69)
X:$000xAF/2F	Motor flag address (Ixx25)
Y:$000xAF/2F	Motor flag control bits (Ixx24)

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address	$0000Bx	$00013x	$0001Bx	$00023x	$0002Bx	$00033x	$0003Bx	$00043x
Motor #	9	10	11	12	13	14	15	16
Address	$0004Bx	$00053x	$0005Bx	$00063x	$0006Bx	$00073x	$0007Bx	$00083x
Motor #	17	18	19	20	21	22	23	24
Address	$0008Bx	$00093x	$0009Bx	$000A3x	$000ABx	$000B3x	$000BBx	$000C3x
Motor #	25	26	27	28	29	30	31	32
Address	$000CBx	$000D3x	$000DBx	$000E3x	$000EBx	$000F3x	$000FBx	$00103x

X:$000xB0/30 Motor servo status/control bits

(First word returned on **?** command. See Y:$000xC0/40 for second word)

(Refer to **?** description in on-line commands for detailed description of bits)

0	Rapid move maximum velocity select (Ixx90)
1	Alternate command output mode (Ixx96)
2	Software-capture enable (Ixx97 bit 0)
3	Trigger-on-error enable (Ixx97 bit 1)
4	Position-following (master) enable (Ixx06 bit 0)
5	Position-following (master) offset mode (Ixx06 bit 1)

	6	Commutation enable (Ixx01 bit 0)
	7	Y-address commutation encoder (Ixx01 bit 1)
	8	User-written servo enable (Ixx59 bit 0)
	9	User-written phase enable (Ixx59 bit 1)
	10	Home search in progress
	11	Block request
	12	Abort/limit deceleration in progress
	13	Desired velocity zero
	14	Data block error
	15	Dwell in progress
	16	Integrate only at zero desired velocity (Ixx34)
	17	Move timer active
	18	Open loop mode
	19	Amplifier enabled
	20	Extended servo algorithm enable (Iyy00/Iyy50)
	21	Positive end limit set (soft or hard)
	22	Negative end limit set (soft or hard)
	23	Motor activated (Ixx00)
Y:$000xB0/30		Motor phase address pointer (Ixx83)
X:$000xB1/31		Motor block execution pointer
Y:$000xB1/31		Motor friction feedforward gain (Ixx68)
X:$000xB2/32		Motor number of commutation cycles (Ixx70)
Y:$000xB2/32		Motor previous phase position
X:$000xB3/33		Motor B phase bias (Ixx79)
Y:$000xB3/33		Motor A phase bias (Ixx29)
D:$000xB4/34		Motor present phase position (X-register units: counts*Ixx70)
X:$000xB5/35		Motor counts per Ixx70 commutation cycles (Ixx71)
Y:$000xB5/35		Motor phase offset (Ixx72)
X:$000xB6/36		Motor ADC mask word (Ixx84)
Y:$000xB6/36		Motor ADC address (Ixx82)
X:$000xB7/37		Motor slip gain (Ixx78)
Y:$000xB7/37		Motor estimated rotor magnetization current
X:$000xB8/38		Motor commanded quadrature current (servo command+bias)
Y:$000xB8/38		Motor commanded quadrature current (0 for microstep)
X:$000xB9/39		Motor actual quadrature current
Y:$000xB9/39		Motor actual direct current
X:$000xBA/3A		Motor current-loop back path proportional gain (Ixx76)
Y:$000xBA/3A		Motor current-loop forward path proportional gain (Ixx62)
X:$000xBB/3B		Motor commutation table pointer
Y:$000xBB/3B		Motor current-loop integral gain (Ixx61)
X:$000xBC/3C		Motor quadrature current loop integrator output
Y:$000xBC/3C		Motor direct current loop integrator output
X:$000xBD/3D		Motor commutation table pointer offset (Ixx55)
Y:$000xBD/3D		Motor PWM scale factor (Ixx66)
X:$000xBE/3E		Motor command output address (Ixx02)
Y:$000xBE/3E		Motor delay compensation gain (Ixx56)
X:$000xBF/3F		Motor (Quadrature/torque) command value (includes bias)
Y:$000xBF/3F		Motor (Quadrature/torque) command bias (from TCOMP table)

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address	$0000Cx	$00014x	$0001Cx	$00024x	$0002Cx	$00034x	$0003Cx	$00044x
Motor #	9	10	11	12	13	14	15	16
Address	$0004Cx	$00054x	$0005Cx	$00064x	$0006Cx	$00074x	$0007Cx	$00084x
Motor #	17	18	19	20	21	22	23	24
Address	$0008Cx	$00094x	$0009Cx	$000A4x	$000ACx	$000B4x	$000BCx	$000C4x
Motor #	25	26	27	28	29	30	31	32
Address	$000CCx	$000D4x	$000DCx	$000E4x	$000ECx	$000F4x	$000FCx	$00104x

`X:$000xC0/40` Motor fatal following error limit (Ixx11)

`Y:$000xC0/40` Motor status bits

(Second word returned on **?** command. See `X:$000xB0/30` for first word)

(Refer to **?** specification in On-line Commands for detailed description of bit meanings)

0	Background-check in-position
1	Warning following error limit exceeded
2	Fatal following error limit exceeded
3	Amplifier fault error
4	Backlash direction flag
5	I^2T Amplifier fault error
6	Integrated fatal following error
7	Trigger move
8	Phasing reference error
9	Phasing reference in progress
10	Home complete
11	Stopped on actual position limit
12	Stopped on desired position limit
13	Foreground-check in-position
14	(Reserved for future use)
15	Assigned to C.S.
16-19	Coordinate definition (1=A, 2=B, 3=C, 4=UVW, 5=I, 7=XYZ)
20-23	Number of C.S. defined in (-1)

`L:$000xC1/41` Motor jog speed (Ixx22) [floating point]

`L:$000xC2/42` Motor jog max. acceleration (Ixx19) [floating point]

`X:$000xC3/43` Motor jog/home S-curve time (Ixx21)

`Y:$000xC3/43` Motor jog/home acceleration time (Ixx20)

`L:$000xC4/44` Motor limit/abort acceleration (Ixx15) [floating point]

`X:$000xC5/45` Motor in-position band (Ixx28)

`Y:$000xC5/45` Motor warning following error limit (Ixx12)

`D:$000xC6/46` Motor target velocity

`D:$000xC7/47` Motor target (& pre-jog) position (1/[Ix08*32] cts)

`D:$000xC8/48` Motor delta target position (1/[Ix08*32] cts)

`L:$000xC9/49` Motor delta position (floating point)

`L:$000xCA/4A` Motor maximum velocity (Ixx16, cts/msec, floating point)

`L:$000xCB/4B` Motor maximum acceleration (Ixx17, cts/msec^2, floating point)

`D:$000xCC/4C` Motor position bias (1/[Ixx08*32] cts)

`X:$000xCD/4D` Motor home offset position (Ixx26)

`Y:$000xCD/4D` Motor I-variable bits

0 – 3	Power-on Mode (Ixx80)

4 – 23 (Reserved for future use)

`X:$000xCE/4E` Motor phase-finding parameters

0 – 7 Phase-finding time (Ixx74)

8 – 23 Phase-finding torque (Ixx73)

`Y:$000xCE/4E` Motor encoder home capture position offset (counts)

`L:$000xCF/4F` Motor X/U/A/B/C-Axis coefficient (floating-point)

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address	$0000Dx	$00015x	$0001Dx	$00025x	$0002Dx	$00035x	$0003Dx	$00045x
Motor #	9	10	11	12	13	14	15	16
Address	$0004Dx	$00055x	$0005Dx	$00065x	$0006Dx	$00075x	$0007Dx	$00085x
Motor #	17	18	19	20	21	22	23	24
Address	$0008Dx	$00095x	$0009Dx	$000A5x	$000ADx	$000B5x	$000BDx	$000C5x
Motor #	25	26	27	28	29	30	31	32
Address	$000CDx	$000D5x	$000DDx	$000E5x	$000EDx	$000F5x	$000FDx	$00105x

`L:$000xD0/50` Motor Y/V-Axis coefficient (floating point)

`L:$000xD1/51` Motor Z/W-Axis coefficient (floating point)

`L:$000xD2/52` Motor axis offset (floating point)

`X:$000xD3/53` Motor in-position scan counter

`Y:$000xD3/53` Motor in-position number of scans (Ixx88)

`X:$000xD4/54` Motor power-on phase position offset (Ixx75)

`Y:$000xD4/54` Motor multiple resolver gear ratios (Ixx98, Ixx99)

`X:$000xD5/55` Motor power-on phase position format (Ixx91)

`Y:$000xD5/55` Motor power-on phase position address (Ixx81)

`X:$000xD6/56` Motor power-on servo position format (Ixx95)

`Y:$000xD6/56` Motor power-on servo position address (Ixx10)

`D:$000xD7/57` Motor variable jog position/distance (1/[Ixx08*32] cts)

`X:$000xD8/58` Motor I^2T time value

`Y:$000xD8/58` Motor trigger capture position

`X:$000xD9/59` Motor continuous current limit (Ixx57)

`Y:$000xD9/59` Motor integrated current limit (Ixx58)

`D:$000xDA/5A` Motor integrated current value (X-reg units of Ixx58)

`L:$000xDB/5B` Motor homing speed (cts/msec; floating point)

`D:$000xDC/5C` Motor positive software position limit (Ixx13, cts)

`D:$000xDD/5D` Motor negative software position limit (Ixx14, cts)

`D:$000xDE/5E` Motor position rollover range (Ixx27, cts)

`X:$000xDF/5F` Motor backlash slew rate (Ixx85)

`Y:$000xDF/5F` Motor backlash hysteresis (Ixx87)

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address	$0000Ex	$00016x	$0001Ex	$00026x	$0002Ex	$00036x	$0003Ex	$00046x
Motor #	9	10	11	12	13	14	15	16
Address	$0004Ex	$00056x	$0005Ex	$00066x	$0006Ex	$00076x	$0007Ex	$00086x
Motor #	17	18	19	20	21	22	23	24
Address	$0008Ex	$00096x	$0009Ex	$000A6x	$000AEx	$000B6x	$000BEx	$000C6x
Motor #	25	26	27	28	29	30	31	32
Address	$000CEx	$000D6x	$000DEx	$000E6x	$000EEx	$000F6x	$000FEx	$00106x

`D:$000xE0/60` Motor backlash saved position

```
X:$000xE1/61              Motor backlash size (Ixx86)
Y:$000xE1/61              Motor present backlash
X:$000xE2/62              Motor software position limit offset (Ixx41)
D:$000xE3/63 - $000xEA/6A (Reserved for future use)
D:$000xEB/6B - $000xEE/6E Motor spline table
D:$000xEF/6F              Motor filtered velocity
```

Motor #	1	2	3	4	5	6	7	8
Address	$0000Fx	$00017x	$0001Fx	$00027x	$0002Fx	$00037x	$0003Fx	$00047x
Motor #	9	10	11	12	13	14	15	16
Address	$0004Fx	$00057x	$0005Fx	$00067x	$0006Fx	$00077x	$0007Fx	$00087x
Motor #	17	18	19	20	21	22	23	24
Address	$0008Fx	$00097x	$0009Fx	$000A7x	$000AFx	$000B7x	$000BFx	$000C7x
Motor #	25	26	27	28	29	30	31	32
Address	$000CFx	$000D7x	$000DFx	$000E7x	$000EFx	$000F7x	$000FFx	$00107x

```
D:$000xF0/70 - $000xFF/7F Motor position history for filtered velocity
```

Global Registers

```
Y:$001080 - $0010CF 80-character display buffer
X:$001080           Running software checksum value (frozen if any error)
X:$001083           VME bus address modifier (I90)
X:$001084           VME bus address modifier "don't care" bits (I91)
X:$001085           VME bus base address bits A31-A24 (I92)
X:$001086           VME bus mailbox base address bits A23-A16 /
                    ISA bus DPRAM base address bits A23-A16 (I93)
X:$001087           VME bus mailbox base address bits A15-A08 /
                    ISA bus DPRAM base address bits A15-A14 & Enable (I94)
X:$001088           VME bus interrupt level (I95)
X:$001089           VME bus interrupt vector (I96)
X:$00108A           VME bus DPRAM base address bits A23-A20 (I97)
X:$00108B           VME bus DPRAM enable (I98)
X:$00108C           VME bus address width control (I99)
X:$00108D           Baud-rate / card-address word (I54 / I0)
X:$00108E           Multiplexed A/D converter ring size (I5060)
X:$00108F           Control-character reporting motor set (I59)
X:$001090           User program reference checksum value (set on CLOSE)
X:$001091           Rotary buffer interrupt start point (I16)
X:$001092           Rotary buffer interrupt stop point (I17)
X:$001093           Fixed buffer full point (I18)
X:$001094           Data gathering mask 1 word (I5049)
X:$001095           Data gathering mask 2 word (I5050)
X:$001096 – X:$0010C5 Data gathering addresses (I5001 – I5048)
Y:$0010D0           Display setup register
Y:$0010D1           Display setup register
X/Y:$0010F0 - $0010FF Open registers (set to 0 on power-up/reset)
```

Communication Buffers

```
X/Y:$001900 - $0019FF Foreground Program Command/Response Buffers
X/Y:$001A00 - $001AFF Background PLC Command/Response Buffers
X/Y:$001B00 - $001BFF DPRAM Command/Response Buffers
X/Y:$001C00 - $001CFF Auxiliary Serial Command/Response Buffers
X/Y:$001D00 - $001DFF MACRO Master-to-Master Command/Response Buffers
X/Y:$001E00 - $001FFF Synchronous M-variable queue (non-lookahead)
                      (>= V1.939 only; at X/Y:$003600 - $0037FF in older)
X/Y:$001E00 - $001EFF Main Serial Command/Response Buffers
                      (<= V1.938 only; at X/Y:$003600 - $0036FF in newer)
X/Y:$001F00 - $001FFF Bus Command/Response Buffers
                      (<= V1.938 only; at X/Y:$003700 - $0037FF in newer)
```

Coordinate System Registers

C. S. #	1	2	3	4	5	6	7	8
Address	$00200x	$00210x	$00220x	$00230x	$00240x	$00250x	$00260x	$00270x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00280x	$00290x	$002A0x	$002B0x	$002C0x	$002D0x	$002E0x	$002F0x

```
X:$002x00             C.S. commanded time base value
Y:$002x00             C.S. Q-variable starting address
X:$002x01             C.S. time base source address (Isx93)
Y:$002x01             C.S. time base active address (from Isx93 or feed hold)
X:$002x02             C.S. actual time base value (units of I10)
Y:$002x02             C.S. actual time base slew rate (from Isx94 or Isx95)
X:$002x03             C.S. feed hold slew rate (Isx95)
Y:$002x03             C.S. time base slew rate (Isx94)
D:$002x04             C.S. motor definition word
X:$002x05             C.S. S-curve time (Isx88, TS)
Y:$002x05             C.S. acceleration time (Isx87, TA)
L:$002x06             C.S. feedrate / move time (Isx89, F, TM)
L:$002x07             C.S. velocity time units (Isx90)
D:$002x08             C.S. PRELUDE command buffer
X:$002x09             C.S. coordinate transformation matrix number
Y:$002x09             C.S. PRELUDE command stack pointer
X/Y:$002x0A - $002x0E C.S. execution pointers
L:$002x0F             C.S. cutter compensation radius (from CCR)
```

C. S. #	1	2	3	4	5	6	7	8
Address	$00201x	$00211x	$00221x	$00231x	$00241x	$00251x	$00261x	$00271x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00281x	$00291x	$002A1x	$002B1x	$002C1x	$002D1x	$002E1x	$002F1x

```
L:$002x10             C.S. maximum feedrate (Isx98)
L:$002x12             C.S. alternate feedrate (Isx86)
L:$002x12             C.S. circle radius error limit (Isx96)
L:$002x13             C.S. cutter compensation break angle cosine (Isx99)
X:$002x14             C.S. move calculation time
Y:$002x14             C.S. run-time error code
```

= 0: No run-time error
= 1: Insufficient calculation time
= 2: Program counter out of range (too low)
= 3: Program counter out of range (too high)
= 4: Unlinked conditional statement
= 5: Subroutine stack overflow
= 6: Jump to non-existent label
= 7: Cutter compensation interference error
= 8: Forward kinematic execution error
= 9: Inverse kinematic execution error
= 10: No axes remaining in C.S.

`X:$002x15` C.S. user countdown timer 1 (Isx11)
`Y:$002x15` C.S. user countdown timer 2 (Isx12)
`X/Y:$002x16 - $002x1F` (Reserved for future use)

C. S. #	1	2	3	4	5	6	7	8
Address	$00202x	$00212x	$00222x	$00232x	$00242x	$00252x	$00262x	$00272x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00282x	$00292x	$002A2x	$002B2x	$002C2x	$002D2x	$002E2x	$002F2x

`X/Y:$002x20 - $002x2F` (Reserved for future use)

C. S. #	1	2	3	4	5	6	7	8
Address	$00203x	$00213x	$00223x	$00233x	$00243x	$00253x	$00263x	$00273x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00283x	$00293x	$002A3x	$002B3x	$002C3x	$002D3x	$002E3x	$002F3x

`X/Y:$002x30 - $002x34` (Reserved for future use)
`L:$002x35` C.S. minimum arc angle (Isx97)
`X/Y:$002x36 - $002x3D` Lookahead internal registers
`X:$002x3E` C.S. default program number (Isx91)
`Y:$002x3E` C.S. I-variables

0	Kinematics enable (Isx50)
1-19	Coordinate definition (0=none, 1=A, 2=B, 3=C, 4=UVW, 5=I, 7=XYZ)
20	Line-by-line step mode (Isx53)
21,22	(Reserved for future use)
23	Blend disable (Isx92)

`X:$002x3F` C.S. program execution address
`Y:$002x3F` C.S. program execution status
(Second word returned on **??** command. See `X:$002x40` for first word)
(Refer to **??** specification in On-Line Commands for detailed description of bit meanings)

0	CIRCLE/SPLINE move mode
1	CCW CIRCLE/RAPID move mode
2	Cutter compensation on
3	Cutter compensation left
4	PVT/SPLINE move mode
5	Segmented move stop request
6	Segmented move acceleration in progress
7	Segmented move in progress
8	Pre Jog move flag

9	Cutter compensation move buffered
10	Cutter compensation move stop request
11	Cutter compensation outside corner
12	Dwell move buffered
13	Synchronous M-variable one-shot
14	End-of-block (/) stop in progress
15	Delayed calculation flag
16	Rotary buffer full
17	In-position (logical AND of motor bits)
18	Warning following error (logical OR of motor bits)
19	Fatal following error (logical OR of motor bits)
20	Amplifier fault error (logical OR of motor bits)
21	Circle radius error
22	Run time error (error code in Y:$002x14)
23	Lookahead in progress

C. S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Address	$00204x	$00214x	$00224x	$00234x	$00244x	$00254x	$00264x	$00274x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00284x	$00294x	$002A4x	$002B4x	$002C4x	$002D4x	$002E4x	$002F4x

`X:$002x40` C.S. Status/Control Bits

(First word returned on **??** command. See `Y:$002x3F` for first word.)

(Refer to **??** specification in On-Line Commands for detailed bit meanings)

0	Program running
1	Single step mode
2	Continuous-motion mode
3	Move-specified-by-time mode
4	Continuous-motion request
5	Radius vector incremental mode
6	A-axis incremental mode
7	A-axis used in feedrate calculations
8	B-axis incremental mode
9	B-axis used in feedrate calculations
10	C-axis incremental mode
11	C-axis used in feedrate calculations
12	U-axis incremental mode
13	U-axis used in feedrate calculations
14	V-axis incremental mode
15	V-axis used in feedrate calculations
16	W-axis incremental mode
17	W-axis used in feedrate calculations
18	X-axis incremental mode
19	X-axis used in feedrate calculations
20	Y-axis incremental mode
21	Y-axis used in feedrate calculations
22	Z-axis incremental mode
23	Z-axis used in feedrate calculations

`Y:$002x40` C.S. Internal Status/Control Bits

(Third word returned on **??** command. See `Y:$002x3F` for second word.)

(Refer to **??** specification in On-Line Commands for detailed bit meanings)

0	In-program PMATCH command
1	Stopped on desired-position limit
2	Program start error
3	Circle radius error
4 – 7	(Reserved for future use)
	(Bits 8-11 form Isx21)
8	Lookahead reversal in progress
9	Lookahead function running
10	Lookahead buffer active
11	Lookahead state-change command
12	Lookahead buffer last move
13	Lookahead buffer flush
14	Lookahead buffer recalculate
15	Lookahead buffer last segment
16	Lookahead buffer change
17	Lookahead buffer stop
18	Lookahead buffer direction
19	Lookahead synchronous M-variable overflow
20	Lookahead synchronous M-variable assignment
21	Lookahead buffer end
22	Lookahead buffer active
23	Lookahead buffer wrap

`L:$002x41` C.S. A-Axis desired move position (floating-point)
`L:$002x42` C.S. B-Axis desired move position (floating-point)
`L:$002x43` C.S. C-Axis desired move position (floating-point)
`L:$002x44` C.S. U-Axis desired move position (floating-point)
`L:$002x45` C.S. V-Axis desired move position (floating-point)
`L:$002x46` C.S. W-Axis desired move position (floating-point)
`L:$002x47` C.S. X-Axis desired move position (floating-point)
`L:$002x48` C.S. Y-Axis desired move position (floating-point)
`L:$002x49` C.S. Z-Axis desired move position (floating-point)
`L:$002x4A` C.S. Normal vector I component (floating-point)
`L:$002x4B` C.S. Normal vector J component (floating-point)
`L:$002x4C` C.S. Normal vector K component (floating-point)
`L:$002x4D - $002x4F` C.S. ABC propagated positions (floating-point)

C. S. #	1	2	3	4	5	6	7	8
Address	$00205x	$00215x	$00225x	$00235x	$00245x	$00255x	$00265x	$00275x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00285x	$00295x	$002A5x	$002B5x	$002C5x	$002D5x	$002E5x	$002F5x

`L:$002x50 - $002x55` C.S. UVWXYZ propagated positions (floating-point)
`L:$002x56 - $002x5E` C.S. ABCUVWXYZ buffered positions (floating-point)
`L:$002x5F` C.S. Arc initial radius (floating-point)

C. S. #	1	2	3	4	5	6	7	8
Address	$00206x	$00216x	$00226x	$00236x	$00246x	$00256x	$00266x	$00276x
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Address	$00286x	$00296x	$002A6x	$002B6x	$002C6x	$002D6x	$002E6x	$002F6x

`L:$002x60 - $002xBF` C.S. Internal calculation registers

C. S. #	1	2	3	4	5	6	7	8
Address	$0020Cx	$0021Cx	$0022Cx	$0023Cx	$0024Cx	$0025Cx	$0026Cx	$0027Cx
C. S. #	9	10	11	12	13	14	15	16
Address	$0028Cx	$0029Cx	$002ACx	$002BCx	$002CCx	$002DCx	$002ECx	$002FCx

L:$002xC0 - $002xFF C.S. Subroutine Stack

Program and Buffer Pointers

X:$003000 1st motion program number (low 16 bits)
and entry status (high 8 bits)
Y:$003000 1st motion program buffer storage address
X:$003001 - $0030DF 2nd to 224th program # and entry status
Y:$003001 - $0030DF 2nd to 224th program buffer storage address
X:$0030E0 - $0030EF C.S.1 – C.S.16 forward kinematic buffer start address
Y:$0030E0 - $0030EF C.S.1 – C.S.16 forward kinematic buffer storage address
X:$0030F0 - $0030FF C.S.1 – C.S.16 inverse kinematic buffer start address
Y:$0030F0 - $0030FF C.S.1 – C.S.16 inverse kinematic buffer storage address
X:$003100 PLC 0 execution address

Bits	
0 – 18	PLC execution address
23	PLC execution error

Y:$003100 PLC 0 buffer storage pointer

Bits	
0 – 18	PLC base address
22	PLC disabled
23	PLC open

X:$003101 - $00311F PLC 1 - 31 execution address
Y:$003101 - $00311F PLC 1 - 31 storage pointer
X:$003120 Data gather buffer start address
Y:$003120 Data gather buffer storage address
X:$003121 - $003130 Lookahead buffer 1 – 16 start address
Y:$003121 - $003130 Lookahead buffer 1 – 16 storage address
X:$003131 - $003140 Cutter comp buffer 1 – 16 start address
Y:$003131 - $003140 Cutter comp buffer 1 – 16 storage address
X:$003141 - $003150 Rotary buffer 1 – 16 start address
Y:$003141 - $003150 Rotary buffer 1 – 16 storage address
X:$003151 Transformation matrix buffer start address
Y:$003151 Transformation matrix buffer storage address
X:$003152 - $003171 Motor 1 – 32 backlash comp table start address
Y:$003152 - $003171 Motor 1 – 32 backlash comp table storage address
X:$003172 - $003191 Motor 1 – 32 torque comp table start address
Y:$003172 - $003191 Motor 1 – 32 torque comp table storage address
X:$003192 - $0031B1 Motor 1 – 32 leadscrew comp table start address
Y:$003192 - $0031B1 Motor 1 – 32 leadscrew comp table storage address
X:$0031B2 End of unreserved RAM address
X:$0031EE MACRO auxiliary communications error number
$003300 - $0033FF Background stack

Processed A/D Registers

Y:$003400 - $00341F	Processed A/D Table Registers (bits 12-23)
Y:$003400	Processed A/D value from I5061 address low ADC
X:$003400	Processed A/D value from I5061 address high ADC
Y:$003401	Processed A/D value from I5062 address low ADC
X:$003401	Processed A/D value from I5062 address high ADC
Y:$00341F	Processed A/D value from I5076 address low ADC
X:$00341F	Processed A/D value from I5076 address high ADC
X:$003430	Total MACRO ring error count (from power-up/reset)
Y:$003430	Latest check-cycle MACRO ring error count
X:$00343E	MACRO ring test time (I80)
Y:$00343E	MACRO ring sync packet count

MACRO Flag Registers

X:$003440 - $00347F MACRO Status Flag Registers

Bit	0 – 10	(Reserved for future use)
	11	Position captured flag
	12	MACRO node reset (power-on or command)
	13	Ring break detected elsewhere
	14	Amplifier enabled at station
	15	Amplifier/node shutdown fault
	16	Home flag input value
	17	Positive limit flag value
	18	Negative limit flag value
	19	User flag value
	20	W flag value
	21	V flag value
	22	U flag value
	23	T flag value

Y:$003440 - $00347F MACRO Command Flag Registers

Bit	0 – 10	(Reserved for future use)
	11	Position capture enable
	12	Node position reset flag
	13	Ring break detected
	14	Amplifier enable
	15 – 23	(Reserved for future use)

Y:$003440	MACRO IC 0 Node 0 Command Flag Register
X:$003440	MACRO IC 0 Node 0 Status Flag Register
Y:$00344F	MACRO IC 0 Node 15 Command Flag Register
X:$00344F	MACRO IC 0 Node 15 Status Flag Register
Y:$003450	MACRO IC 1 Node 0 Command Flag Register
X:$003450	MACRO IC 1 Node 0 Status Flag Register
Y:$00345F	MACRO IC 1 Node 15 Command Flag Register
X:$00345F	MACRO IC 1 Node 15 Status Flag Register
Y:$003460	MACRO IC 2 Node 0 Command Flag Register
X:$003460	MACRO IC 2 Node 0 Status Flag Register

Y:$00346F	MACRO IC 2 Node 15 Command Flag Register
X:$00346F	MACRO IC 2 Node 15 Status Flag Register
Y:$003460	MACRO IC 3 Node 0 Command Flag Register
X:$003460	MACRO IC 3 Node 0 Status Flag Register
Y:$00346F	MACRO IC 3 Node 15 Command Flag Register
X:$00346F	MACRO IC 3 Node 15 Status Flag Register

Encoder Conversion Table Registers

Y:$003501 - $0035C0	Conversion Table Setup Registers (I8000 - I8191)
X:$003501 - $0035C0	Conversion Table Result Registers

Buffer Pointers

X/Y:$003600 - $0036FF	Main Serial Command/Response Buffers (>= V1.939 only: at X/Y:$001E00 - $001EFF in older)
X/Y:$003700 - $0037FF	Bus Command/Response Buffers (>= V1.939 only: at X/Y:$001F00 - $001FFF in older)
X/Y:$003600 - $0037FF	Synchronous M-variable queue (non-lookahead) (<= V1.938 only; at X/Y:$001E00 - $001FFF in newer)

Commutation Sine Table

X:$003800 – $003FFF	Commutation cos table [2^{23}*cos((*address*-$3800)*360°/2048)]
Y:$003800 – $003FFF	Commutation sine table [2^{23}*sin((*address*-$3800)*360°/2048)]

User Variable Registers

L:$004000	Variable M0 definition
L:$004001 - $005FFF	Variables M1 - M8191 definitions
L:$006000	Standard location for P0 (when I46 = 0 or 2)
L:$006001 - $007FFF	Standard location for P1 - P8191 (when I46 = 0 or 2)
L:$006000	Standard location for C.S.1 Q0 (when I46 = 1)
L:$006001 - $007FFF	Standard location for C.S.1 Q1 - Q8191 (when I46 = 1)
L:$008000	Standard location for C.S.1 Q0 (when I46 = 0)
L:$008001 - $009FFF	Standard location for C.S.1 Q1 - Q8191 (when I46 = 0)

User Program and Buffer Storage

X/Y:$006000	Start of User Memory (when I46 = 3)
X/Y:$008000	Start of User Memory (when I46 = 1 or 2)
X/Y:$00A000	Start of User Memory (when I46 = 0)
X/Y:$0107FF	End of User Memory (Option 5C0, standard data memory)
X/Y:$011BFF	End of User Memory (Option 5D0, standard data memory)
X/Y:$01BFFF	End of User Memory (Option 5E0, standard data memory)
X/Y:$023FFF	End of User Memory (Option 5F0, standard data memory)
X/Y:$03FFFF	End of User Memory (Option 5x3, extended data memory)

Battery-Backed RAM Registers (Option 16x required)

X/Y:$050000 - $053FFF	Standard BBRAM Option (32k x 24 mapped as 16k x 48)
X/Y:$050000 - $05FFFF	Extended BBRAM Option (128k x 24 mapped as 64k x 48)
L:$050000 - $051FFF	Alternate location for P0 - P8191 (when I46 = 1 or 3)
L:$052000 - $053FFF	Alternate location for Q0 - Q8191 (when I46 = 2 or 3)

Dual Ported RAM Registers (Option 2x required)

Note:

This listing of dual-ported RAM addresses assumes a Turbo PMAC base address of $060000, which is the base address for DPRAM ICs on a Turbo PMAC board, or on a UMAC CPU board. UMAC accessory boards with DPRAM (such as the ACC-54E USB/Ethernet board) will have different base addresses for their DPRAM IC (most commonly $06C000). Turbo PMAC I24 specifies the base address of the DPRAM IC that can be used for the automatic DPRAM functions and structures shown here (if I24 is 0, the base address is $060000). Variable I4904 shows where DPRAM ICs are present in a UMAC system.

```
X/Y:$060000 - $060FFF Standard DPRAM Option 2 (8k x 16)
                      Mapped to PMAC as 4k x 32, mapped to host as 16k x 8
X/Y:$060000 - $063FFF Extended DPRAM Option 2B (32k x 16)
                      Mapped to PMAC as 16k x 32, mapped to host as 64k x 8
```

Note:

Addresses specified with '0x' prefix are host-computer address offsets from the base address of the DPRAM, as set by I92, I93, I94, and I97.

Motor Data Reporting Buffer Control (used if I48=1 or I57=1)

X:$06001A		Motor foreground data reporting control (used if I48=1)
(0x006A)	Bit 15	0 = Host ready; 1 = Host accessing buffer
Y:$06001B		Servo counter bits 0-15 (from X:$0)
(0x006C)		
X:$06001B		Motor foreground data reporting control (used if I48=1)
(0x006E)	Bit 15	0 = PMAC accessing buffer; 1 = PMAC ready
	Bits 0-7	Servo counter bits 16-23 (from X:$0)
Y:$06001C		Motor data reporting control mask
(0x0070)	Bit 15	0 = Motor 16 not reported; 1 = Motor 16 reported
	Bit 14	0 = Motor 15 not reported; 1 = Motor 15 reported
	…	
	Bit 1	0 = Motor 2 not reported; 1 = Motor 2 reported
	Bit 0	0 = Motor 1 not reported; 1 = Motor 1 reported
X:$06001C		Motor data reporting control mask
(0x0072)	Bit 15	0 = Motor 32 not reported; 1 = Motor 32 reported
	Bit 14	0 = Motor 31 not reported; 1 = Motor 31 reported
	…	
	Bit 1	0 = Motor 18 not reported; 1 = Motor 18 reported
	Bit 0	0 = Motor 17 not reported; 1 = Motor 17 reported

Motor Data Reporting Buffer (Used if I48 = 1 or I57 = 1)

Note: Turbo PMAC 24-bit registers are copied into the low 24 bits of a 32-bit DPRAM word. Bit 23 is sign-extended into bits 24 – 31. Turbo PMAC 48-bit registers are treated as 2 24-bit registers. Each 24-bit register is copied into the low 24 bits of a 32-bit DPRAM word. Bit 23 is sign-extended into bits 24 – 31. The host computer must re-assemble these values into a single fixed-point or floating-point value.

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0074	0x00A4	0x00D4	0x0104	0x0134	0x0164	0x0194	0x01C4
Address	$06001D	$060029	$060035	$060041	$06004D	$060059	$060065	$060071
Motor #	9	10	11	12	13	14	15	16
Host Address	0x01F4	0x0224	0x0254	0x0284	0x02B4	0x02E4	0x0314	0x0344
Address	$06007D	$060089	$060095	$0600A1	$0600AD	$0600B9	$0600C5	$0600D1
Motor #	17	18	19	20	21	22	23	24
Host Address	0x0374	0x03A4	0x03D4	0x0404	0x0434	0x0464	0x0494	0x04C4
Address	$0600DD	$0600E9	$0600F5	$060101	$06010D	$060119	$060125	$060131
Motor #	25	26	27	28	29	30	31	32
Host Address	0x04F4	0x0524	0x0554	0x0584	0x05B4	0x05E4	0x0614	0x0644
Address	$06013D	$060149	$060155	$060161	$06016D	$060179	$060185	$060191

`DP:$060xxx-$060xxx+1` Motor following error (1/[Ixx08*32] cts)
(=CmdPos+MasterPos-CompPos-ActPos)
[from D:$88 + D:$8D + D:$90 – D:$8B, etc.]

Motor #	1	2	3	4	5	6	7	8
Host Address	0x007C	0x00AC	0x00DC	0x010C	0x013C	0x016C	0x019C	0x01CC
Address	$06001F	$06002B	$060037	$060043	$06004F	$06005B	$060067	$060073
Motor #	9	10	11	12	13	14	15	16
Host Address	0x01FC	0x022C	0x025C	0x028C	0x02BC	0x02EC	0x031C	0x034C
Address	$06007F	$06008B	$060097	$0600A3	$0600AF	$0600BB	$0600C7	$0600D3
Motor #	17	18	19	20	21	22	23	24
Host Address	0x037C	0x03AC	0x03DC	0x040C	0x043C	0x046C	0x049C	0x04CC
Address	$0600DF	$0600EB	$0600F7	$060103	$06010F	$06011B	$060127	$060133
Motor #	25	26	27	28	29	30	31	32
Host Address	0x04FC	0x052C	0x055C	0x058C	0x05BC	0x05EC	0x061C	0x064C
Address	$06013F	$06014B	$060157	$060163	$06016F	$06017B	$060187	$060193

`DP:$060xxx` Motor servo command (upper 16 bits have units of 16-bit DAC)
[from X:$BF, etc.]

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0080	0x00B0	0x00E0	0x0110	0x0140	0x0170	0x01A0	0x01D0
Address	$060020	$06002C	$060038	$060044	$060050	$06005C	$060068	$060074
Motor #	9	10	11	12	13	14	15	16
Host Address	0x0200	0x0230	0x0260	0x0290	0x02C0	0x02F0	0x0320	0x0350
Address	$060080	$06008C	$060098	$0600A4	$0600B0	$0600BC	$0600C8	$0600D4
Motor #	17	18	19	20	21	22	23	24
Host Address	0x0380	0x03B0	0x03E0	0x0410	0x0440	0x0470	0x04A0	0x04D0
Address	$0600E0	$0600EC	$0600F8	$060104	$060110	$06011C	$060128	$060134
Motor #	25	26	27	28	29	30	31	32
Host Address	0x0500	0x0530	0x0560	0x0590	0x05C0	0x05F0	0x0620	0x0650
Address	$060140	$06014C	$060158	$060164	$060170	$06017C	$060188	$060194

`DP:$060xxx` Motor servo status (individual bits – see $ description)
[from X:$B0, etc]

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0084	0x00B4	0x00E4	0x0114	0x0144	0x0174	0x01A4	0x01D0
Address	$060021	$06002D	$060039	$060045	$060051	$06005D	$060069	$060075
Motor #	9	10	11	12	13	14	15	16
Host Address	0x0204	0x0234	0x0264	0x0294	0x02C4	0x02F4	0x0324	0x0354
Address	$060081	$06008D	$060099	$0600A5	$0600B1	$0600BD	$0600C9	$0600D5
Motor #	17	18	19	20	21	22	23	24
Host Address	0x0384	0x03B4	0x03E4	0x0414	0x0444	0x0474	0x04A4	0x04D4
Address	$0600E1	$0600ED	$0600F9	$060105	$060111	$06011D	$060129	$060135
Motor #	25	26	27	28	29	30	31	32
Host Address	0x0504	0x0534	0x0564	0x0594	0x05C4	0x05F4	0x0624	0x0654
Address	$060141	$06014D	$060159	$060165	$060171	$06017D	$060189	$060195

`DP:$060xxx` Motor general status (individual bits – see $ description)
[from Y:$C0, etc]

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0088	0x00B8	0x00E8	0x0118	0x0148	0x0178	0x01A8	0x01D8
Address	$060022	$06002E	$06003A	$060046	$060052	$06005E	$06006A	$060076
Motor #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Address	0x0208	0x0238	0x0268	0x0298	0x02C8	0x02F8	0x0328	0x0358
Address	$060082	$06008E	$06009A	$0600A6	$0600B2	$0600BE	$0600CA	$0600D6
Motor #	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Host Address	0x0388	0x03B8	0x03E8	0x0418	0x0448	0x0478	0x04A8	0x04D8
Address	$0600E2	$0600EE	$0600FA	$060106	$060112	$06011E	$06012A	$060136
Motor #	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Host Address	0x0508	0x0538	0x0568	0x0598	0x05C8	0x05F8	0x0628	0x0658
Address	$060142	$06014E	$06015A	$060166	$060172	$06017E	$06018A	$060196

`DP:$060xxx-$060xxx+1` Motor position bias (1/[Ixx08*32] cts)
[from D:$CC, etc]

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0090	0x00C0	0x00F0	0x0120	0x0150	0x0180	0x01B0	0x01E0
Address	$060024	$060030	$06003C	$060048	$060054	$060060	$06006C	$060078
Motor #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Address	0x0210	0x0240	0x0270	0x02A0	0x02D0	0x0300	0x0330	0x0360
Address	$060084	$060090	$06009C	$0600A8	$0600B4	$0600C0	$0600CC	$0600D8
Motor #	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Host Address	0x0390	0x03C0	0x03F0	0x0420	0x0450	0x0480	0x04B0	0x04E0
Address	$0600E4	$0600F0	$0600FC	$060108	$060114	$060120	$06012C	$060138
Motor #	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Host Address	0x0510	0x0540	0x0570	0x05A0	0x05D0	0x0600	0x0630	0x0660
Address	$060144	$060150	$06015C	$060168	$060174	$060180	$06018C	$060198

`DP:$060xxx` Motor filtered actual velocity (1/[Ixx09*32] cts/servo cycle)
[from D:$EF, etc.]

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x0094	0x00C4	0x00F4	0x0124	0x0154	0x0184	0x01B4	0x01E4
Address	$060025	$060031	$06003D	$060049	$060055	$060061	$06006D	$060079
Motor #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Address	0x0214	0x0244	0x0274	0x02A4	0x02D4	0x0304	0x0334	0x0364
Address	$060085	$060091	$06009D	$0600A9	$0600B5	$0600C1	$0600CD	$0600D9
Motor #	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Host Address	0x0394	0x03C4	0x03F4	0x0424	0x0454	0x0484	0x04B4	0x04E4
Address	$0600E5	$0600F1	$0600FD	$060109	$060115	$060121	$06012D	$060139
Motor #	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Host Address	0x0514	0x0544	0x0574	0x05A4	0x05D4	0x0604	0x0634	0x0664
Address	$060145	$060151	$06015B	$060169	$060175	$060181	$06018D	$060199

`DP:$060xxx-$060xxx+1` Motor master position (1/[Ixx08*32] cts)
[from D:$8D, etc.]

Motor #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x009C	0x00CC	0x00FC	0x012C	0x015C	0x018C	0x01BC	0x01EC
Address	$060027	$060033	$06003F	$06004B	$060057	$060063	$06006F	$06007B
Motor #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Address	0x021C	0x024C	0x027C	0x02AC	0x02DC	0x030C	0x033C	0x036C
Address	$060087	$060093	$06009F	$0600AB	$0600B7	$0600C3	$0600CF	$0600DB
Motor #	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Host Address	0x039C	0x03CC	0x03FC	0x042C	0x045C	0x048C	0x04BC	0x04EC
Address	$0600E7	$0600F3	$0600FF	$06010B	$060117	$060123	$06012F	$06013B
Motor #	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Host Address	0x051C	0x054C	0x057C	0x05AC	0x05DC	0x060C	0x063C	0x066C
Address	$060147	$060153	$06015F	$06016B	$060177	$060183	$06018F	$06019B

`DP:$060xxx-$060xxx+1` Motor net actual position (1/[Ixx08*32] cts)
(=ActPos+BiasPos+CompPos-MasterPos*OffsetModeBit)
[from D:$8B + D:$CC + D:$90 – D:$8D*X:$B0.5, etc.]

Background Data Reporting Buffer Control (used if I49 = 1 or I57 = 1)

Y:$06019D		Background data reporting control
(0x0674)	Bits 0-4	Number of coordinate systems to report
X:$06019D		Background data reporting control
(0x0676)	Bit 15	0 = Host ready; 1 = Host accessing buffer

Global Background Data Reporting Buffer (used if I49 = 1)

Y:$06019E		Background data reporting time stamp
(0x0678)		Servo counter bits 0-15 (from X:$0)
X:$06019E		Background data reporting control & time stamp
(0x067A)	Bit 15	0 = PMAC accessing buffer; 1 = PMAC ready
	Bits 0-7	Servo counter bits 16-23 (from X:$0)
DP:$06019F		JPAN control panel port word (PMAC1)
(0x067C)		JIO port word bits 0 –23 (PMAC2)
DP:$0601A0		JOPTO I/O port word (PMAC1)
(0x0680)		JIO port word bits 24 –31 (PMAC2)
DP:$0601A1		JTHW multiplexer port word
(0x0684)		
DP:$0601A2		Global status word 1 *[from X:$6]*
(0x0688)		
DP:$0601A3		Global status word 2 *[from Y:$6]*
(0x068C)		
DP:$0601A4-$0601A6		(Reserved for future use)
(0x0690 – 0x069A)		

Coordinate System Background Data Reporting Buffer (used if I49 = 1)

Note:

Turbo PMAC 24-bit registers are copied into the low 24 bits of a 32-bit DPRAM word. Bit 23 is sign-extended into bits 24 – 31. Turbo PMAC 48-bit registers are treated as 2 24-bit registers. Each 24-bit register is copied into the low 24 bits of a 32-bit DPRAM word. Bit 23 is sign-extended into bits 24 – 31. The host computer must re-assemble these values into a single fixed-point or floating-point value.

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x069C	0x071C	0x079C	0x081C	0x089C	0x091C	0x099C	0x0A1C
Address	$0601A7	$0601C7	$0601E7	$060207	$060227	$060247	$060267	$060287
C. S. #	9	10	11	12	13	14	15	16
Host Adr	0x0A9C	0x0B1C	0x0B9C	0x0C1C	0x0C9C	0x0D1C	0x0D9C	0x0E1C
Address	$0602A7	$0602C7	$0602E7	$060307	$060327	$060347	$060367	$060387

DP:$060xx7-$060xx8 C.S. programmed feedrate/move time (engrg. vel. units/msec)
[from L:$2006, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06A4	0x0724	0x07A4	0x0824	0x08A4	0x0924	0x09A4	0x0A24
Address	$0601A9	$0601C9	$0601E9	$060209	$060229	$060249	$060269	$060289
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AA4	0x0B24	0x0BA4	0x0C24	0x0CA4	0x0D24	0x0DA4	0x0E24
Address	$0602A9	$0602C9	$0602E9	$060309	$060329	$060349	$060369	$060389

DP:$060xx9 C.S. feedrate override (units of I10)
[from X:$2002, etc]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06A8	0x0728	0x07A8	0x0828	0x08A8	0x0928	0x09A8	0x0A28
Address	$0601AA	$0601CA	$0601EA	$06020A	$06022A	$06024A	$06026A	$06028A
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AA8	0x0B24	0x0BA8	0x0C28	0x0CA8	0x0D28	0x0DA8	0x0E28
Address	$0602AA	$0602CA	$0602EA	$06030A	$06032A	$06034A	$06036A	$06038A

DP:$060xxA C.S. feedrate override source address
[from X:$2001, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06AC	0x072C	0x07AC	0x082C	0x08AC	0x092C	0x09AC	0x0A2C
Address	$0601AB	$0601CB	$0601EB	$06020B	$06022B	$06024B	$06026B	$06028B
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AAC	0x0B2C	0x0BAC	0x0C2C	0x0CAC	0x0D2C	0x0DAC	0x0E2C
Address	$0602AB	$0602CB	$0602EB	$06030B	$06032B	$06034B	$06036B	$06038B

DP:$060xxB-$060xxC C.S. status words (individual bits)
[from D:$2040, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06B4	0x0734	0x07B4	0x0834	0x08B4	0x0934	0x09B4	0x0A34
Address	$0601AD	$0601CD	$0601ED	$06020D	$06022D	$06024D	$06026D	$06028D
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AB4	0x0B34	0x0BB4	0x0C34	0x0CB4	0x0D34	0x0DB4	0x0E34
Address	$0602AD	$0602CD	$0602ED	$06030D	$06032D	$06034D	$06036D	$06038D

DP:$060xxD-$060xxE C.S. A-axis target position (engineering units)
[from L:$2041, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06BC	0x073C	0x07BC	0x083C	0x08BC	0x093C	0x09BC	0x0A3C
Address	$0601AF	$0601CF	$0601EF	$06020F	$06022F	$06024F	$06026F	$06028F
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0ABC	0x0B3C	0x0BBC	0x0C3C	0x0CBC	0x0D3C	0x0DBC	0x0E3C
Address	$0602AF	$0602CF	$0602EF	$06030F	$06032F	$06034F	$06036F	$06038F

DP:$060xxF-$060xy0 C.S. B-axis target position (engineering units)
[from L:$2042, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06C4	0x0744	0x07C4	0x0844	0x08C4	0x0944	0x09C4	0x0A44
Address	$0601B1	$0601D1	$0601F1	$060211	$060231	$060251	$060271	$060291
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AC4	0x0B44	0x0BC4	0x0C44	0x0CC4	0x0D44	0x0DC4	0x0E44
Address	$0602B1	$0602D1	$0602F1	$060311	$060331	$060351	$060371	$060391

DP:$060xx1-$060xx2 C.S. C-axis target position (engineering units)
[from L:$2043, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06CC	0x074C	0x07CC	0x084C	0x08CC	0x094C	0x09CC	0x0A4C
Address	$0601B3	$0601D3	$0601F3	$060213	$060233	$060253	$060273	$060293
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0ACC	0x0B4C	0x0BCC	0x0C4C	0x0CCC	0x0D4C	0x0DCC	0x0E4C
Address	$0602B3	$0602D3	$0602F3	$060313	$060333	$060353	$060373	$060393

DP:$060xx3-$060xx4 C.S. U-axis target position(engineering units)
[from L:$2044, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06D4	0x0754	0x07D4	0x0854	0x08D4	0x0954	0x09D4	0x0A54
Address	$0601B5	$0601D5	$0601F5	$060215	$060235	$060255	$060275	$060295
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AD4	0x0B54	0x0BD4	0x0C54	0x0CD4	0x0D54	0x0DD4	0x0E54
Address	$0602B5	$0602D5	$0602F5	$060315	$060335	$060355	$060375	$060395

DP:$060xx5-$060xx6 C.S. V-axis target position (engineering units)
[from L:$2045, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06DC	0x075C	0x07DC	0x085C	0x08DC	0x095C	0x09DC	0x0A5C
Address	$0601B7	$0601D7	$0601F7	$060217	$060237	$060257	$060277	$060297
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0ADC	0x0B5C	0x0BDC	0x0C5C	0x0CDC	0x0D5C	0x0DDC	0x0E5C
Address	$0602B7	$0602D7	$0602F7	$060317	$060337	$060357	$060377	$060397

DP:$060xx7-$060xx8 C.S. W-axis target position (engineering units)
[from L:$2046, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06E4	0x0764	0x07E4	0x0864	0x08E4	0x0964	0x09E4	0x0A64
Address	$0601B9	$0601D9	$0601F9	$060219	$060239	$060259	$060279	$060299
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AE4	0x0B64	0x0BE4	0x0C64	0x0CE4	0x0D64	0x0DE4	0x0E64
Address	$0602B9	$0602D9	$0602F9	$060319	$060339	$060359	$060379	$060399

DP:$060xx9-$060xxA C.S. X-axis target position (engineering units)
[from L:$2047, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06EC	0x076C	0x07EC	0x086C	0x08EC	0x096C	0x09EC	0x0A6C
Address	$0601BB	$0601DB	$0601FB	$06021B	$06023B	$06025B	$06027B	$06029B
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AEC	0x0B6C	0x0BEC	0x0C6C	0x0CEC	0x0D6C	0x0DEC	0x0E6C
Address	$0602BB	$0602DB	$0602FB	$06031B	$06033B	$06035B	$06037B	$06039B

DP:$060xxB-$060xxC C.S. Y-axis target position (engineering units)
[from L:$2048, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06F4	0x0774	0x07F4	0x0874	0x08F4	0x0974	0x09F4	0x0A74
Address	$0601BD	$0601DD	$0601FD	$06021D	$06023D	$06025D	$06027D	$06029D
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AF4	0x0B74	0x0BF4	0x0C74	0x0CF4	0x0D74	0x0DF4	0x0E74
Address	$0602BD	$0602DD	$0602FD	$06031D	$06033D	$06035D	$06037D	$06039D

DP:$060xxD-$060xxE C.S. Z-axis target position (engineering units)
[from L:$2049, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x06FC	0x077C	0x07FC	0x087C	0x08FC	0x097C	0x09FC	0x0A7C
Address	$0601BF	$0601DF	$0601FF	$06021F	$06023F	$06025F	$06027F	$06029F
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0AFC	0x0B7C	0x0BFC	0x0C7C	0x0CFC	0x0D7C	0x0DFC	0x0E7C
Address	$0602BF	$0602DF	$0602FF	$06031F	$06033F	$06035F	$06037F	$06039F

DP:$060xxF C.S. program status (individual bits)
[from Y:$203F, etc.]

C. S. #	1	2	3	4	5	6	7	8
Host Address	0x0700	0x0780	0x0800	0x0880	0x0900	0x0980	0x0A00	0x0A80
Address	$0601C0	$0601E0	$060200	$060220	$060240	$060260	$060280	$0602A0
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Addess	0x0B00	0x0B80	0x0C00	0x0C80	0x0D00	0x0D80	0x0E00	0x0E80
Address	$0602C0	$0602E0	$060300	$060320	$060340	$060360	$060380	$0603A0

DP:$060xx0 C.S. program lines remaining in buffer

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x0704	0x0784	0x0804	0x0884	0x0904	0x0984	0x0A04	0x0A84
Address	$0601C1	$0601E1	$060201	$060221	$060241	$060261	$060281	$0602A1
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0B04	0x0B84	0x0C04	0x0C84	0x0D04	0x0D84	0x0E04	0x0E84
Address	$0602C1	$0602E1	$060301	$060321	$060341	$060361	$060381	$0603A1

DP:$060xx1 C.S. time remaining in move

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x0708	0x0788	0x0808	0x0888	0x0908	0x0988	0x0A08	0x0A88
Address	$0601C2	$0601E2	$060202	$060222	$060242	$060262	$060282	$0602A2
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0B08	0x0B88	0x0C08	0x0C88	0x0D08	0x0D88	0x0E08	0x0E88
Address	$0602C2	$0602E2	$060302	$060322	$060342	$060362	$060382	$0603A2

DP:$060xx2 C.S. time remaining in accel/decel

C. S. #	1	2	3	4	5	6	7	8
Host Address	0x070C	0x078C	0x080C	0x088C	0x090C	0x098C	0x0A0C	0x0A8C
Address	$0601C3	$0601E3	$060203	$060223	$060243	$060263	$060283	$0602A3
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Address	0x0B0C	0x0B8C	0x0C0C	0x0C8C	0x0D0C	0x0D8C	0x0E0C	0x0E8C
Address	$0602C3	$0602E3	$060303	$060323	$060343	$060363	$060383	$0603A3

DP:$060xx3 C.S. program execution address offset

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x0710	0x0790	0x0810	0x0890	0x0910	0x0990	0x0A10	0x0A90
Address	$0601C4	$0601E4	$060204	$060224	$060244	$060264	$060284	$0602A4
C. S. #	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Host Adr	0x0B10	0x0B90	0x0C10	0x0C90	0x0D10	0x0D90	0x0E10	0x0E90
Address	$0602C4	$0602E4	$060304	$060324	$060344	$060364	$060384	$0603A4

DP:$060xx4-$060xx6 (Reserved for future use)

DPRAM ASCII Buffers (used if I58 = 1)

Y:$0603A7 DPRAM ASCII Command Buffer Control
(0x0E9C) Bit 0 = 1: Data ready from host
X:$0603A7 DPRAM ASCII Command Buffer Control Character (Bits 0 – 7)
(0x0E9E)
DP:$0603A8-$0603CF DPRAM ASCII Command Buffer
(0x0EA0 – 0x0F3E) <= 159 characters, null terminated
Y:$0603D0 DPRAM ASCII Response Buffer Control Characters
(0x0F40) Bits 0 – 7 Response handshake control character
Bits 8 – 14 Response type (0 = host, 1 = CMDR, 2 = SENDR)
Bit 15 (0 = valid command, 1 = error in command)
X:$0603D0 DPRAM ASCII Response Buffer: No. of characters + 1
(0x0F42)
DP:$0603D1-$060410 DPRAM ASCII Response Buffer
(0x0F44 – 0x1040) <= 255 characters, null terminated

Background Variable Read and Write Buffer Control (used if I55 = 1)

`Y:$060411` Background Variable-Read Control
(0x1044) Bit 0 (0 = no new data ready; 1 = new data ready)
`X:$060411` Background Variable-Read Servo Time (from X:$0)
(0x1046)
`Y:$060412` Background Variable-Read Buffer Size
(0x1048)
`X:$060412` Background Variable-Read Buffer Start Address Offset
(0x104A)
`Y:$060413` Background Variable-Write Buffer Size
(0x104C)
`X:$060413` Background Variable-Write Buffer Start Address Offset
(0x104E)

Binary Rotary Program Buffer Control (used after OPEN BIN ROT)

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x1050	0x105C	0x1068	0x1074	0x1080	0x108C	0x1098	0x10A4
Address	$060414	$060417	$06041A	$06041D	$060420	$060423	$060426	$060429
C. S. #	9	10	11	12	13	14	15	16
Host Adr	0x10B0	0x10BC	0x10C8	0x10D4	0x10E0	0x10EC	0x10F8	0x1104
Address	$06042C	$06042F	$060432	$060435	$060438	$06043B	$06043E	$060441

`Y:$0604xx` C.S. Binary Rotary Buffer Status
`X:$0604xx` C.S. Binary Rotary Buffer C.S. # and Enable

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x1054	0x1060	0x106C	0x1078	0x1084	0x1090	0x109C	0x10A8
Address	$060415	$060418	$06041B	$06041E	$060421	$060424	$060427	$06042A
C. S. #	9	10	11	12	13	14	15	16
Host Adr	0x10B4	0x10C0	0x10CC	0x10D8	0x10E4	0x10F0	0x10FC	0x1108
Address	$06042D	$060430	$060433	$060436	$060439	$06043C	$06043F	$060442

`Y:$0604xx` C.S. Binary Rotary Buffer Host Index
`X:$0604xx` C.S. Binary Rotary Buffer PMAC Index

C. S. #	1	2	3	4	5	6	7	8
Host Adr	0x1058	0x1064	0x1070	0x107C	0x1088	0x1094	0x10A0	0x10AC
Address	$060416	$060419	$06041C	$06041F	$060422	$060425	$060428	$06042A
C. S. #	9	10	11	12	13	14	15	16
Host Adr	0x10B8	0x10C4	0x10D0	0x10DC	0x10E8	0x10F4	0x1100	0x110C
Address	$06042E	$060431	$060434	$060437	$06043A	$06043D	$060440	$060443

`Y:$0604xx` C.S. Binary Rotary Buffer Size
`X:$0604xx` C.S. Binary Rotary Buffer Start Address Offset
`DP:$060444-$06044E` (Reserved for future use)
(0x1110 – 0x113A)

Data Gathering Control (used if I5000 = 2 or 3)

Y:$06044F (0x113C)	DPRAM Data Gathering Buffer Size
X:$06044F (0x113E)	DPRAM Data Gathering Buffer Start Address Offset

Variable-Sized Buffers/Open-Use Space

DP:$060450 (0x1140)	DPRAM variable-size buffers start
DP:$060FFF (0x3FFC)	DPRAM end address (Option 2: 8K x 16)
DP:$063FFF (0xFFFC)	DPRAM end address (Option 2B: 32K x 16)

VME Bus/DPRAM Interface Registers

$070000 - $077FFF	Used for VME-bus/DPRAM functions
X:$070006	VME Active Address Modifier (Bits 0-7; from I90)
X:$070007	VME Active Address Modifier Don't Care Bits (Bits 0-7; from I91)
X:$070008	VME Active Base Address Bits A31-A24 (Bits 0-7; from I92)
X:$070009	VME Active Mailbox Base Address Bits A23-A16 ISA Active DPRAM Base Address bits A23-A16 (Bits 0-7; from I93)
X:$07000A	VME Active Mailbox Base Address Bits A15-A08 ISA Active DPRAM Base Address bits A15-A14, Enable & Bank Select (Bits 0-7; from I94)
X:$07000B	VME Active Interrupt Level (Bits 0-7; from I95)
X:$07000C	VME Active Interrupt Vector (Bits 0-7; from I96)
X:$07000D	VME Active DPRAM Base Address Bits A23-A20 (Bits 0-7; from I97)
X:$07000E	VME Active DPRAM Enable State (Bits 0-7; from I98)
X:$07000F	VME Active Address Width Control (Bits 0-7; from I99)
Y:$070000	VME Mailbox Register 0 (Bits 0-7)
Y:$070001	VME Mailbox Register 1 (Bits 0-7)
Y:$070002	VME Mailbox Register 2 (Bits 0-7)
Y:$070003	VME Mailbox Register 3 (Bits 0-7)
Y:$070004	VME Mailbox Register 4 (Bits 0-7)
Y:$070005	VME Mailbox Register 5 (Bits 0-7)
Y:$070006	VME Mailbox Register 6 (Bits 0-7)
Y:$070007	VME Mailbox Register 7 (Bits 0-7)
Y:$070008	VME Mailbox Register 8 (Bits 0-7)
Y:$070009	VME Mailbox Register 9 (Bits 0-7)
Y:$07000A	VME Mailbox Register A (Bits 0-7)
Y:$07000B	VME Mailbox Register B (Bits 0-7)
Y:$07000C	VME Mailbox Register C (Bits 0-7)
Y:$07000D	VME Mailbox Register D (Bits 0-7)
Y:$07000E	VME Mailbox Register E (Bits 0-7)
Y:$07000F	VME Mailbox Register F (Bits 0-7)

Turbo PMAC2 I/O Control Registers

Note:

These registers can be addressed either as X or Y registers)

PC/ISA Bus Turbo PMAC2 Versions (PMAC2-PC, PMAC2-Lite, PMAC2-PC UltraLite):

`X/Y:$070800` I/O Buffer IC Direction Control

Bits:		
	0	OUT0: Buffer direction control for I/O00 to I/O07 on JIO
	1	OUT1: Buffer direction control for I/O08 to I/O15 on JIO
	2	OUT2: Buffer direction control for I/O16 to I/O23 on JIO
	3	OUT3: Buffer direction control for I/O24 to I/O31 on JIO
	4	OUT4: Buffer direction control for DAT0 to DAT7 on JTHW
	5	OUT5: Buffer direction control for SEL0 to SEL7 on JTHW
	6	OUT6: Buffer direction control for DISP0 to DISP7 on JDISP, inverted as R/W- output on JDISP (pin 5)
	7	OUT7: Inverted as E output on JDISP
	16	ACC_FLT8: Channel 8 Accessory Fault
	17	ACC_FLT7: Channel 7 Accessory Fault
	18	ACC_FLT6: Channel 6 Accessory Fault
	19	ACC_FLT5: Channel 5 Accessory Fault
	20	ACC_FLT4: Channel 4 Accessory Fault
	21	ACC_FLT3: Channel 3 Accessory Fault
	22	ACC_FLT2: Channel 2 Accessory Fault
	23	ACC_FLT1: Channel 1 Accessory Fault

`X/Y:$070801` I/O Buffer IC Direction Control

Bits:		
	0	OUT8: Inverted as RS output on JDISP
	1	XIN_1: Jumper E1 input; phase/servo clock direction status
	2	XIN_2: Jumper E2 input (40/60 MHz control)
	3	XIN_3: Jumper E3 input (re-initialize on reset)
	4	XIN_4: Jumper E4 input
	5	XIN_5: Jumper E5 input
	6	XIN_6: Jumper E6 input
	7	XIN_7: Option 12 ADC convert status

VME Bus Turbo PMAC2 Versions (PMAC2-VME, PMAC2-VME UltraLite):

`X/Y:$070800` I/O Buffer IC Direction Control

Bits:		
	0	OUT0: Buffer direction control for I/O00 to I/O07 on JIO
	1	OUT1: Buffer direction control for I/O08 to I/O15 on JIO
	2	OUT2: Buffer direction control for I/O16 to I/O23 on JIO
	3	OUT3: Buffer direction control for I/O24 to I/O31 on JIO
	16	ACC_FLT8: Channel 8 Accessory Fault
	17	ACC_FLT7: Channel 7 Accessory Fault
	18	ACC_FLT6: Channel 6 Accessory Fault
	19	ACC_FLT5: Channel 5 Accessory Fault
	20	ACC_FLT4: Channel 4 Accessory Fault
	21	ACC_FLT3: Channel 3 Accessory Fault
	22	ACC_FLT2: Channel 2 Accessory Fault
	23	ACC_FLT1: Channel 1 Accessory Fault

`X/Y:$070801` I/O Buffer IC Direction Control

Bits:		
	0	OUT8: Inverted as RS output on JDISP
	1	XIN_1: Jumper E1 input; phase/servo clock direction status

2 XIN_2: Jumper E2 input (40/60 MHz control)
3 XIN_3: Jumper E3 input (re-initialize on reset)

`X/Y:$070802` I/O Buffer IC Direction Control
0 OUT4: Buffer direction control for DAT0 to DAT7 on JTHW
1 OUT5: Buffer direction control for SEL0 to SEL7 on JTHW
2 OUT6: Buffer direction control for DISP0 to DISP7 on JDISP, inverted as R/W- output on JDISP (pin 5)
3 OUT7: Inverted as E output on JDISP

`X/Y:$070803` I/O Buffer IC Direction Control
0 XIN_4: Jumper E4 input
1 XIN_5: Jumper E5 input
2 XIN_6: Jumper E6 input
3 XIN_7: Option 12 ADC convert status

PMAC(1)-Style Servo ASIC Registers

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC(1) - 1	PMAC(1) - 2	PMAC(1) - 3	PMAC(1) - 4	PMAC(1) - 5	PMAC(1) - 6	PMAC(1) - 7	PMAC(1) - 8
Address	$078000	$078004	$078008	$07800C	$078100	$078104	$078108	$07810C
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V #1 - 1	24P/V #1 - 2	24P/V #1 - 3	24P/V #1 - 4	24P/V #1 - 5	24P/V #1 - 6	24P/V #1 - 7	24P/V #1 - 8
Address	$078200	$078204	$078208	$07820C	$078300	$078304	$078308	$07830C
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V #2 - 1	24P/V #2 - 2	24P/V #2 - 3	24P/V #2 - 4	24P/V #2 - 5	24P/V #2 - 6	24P/V #2 - 7	24P/V #2 - 8
Address	$079200	$079204	$079208	$07920C	$079300	$079304	$079308	$07930C
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V #3 - 1	24P/V #3 - 2	24P/V #3 - 3	24P/V #3 - 4	24P/V #3 - 5	24P/V #3 - 6	24P/V #3 - 7	24P/V #3 - 8
Address	$07A200	$07A204	$07A208	$07A20C	$07A300	$07A304	$07A308	$07A30C
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V #4 - 1	24P/V #4 - 2	24P/V #4 - 3	24P/V #4 - 4	24P/V #4 - 5	24P/V #4 - 6	24P/V #4 - 7	24P/V #4 - 8
Address	$07B200	$07B204	$07B208	$07B20C	$07B300	$07B304	$07B308	$07B30C

`Y:$07xx0x` Time between last two encoder counts (SCLK cycles)
`X:$07xx0x` Encoder Status/Control Bits
(Bits 0-15: control; Bits 16-23: status)
0-3 Decode control (I7mn0)
4-7 Position capture control (I7mn2)
8-9 Flag select control (I7mn3)
10 Count write enable
11 Compare equal flag latch control
12 Compare-equal output enable
13 EQU output invert enable
14 OUT value (AENAn)
15 Digital delay filter disable (I7mn1)
16 Compare-equal flag
17 Position-captured flag

18	Count error flag
19	Encoder C Channel Status
20	Flag 1 Status (HMFLn)
21	Flag 2 Status (-LIMn)
22	Flag 3 Status (+LIMn)
23	Flag 4 Status (FAULTn)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC(1) - 1	PMAC(1) - 2	PMAC(1) - 3	PMAC(1) - 4	PMAC(1) - 5	PMAC(1) - 6	PMAC(1) - 7	PMAC(1) - 8
Address	$078001	$078005	$078009	$07800D	$078101	$078105	$078109	$07810D
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V #1 - 1	24P/V #1 - 2	24P/V #1 - 3	24P/V #1 - 4	24P/V #1 - 5	24P/V #1 - 6	24P/V #1 - 7	24P/V #1 - 8
Address	$078201	$078205	$078209	$07820D	$078301	$078305	$078309	$07830D
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V #2 - 1	24P/V #2 - 2	24P/V #2 - 3	24P/V #2 - 4	24P/V #2 - 5	24P/V #2 - 6	24P/V #2 - 7	24P/V #2 - 8
Address	$079201	$079205	$079209	$07920D	$079301	$079305	$079309	$07930D
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V #3 - 1	24P/V #3 - 2	24P/V #3 - 3	24P/V #3 - 4	24P/V #3 - 5	24P/V #3 - 6	24P/V #3 - 7	24P/V #3 - 8
Address	$07A201	$07A205	$07A209	$07A20D	$07A301	$07A305	$07A309	$07A30D
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V #4 - 1	24P/V #4 - 2	24P/V #4 - 3	24P/V #4 - 4	24P/V #4 - 5	24P/V #4 - 6	24P/V #4 - 7	24P/V #4 - 8
Address	$07B201	$07B205	$07B209	$07B20D	$07B301	$07B305	$07B309	$07B30D

`Y:$07xx0x` Time since last encoder count (SCLK cycles)
`X:$07xx0x` Encoder phase position (counts)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC(1) - 1	PMAC(1) - 2	PMAC(1) - 3	PMAC(1) - 4	PMAC(1) - 5	PMAC(1) - 6	PMAC(1) - 7	PMAC(1) - 8
Address	$078002	$078006	$07800A	$07800E	$078102	$078106	$07810A	$07810E
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V #1 - 1	24P/V #1 - 2	24P/V #1 - 3	24P/V #1 - 4	24P/V #1 - 5	24P/V #1 - 6	24P/V #1 - 7	24P/V #1 - 8
Address	$078202	$078206	$07820A	$07820E	$078302	$078306	$07830A	$07830E
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V #2 - 1	24P/V #2 - 2	24P/V #2 - 3	24P/V #2 - 4	24P/V #2 - 5	24P/V #2 - 6	24P/V #2 - 7	24P/V #2 - 8
Address	$079202	$079206	$07920A	$07920E	$079302	$079306	$07930A	$07930E
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V #3 - 1	24P/V #3 - 2	24P/V #3 - 3	24P/V #3 - 4	24P/V #3 - 5	24P/V #3 - 6	24P/V #3 - 7	24P/V #3 - 8
Address	$07A202	$07A206	$07A20A	$07A20E	$07A302	$07A306	$07A30A	$07A30E
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V #4 - 1	24P/V #4 - 2	24P/V #4 - 3	24P/V #4 - 4	24P/V #4 - 5	24P/V #4 - 6	24P/V #4 - 7	24P/V #4 - 8
Address	$07B202	$07B206	$07B20A	$07B20E	$07B302	$07B306	$07B30A	$07B30E

`X:$07xx0x` Encoder servo position (2*counts; LSB is direction)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC(1) - 1	PMAC(1) - 2	PMAC(1) - 3	PMAC(1) - 4	PMAC(1) - 5	PMAC(1) - 6	PMAC(1) - 7	PMAC(1) - 8
Address	$078003	$078002	$07800B	$07800A	$078103	$078102	$07810B	$07810A
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V #1 - 1	24P/V #1 - 2	24P/V #1 - 3	24P/V #1 - 4	24P/V #1 - 5	24P/V #1 - 6	24P/V #1 - 7	24P/V #1 - 8
Address	$078203	$078202	$07820B	$07820A	$078303	$078302	$07830B	$07830A
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V #2 - 1	24P/V #2 - 2	24P/V #2 - 3	24P/V #2 - 4	24P/V #2 - 5	24P/V #2 - 6	24P/V #2 - 7	24P/V #2 - 8
Address	$079203	$079202	$07920B	$07920A	$079303	$079302	$07930B	$07930A
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V #3 - 1	24P/V #3 - 2	24P/V #3 - 3	24P/V #3 - 4	24P/V #3 - 5	24P/V #3 - 6	24P/V #3 - 7	24P/V #3 - 8
Address	$07A203	$07A202	$07A20B	$07A20A	$07A303	$07A302	$07A30B	$07A30A
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V #4 - 1	24P/V #4 - 2	24P/V #4 - 3	24P/V #4 - 4	24P/V 4 - 5	24P/V #4 - 6	24P/V #4 - 7	24P/V #4 - 8
Address	$07B203	$07B202	$07B20B	$07B20A	$07B303	$07B302	$07B30B	$07B30A

`Y:$07xx0x` DAC output value (high 16 bits)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC(1) - 1	PMAC(1) - 2	PMAC(1) - 3	PMAC(1) - 4	PMAC(1) - 5	PMAC(1) - 6	PMAC(1) - 7	PMAC(1) - 8
Address	$078006	$078007	$07800E	$07800F	$078106	$078107	$07810E	$07810F
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V #1 - 1	24P/V #1 - 2	24P/V #1 - 3	24P/V #1 - 4	24P/V #1 - 5	24P/V #1 - 6	24P/V #1 - 7	24P/V #1 - 8
Address	$078206	$078207	$07820E	$07820F	$078306	$078307	$07830E	$07830F
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V #2 - 1	24P/V #2 - 2	24P/V #2 - 3	24P/V #2 - 4	24P/V #2 - 5	24P/V #2 - 6	24P/V #2 - 7	24P/V #2 - 8
Address	$079206	$079207	$07920E	$07920F	$079306	$079307	$07930E	$07930F
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V #3 - 1	24P/V #3 - 2	24P/V #3 - 3	24P/V #3 - 4	24P/V #3 - 5	24P/V #3 - 6	24P/V #3 - 7	24P/V #3 - 8
Address	$07A206	$07A207	$07A20E	$07A20F	$07A306	$07A307	$07A30E	$07A30F
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V #4 - 1	24P/V #4 - 2	24P/V #4 - 3	24P/V #4 - 4	24P/V #4 - 5	24P/V #4 - 6	24P/V #4 - 7	24P/V #4 - 8
Address	$07B206	$07B207	$07B20E	$07B20F	$07B306	$07B307	$07B30E	$07B30F

`Y:$07xx0x` ADC input value (high 16 bits)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC(1) - 1	PMAC(1) - 2	PMAC(1) - 3	PMAC(1) - 4	PMAC(1) - 5	PMAC(1) - 6	PMAC(1) - 7	PMAC(1) - 8
Address	$078003	$078007	$07800B	$07800F	$078103	$078107	$07810B	$07810F
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V #1 - 1	24P/V #1 - 2	24P/V #1 - 3	24P/V #1 - 4	24P/V #1 - 5	24P/V #1 - 6	24P/V #1 - 7	24P/V #1 - 8
Address	$078203	$078207	$07820B	$07820F	$078303	$078307	$07830B	$07830F
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V #2 - 1	24P/V #2 - 2	24P/V #2 - 3	24P/V #2 - 4	24P/V #2 - 5	24P/V #2 - 6	24P/V #2 - 7	24P/V #2 - 8
Address	$079203	$079207	$07920B	$07920F	$079303	$079307	$07930B	$07930F
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V #3 - 1	24P/V #3 - 2	24P/V #3 - 3	24P/V #3 - 4	24P/V #3 - 5	24P/V #3 - 6	24P/V #3 - 7	24P/V #3 - 8
Address	$07A203	$07A207	$07A20B	$07A20F	$07A303	$07A307	$07A30B	$07A30F
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V #4 - 1	24P/V #4 - 2	24P/V #4 - 3	24P/V #4 - 4	24P/V #4 - 5	24P/V #4 - 6	24P/V #4 - 7	24P/V #4 - 8
Address	$07B203	$07B207	$07B20B	$07B20F	$07B303	$07B307	$07B30B	$07B30F

`X:$07xx0x` Encoder Capture/Compare position
(Capture register is read-only; compare register is write-only)

PMAC2-Style Servo ASIC Registers

Note:

For addressing of alternate Servo ICs ‘n*’ on the UBUS expansion port, add $20 to the addresses for Servo IC ‘n’ shown in these tables.

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078000	$078008	$078010	$078018	$078100	$078108	$078110	$078118
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078200	$078208	$078210	$078218	$078300	$078308	$078310	$078318
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079200	$079208	$079210	$079218	$079300	$079308	$079310	$079318
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A200	$07A208	$07A210	$07A218	$07A300	$07A308	$07A310	$07A318
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B200	$07B208	$07B210	$07B218	$07B300	$07B308	$07B310	$07B318

`Y:$07x0xx .` Channel n Time between last two encoder counts (SCLK cycles)

Bits: 0-22: Timer (units of SCLK cycles)
23: Change-of-direction flag

Alternate use if Channel Control Word bit 18 set to 1 (Rev "D" or newer IC only):
Channel n Timer-Based Fractional Count Data (unsigned)

Bits: 0-11 Compare "B" fractional count (bit 11 = ½-count; bit 10 = ¼-count; etc.)
12-23 Flag-captured fractional count (bit 23 = ½-count; bit 22 = ¼-count; etc.)

`X:$07xx0x` Channel n Status Word

Bits:

0-2	Capture Hall Effect Device State
3	Invalid demultiplex of C, U, V, & W
4-7	Reserved for future use (reports as 0)
8	Encoder Count Error (0 on counter reset, 1 on illegal transition)
9	Position Compare (EQUn) output value
10	Position-Captured-On-Gated-Index Flag (=0 on read of captured position register, =1 on trigger capture)
11	Position-Captured Flag (on any trigger) (=0 on read of captured position register, =1 on trigger capture)
12	Encoder Channel A (CHAn) Input Value
13	Encoder Channel B (CHBn) Input Value
14	Encoder Channel C (Index, CHCn) Input Value (ungated)
15	Amplifier Fault (FAULTn) Input Value
16	Home Flag (HMFLn) Input Value
17	Positive End Limit (PLIMn) Input Value
18	Negative End Limit (MLIMn) Input Value
19	User Flag (USERn) Input Value
20	FlagWn Input Value
21	FlagVn Input Value
22	FlagUn Input Value
23	FlagTn Input Value

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078001	$078009	$078011	$078019	$078101	$078109	$078111	$078119
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078201	$078209	$078211	$078219	$078301	$078309	$078311	$078319
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079201	$079209	$079211	$079219	$079301	$079309	$079311	$079319
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A201	$07A209	$07A211	$07A219	$07A301	$07A309	$07A311	$07A319
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B201	$07B209	$07B211	$07B219	$07B301	$07B309	$07B311	$07B319

`Y:$07x0xx .` Channel n Time between last two encoder counts (SCLK cycles)

Bits: 0-22: Timer (units of SCLK cycles)
23: Change-of-direction flag

Alternate use if Channel Control Word bit 18 set to 1 (Rev "D" or newer IC only):
Channel n Timer-Based Fractional Count Data (unsigned)

Bits: 0-11 Compare "B" fractional count (bit 11 = ½-count; bit 10 = ¼-count; etc.)
12-23 Flag-captured fractional count (bit 23 = ½-count; bit 22 = ¼-count; etc.)

`X:$07xx0x` Channel n Encoder phase position (counts)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078002	$07800A	$078012	$07801A	$078102	$07810A	$078112	$07811A
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078202	$07820A	$078212	$07821A	$078302	$07830A	$078312	$07831A
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079202	$07920A	$079212	$07921A	$079302	$07930A	$079312	$07931A
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A202	$07A20A	$07A212	$07A21A	$07A302	$07A30A	$07A312	$07A31A
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B202	$07B20A	$07B212	$07B21A	$07B302	$07B30A	$07B312	$07B31A

`Y:$07xx0x` Channel n Output A Command Value
Bits: 8-23: PWM Command Value
6-23: Serial DAC Command Value
0-5: Not Used

`X:$07xx0x` Channel n Encoder Servo Position Capture Register
Bits: 0: Direction of last count (0=up, 1=down)
1-23: Position counter (units of counts)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078003	$07800B	$078013	$07801B	$078103	$07810B	$078113	$07811B
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078203	$07820B	$078213	$07821B	$078303	$07830B	$078313	$07831B
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079203	$07920B	$079213	$07921B	$079303	$07930B	$079313	$07931B
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A203	$07A20B	$07A213	$07A21B	$07A303	$07A30B	$07A313	$07A31B
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B203	$07B20B	$07B213	$07B21B	$07B303	$07B30B	$07B313	$07B31B

`Y:$07xx0x` Channel n Output B Command Value
Bits: 8-23: PWM Command Value
6-23: Serial DAC Command Value
0-5: Not used
`X:$07xx0x` Channel n Flag Position Capture Value; 24 bits, units of counts

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078004	$07800C	$078014	$07801C	$078104	$07810C	$078114	$07811C
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078204	$07820C	$078214	$07821C	$078304	$07830C	$078314	$07831C
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079204	$07920C	$079214	$07921C	$079304	$07930C	$079314	$07931C
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A204	$07A20C	$07A214	$07A21C	$07A304	$07A30C	$07A314	$07A31C
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B204	$07B20C	$07B214	$07B21C	$07B304	$07B30C	$07B314	$07B31C

`Y:$07xx0x` Channel n Output C Command Value
Bits: 8-23: PWM Command Value
0-23: PFM Command Value

IC# - Chan#	0-1	1-1
Board - Chan#	PMAC2-1	PMAC2-5
Address	$078004	$078104
IC# - Chan#	2-1	3-1
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 5
Address	$078204	$078304
IC# - Chan#	4-1	5-1
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 5
Address	$079204	$079304
IC# - Chan#	6-1	7-1
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 5
Address	$07A204	$07A304
IC# - Chan#	8-1	9-1
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 5
Address	$07B204	$07B304

`X:$07xx04` Servo IC Clock Control Word

Bits

(Bits 0-11 form I7m03)

0-2: SCLK Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

3-5: PFM Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

6-8: DAC Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

9-11: ADC Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

(Bits 12-13 form I7m07)

12: Phase Clock Direction (0=output, 1=input)
(This must be 0 in X:$C004; 1 in X:$C024--if 2nd ASIC is used)

13: Servo Clock Direction (0=output, 1=input)
(This must be 0 in X:$C004; 1 in X:$C024--if 2nd ASIC is used)

14-15: Reserved for future use (report as zero)

(Bits 16-19 form I7m01)

16-19: Phase Clock Frequency Control n (f = MaxPhase / [n+1], n=0-15)

(Bits 20-23 form I7m02)

20-23: Servo Clock Frequency Control n (f = Phase / [n+1], n=0--15)

IC# - Chan#	0-2	1-2
Board - Chan#	PMAC2-2	PMAC2-6
Address	$07800C	$07810C
IC# - Chan#	2-2	3-2
Board - Chan#	24P/V/E2 #1 - 2	24P/V/E2 #1 - 6
Address	$07820C	$07830C
IC# - Chan#	4-2	5-2
Board - Chan#	24P/V/E2 #2 - 2	24P/V/E2 #2 - 6
Address	$07920C	$07930C
IC# - Chan#	6-2	7-2
Board - Chan#	24P/V/E2 #3 - 2	24P/V/E2 #3 - 6
Address	$07A20C	$07A30C
IC# - Chan#	8-2	9-2
Board - Chan#	24P/V/E2 #4 - 2	24P/V/E2 #4 - 6
Address	$07B20C	$07B30C

`X:$07xx0C` DAC Strobe Word, 24 bits (I7m05)
(Shifted out MSB first one bit per DAC_CLK cycle, starting on rising edge of phase clock)

IC# - Chan#	0-3	1-3
Board - Chan#	PMAC2-3	PMAC2-7
Address	$078014	$078114
IC# - Chan#	2-3	3-3
Board - Chan#	24P/V/E2 #1 - 3	24P/V/E2 #1 - 7
Address	$078214	$078314
IC# - Chan#	4-3	5-3
Board - Chan#	24P/V/E2 #2 - 3	24P/V/E2 #2 - 7
Address	$079214	$079314
IC# - Chan#	6-3	7-3
Board - Chan#	24P/V/E2 #3 - 3	24P/V/E2 #3 - 7
Address	$07A214	$07A314
IC# - Chan#	8-3	9-3
Board - Chan#	24P/V/E2 #4 - 3	24P/V/E2 #4 - 7
Address	$07B214	$07B314

`X:$07xx14` Servo IC ADC Strobe Word, 24 bits (I7m06)
(Shifted out MSB first one bit per ADC_CLK cycle, starting on rising edge of phase clock)

IC# - Chan#	**0-4**	**1-4**
Board - Chan#	PMAC2-4	PMAC2-8
Address	$07801C	$07811C
IC# - Chan#	2-4	3-4
Board - Chan#	24P/V/E2 #1 - 4	24P/V/E2 #1 - 8
Address	$07821C	$07831C
IC# - Chan#	4-4	5-4
Board - Chan#	24P/V/E2 #2 - 4	24P/V/E2 #2 - 8
Address	$07921C	$07931C
IC# - Chan#	6-4	7-4
Board - Chan#	24P/V/E2 #3 - 4	24P/V/E2 #3 - 8
Address	$07A21C	$07A31C
IC# - Chan#	8-4	9-4
Board - Chan#	24P/V/E2 #4 - 4	24P/V/E2 #4 - 8
Address	$07B21C	$07B31C

`X:$07xx1C` Servo IC PWM, PFM, MaxPhase Control Word
(Bits 0-7 form I7m04)
Bits: 0-7: PWM Dead Time (16*PWM CLK cycles)
also PFM pulse width (PFM CLK cycles)
(Bits 8-23 form I7m00)

8-23: PWM MaxCount Value
PWM Frequency = 117.9648 MHz / [4*MaxCount + 6]
"MaxPhase" Frequency = 2*PWM Frequency
= 117.9648 MHz / [2*MaxCount + 3]

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078005	$07800D	$078015	$07801D	$078105	$07810D	$078115	$07811D
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078205	$07820D	$078215	$07821D	$078305	$07830D	$078315	$07831D
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079205	$07920D	$079215	$07921D	$079305	$07930D	$079315	$07931D
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A205	$07A20D	$07A215	$07A21D	$07A305	$07A30D	$07A315	$07A31D
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B205	$07B20D	$07B215	$07B21D	$07B305	$07B30D	$07B315	$07B31D

`Y:$07xx0x` Channel n ADC A Input Value
Bits: 6-23: Serial ADC Value
0-5: Not used

`X:$07xx0x` Channel n Control Word
(Bits 0-3 form I7mn0)
Bits ..... 0-3: Encoder/Timer Decode Control
0000: External pulse-and-direction decode, clockwise
0001: External x1 quadrature decode, clockwise
0010: External x2 quadrature decode, clockwise
0011: External x4 quadrature decode, clockwise
0100: External pulse-and-direction decode, counter-clockwise
0101: External x1 quadrature decode, counter-clockwise
0110: External x2 quadrature decode, counter-clockwise
0111: External x4 quadrature decode, counter-clockwise
1000: Internal pulse-and-direction decode (clockwise)
1001: (Reserved for future use)
1010: (Reserved for future use)
1011: External x6 hall-sensor decode, clockwise
1110: MLDT timer mode
1101: (Reserved for future use)
1110: (Reserved for future use)
1111: External x6 hall-sensor decode, clockwise
(Bits 4-7 form I7mn2)
4-5: Position Capture Control
00: Immediate capture

01: Use encoder index alone
10: Use capture flag alone
11: Use encoder index and capture flag

6: Index Capture Invert Control (0=no inversion, 1=inversion)
7: Flag Capture Invert Control (0=no inversion, 1=inversion)
8-9: Capture Flag Select Control (I7mn3)
00: Home Flag (HMFLn)
01: Positive End Limit (PLIMn)
10: Negative End Limit (MLIMn)
11: User Flag (USERn)
10: Encoder Counter Reset Control (1=reset)
11: Position Compare Initial State Write Enable
12: Position Compare Initial State Value
13: Position Compare Channel Select (I7mn1)
(0= use this channel's encoder; 1=use first encoder on IC)
14: AENAn output value
15: Gated Index Select for Position Capture (I7mn4)
(0=ungated index, 1=gated index)
16: Invert AB for Gated Index (I7mn5)
(0: Gated Signal=A&B&C; 1: Gated Signal=A/&B/&C)
17: Demultiplex Index Channel Control
(0=no demux; 1=demux hall signals from index input)
18: Hardware 1/T Enable (I7mn9)
19: Invert PFM Direction Control (0=no inversion, 1=invert) (I7mn8)
(Bits 20-21: I7mn7)
20: Invert A & B Output Control (0=no inversion, 1=invert)
21: Invert C Output Control (0=no inversion, 1=invert)
(Bits 22-23: I7mn6)
22: Output A & B Mode Select (0=PWM, 1=DAC)
23: Output C Mode Select (0=PWM, 1=PFM)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078006	$07800E	$078016	$07801E	$078106	$07810E	$078116	$07811E
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078206	$07820E	$078216	$07821E	$078306	$07830E	$078316	$07831E
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079206	$07920E	$079216	$07921E	$079306	$07930E	$079316	$07931E
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A206	$07A20E	$07A216	$07A21E	$07A306	$07A30E	$07A316	$07A31E
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B206	$07B20E	$07B216	$07B21E	$07B306	$07B30E	$07B316	$07B31E

`Y:$07xx0x` Channel n ADC B Input Value
Bits: 6-23: Serial ADC Value
0-5: Not used
`X:$07xx0x` Channel n Encoder Compare Auto-increment value (24 bits, units of counts)

IC# - Chan#	0-1	0-2	0-3	0-4	1-1	1-2	1-3	1-4
Board - Chan#	PMAC2-1	PMAC2-2	PMAC2-3	PMAC2-4	PMAC2-5	PMAC2-6	PMAC2-7	PMAC2-8
Address	$078007	$07800F	$078017	$07801F	$078107	$07810F	$078117	$07811F
IC# - Chan#	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Board - Chan#	24P/V/E2 #1 - 1	24P/V/E2 #1 - 2	24P/V/E2 #1 - 3	24P/V/E2 #1 - 4	24P/V/E2 #1 - 5	24P/V/E2 #1 - 6	24P/V/E2 #1 - 7	24P/V/E2 #1 - 8
Address	$078207	$07820F	$078217	$07821F	$078307	$07830F	$078317	$07831F
IC# - Chan#	4-1	4-2	4-3	4-4	5-1	5-2	5-3	5-4
Board - Chan#	24P/V/E2 #2 - 1	24P/V/E2 #2 - 2	24P/V/E2 #2 - 3	24P/V/E2 #2 - 4	24P/V/E2 #2 - 5	24P/V/E2 #2 - 6	24P/V/E2 #2 - 7	24P/V/E2 #2 - 8
Address	$079207	$07920F	$079217	$07921F	$079307	$07930F	$079317	$07931F
IC# - Chan#	6-1	6-2	6-3	6-4	7-1	7-2	7-3	7-4
Board - Chan#	24P/V/E2 #3 - 1	24P/V/E2 #3 - 2	24P/V/E2 #3 - 3	24P/V/E2 #3 - 4	24P/V/E2 #3 - 5	24P/V/E2 #3 - 6	24P/V/E2 #3 - 7	24P/V/E2 #3 - 8
Address	$07A207	$07A20F	$07A217	$07A21F	$07A307	$07A30F	$07A317	$07A31F
IC# - Chan#	8-1	8-2	8-3	8-4	9-1	9-2	9-3	9-4
Board - Chan#	24P/V/E2 #4 - 1	24P/V/E2 #4 - 2	24P/V/E2 #4 - 3	24P/V/E2 #4 - 4	24P/V/E2 #4 - 5	24P/V/E2 #4 - 6	24P/V/E2 #4 - 7	24P/V/E2 #4 - 8
Address	$07B207	$07B20F	$07B217	$07B21F	$07B307	$07B30F	$07B317	$07B31F

`Y:$07xx0x` Channel n Encoder Compare A Value (24 bits, units of counts)
`X:$07xx0x` Channel n Encoder Compare B Value (24 bits, units of counts)

Turbo PMAC2 MACRO and I/O ASIC Registers

Note:

Starting with firmware version 1.936, it is possible to assign any of 16 possible base addresses to MACRO ICs with I20, I21, I22, and I23. This is really useful only for certain extended UMAC Turbo systems.

Turbo PMAC2 boards that are not "Ultralite" and not "UMAC" have only one MACRO IC, with a fixed base address of $078400. Turbo PMAC2 Ultralite boards may have up to 4 MACRO ICs, with base addresses of $078400, $078500, $078600, and $078700.

UMAC Turbo systems may have up to 16 MACRO ICs, although only 4 at any given time can support automatic firmware functions by designation as MACRO ICs 0 – 3 with I20 – I23. The 16 possible base addresses are $07xy00, where 'x' can be 8, 9, A, or B, and 'y' can be 4, 5, 6, or 7.

This section assumes that MACRO ICs 0 – 3 have the default base addresses of $078400, $078500, $078600, and $078700.

I/O Control and Data Registers (MACRO IC 0 only)

`Y:$078400` JI/O Port Data Register (Input or output; when used as general I/O; see Y:$078404)

Bits: 0 I/O00 Data Value
...
23 I/O23 Data Value

`X:$078400` JI/O Port Data Direction Control Register (when used as general I/O; see Y:$078404)

Bits: 0 I/O00 Direction Control
...
23 I/O23 Direction Control
(All bits: 0=Input; 1=Output)

`Y:$078401` JI/O Port Data Register (Input or output; when used as general I/O; see Y:$078405)

Bits: 0 I/O24 Data Value
...
7 I/O31 Data Value
8 I/O24 Latched Data Value
...
15 I/O31 Latched Data Value
16-23 Not used

`X:$078401` JI/O Port Data Direction Control Register (when used as general I/O; see Y:$078405)

Bits: 0 I/O24 Direction Control
...
7 I/O31 Direction Control
(All bits: 0=Input; 1=Output)
8-23 Not used

`Y:$078402` JTHW Port Data Register (Input or output; when used as general I/O; see Y:$078406)

Bits: 0 DAT0 Data Value
...
7 DAT7 Data Value
8 SEL0 Data Value
...
15 SEL7 Data Value
16-23 Not used

`X:$078402` JTHW Port Data Direction Control Register (when used as general I/O; see Y:$078406)

Bits: 0 DAT0 Direction Control (Must be 0 to use standard port accessories)
...
7 DAT7 Direction Control (Must be 0 to use standard port accessories)
8 SEL0 Direction Control (Must be 1 to use standard port accessories)
...
15 SEL7 Direction Control (Must be 1 to use standard port accessories)
(All bits: 0=Input; 1=Output)
16-23 Not used

`Y:$078403` JDISP, J??? Port Data Register

Bits:
0 DISP0 Data Value
...
7 DISP7 Data Value
8 CTRL0 Data Value
...
11 CTRL3 Data Value
12-23 Not used

`X:$078403` JDISP, J??? Port Data Direction Control Register

Bits:
0 DISP0 Direction Control (Must be 1 for display to function)
...
7 DISP7 Direction Control (Must be 1 for display to function)
8 CTRL0 Direction Control (Must be 1 for ??? to function)
...
11 CTRL3 Direction Control (Must be 1 for ??? to function)
(All bits: 0=Input; 1=Output)
12-23 Not used

`Y:$078404` JI/O Port Data Type Control Register

Bits:
0 I/O00 Data Type Control (0=FlagW1*; 1=I/O00)
1 I/O01 Data Type Control (0=FlagV1*; 1=I/O01)
2 I/O02 Data Type Control (0=FlagU1*; 1=I/O02)
3 I/O03 Data Type Control (0=FlagT1*; 1=I/O03)
4 I/O04 Data Type Control (0=USER1*; 1=I/O04)
5 I/O05 Data Type Control (0=MLIM1*; 1=I/O05)
6 I/O06 Data Type Control (0=PLIM1*; 1=I/O06)
7 I/O07 Data Type Control (0=HMFL1*; 1=I/O07)
8 I/O08 Data Type Control (0=PWM_B_BOT1*; 1=I/O08)
9 I/O09 Data Type Control (0=PWM_B_TOP1*; 1=I/O09)
10 I/O10 Data Type Control (0=PWM_A_BOT1*; 1=I/O10)
11 I/O11 Data Type Control (0=PWM_A_TOP1*; 1=I/O11)
12 I/O12 Data Type Control (0=PWM_B_BOT2*; 1=I/O12)
13 I/O13 Data Type Control (0=PWM_B_TOP2*; 1=I/O13)
14 I/O14 Data Type Control (0=PWM_A_BOT2*; 1=I/O14)
15 I/O15 Data Type Control (0=PWM_A_TOP2*; 1=I/O15)
16 I/O16 Data Type Control (0=HMFL2*; 1=I/O16)
17 I/O17 Data Type Control (0=PLIM2*; 1=I/O17)
18 I/O18 Data Type Control (0=MLIM2*; 1=I/O18)
19 I/O19 Data Type Control (0=USER2*; 1=I/O19)
20 I/O20 Data Type Control (0=FlagT2*; 1=I/O20)
21 I/O21 Data Type Control (0=FlagU2*; 1=I/O21)
22 I/O22 Data Type Control (0=FlagV2*; 1=I/O22)
23 I/O23 Data Type Control (0=FlagW2*; 1=I/O23)
(All bits: 0=dedicated hardware I/O; 1=general I/O)
(All bits must be 1 for JI/O Port to function as general I/O)

`X:$078404` JI/O Port Data Inversion Control Register (when used as general I/O; see Y:$078404)

Bits: 0 I/O00 Inversion Control
...
23 I/O23 Inversion Control
(All bits: 0=Non-inverting; 1=Inverting)

`Y:$078405` JI/O Port Data Type Control Register

Bits: 0 I/O24 Data Type Control
...
7 I/O31 Data Type Control
(These bits are always 1; there is no alternate mode for these lines)
8-23 Not used

`X:$078405` JI/O Port Data Inversion Control

Bits: 0 I/O24 Inversion Control
...
7 I/O31 Inversion Control
(All bits: 0=Non-inverting; 1=Inverting)
8-23 Not used

`Y:$078406` JTHW Port Data Type Control Register

Bits: 0 DAT0 Data Type Control (0=HWC1; 1 =DAT0)
1 DAT1 Data Type Control (0=HWC2; 1 =DAT1)
2 DAT2 Data Type Control (0=Fault1*; 1 =DAT2)
3 DAT3 Data Type Control (0=Fault2*; 1 =DAT3)
4 DAT4 Data Type Control (0=EQU1*; 1 =DAT4)
5 DAT5 Data Type Control (0=EQU2*; 1 =DAT5)
6 DAT6 Data Type Control (0=AENA1*; 1 =DAT6)
7 DAT7 Data Type Control (0=AENA2*; 1 =DAT7)
8 SEL0 Data Type Control (0=ADC_STROB*; 1=SEL0)
9 SEL1 Data Type Control (0=ADC_CLK*; 1=SEL1)
10 SEL2 Data Type Control (0=ADC_A1*; 1=SEL2)
11 SEL3 Data Type Control (0=ADC_B1*; 1=SEL3)
12 SEL4 Data Type Control (0=ADC_A2*; 1=SEL4)
13 SEL5 Data Type Control (0=ADC_B2*; 1=SEL5)
14 SEL6 Data Type Control (0=SCLK*; 1=SEL6)
15 SEL7 Data Type Control (0=SCLK_DIR*; 1=SEL7)
(All bits: 0=dedicated hardware I/O; 1=general I/O)
(All bits must be 1 for JTHW Port to function as general I/O or with multiplexed accessories)
16-23 Not used

`X:$078406` JTHW Port Data Inversion Control Register (when used as general I/O; see Y:$078406)

Bits: 0 DAT0 Inversion Control
...
7 DAT7 Inversion Control
8 SEL0 Inversion Control
...
15 SEL7 Inversion Control
(All bits: 0=Non-inverting; 1=Inverting)

(All bits must be 0 to use standard port accessories)
16-23 Not used

`Y:$078407` JDISP, J??? Port Data Type Control Register
Bits: 0 DISP0 Data Type Control
...
7 DISP7 Data Type Control
8 CTRL0 Data Type Control
...
11 CTRL3 Data Type Control
(These bits are always 1; there is no alternate mode for these pins)
12-23 Not used

`X:$078407` JDISP, J??? Port Data Inversion Control Register
Bits: 0 DISP0 Inversion Control
...
7 DISP7 Inversion Control
8 CTRL0 Inversion Control
...
11 CTRL3 Inversion Control
(All bits: 0=Non-inverting; 1=Inverting)
(All bits must be 0 to use standard port accessories)
12-23 Not used

`Y:$078408-$07840B` Not used
`X:$078408-$07840B` Not used
`Y:$07840C` Pure binary conversion from gray code input on I/O00 to I/O23
`X:$07840C` Not used
`Y:$07840D` Gray-to-binary conversion bit-length control
Bits: 0-3 Bit length of less significant word portion (I/O00 - I/Onn)
4 =1 specifies 16-bit lower / 8-bit upper conversion
5-23 Not used
`X:$07840D` Not used

MACRO Ring Control Registers

MACRO IC#	0	1	2	3
Address	$07840E	$07940E	$07A40E	$07B40E

`Y:$07x40E` MACRO Node Enable Control (I6841, I689 I6941, I6991)
Bits: 0 Node 0 enable control
...
15 Node 15 enable control
(0=node disable; 1=node enable)
16-19 Sync packet slave node number control
20-23 Master number control

`X:$07840E` Not used

MACRO IC#	0	1	2	3
Address	$07840F	$07940F	$07A40F	$07B40F

Y:$07x40F MACRO Ring Status and Control (I6840, I6890, I6940, I6990)

Bits:
- 0 Data overrun error (cleared when read)
- 1 Byte violation error (cleared when read)
- 2 Packet parity error (cleared when read)
- 3 Data underrun error (cleared when read)
- 4 Master station enable
- 5 Synchronizing master station enable
- 6 Sync packet received (cleared when read)
- 7 Sync packet phase lock enable
- 8 Node 8 master address check disable
- 9 Node 9 master address check disable
- 10 Node 10 master address check disable
- 11 Node 11 master address check disable
- 12 Node 12 master address check disable
- 13 Node 13 master address check disable
- 14 Node 14 master address check disable
- 15 Node 15 master address check disable

X:$07x40F MACRO IC clock control register

Bits

(Bits 0-11 form I6803 – MACRO IC 0 only)

- 0-2: Handwheel SCLK* Frequency Control n (f=39.3216MHz/2^n, n=0-7)
- 3-5: JHW/PD PFM Clock* Frequency Control n (f=39.3216MHz/2^n, n=0-7)
- 6-8: DAC Clock* Frequency Control n (f=39.3216MHz / 2^n, n=0-7)
- 9-11: ADC Clock* Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

(Bits 12-13 form I6807, I6857, I6907, I6957)

- 12: Phase Clock Direction (0=output, 1=input)
  (This must be 1)
- 13: Servo Clock Direction (0=output, 1=input)
  (This must be 1)
- 14-15: Not used (report as zero)

(Bits 16-19 form I6801, I6851, I6901, I6951)

- 16-19: Phase Clock* Frequency Control n
  (f = MAXPHASE* / [n+1], n=0-15)

(Bits 20-23 form I6802 – MACRO IC 0 only)

- 20-23: Servo Clock* Frequency Control n
  (f = PHASE* / [n+1], n=0-15)

Supplemental Servo Channel Registers (MACRO IC 0 only)

Chan #	1*	2*
Address	$078410	$078418

Y:$07841x Handwheel n Time between last two encoder counts (SCLK cycles)

X:$07841x Supplementary Channel n* (Handwheel n) Status Word

Bits:
- 0-2 Captured Hall Effect Device (UVW) State
- 3-7 Not used (reports as 0)
- 8 Encoder Count Error (0 on counter reset, 1 on illegal transition)
- 9 Position Compare (EQUn*) output value
- 10 Position-Captured-On-Gated-Index Flag
  (=0 on read of captured position register, =1 on trigger capture)
- 11 Position-Captured Flag (on any trigger)

(=0 on read of captured position register, =1 on trigger capture)
12 Handwheel 1 Channel A (HWAn) Input Value
13 Handwheel 1 Channel B (HWBn) Input Value
14 Handwheel 1 Channel C (Index, HWCn) Input Value (ungated)
15 Amplifier Fault (FAULTn*) Input Value
16 Home Flag (HMFLn*) Input Value
17 Positive End Limit (PLIMn*) Input Value
18 Negative End Limit (MLIMn*) Input Value
19 User Flag (USERn*) Input Value
20 FlagWn* Input Value
21 FlagVn* Input Value
22 FlagUn* Input Value
2 3 FlagTn* Input Value

Chan #	1*	2*
Address	$078411	$078419

`Y:$07841x` Handwheel n Time Since Last Encoder Count (SCLK cycles)
`X:$07841x` Handwheel n Phase Position Capture Register (counts)

Chan #	1*	2*
Address	$078412	$07841A

`Y:$07841x` Supplementary Channel n* Output A Command Value
Bits: 8-23: PWM Command Value
(appears on I/O10 & I/O11 if in dedicated mode)
0-5: Not Used
`X:$07841x` Handwheel n* Servo Position Capture Register
Bits: 0: Direction of last count (0=up, 1=down)
1-23: Position counter (units of counts)

Chan #	1*	2*
Address	$078413	$07841B

`Y:$07841x` Supplementary Channel n* Output B Command Value
Bits: 8-23: PWM Command Value
(appears on I/O08 & I/O09 if in dedicated mode)
0-5: Not used
`X:$07841x` Handwheel n Flag Position Capture Value; 24 bits, units of counts

Chan #	1*	2*
Address	$078414	$07841C

`Y:$07841x` Supplementary Channel n* Output C Command Value
Bits: 8-23: PWM Command Value (appears on PUL1 & DIR1 if in PWM mode)
0-23: PFM Command Value (appears on PUL1 & DIR1 if in PFM mode)
`X:$078414` Supplementary ADC Strobe Word, 24 bits
(Shifted out MSB first one bit per DAC_CLK cycle, starting on rising edge of phase clock; appears on SEL0 if in dedicated mode)
`X:$07841C` Supplementary PWM, PFM, MaxPhase* Control Word
Bits: 0-7: PWM* Dead Time (16*PWM* CLK cycles) (I6804)
also PFM* pulse width (PFM* CLK cycles)
8-23: PWM* Max Count Value (I6800)

PWM* Frequency = 117.96MHz / [2*(MaxCount+1)]
"MaxPhase*" Frequency = 2*PWM* Frequency

Chan #	1*	2*
Address	$078415	$07841D

`Y:$07841x` Supplementary Channel n* ADC A Input Value (uses SEL2 in dedicated mode)
Bits: 6-23: Serial ADC Value
0-5: Not used

`X:$07841x` Supplementary Channel n* (Handwheel n) Control Word
(Bits 0-3 form I68n0)
Bits 0-1: Encoder Decode Control
00: Pulse and direction decode
01: x1 quadrature decode
10: x2 quadrature decode
11: x4 quadrature decode
2-3: Direction & Timer Control
00: Standard timer control, external signal source, no inversion
01: Standard timer control, external signal source, invert direction
10: Standard timer control, internal PFM source, no inversion
11: Alternate timer control, external signal source
(Bits 4-7 form I68n2)
4-5: Position Capture Control
00: Software capture (by setting bit 6)
01: Use encoder index alone
10: Use capture flag alone
11: Use encoder index and capture flag
6: Index Capture Invert Control (0=no inversion, 1=inversion)
7: Flag Capture Invert Control (0=no inversion, 1=inversion)
8-9: Capture Flag Select Control (I68n3)
00: Home Flag (HMFLn*)
01: Positive Limit (PLIMn*)
10: Negative Limit (MLIMn*)
11: User Flag (USERn*)
10: Encoder Counter Reset Control (1=reset)
11: Position Compare Initial State Write Enable
12: Position Compare Initial State Value
13: Position Compare Channel Select (I68n1)
(0= use this channel's encoder; 1=use first encoder on IC)
14: AENAn* output value
15: Gated Index Select for Position Capture (I68n4)
(0=ungated index, 1=gated index)
16: Invert AB for Gated Index (I68n5)
(0: Gated Signal=A&B&C; 1: Gated Signal=A/&B/&C)
17: De-multiplex Index Channel Control
(0=no demux; 1=demux hall signals from index input)
18: Reserved for future use (reports as 0)
19: Invert PFM Direction Control (0=no inversion, 1=invert) (I68n8)
(Bits 20-21 form I68n7)
20: Invert A & B Output Control (0=no inversion, 1=invert)
21: Invert C Output Control (0=no inversion, 1=invert)

(Bits 22-23 form I68n6)
22: Output A & B Mode Select (0=PWM, 1=DAC)
23: Output C Mode Select (0=PWM, 1=PFM)

Chan #	1*	2*
Address	$078416	$07841E

Y:$07841x Supplementary Channel n* ADC B Input Value (uses SEL3 in dedicated mode)
Bits: 6-23: Serial ADC Value
0-5: Not used
X:$07841x Handwheel n Compare Auto-increment value (24 bits, units of counts)

Chan #	1*	2*
Address	$078417	$07841F

Y:$07841x Handwheel n Compare A Value (24 bits, units of counts)
X:$07841x Handwheel n Compare B Value (24 bits, units of counts)

Turbo PMAC2 MACRO Node Registers

MACRO IC#	0	1	2	3
Address	$07842x	$07942x	$07A42x	$07B42x

Y:$07x420 MACRO Node 0 24-bit command(write) and feedback (read) register
X:$07x420 MACRO Node 2 24-bit command(write) and feedback (read) register
Y:$07x421 MACRO Node 0 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x421 MACRO Node 2 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x422 MACRO Node 0 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x422 MACRO Node 2 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x423 MACRO Node 0 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x423 MACRO Node 2 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x424 MACRO Node 1 24-bit command(write) and feedback (read) register
X:$07x424 MACRO Node 3 24-bit command(write) and feedback (read) register
Y:$07x425 MACRO Node 1 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x425 MACRO Node 3 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x426 MACRO Node 1 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x426 MACRO Node 3 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x427 MACRO Node 1 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x427 MACRO Node 3 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x428 MACRO Node 4 24-bit command(write) and feedback (read) register

X:$07x428 MACRO Node 6 24-bit command(write) and feedback (read) register
Y:$07x429 MACRO Node 4 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x429 MACRO Node 6 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x42A MACRO Node 4 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x42A MACRO Node 6 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x42B MACRO Node 4 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x42B MACRO Node 6 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x42C MACRO Node 5 24-bit command(write) and feedback (read) register
X:$07x42C MACRO Node 7 24-bit command(write) and feedback (read) register
Y:$07x42D MACRO Node 5 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x42D MACRO Node 7 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x42E MACRO Node 5 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x42E MACRO Node 7 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x42F MACRO Node 5 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x42F MACRO Node 7 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

MACRO IC#	0	1	2	3
Address	$07843x	$07943x	$07A43x	$07B43x

Y:$07x430 MACRO Node 8 24-bit command(write) and feedback (read) register
X:$07x430 MACRO Node 10 24-bit command(write) and feedback (read) register
Y:$07x431 MACRO Node 8 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x431 MACRO Node 10 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x432 MACRO Node 1 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x432 MACRO Node 10 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x433 MACRO Node 8 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x433 MACRO Node 10 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$07x434 MACRO Node 9 24-bit command(write) and feedback (read) register
X:$07x434 MACRO Node 11 24-bit command(write) and feedback (read) register
Y:$07x435 MACRO Node 9 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$07x435 MACRO Node 11 1st 16-bit command(write) and feedback (read) register

(bits 8-23; bits 0-7 not used)

Y:$07x436 MACRO Node 9 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x436 MACRO Node 11 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x437 MACRO Node 9 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x437 MACRO Node 11 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x438 MACRO Node 12 24-bit command(write) and feedback (read) register

X:$07x438 MACRO Node 14 24-bit command(write) and feedback (read) register

Y:$07x439 MACRO Node 12 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x439 MACRO Node 14 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x43A MACRO Node 12 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x43A MACRO Node 14 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x43B MACRO Node 12 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x43B MACRO Node 14 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x43C MACRO Node 13 24-bit command(write) and feedback (read) register

X:$07x43C MACRO Node 15 24-bit command(write) and feedback (read) register

Y:$07x43D MACRO Node 13 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x43D MACRO Node 15 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x43E MACRO Node 13 1st 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x43E MACRO Node 15 2nd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$07x43F MACRO Node 13 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$07x43F MACRO Node 15 3rd 16-bit command(write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Turbo PMAC(1) I/O Registers

	Bits		
Y:$078800	0-7	Display & EAROM I/O (dedicated)	
	8	Jog Minus Input	(J2-4)
	9	Jog Plus Input	(J2-6)
	10	Prejog Input	(J2-7)
	11	Start (Run) Input	(J2-8)
	12	Step/Quit Input	(J2-9)
	13	Stop (Abort) Input	(J2-10)
	14	Home Command Input	(J2-11)
	15	Feed Hold Input	(J2-12)

	16	Motor/CoordSys Select Inp. Bit 0 (J2-3)	
	17	Motor/CoordSys Select Inp. Bit 1 (J2-5)	
	18	Motor/CoordSys Select Inp. Bit 2 (J2-13)	
	19	Motor/CoordSys Select Inp. Bit 3 (J2-14)	
	20	E51 Jumper (Reinitialize on power-on)	
	21	E50 Jumper: EAROM Write Enable	
	22	E49 Jumper: Wait State Control	
	23	E48 Jumper: Parity Control	
Y:$078801	0	Thumbwheel Port Inp. Bit 0 (DAT0:J3-3)	
	1	Thumbwheel Port Inp. Bit 1 (DAT1:J3-5)	
	2	Thumbwheel Port Inp. Bit 2 (DAT2:J3-7)	
	3	Thumbwheel Port Inp. Bit 3 (DAT3:J3-9)	
	4	Thumbwheel Port Inp. Bit 4 (DAT4:J3-11)	
	5	Thumbwheel Port Inp. Bit 5 (DAT5:J3-13)	
	6	Thumbwheel Port Inp. Bit 6 (DAT6:J3-15)	
	7	Thumbwheel Port Inp. Bit 7 (DAT7:J3-17)	
	8	Thumbwheel Port Out. Bit 0 (SEL0:J3-4)	
	9	Thumbwheel Port Out. Bit 1 (SEL1:J3-6)	
	10	Thumbwheel Port Out. Bit 2 (SEL2:J3-8)	
	11	Thumbwheel Port Out. Bit 3 (SEL3:J3-10)	
	12	Thumbwheel Port Out. Bit 4 (SEL4:J3-12)	
	13	Thumbwheel Port Out. Bit 5 (SEL5:J3-14)	
	14	Thumbwheel Port Out. Bit 6 (SEL6:J3-16)	
	15	Thumbwheel Port Out. Bit 7 (SEL7:J3-18)	
	16	Jumper E40: Software Card Address Bit 0	
	17	Jumper E41: Software Card Address Bit 1	
	18	Jumper E42: Software Card Address Bit 2	
	19	Jumper E43: Software Card Address Bit 3	
	20	Jumper E44: Baud Rate Select Bit 0	
	21	Jumper E45: Baud Rate Select Bit 1	
	22	Jumper E46: Baud Rate Select Bit 2	
	23	Jumper E47: Baud Rate Select Bit 3	
Y:$078802	0	Machine Input 1 (MI1)	(J5-15)
	1	Machine Input 2 (MI2)	(J5-13)
	2	Machine Input 3 (MI3)	(J5-11)
	3	Machine Input 4 (MI4)	(J5-9)
	4	Machine Input 5 (MI5)	(J5-7)
	5	Machine Input 6 (MI6)	(J5-5)
	6	Machine Input 7 (MI7)	(J5-3)
	7	Machine Input 8 (MI8)	(J5-1)
	8	Machine Output 1 (MO1)	(J5-31)
	9	Machine Output 2 (MO2)	(J5-29)
	10	Machine Output 3 (MO3)	(J5-27)
	11	Machine Output 4 (MO4)	(J5-25)
	12	Machine Output 5 (MO5)	(J5-23)
	13	Machine Output 6 (MO6)	(J5-21)
	14	Machine Output 7 (MO7)	(J5-19)
	15	Machine Output 8 (MO8)	(J5-17)
	16	D_RS Line (dedicated use)	
	17	Read/Write Line (dedicated use)	
	18	EACLK (EAROM Clock -- dedicated)	

19	ENA422 (Enable RS422 -- dedicated)
20	INPOS (In Position Status Line)
21	BFUL (Buffer Full Status Line)
22	EROR (Error Status Line)
23	F1ER (Following Error Status Line)

Turbo PMAC2 Option 12 A/D Register

`Y:$078800` Option 12 Multiplexed A/D register
Bits 0-11: Option 12; Bits 12-23: Option 12A
Configuration Register (write-only)
Converted Data (read-only)

3U Turbo PMAC2 Stack I/O Registers

`Y:$078800` ACC-1E/6E Analog-to-Digital Converters (low 12 bits)
Write operation: channel select (Channels 0-7) and mode;
Read operation: converted value of selected channel
ACC-1E/6E Analog-to-Digital Converters (high 12 bits)
Write operation: channel select (Channels 8-15) and mode;
Read operation: converted value of selected channel

`Y:$078800` ACC-3E/4E Board with Jumper E1 selected
`Y:$078900` ACC-3E/4E Board with Jumper E2 selected
`Y:$078A00` ACC-3E/4E Board with Jumper E3 selected
`Y:$078B00` ACC-3E/4E Board with Jumper E4 selected

Bits:	0 – 7	ACC-4E IN00 – IN07
	0 – 7	ACC-3E I/O00 – I/O07
	8 – 15	ACC-3E I/O48 – I/O55
	16 – 23	ACC-3E I/O96 – I/O103

`Y:$078801` ACC-3E/4E Board with Jumper E1 selected
`Y:$078901` ACC-3E/4E Board with Jumper E2 selected
`Y:$078A01` ACC-3E/4E Board with Jumper E3 selected
`Y:$078B01` ACC-3E/4E Board with Jumper E4 selected

Bits:	0 – 7	ACC-4E IN08 – IN15
	0 – 7	ACC-3E I/O08 – I/O15
	8 – 15	ACC-3E I/O56 – I/O63
	16 – 23	ACC-3E I/O104 – I/O111

`Y:$078802` ACC-3E/4E Board with Jumper E1 selected
`Y:$078902` ACC-3E/4E Board with Jumper E2 selected
`Y:$078A02` ACC-3E/4E Board with Jumper E3 selected
`Y:$078B02` ACC-3E/4E Board with Jumper E4 selected

Bits:	0 – 7	ACC-4E IN16 – IN23
	0 – 7	ACC-3E I/O16 – I/O23
	8 – 15	ACC-3E I/O64 – I/O71
	16 – 23	ACC-3E I/O112 – I/O119

`Y:$078803` ACC-3E/4E Board with Jumper E1 selected
`Y:$078903` ACC-3E/4E Board with Jumper E2 selected
`Y:$078A03` ACC-3E/4E Board with Jumper E3 selected
`Y:$078B03` ACC-3E/4E Board with Jumper E4 selected

Bits:	0 – 7	ACC-4E IN24 – IN31
	0 – 7	ACC-3E I/O24 – I/O31
	8 – 15	ACC-3E I/O72 – I/O79

		16 – 23 ACC-3E I/O120 – I/O127
Y:$078804		ACC-3E/4E Board with Jumper E1 selected
Y:$078904		ACC-3E/4E Board with Jumper E2 selected
Y:$078A04		ACC-3E/4E Board with Jumper E3 selected
Y:$078B04		ACC-3E/4E Board with Jumper E4 selected
	Bits:	0 – 7 ACC-4E IN32 – IN39
		0 – 7 ACC-3E I/O32 – I/O39
		8 – 15 ACC-3E I/O80 – I/O87
		16 – 23 ACC-3E I/O128 – I/O135
Y:$078805		ACC-3E/4E Board with Jumper E1 selected
Y:$078905		ACC-3E/4E Board with Jumper E2 selected
Y:$078A05		ACC-3E/4E Board with Jumper E3 selected
Y:$078B05		ACC-3E/4E Board with Jumper E4 selected
	Bits:	0 – 7 ACC-4E IN40 – IN47
		0 – 7 ACC-3E I/O40 – I/O47
		8 – 15 ACC-3E I/O88 – I/O95
		16 – 23 ACC-3E I/O136 – I/O143

JEXP Expansion Port I/O Registers

X/Y:$078A00	1st Expansion Port I/O Base Address (CS04)
Y:$078A00	1st Accessory-36 (configuration – write only, data – read only)
Y:$078A00	1st Accessory-14 Port A
Y:$078A01	1st Accessory-14 Port B
Y:$078A02	1st Accessory-14 Multimodule
Y:$078A03	1st Accessory-14 Control Word
X/Y:$078B00	2nd Expansion Port I/O Base Address (CS06)
Y:$078B00	2nd Accessory-36 (configuration – write only, data – read only)
Y:$078B00	2nd Accessory-14 Port A
Y:$078B01	2nd Accessory-14 Port B
Y:$078B02	2nd Accessory-14 Multimodule
Y:$078B03	2nd Accessory-14 Control Word
X/Y:$078C00	3rd Expansion Port I/O Base Address (CS10)
Y:$078C00	3rd Accessory-36 (configuration – write only, data – read only)
Y:$078C00	3rd Accessory-14 Port A
Y:$078C01	3rd Accessory-14 Port B
Y:$078C02	3rd Accessory-14 Multimodule
Y:$078C03	3rd Accessory-14 Control Word
X/Y:$078D00	4th Expansion Port I/O Base Address (CS12)
Y:$078D00	4th Accessory-36 (configuration – write only, data – read only)
Y:$078D01	4th Accessory-14 Port A
Y:$078D02	4th Accessory-14 Port B
Y:$078D03	4th Accessory-14 Multimodule
Y:$078D04	4th Accessory-14 Control Word
X/Y:$078E00	5th Expansion Port I/O Base Address (CS14)
Y:$078E00	5th Accessory-36 (configuration – write only, data – read only)
Y:$078E00	5th Accessory-14 Port A
Y:$078E01	5th Accessory-14 Port B
Y:$078E02	5th Accessory-14 Multimodule

Y:$078E03	5th Accessory-14 Control Word
X/Y:$078F00	6th Expansion Port I/O Base Address (CS16)
Y:$078F00	6th Accessory-36 (configuration – write only, data – read only)
Y:$078F00	6th Accessory-14 Port A
Y:$078F01	6th Accessory-14 Port B
Y:$078F02	6th Accessory-14 Multimodule
Y:$078F03	6th Accessory-14 Control Word

UMAC UBUS Expansion Port I/O Registers

Note:

Presently, ACC-9E, 10E, 11E, and 12E boards make no distinction between A, B, C, and D base addresses, because they do not use address lines A13 and A12.. If one of these boards is set up for a certain base address 0, 2, 4, or 6, it will respond to any of the four possible settings for this address (A, B, C, or D), and no other board may be placed at any of the settings for this numerical base address.

X/Y:$078C00	UBUS Port I/O Base Address 0A (CS10, A13=0, A12=0)
X/Y:$078D00	UBUS Port I/O Base Address 2A (CS12, A13=0, A12=1)
X/Y:$078E00	UBUS Port I/O Base Address 4A (CS14, A13=1, A12=0)
X/Y:$078F00	UBUS Port I/O Base Address 6A (CS16, A13=1, A12=1)
X/Y:$079C00	UBUS Port I/O Base Address 0B (CS10, A13=0, A12=0)
X/Y:$079D00	UBUS Port I/O Base Address 2B (CS12, A13=0, A12=1)
X/Y:$079E00	UBUS Port I/O Base Address 4B (CS14, A13=1, A12=0)
X/Y:$079F00	UBUS Port I/O Base Address 6B (CS16, A13=1, A12=1)
X/Y:$07AC00	UBUS Port I/O Base Address 0C (CS10, A13=0, A12=0)
X/Y:$07AD00	UBUS Port I/O Base Address 2C (CS12, A13=0, A12=1)
X/Y:$07AE00	UBUS Port I/O Base Address 4C (CS14, A13=1, A12=0)
X/Y:$07AF00	UBUS Port I/O Base Address 6C (CS16, A13=1, A12=1)
X/Y:$07BC00	UBUS Port I/O Base Address 0D (CS10, A13=0, A12=0)
X/Y:$07BD00	UBUS Port I/O Base Address 2D (CS12, A13=0, A12=1)
X/Y:$07BE00	UBUS Port I/O Base Address 4D (CS14, A13=1, A12=0)
X/Y:$07BF00	UBUS Port I/O Base Address 6D (CS16, A13=1, A12=1)

TURBO PMAC MATHEMATICAL FEATURES

Mathematical Operators

+

Function: Addition

The + sign implements the addition of the numerical values preceding and following it.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

The + sign may not be used as a "unary" operator to emphasize that the positive value of the following variable or constant is to be used (e.g. **P1=+P2**). This syntax will be rejected with an error.

Execution time, 80 MHz CPU: 2.9 µsec interpreted, 1.2 µsec compiled floating-point, 0.08 µsec compiled fixed-point

-

Function: Subtraction, negation

The - sign implements the subtraction of the numerical value following it from the numerical value preceding it. If there is no numerical value preceding it, it causes the negation of the numerical value following it.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

Execution time, 80 MHz CPU: 2.9 µsec interpreted, 1.2 µsec compiled floating-point, 0.08 µsec compiled fixed-point

*

Function: Multiplication

The * sign implements the multiplication of the numerical values preceding and following it.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

Execution time, 80 MHz CPU: 3.0 µsec interpreted, 1.0 µsec compiled floating-point, 0.13 µsec compiled fixed-point

/

Function: Division

The / sign implements the division of the numerical value preceding it by the numerical value following it. Unless the division is executed in a compiled PLC on a line with only L-variables and integers, the division operation is always a floating-point calculation (even if integer values are used). The quotient is computed as a floating-point value and used as such in subsequent calculations in the same expression; if it is then stored to an integer, it is rounded at the time of storage.

If the division operation is performed as an integer operation in a compiled PLC (only L-variables and integer constants on the line), the quotient is computed as an integer (rounded to the nearest integer value) and used as such in subsequent calculations in the same expression.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

If the divisor is equal to 0, the result will saturate at +/-2^{47} (+/-2^{23} for an integer division in a compiled PLC). No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible division-by-zero errors.

Execution time, 80 MHz CPU: 3.6 µsec interpreted, 1.6 µsec compiled floating-point, 0.73 µsec compiled fixed-point

Examples:

(L1 and M1 are integer variables; P1 is a floating-point variable)

Command	Result
`P1=10*2/3`	6.666666667
`P1=10*(2/3)`	6.666666667
`M1=10*2/3`	7
`M1=10*(2/3)`	7
`L1=10*2/3`	7
`L1=10*(2/3)`	10

%

Function: Modulo (remainder)

The % sign causes the calculation of the remainder due to the division of the numerical value preceding it by the numerical value following it. Unless the division is executed in a compiled PLC on a line with only L-variables and integers, the division operation is always a floating-point calculation (even if integer values are used). The quotient is computed as a floating-point value, then truncated to the next lowest (i.e. toward -∞) integer so the remainder can be computed.

If the divisor "n" is a positive value, the modulo result is in the range $0 \leq \text{Result} < n$. If the divisor "n" is a negative value, the modulo result is in the range $-n \leq \text{Result} < n$.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

If the divisor is equal to 0, the division will saturate and the modulo result will be 0. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible division-by-zero errors.

Execution time, 80 MHz CPU: 3.3 µsec interpreted, 1.4 µsec compiled floating-point, 0.78 µsec compiled fixed-point

Examples:

Operation	Result
`11%4`	3
`-11%4`	1
`11%-4`	3
`-11%-4`	-3
`3%2.5`	0.5
`-3%2.5`	2
`3%-2.5`	-2
`-3%-2.5`	2

&

Function: Bit-by-bit "and"

The & sign implements the bit-by-bit logical "and" of the numerical value preceding it and the numerical value following it. A given bit of the result is equal to 1 if and only if the matching bits of both operands are equal to 1. The operation is done both on integer bits and fractional bits (if any).

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

This bit-by-bit "and" operator that logically combines the bits of numerical values is not to be confused with the **AND** command, which logically combines conditions.

Execution time, 80 MHz CPU: 3.4 µsec interpreted, 1.4 µsec compiled floating-point, 0.10 µsec compiled fixed-point

Examples:

Operation	Result
3&1	1
3&2	2
3&3	3
3&4	0
3&-3	1
0.875&1.75	0.75
0.875&-1.75	0.25

|

Function: Bit-by-bit "or"

The | sign implements the bit-by-bit logical "or" of the numerical value preceding it and the numerical value following it. A given bit of the result is equal to 1 if the matching bit of either operand is equal to 1. The operation is done both on integer bits and fractional bits (if any).

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

This bit-by-bit "or" operator that logically combines the bits of numerical values is not to be confused with the **OR** command, which logically combines conditions.

Execution time, 80 MHz CPU: 3.2 µsec interpreted, 1.4 µsec compiled floating-point, 0.10 µsec compiled fixed-point

Examples:

Operation	Result
4|3	7
3|2	3
3|3	3
$F0|$4	$F4
3|-3	-1
0.5|0.375	0.875
0.875|-1.75	-0.375

^

Function: Bit-by-bit "exclusive or"

The ^ sign implements the bit-by-bit logical "exclusive or" (xor) of the numerical value preceding it and the numerical value following it. A given bit of the result is equal to 1 if and only if the matching bits of the two operands are different from each other. The operation is done both on integer bits and fractional bits (if any).

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

Execution time, 80 MHz CPU: 3.1 µsec interpreted, 1.4 µsec compiled floating-point, 0.10 µsec compiled fixed-point

Examples:

Operation	Result
2^1	3
2^2	0
5^7	2
$AA^$55	$FF
3^-3	-2
0.5^0.875	0.375

Mathematical Functions

ABS

Function: Absolute value

Syntax: **ABS({expression})**

Domain: All real numbers

Domain units: User-determined

Range: Non-negative real numbers

Range units: User-determined

ABS implements the absolute value, or magnitude, function, of the mathematical expression contained inside the following parentheses.

Execution time, 80 MHz CPU: 3.1 µsec interpreted, 0.6 µsec compiled

Examples:

```
P8=ABS(P7)                    ; Computes magnitude of P7
IF(Q200!=0)                   ; Divide by 0 check
  Q240=ABS(Q200)/Q200         ; Computes sign (-1 or 1) of Q200
ELSE
  Q240=0                      ; Sign value is 0
ENDIF
```

ACOS

Function: Trigonometric arc-cosine

Syntax: **ACOS({expression})**

Domain: -1.0 to +1.0

Domain units: none

Range: 0 to Pi radians (0 to 180 degrees)

Range units: Radians/degrees

ACOS implements the inverse cosine, or arc-cosine, function, of the mathematical expression contained inside the following parentheses.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

If the argument inside the parentheses is outside of the legal domain of –1.0 to +1.0, an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Execution time, 80 MHz CPU: 6.7 µsec interpreted, 4.3 µsec compiled

Examples:

```
P50=ACOS(P48/P49)          ; Computes angle whose cos is P48/P49
C(ACOS(Q70/10))            ; Move C axis to specified angle
```

ASIN

Function: Trigonometric arc-sine

Syntax: **ASIN({expression})**

Domain: -1.0 to +1.0

Domain units: none

Range: 0 to Pi radians (0 to 180 degrees)

Range units: Radians/degrees

ASIN implements the inverse sine, or arc-sine, function, of the mathematical expression contained inside the following parentheses.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

If the argument inside the parentheses is outside of the legal domain of –1.0 to +1.0, an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Execution time, 80 MHz CPU: 7.0 µsec interpreted, 4.7 µsec compiled

Examples:

```
P50=ASIN(P48/P49)          ; Computes angle whose sin is P48/P49
C(ASIN(Q70/10))            ; Move C axis to specified angle
```

ATAN

Function: Trigonometric arc-tangent

Syntax: **ATAN({expression})**

Domain: All real numbers

Domain units: none

Range: -Pi/2 to +Pi/2 radians (-90 to +90 degrees)

Range units: Radians/degrees

ATAN implements the standard inverse tangent, or arc-tangent, function, of the mathematical expression contained inside the following parentheses. This standard arc-tangent function returns values only in the +/-90-degree range; if a full +/-180-degree range is desired, the **ATAN2** function should be used instead.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

Execution time, 80 MHz CPU: 5.9 µsec interpreted, 3.5 µsec compiled

Examples:

```
P50=ATAN(P48/P49)          ; Computes angle whose tan is P48/P49
C(ATAN(Q70/10))            ; Move C axis to specified angle
```

ATAN2

Function: Two-argument trigonometric arc-tangent

Syntax: **ATAN2({expression})**

Domain: All real numbers in both arguments

Domain units: none

Range: -Pi to +Pi radians (-180 to +180 degrees)

Range units: Radians/degrees

ATAN2 implements the expanded (two-argument) inverse tangent, or arc-tangent, function, of the mathematical expression contained inside the following parentheses, and the value of variable Q0 for the coordinate system used. (If this function is used inside a PLC program, make sure the desired coordinate system has been selected with the **ADDRESS** command.)

This expanded arc-tangent function returns values in the full +/-180-degree range; if only the +/-90-degree range is desired, the standard **ATAN** function should be used instead. The **ATAN2** function makes use of the signs of both arguments, as well as the ratio of their magnitudes, to extend the range to a full 360 degrees. The value in the parentheses following **ATAN2** is the "sine" argument; the value in Q0 is the "cosine" argument.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

If both arguments for the **ATAN2** function are equal to exactly 0.0, an internal division-by-zero error will result, and an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for these possible domain errors.

Execution time, 80 MHz CPU: 5.1 µsec interpreted, 3.5 µsec compiled

Examples:

```
Q30=-0.707                 ; "Cosine" argument
Q31=-0.707                 ; "Sine" argument
Q32=ATAN(Q31/Q30)          ; Single-argument arctangent
Q32                        ; Query resulting value
45                         ; Returns value in +/-90 range
Q0=Q30                     ; Prepare "cosine" for ATAN2
Q33=ATAN2(Q31)             ; Two-argument arctangent
Q33                        ; Query resulting value
-135                       ; Note different result

Q0=M163-M161               ; X target – X present position
Q1=M263-M261               ; Y target – Y present position
IF (ABS(Q0)>0.001 OR ABS(Q1)>0.001)  ; Div by 0 check
  Q2=ATAN2(Q1)             ; Calculate directed angle
ENDIF
```

COS

Function: Trigonometric cosine

Syntax: **COS({expression})**

Domain: All real numbers

Domain units: Radians/degrees

Range: -1.0 to +1.0

Range units: none

COS implements the trigonometric cosine function of the mathematical expression contained inside the following parentheses.

This function interprets its argument in degrees if I15 is set to the default value of 0; it interprets its argument in radians if I15 is set to 1.

Execution time, 80 MHz CPU: 5.6 µsec interpreted, 3.2 µsec compiled

Examples:

```
P60=COS(30)              ; Computes cosine of 30
X(Q80*COS(Q81))          ; Move X axis to calculated value
```

EXP

Function: Exponentiation (e^x)

Syntax: **EXP({expression})**

Domain: All real numbers

Domain units: User-determined

Range: Positive real numbers

Range units: User-determined

EXP implements the standard exponentiation function of the mathematical expression contained inside the following parentheses, raising "e" to the power of this expression.

To implement the y^x function, use $e^{x\ ln(y)}$ instead.

Execution time, 80 MHz CPU: 5.6 µsec interpreted, 3.2 µsec compiled

Examples:

```
P20=EXP(P19)             ; Raises e to the power of P19
P3=EXP(P2*LN(P1))        ; Raises P1 to the power of P2
```

INT

Function: Truncation to integer

Syntax: **INT({expression})**

Domain: All real numbers

Domain units: User-determined

Range: All integers

Range units: User-determined

INT implements the truncation to integer function of the mathematical expression contained inside the following parentheses. The truncation is always done in the negative direction.

Note that while the result is an integer number, it is still represented as a floating-point value.

Execution time, 80 MHz CPU: 3.3 µsec interpreted, 1.0 µsec compiled

Examples:

```
P50=2.5                  ;
P51=INT(P50)             ; Take INT of positive value
P51                      ; Query resulting value
2                        ; Next lower integer value
P52=INT(-P50)            ; Take INT of negative value
P52                      ; Query resulting value
-3                       ; Next lower integer value
```

LN

Function: Natural logarithm

Syntax: **EXP({expression})**

Domain: Positive real numbers

Domain units: User-determined

Range: All real numbers

Range units: User-determined

LN implements the natural logarithm (logarithm base "e") function of the mathematical expression contained inside the following parentheses.

To implement the logarithm using another base, divide the natural logarithm of the value by the natural logarithm of the base ($\log_y x = \ln x / \ln y$). The natural logarithm of 10 is equal to 2.302585.

If the argument inside the parentheses is outside of the legal domain of positive numbers, a 0 value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Execution time, 80 MHz CPU: 4.3 µsec interpreted, 1.4 µsec compiled

Examples:

```
P19=LN(P20)              ; Takes the natural log of P20
P6=LN(P5)/LN(10)         ; Takes the log base 10 of P5
```

SIN

Function: Trigonometric sine

Syntax: **SIN({expression})**

Domain: All real numbers

Domain units: Radians/degrees

Range: -1.0 to +1.0

Range units: none

SIN implements the trigonometric sine function of the mathematical expression contained inside the following parentheses.

This function interprets its argument in degrees if I15 is set to the default value of 0; it interprets its argument in radians if I15 is set to 1.

Execution time, 80 MHz CPU: 5.6 µsec interpreted, 3.2 µsec compiled

Examples:

```
P60=SIN(30)              ; Computes cosine of 30
Y(Q80*SIN(Q81))          ; Move Y axis to calculated value
```

SQRT

Function: Square root

Syntax: **SQRT({expression})**

Domain: Non-negative real numbers

Domain units: User-determined

Range: Non-negative real numbers

Range units: User-determined

SQRT implements the positive square-root function of the mathematical expression contained inside the following parentheses.

If the argument inside the parentheses is outside of the legal domain of non-negative numbers, an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Execution time, 80 MHz CPU: 3.7 µsec interpreted, 1.4 µsec compiled

Examples:

```
P19=SQRT(P20)                ; Takes the square root of P20
P50=SQRT(P8*P8+P9*P9)        ; Pythagorean theorem calculation
```

TAN

Function:	Trigonometric tangent
Syntax:	**TAN({expression})**
Domain:	All real numbers except +/-(2N-1)*90 degrees
Domain units:	Radians/degrees
Range:	All real numbers
Range units:	none

TAN implements the trigonometric tangent function of the mathematical expression contained inside the following parentheses.

This function interprets its argument in degrees if I15 is set to the default value of 0; it interprets its argument in radians if I15 is set to 1.

If the argument inside the parentheses approaches +/-(2N-1)*90 degrees (+/-90, 270, 450, etc.), the **TAN** function will "blow up" and a very large value will be returned. If the argument inside the parentheses is exactly equal to one of these quantities, an internal division-by-zero error will occur and the resulting value will saturate at $+/-2^{47}$. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Execution time, 80 MHz CPU: 6.8µsec interpreted, 4.5 µsec compiled

Examples:

```
P60=TAN(30)                  ; Computes cosine of 30
Y(Q80*TAN(Q81))              ; Move Y axis to calculated value
```

TURBO PMAC(1) SUGGESTED M-VARIABLE DEFINITIONS

; This file contains suggested definitions for M-variables on the Turbo PMAC(1). Note that these are only suggestions; the user is free to make whatever definitions are desired.

```
; Clear existing definitions
CLOSE                       ; Make sure no buffer is open
M0..8191->*                 ; All M-variables are now self-referenced
M0->X:$000000,0,24,U        ; Servo cycle counter
; JOPTO Port General Purpose inputs and outputs
M1->Y:$078802,8,1           ; Machine Output 1
M2->Y:$078802,9,1           ; Machine Output 2
M3->Y:$078802,10,1          ; Machine Output 3
M4->Y:$078802,11,1          ; Machine Output 4
M5->Y:$078802,12,1          ; Machine Output 5
M6->Y:$078802,13,1          ; Machine Output 6
M7->Y:$078802,14,1          ; Machine Output 7
M8->Y:$078802,15,1          ; Machine Output 8
M9->Y:$078802,8,8,U         ; Machine Outputs 1-8 treated as byte
M11->Y:$078802,0,1          ; Machine Input 1
M12->Y:$078802,1,1          ; Machine Input 2
M13->Y:$078802,2,1          ; Machine Input 3
M14->Y:$078802,3,1          ; Machine Input 4
M15->Y:$078802,4,1          ; Machine Input 5
M16->Y:$078802,5,1          ; Machine Input 6
M17->Y:$078802,6,1          ; Machine Input 7
M18->Y:$078802,7,1          ; Machine Input 8
M19->Y:$078802,0,8,U        ; Machine Inputs 1-8 treated as byte
; Control-Panel Port Input Bits (so can be used as general I/O if I2=1)
M20->Y:$078800,8,1          ; Jog Minus Input
M21->Y:$078800,9,1          ; Jog Plus Input
M22->Y:$078800,10,1         ; Prejog Input
M23->Y:$078800,11,1         ; Start (Run) Input
M24->Y:$078800,12,1         ; Step/Quit Input
M25->Y:$078800,13,1         ; Stop (Abort) Input
M26->Y:$078800,14,1         ; Home Command Input
M27->Y:$078800,15,1         ; Feed Hold Input
M28->Y:$078800,16,1         ; Motor/C.S. Select Input Bit 0
M29->Y:$078800,17,1         ; Motor/C.S. Select Input Bit 1
M30->Y:$078800,18,1         ; Motor/C.S. Select Input Bit 2
M31->Y:$078800,19,1         ; Motor/C.S. Select Input Bit 3
M32->Y:$078800,16,4,C       ; Selected Motor/C.S. Number
; Thumbwheel Port Bits (So they can be used as general-purpose I/O)
M40->Y:$078801,8,1          ; SEL0 Output
M41->Y:$078801,9,1          ; SEL1 Output
M42->Y:$078801,10,1         ; SEL2 Output
M43->Y:$078801,11,1         ; SEL3 Output
M44->Y:$078801,12,1         ; SEL4 Output
```

```
M45->Y:$078801,13,1          ; SEL5 Output
M46->Y:$078801,14,1          ; SEL6 Output
M47->Y:$078801,15,1          ; SEL7 Output
M48->Y:$078801,8,8,U         ; SEL0-7 Outputs treated as a byte
M50->Y:$078801,0,1           ; DAT0 Input
M51->Y:$078801,1,1           ; DAT1 Input
M52->Y:$078801,2,1           ; DAT2 Input
M53->Y:$078801,3,1           ; DAT3 Input
M54->Y:$078801,4,1           ; DAT4 Input
M55->Y:$078801,5,1           ; DAT5 Input
M56->Y:$078801,6,1           ; DAT6 Input
M57->Y:$078801,7,1           ; DAT7 Input
M58->Y:$078801,0,8,U         ; DAT0-7 Inputs treated as a byte
; Miscellaneous global registers
M70->X:$FFFF8C,0,24          ; Time between phase interrupts (CPU cycles/2)
M71->X:$000037,0,24          ; Time for phase tasks (CPU cycles/2)
M72->Y:$000037,0,24          ; Time for servo tasks (CPU cycles/2)
M73->X:$00000B,0,24          ; Time for RTI tasks (CPU cycles/2)
M80->X:$000025,0,24          ; Minimum watchdog timer count
M81->X:$000024,0,24          ; Pointer to display buffer
M82->Y:$001080,0,8           ; First character of display buffer
M83->X:$000006,12,1          ; Firmware checksum error bit
M84->X:$000006,13,1          ; Any memory checksum error bit
M85->X:$000006,5,1           ; MACRO auxiliary communications error bit
M86->X:$000006,6,1           ; ACC-34 serial parity error bit
; VME/DPRAM active setup registers
M90->X:$070006,0,8      ; VME Active Address Modifier (Bits 0-7; from I90)
M91->X:$070007,0,8      ; VME Active Address Modifier Don't Care Bits
                        ; (Bits 0-7; from I91)
M92->X:$070008,0,8      ; VME Active Base Address Bits A31-A24 (Bits 0-7; from I92)
M93->X:$070009,0,8      ; VME Active Mailbox Base Address Bits A23-A16
                        ; ISA Active DPRAM Base Address bits A23-A16
                        ; (Bits 0-7; from I93)
M94->X:$07000A,0,8      ; VME Active Mailbox Base Address Bits A15-A08
                        ; ISA Active DPRAM Base Address bits A15-A14, Enable &
                        ; Bank Select
                        ; (Bits 0-7; from I94)
M95->X:$07000B,0,8      ; VME Active Interrupt Level (Bits 0-7; from I95)
M96->X:$07000C,0,8      ; VME Active Interrupt Vector (Bits 0-7; from I96)
M97->X:$07000D,0,8      ; VME Active DPRAM Base Address Bits A23-A20
                        ; (Bits 0-7; from I97)
M98->X:$07000E,0,8      ; VME Active DPRAM Enable State (Bits 0-7; from I98)
M99->X:$07000F,0,8      ; VME Active Address Width Control (Bits 0-7; from I99)
; Servo IC 0 Registers for PMAC(1) Channel 1 (usually for Motor #1)
M101->X:$078001,0,24,S       ; ENC1 24-bit counter position
M102->Y:$078003,8,16,S       ; DAC1 16-bit analog output
M103->X:$078003,0,24,S       ; ENC1 capture/compare position register
```

```
M105->Y:$078006,8,16,S      ; ADC1 16-bit analog input
M106->Y:$078000,0,24,U      ; ENC1 time between counts (SCLK cycles)
M110->X:$078000,10,1        ; ENC1 count-write enable control
M111->X:$078000,11,1        ; EQU1 compare flag latch control
M112->X:$078000,12,1        ; EQU1 compare output enable
M113->X:$078000,13,1        ; EQU1 compare invert enable
M114->X:$078000,14,1        ; AENA1/DIR1 Output
M116->X:$078000,16,1        ; EQU1 compare flag
M117->X:$078000,17,1        ; ENC1 position-captured flag
M118->X:$078000,18,1        ; ENC1 Count-error flag
M119->X:$078000,19,1        ; ENC1 3rd channel input status
M120->X:$078000,20,1        ; HMFL1 input status
M121->X:$078000,21,1        ; -LIM1 (positive end) input status
M122->X:$078000,22,1        ; +LIM1 (negative end) input status
M123->X:$078000,23,1        ; FAULT1 input status
; Motor #1 Status Bits
M130->Y:$0000C0,11,1        ; #1 Stopped-on-position-limit bit
M131->X:$0000B0,21,1        ; #1 Positive-end-limit-set bit
M132->X:$0000B0,22,1        ; #1 Negative-end-limit-set bit
M133->X:$0000B0,13,1        ; #1 Desired-velocity-zero bit
M135->X:$0000B0,15,1        ; #1 Dwell-in-progress bit
M137->X:$0000B0,17,1        ; #1 Running-program bit
M138->X:$0000B0,18,1        ; #1 Open-loop-mode bit
M139->X:$0000B0,19,1        ; #1 Amplifier-enabled status bit
M140->Y:$0000C0,0,1         ; #1 Background in-position bit
M141->Y:$0000C0,1,1         ; #1 Warning-following error bit
M142->Y:$0000C0,2,1         ; #1 Fatal-following-error bit
M143->Y:$0000C0,3,1         ; #1 Amplifier-fault-error bit
M144->Y:$0000C0,13,1        ; #1 Foreground in-position bit
M145->Y:$0000C0,10,1        ; #1 Home-complete bit
M146->Y:$0000C0,6,1         ; #1 Integrated following error fault bit
M147->Y:$0000C0,5,1         ; #1 I2T fault bit
M148->Y:$0000C0,8,1         ; #1 Phasing error fault bit
M149->Y:$0000C0,9,1         ; #1 Phasing search-in-progress bit
; Motor #1 Move Registers
M161->D:$000088             ; #1 Commanded position (1/[Ixx08*32] cts)
M162->D:$00008B             ; #1 Actual position (1/[Ixx08*32] cts)
M163->D:$0000C7             ; #1 Target (end) position (1/[Ixx08*32] cts)
M164->D:$0000CC             ; #1 Position bias (1/[Ixx08*32] cts)
M166->X:$00009D,0,24,S      ; #1 Actual velocity (1/[Ixx09*32] cts/cyc)
M167->D:$00008D             ; #1 Present master pos (1/[Ixx07*32] cts)
M168->X:$0000BF,8,16,S      ; #1 Filter Output (16-bit DAC bits)
M169->D:$000090             ; #1 Compensation correction (1/[Ixx08*32] cts)
M170->D:$0000B4             ; #1 Present phase position (including fraction)
M171->X:$0000B4,24,S        ; #1 Present phase position (counts *Ixx70)
M172->L:$0000D7             ; #1 Variable jog position/distance (cts)
M173->Y:$0000CE,0,24,S      ; #1 Encoder home capture position (cts)
```

```
M174->D:$0000EF                ; #1 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M175->X:$0000B9,8,16,S         ; #1 Actual quadrature current
M176->Y:$0000B9,8,16,S         ; #1 Actual direct current
M177->X:$0000BC,8,16,S         ; #1 Quadrature current-loop integrator output
M178->Y:$0000BC,8,16,S         ; #1 Direct current-loop integrator output
M179->X:$0000AE,8,16,S         ; #1 PID internal filter result (16-bit DAC bits)
; Motor #1 Axis Definition Registers
M191->L:$0000CF                ; #1 X/U/A/B/C-Axis scale factor (cts/unit)
M192->L:$0000D0                ; #1 Y/V-Axis scale factor (cts/unit)
M193->L:$0000D1                ; #1 Z/W-Axis scale factor (cts/unit)
M194->L:$0000D2                ; #1 Axis offset (cts)
; Servo IC 0 Registers for PMAC(1) Channel 2 (usually for Motor #2)
M201->X:$078005,0,24,S         ; ENC2 24-bit counter position
M202->Y:$078002,8,16,S         ; DAC2 16-bit analog output
M203->X:$078007,0,24,S         ; ENC2 capture/compare position register
M205->Y:$078007,8,16,S         ; ADC2 16-bit analog input
M206->Y:$078004,0,24,U         ; ENC2 time between counts (SCLK cycles)
M210->X:$078004,10,1           ; ENC2 count-write enable control
M211->X:$078004,11,1           ; EQU2 compare flag latch control
M212->X:$078004,12,1           ; EQU2 compare output enable
M213->X:$078004,13,1           ; EQU2 compare invert enable
M214->X:$078004,14,1           ; AENA2/DIR2 Output
M216->X:$078004,16,1           ; EQU2 compare flag
M217->X:$078004,17,1           ; ENC2 position-captured flag
M218->X:$078004,18,1           ; ENC2 Count-error flag
M219->X:$078004,19,1           ; ENC2 3rd channel input status
M220->X:$078004,20,1           ; HMFL2 input status
M221->X:$078004,21,1           ; -LIM2 (positive end) input status
M222->X:$078004,22,1           ; +LIM2 (negative end) input status
M223->X:$078004,23,1           ; FAULT2 input status
; Motor #2 Status Bits
M230->Y:$000140,11,1           ; #2 Stopped-on-position-limit bit
M231->X:$000130,21,1           ; #2 Positive-end-limit-set bit
M232->X:$000130,22,1           ; #2 Negative-end-limit-set bit
M233->X:$000130,13,1           ; #2 Desired-velocity-zero bit
M235->X:$000130,15,1           ; #2 Dwell-in-progress bit
M237->X:$000130,17,1           ; #2 Running-program bit
M238->X:$000130,18,1           ; #2 Open-loop-mode bit
M239->X:$000130,19,1           ; #2 Amplifier-enabled status bit
M240->Y:$000140,0,1            ; #2 Background in-position bit
M241->Y:$000140,1,1            ; #2 Warning-following error bit
M242->Y:$000140,2,1            ; #2 Fatal-following-error bit
M243->Y:$000140,3,1            ; #2 Amplifier-fault-error bit
M244->Y:$000140,13,1           ; #2 Foreground in-position bit
M245->Y:$000140,10,1           ; #2 Home-complete bit
M246->Y:$000140,6,1            ; #2 Integrated following error fault bit
M247->Y:$000140,5,1            ; #2 I2T fault bit
```

```
M248->Y:$000140,8,1          ; #2 Phasing error fault bit
M249->Y:$000140,9,1          ; #2 Phasing search-in-progress bit
; Motor #2 Move Registers
M261->D:$000108              ; #2 Commanded position (1/[Ixx08*32] cts)
M262->D:$00010B              ; #2 Actual position (1/[Ixx08*32] cts)
M263->D:$000147              ; #2 Target (end) position (1/[Ixx08*32] cts)
M264->D:$00014C              ; #2 Position bias (1/[Ixx08*32] cts)
M266->X:$00011D,0,24,S       ; #2 Actual velocity (1/[Ixx09*32] cts/cyc)
M267->D:$00010D              ; #2 Present master pos (1/[Ixx07*32] cts)
M268->X:$00013F,8,16,S       ; #2 Filter Output (16-bit DAC bits)
M269->D:$000110              ; #2 Compensation correction (1/[Ixx08*32] cts)
M270->D:$000134              ; #2 Present phase position (including fraction)
M271->X:$000134,24,S         ; #2 Present phase position (counts *Ixx70)
M272->L:$000157              ; #2 Variable jog position/distance (cts)
M273->Y:$00014E,0,24,S       ; #2 Encoder home capture position (cts)
M274->D:$00016F              ; #2 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M275->X:$000139,8,16,S       ; #2 Actual quadrature current
M276->Y:$000139,8,16,S       ; #2 Actual direct current
M277->X:$00013C,8,16,S       ; #2 Quadrature current-loop integrator output
M278->Y:$00013C,8,16,S       ; #2 Direct current-loop integrator output
M279->X:$00012E,8,16,S       ; #2 PID internal filter result (16-bit DAC bits)
; Motor #2 Axis Definition Registers
M291->L:$00014F              ; #2 X/U/A/B/C-Axis scale factor (cts/unit)
M292->L:$000150              ; #2 Y/V-Axis scale factor (cts/unit)
M293->L:$000151              ; #2 Z/W-Axis scale factor (cts/unit)
M294->L:$000152              ; #2 Axis offset (cts)
; Servo IC 0 Registers for PMAC(1) Channel 3 (usually for Motor #3)
M301->X:$078009,0,24,S       ; ENC3 24-bit counter position
M302->Y:$07800B,8,16,S       ; DAC3 16-bit analog output
M303->X:$07800B,0,24,S       ; ENC3 capture/compare position register
M305->Y:$07800E,8,16,S       ; ADC3 16-bit analog input
M306->Y:$078008,0,24,U       ; ENC3 time between counts (SCLK cycles)
M310->X:$078008,10,1         ; ENC3 count-write enable control
M311->X:$078008,11,1         ; EQU3 compare flag latch control
M312->X:$078008,12,1         ; EQU3 compare output enable
M313->X:$078008,13,1         ; EQU3 compare invert enable
M314->X:$078008,14,1         ; AENA3/DIR3 Output
M316->X:$078008,16,1         ; EQU3 compare flag
M317->X:$078008,17,1         ; ENC3 position-captured flag
M318->X:$078008,18,1         ; ENC3 Count-error flag
M319->X:$078008,19,1         ; ENC3 3rd channel input status
M320->X:$078008,20,1         ; HMFL3 input status
M321->X:$078008,21,1         ; -LIM3 (positive end) input status
M322->X:$078008,22,1         ; +LIM3 (negative end) input status
M323->X:$078008,23,1         ; FAULT3 input status
```

```
; Motor #3 Status Bits
M330->Y:$0001C0,11,1        ; #3 Stopped-on-position-limit bit
M331->X:$0001B0,21,1        ; #3 Positive-end-limit-set bit
M332->X:$0001B0,22,1        ; #3 Negative-end-limit-set bit
M333->X:$0001B0,13,1        ; #3 Desired-velocity-zero bit
M335->X:$0001B0,15,1        ; #3 Dwell-in-progress bit
M337->X:$0001B0,17,1        ; #3 Running-program bit
M338->X:$0001B0,18,1        ; #3 Open-loop-mode bit
M339->X:$0001B0,19,1        ; #3 Amplifier-enabled status bit
M340->Y:$0001C0,0,1         ; #3 Background in-position bit
M341->Y:$0001C0,1,1         ; #3 Warning-following error bit
M342->Y:$0001C0,2,1         ; #3 Fatal-following-error bit
M343->Y:$0001C0,3,1         ; #3 Amplifier-fault-error bit
M344->Y:$0001C0,13,1        ; #3 Foreground in-position bit
M345->Y:$0001C0,10,1        ; #3 Home-complete bit
M346->Y:$0001C0,6,1         ; #3 Integrated following error fault bit
M347->Y:$0001C0,5,1         ; #3 I2T fault bit
M348->Y:$0001C0,8,1         ; #3 Phasing error fault bit
M349->Y:$0001C0,9,1         ; #3 Phasing search-in-progress bit
; Motor #3 Move Registers
M361->D:$000188             ; #3 Commanded position (1/[Ixx08*32] cts)
M362->D:$00018B             ; #3 Actual position (1/[Ixx08*32] cts)
M363->D:$0001C7             ; #3 Target (end) position (1/[Ixx08*32] cts)
M364->D:$0001CC             ; #3 Position bias (1/[Ixx08*32] cts)
M366->X:$00019D,0,24,S      ; #3 Actual velocity (1/[Ixx09*32] cts/cyc)
M367->D:$00018D             ; #3 Present master pos (1/[Ixx07*32] cts)
M368->X:$0001BF,8,16,S      ; #3 Filter Output (16-bit DAC bits)
M369->D:$000190             ; #3 Compensation correction (1/[Ixx08*32] cts)
M370->D:$0001B4             ; #3 Present phase position (including fraction)
M371->X:$0001B4,24,S        ; #3 Present phase position (counts *Ixx70)
M372->L:$0001D7             ; #3 Variable jog position/distance (cts)
M373->Y:$0001CE,0,24,S      ; #3 Encoder home capture position (cts)
M374->D:$0001EF             ; #3 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M375->X:$0001B9,8,16,S      ; #3 Actual quadrature current
M376->Y:$0001B9,8,16,S      ; #3 Actual direct current
M377->X:$0001BC,8,16,S      ; #3 Quadrature current-loop integrator output
M378->Y:$0001BC,8,16,S      ; #3 Direct current-loop integrator output
M379->X:$0001AE,8,16,S      ; #3 PID internal filter result (16-bit DAC bits)
; Motor #3 Axis Definition Registers
M391->L:$0001CF             ; #3 X/U/A/B/C-Axis scale factor (cts/unit)
M392->L:$0001D0             ; #3 Y/V-Axis scale factor (cts/unit)
M393->L:$0001D1             ; #3 Z/W-Axis scale factor (cts/unit)
M394->L:$0001D2             ; #3 Axis offset (cts)
```

```
; Servo IC 0 Registers for PMAC(1) Channel 4 (usually for Motor #4)
M401->X:$07800D,0,24,S        ; ENC4 24-bit counter position
M402->Y:$07800A,8,16,S        ; DAC4 16-bit analog output
M403->X:$07800F,0,24,S        ; ENC4 capture/compare position register
M405->Y:$07800F,8,16,S        ; ADC4 16-bit analog input
M406->Y:$07800C,0,24,U        ; ENC4 time between counts (SCLK cycles)
M410->X:$07800C,10,1          ; ENC4 count-write enable control
M411->X:$07800C,11,1          ; EQU4 compare flag latch control
M412->X:$07800C,12,1          ; EQU4 compare output enable
M413->X:$07800C,13,1          ; EQU4 compare invert enable
M414->X:$07800C,14,1          ; AENA4/DIR4 Output
M416->X:$07800C,16,1          ; EQU4 compare flag
M417->X:$07800C,17,1          ; ENC4 position-captured flag
M418->X:$07800C,18,1          ; ENC4 Count-error flag
M419->X:$07800C,19,1          ; ENC4 3rd channel input status
M420->X:$07800C,20,1          ; HMFL4 input status
M421->X:$07800C,21,1          ; -LIM4 (positive end) input status
M422->X:$07800C,22,1          ; +LIM4 (negative end) input status
M423->X:$07800C,23,1          ; FAULT4 input status
; Motor #4 Status Bits
M430->Y:$000240,11,1          ; #4 Stopped-on-position-limit bit
M431->X:$000230,21,1          ; #4 Positive-end-limit-set bit
M432->X:$000230,22,1          ; #4 Negative-end-limit-set bit
M433->X:$000230,13,1          ; #4 Desired-velocity-zero bit
M435->X:$000230,15,1          ; #4 Dwell-in-progress bit
M437->X:$000230,17,1          ; #4 Running-program bit
M438->X:$000230,18,1          ; #4 Open-loop-mode bit
M439->X:$000230,19,1          ; #4 Amplifier-enabled status bit
M440->Y:$000240,0,1           ; #4 Background in-position bit
M441->Y:$000240,1,1           ; #4 Warning-following error bit
M442->Y:$000240,2,1           ; #4 Fatal-following-error bit
M443->Y:$000240,3,1           ; #4 Amplifier-fault-error bit
M444->Y:$000240,13,1          ; #4 Foreground in-position bit
M445->Y:$000240,10,1          ; #4 Home-complete bit
M446->Y:$000240,6,1           ; #4 Integrated following error fault bit
M447->Y:$000240,5,1           ; #4 I2T fault bit
M448->Y:$000240,8,1           ; #4 Phasing error fault bit
M449->Y:$000240,9,1           ; #4 Phasing search-in-progress bit
; Motor #4 Move Registers
M461->D:$000208               ; #4 Commanded position (1/[Ixx08*32] cts)
M462->D:$00020B               ; #4 Actual position (1/[Ixx08*32] cts)
M463->D:$000247               ; #4 Target (end) position (1/[Ixx08*32] cts)
M464->D:$00024C               ; #4 Position bias (1/[Ixx08*32] cts)
M466->X:$00021D,0,24,S        ; #4 Actual velocity (1/[Ixx09*32] cts/cyc)
M467->D:$00020D               ; #4 Present master pos (1/[Ixx07*32] cts)
M468->X:$00023F,8,16,S        ; #4 Filter Output (16-bit DAC bits)
M469->D:$000210               ; #4 Compensation correction (1/[Ixx08*32] cts)
```

```
M470->D:$000234              ; #4 Present phase position (including fraction)
M471->X:$000234,24,S         ; #4 Present phase position (counts *Ixx70)
M472->L:$000257              ; #4 Variable jog position/distance (cts)
M473->Y:$00024E,0,24,S       ; #4 Encoder home capture position (cts)
M474->D:$00026F              ; #4 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M475->X:$000239,8,16,S       ; #4 Actual quadrature current
M476->Y:$000239,8,16,S       ; #4 Actual direct current
M477->X:$00023C,8,16,S       ; #4 Quadrature current-loop integrator output
M478->Y:$00023C,8,16,S       ; #4 Direct current-loop integrator output
M479->X:$00022E,8,16,S       ; #4 PID internal filter result (16-bit DAC bits)
; Motor #4 Axis Definition Registers
M491->L:$00024F              ; #4 X/U/A/B/C-Axis scale factor (cts/unit)
M492->L:$000250              ; #4 Y/V-Axis scale factor (cts/unit)
M493->L:$000251              ; #4 Z/W-Axis scale factor (cts/unit)
M494->L:$000252              ; #4 Axis offset (cts)
; Servo IC 1 Registers for PMAC(1) Channel 5 (usually for Motor #5)
M501->X:$078101,0,24,S       ; ENC5 24-bit counter position
M502->Y:$078103,8,16,S       ; DAC5 16-bit analog output
M503->X:$078103,0,24,S       ; ENC5 capture/compare position register
M505->Y:$078106,8,16,S       ; ADC5 16-bit analog input
M506->Y:$078100,0,24,U       ; ENC5 time between counts (SCLK cycles)
M510->X:$078100,10,1         ; ENC5 count-write enable control
M511->X:$078100,11,1         ; EQU5 compare flag latch control
M512->X:$078100,12,1         ; EQU5 compare output enable
M513->X:$078100,13,1         ; EQU5 compare invert enable
M514->X:$078100,14,1         ; AENA5/DIR5 Output
M516->X:$078100,16,1         ; EQU5 compare flag
M517->X:$078100,17,1         ; ENC5 position-captured flag
M518->X:$078100,18,1         ; ENC5 Count-error flag
M519->X:$078100,19,1         ; ENC5 3rd channel input status
M520->X:$078100,20,1         ; HMFL5 input status
M521->X:$078100,21,1         ; -LIM5 (positive end) input status
M522->X:$078100,22,1         ; +LIM5 (negative end) input status
M523->X:$078100,23,1         ; FAULT5 input status
; Motor #5 Status Bits
M530->Y:$0002C0,11,1         ; #5 Stopped-on-position-limit bit
M531->X:$0002B0,21,1         ; #5 Positive-end-limit-set bit
M532->X:$0002B0,22,1         ; #5 Negative-end-limit-set bit
M533->X:$0002B0,13,1         ; #5 Desired-velocity-zero bit
M535->X:$0002B0,15,1         ; #5 Dwell-in-progress bit
M537->X:$0002B0,17,1         ; #5 Running-program bit
M538->X:$0002B0,18,1         ; #5 Open-loop-mode bit
M539->X:$0002B0,19,1         ; #5 Amplifier-enabled status bit
M540->Y:$0002C0,0,1          ; #5 Background in-position bit
M541->Y:$0002C0,1,1          ; #5 Warning-following error bit
M542->Y:$0002C0,2,1          ; #5 Fatal-following-error bit
M543->Y:$0002C0,3,1          ; #5 Amplifier-fault-error bit
```

```
M544->Y:$0002C0,13,1          ; #5 Foreground in-position bit
M545->Y:$0002C0,10,1          ; #5 Home-complete bit
M546->Y:$0002C0,6,1           ; #5 Integrated following error fault bit
M547->Y:$0002C0,5,1           ; #5 I2T fault bit
M548->Y:$0002C0,8,1           ; #5 Phasing error fault bit
M549->Y:$0002C0,9,1           ; #5 Phasing search-in-progress bit
; Motor #5 Move Registers
M561->D:$000288               ; #5 Commanded position (1/[Ixx08*32] cts)
M562->D:$00028B               ; #5 Actual position (1/[Ixx08*32] cts)
M563->D:$0002C7               ; #5 Target (end) position (1/[Ixx08*32] cts)
M564->D:$0002CC               ; #5 Position bias (1/[Ixx08*32] cts)
M566->X:$00029D,0,24,S        ; #5 Actual velocity (1/[Ixx09*32] cts/cyc)
M567->D:$00028D               ; #5 Present master pos (1/[Ixx07*32] cts)
M568->X:$0002BF,8,16,S        ; #5 Filter Output (16-bit DAC bits)
M569->D:$000290               ; #5 Compensation correction (1/[Ixx08*32] cts)
M570->D:$0002B4               ; #5 Present phase position (including fraction)
M571->X:$0002B4,24,S          ; #5 Present phase position (counts *Ixx70)
M572->L:$0002D7               ; #5 Variable jog position/distance (cts)
M573->Y:$0002CE,0,24,S        ; #5 Encoder home capture position (cts)
M574->D:$0002EF               ; #5 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M575->X:$0002B9,8,16,S        ; #5 Actual quadrature current
M576->Y:$0002B9,8,16,S        ; #5 Actual direct current
M577->X:$0002BC,8,16,S        ; #5 Quadrature current-loop integrator output
M578->Y:$0002BC,8,16,S        ; #5 Direct current-loop integrator output
M579->X:$0002AE,8,16,S        ; #5 PID internal filter result (16-bit DAC bits)
; Motor #5 Axis Definition Registers
M591->L:$0002CF               ; #5 X/U/A/B/C-Axis scale factor (cts/unit)
M592->L:$0002D0               ; #5 Y/V-Axis scale factor (cts/unit)
M593->L:$0002D1               ; #5 Z/W-Axis scale factor (cts/unit)
M594->L:$0002D2               ; #5 Axis offset (cts)
; Servo IC 1 Registers for PMAC(1) Channel 6 (usually for Motor #6)
M601->X:$078105,0,24,S        ; ENC6 24-bit counter position
M602->Y:$078102,8,16,S        ; DAC6 16-bit analog output
M603->X:$078107,0,24,S        ; ENC6 capture/compare position register
M605->Y:$078107,8,16,S        ; ADC6 16-bit analog input
M606->Y:$078104,0,24,U        ; ENC6 time between counts (SCLK cycles)
M610->X:$078104,10,1          ; ENC6 count-write enable control
M611->X:$078104,11,1          ; EQU6 compare flag latch control
M612->X:$078104,12,1          ; EQU6 compare output enable
M613->X:$078104,13,1          ; EQU6 compare invert enable
M614->X:$078104,14,1          ; AENA6/DIR6 Output
M616->X:$078104,16,1          ; EQU6 compare flag
M617->X:$078104,17,1          ; ENC6 position-captured flag
M618->X:$078104,18,1          ; ENC6 Count-error flag
M619->X:$078104,19,1          ; ENC6 3rd channel input status
M620->X:$078104,20,1          ; HMFL6 input status
M621->X:$078104,21,1          ; -LIM6 (positive end) input status
```

```
M622->X:$078104,22,1          ; +LIM6 (negative end) input status
M623->X:$078104,23,1          ; FAULT6 input status
; Motor #6 Status Bits
M630->Y:$000340,11,1          ; #6 Stopped-on-position-limit bit
M631->X:$000330,21,1          ; #6 Positive-end-limit-set bit
M632->X:$000330,22,1          ; #6 Negative-end-limit-set bit
M633->X:$000330,13,1          ; #6 Desired-velocity-zero bit
M635->X:$000330,15,1          ; #6 Dwell-in-progress bit
M637->X:$000330,17,1          ; #6 Running-program bit
M638->X:$000330,18,1          ; #6 Open-loop-mode bit
M639->X:$000330,19,1          ; #6 Amplifier-enabled status bit
M640->Y:$000340,0,1           ; #6 Background in-position bit
M641->Y:$000340,1,1           ; #6 Warning-following error bit
M642->Y:$000340,2,1           ; #6 Fatal-following-error bit
M643->Y:$000340,3,1           ; #6 Amplifier-fault-error bit
M644->Y:$000340,13,1          ; #6 Foreground in-position bit
M645->Y:$000340,10,1          ; #6 Home-complete bit
M646->Y:$000340,6,1           ; #6 Integrated following error fault bit
M647->Y:$000340,5,1           ; #6 I2T fault bit
M648->Y:$000340,8,1           ; #6 Phasing error fault bit
M649->Y:$000340,9,1           ; #6 Phasing search-in-progress bit
; Motor #6 Move Registers
M661->D:$000308               ; #6 Commanded position (1/[Ixx08*32] cts)
M662->D:$00030B               ; #6 Actual position (1/[Ixx08*32] cts)
M663->D:$000347               ; #6 Target (end) position (1/[Ixx08*32] cts)
M664->D:$00034C               ; #6 Position bias (1/[Ixx08*32] cts)
M666->X:$00031D,0,24,S        ; #6 Actual velocity (1/[Ixx09*32] cts/cyc)
M667->D:$00030D               ; #6 Present master pos (1/[Ixx07*32] cts)
M668->X:$00033F,8,16,S        ; #6 Filter Output (16-bit DAC bits)
M669->D:$000310               ; #6 Compensation correction (1/[Ixx08*32] cts)
M670->D:$000334               ; #6 Present phase position (including fraction)
M671->X:$000334,24,S          ; #6 Present phase position (counts *Ixx70)
M672->L:$000357               ; #6 Variable jog position/distance (cts)
M673->Y:$00034E,0,24,S        ; #6 Encoder home capture position (cts)
M674->D:$00036F               ; #6 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M675->X:$000339,8,16,S        ; #6 Actual quadrature current
M676->Y:$000339,8,16,S        ; #6 Actual direct current
M677->X:$00033C,8,16,S        ; #6 Quadrature current-loop integrator output
M678->Y:$00033C,8,16,S        ; #6 Direct current-loop integrator output
M679->X:$00032E,8,16,S        ; #6 PID internal filter result (16-bit DAC bits)
; Motor #6 Axis Definition Registers
M691->L:$00034F               ; #6 X/U/A/B/C-Axis scale factor (cts/unit)
M692->L:$000350               ; #6 Y/V-Axis scale factor (cts/unit)
M693->L:$000351               ; #6 Z/W-Axis scale factor (cts/unit)
M694->L:$000352               ; #6 Axis offset (cts)
```

```
; Servo IC 1 Registers for PMAC(1) Channel 3 (usually for Motor #3)
M701->X:$078109,0,24,S          ; ENC7 24-bit counter position
M702->Y:$07810B,8,16,S          ; DAC7 16-bit analog output
M703->X:$07810B,0,24,S          ; ENC7 capture/compare position register
M705->Y:$07810E,8,16,S          ; ADC7 16-bit analog input
M706->Y:$078108,0,24,U          ; ENC7 time between counts (SCLK cycles)
M710->X:$078108,10,1            ; ENC7 count-write enable control
M711->X:$078108,11,1            ; EQU7 compare flag latch control
M712->X:$078108,12,1            ; EQU7 compare output enable
M713->X:$078108,13,1            ; EQU7 compare invert enable
M714->X:$078108,14,1            ; AENA7/DIR7 Output
M716->X:$078108,16,1            ; EQU7 compare flag
M717->X:$078108,17,1            ; ENC7 position-captured flag
M718->X:$078108,18,1            ; ENC7 Count-error flag
M719->X:$078108,19,1            ; ENC7 3rd channel input status
M720->X:$078108,20,1            ; HMFL7 input status
M721->X:$078108,21,1            ; -LIM7 (positive end) input status
M722->X:$078108,22,1            ; +LIM7 (negative end) input status
M723->X:$078108,23,1            ; FAULT7 input status
; Motor #7 Status Bits
M730->Y:$0003C0,11,1            ; #7 Stopped-on-position-limit bit
M731->X:$0003B0,21,1            ; #7 Positive-end-limit-set bit
M732->X:$0003B0,22,1            ; #7 Negative-end-limit-set bit
M733->X:$0003B0,13,1            ; #7 Desired-velocity-zero bit
M735->X:$0003B0,15,1            ; #7 Dwell-in-progress bit
M737->X:$0003B0,17,1            ; #7 Running-program bit
M738->X:$0003B0,18,1            ; #7 Open-loop-mode bit
M739->X:$0003B0,19,1            ; #7 Amplifier-enabled status bit
M740->Y:$0003C0,0,1             ; #7 Background in-position bit
M741->Y:$0003C0,1,1             ; #7 Warning-following error bit
M742->Y:$0003C0,2,1             ; #7 Fatal-following-error bit
M743->Y:$0003C0,3,1             ; #7 Amplifier-fault-error bit
M744->Y:$0003C0,13,1            ; #7 Foreground in-position bit
M745->Y:$0003C0,10,1            ; #7 Home-complete bit
M746->Y:$0003C0,6,1             ; #7 Integrated following error fault bit
M747->Y:$0003C0,5,1             ; #7 I2T fault bit
M748->Y:$0003C0,8,1             ; #7 Phasing error fault bit
M749->Y:$0003C0,9,1             ; #7 Phasing search-in-progress bit
; Motor #7 Move Registers
M761->D:$000388                 ; #7 Commanded position (1/[Ixx08*32] cts)
M762->D:$00038B                 ; #7 Actual position (1/[Ixx08*32] cts)
M763->D:$0003C7                 ; #7 Target (end) position (1/[Ixx08*32] cts)
M764->D:$0003CC                 ; #7 Position bias (1/[Ixx08*32] cts)
M766->X:$00039D,0,24,S          ; #7 Actual velocity (1/[Ixx09*32] cts/cyc)
M767->D:$00038D                 ; #7 Present master pos (1/[Ixx07*32] cts)
M768->X:$0003BF,8,16,S          ; #7 Filter Output (16-bit DAC bits)
M769->D:$000390                 ; #7 Compensation correction (1/[Ixx08*32] cts)
```

```
M770->D:$0003B4              ; #7 Present phase position (including fraction)
M771->X:$0003B4,24,S         ; #7 Present phase position (counts *Ixx70)
M772->L:$0003D7              ; #7 Variable jog position/distance (cts)
M773->Y:$0003CE,0,24,S       ; #7 Encoder home capture position (cts)
M774->D:$0003EF              ; #7 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M775->X:$0003B9,8,16,S       ; #7 Actual quadrature current
M776->Y:$0003B9,8,16,S       ; #7 Actual direct current
M777->X:$0003BC,8,16,S       ; #7 Quadrature current-loop integrator output
M778->Y:$0003BC,8,16,S       ; #7 Direct current-loop integrator output
M779->X:$0003AE,8,16,S       ; #7 PID internal filter result (16-bit DAC bits)
; Motor #7 Axis Definition Registers
M791->L:$0003CF              ; #7 X/U/A/B/C-Axis scale factor (cts/unit)
M792->L:$0003D0              ; #7 Y/V-Axis scale factor (cts/unit)
M793->L:$0003D1              ; #7 Z/W-Axis scale factor (cts/unit)
M794->L:$0003D2              ; #7 Axis offset (cts)
; Servo IC 1 Registers for PMAC(1) Channel 8 (usually for Motor #8)
M801->X:$07810D,0,24,S       ; ENC8 24-bit counter position
M802->Y:$07810A,8,16,S       ; DAC8 16-bit analog output
M803->X:$07810F,0,24,S       ; ENC8 capture/compare position register
M805->Y:$07810F,8,16,S       ; ADC8 16-bit analog input
M806->Y:$07810C,0,24,U       ; ENC8 time between counts (SCLK cycles)
M810->X:$07810C,10,1         ; ENC8 count-write enable control
M811->X:$07810C,11,1         ; EQU8 compare flag latch control
M812->X:$07810C,12,1         ; EQU8 compare output enable
M813->X:$07810C,13,1         ; EQU8 compare invert enable
M814->X:$07810C,14,1         ; AENA8/DIR8 Output
M816->X:$07810C,16,1         ; EQU8 compare flag
M817->X:$07810C,17,1         ; ENC8 position-captured flag
M818->X:$07810C,18,1         ; ENC8 Count-error flag
M819->X:$07810C,19,1         ; ENC8 3rd channel input status
M820->X:$07810C,20,1         ; HMFL8 input status
M821->X:$07810C,21,1         ; -LIM8 (positive end) input status
M822->X:$07810C,22,1         ; +LIM8 (negative end) input status
M823->X:$07810C,23,1         ; FAULT8 input status
; Motor #8 Status Bits
M830->Y:$000440,11,1         ; #8 Stopped-on-position-limit bit
M831->X:$000430,21,1         ; #8 Positive-end-limit-set bit
M832->X:$000430,22,1         ; #8 Negative-end-limit-set bit
M833->X:$000430,13,1         ; #8 Desired-velocity-zero bit
M835->X:$000430,15,1         ; #8 Dwell-in-progress bit
M837->X:$000430,17,1         ; #8 Running-program bit
M838->X:$000430,18,1         ; #8 Open-loop-mode bit
M839->X:$000430,19,1         ; #8 Amplifier-enabled status bit
M840->Y:$000440,0,1          ; #8 Background in-position bit
M841->Y:$000440,1,1          ; #8 Warning-following error bit
M842->Y:$000440,2,1          ; #8 Fatal-following-error bit
M843->Y:$000440,3,1          ; #8 Amplifier-fault-error bit
```

```
M844->Y:$000440,13,1          ; #8 Foreground in-position bit
M845->Y:$000440,10,1          ; #8 Home-complete bit
M846->Y:$000440,6,1           ; #8 Integrated following error fault bit
M847->Y:$000440,5,1           ; #8 I2T fault bit
M848->Y:$000440,8,1           ; #8 Phasing error fault bit
M849->Y:$000440,9,1           ; #8 Phasing search-in-progress bit
; Motor #8 Move Registers
M861->D:$000408               ; #8 Commanded position (1/[Ixx08*32] cts)
M862->D:$00040B               ; #8 Actual position (1/[Ixx08*32] cts)
M863->D:$000447               ; #8 Target (end) position (1/[Ixx08*32] cts)
M864->D:$00044C               ; #8 Position bias (1/[Ixx08*32] cts)
M866->X:$00041D,0,24,S        ; #8 Actual velocity (1/[Ixx09*32] cts/cyc)
M867->D:$00040D               ; #8 Present master pos (1/[Ixx07*32] cts)
M868->X:$00043F,8,16,S        ; #8 Filter Output (16-bit DAC bits)
M869->D:$000410               ; #8 Compensation correction (1/[Ixx08*32] cts)
M870->D:$000434               ; #8 Present phase position (including fraction)
M871->X:$000434,24,S          ; #8 Present phase position (counts *Ixx70)
M872->L:$000457               ; #8 Variable jog position/distance (cts)
M873->Y:$00044E,0,24,S        ; #8 Encoder home capture position (cts)
M874->D:$00046F               ; #8 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M875->X:$000439,8,16,S        ; #8 Actual quadrature current
M876->Y:$000439,8,16,S        ; #8 Actual direct current
M877->X:$00043C,8,16,S        ; #8 Quadrature current-loop integrator output
M878->Y:$00043C,8,16,S        ; #8 Direct current-loop integrator output
M879->X:$00042E,8,16,S        ; #8 PID internal filter result (16-bit DAC bits)
; Motor #8 Axis Definition Registers
M891->L:$00044F               ; #8 X/U/A/B/C-Axis scale factor (cts/unit)
M892->L:$000450               ; #8 Y/V-Axis scale factor (cts/unit)
M893->L:$000451               ; #8 Z/W-Axis scale factor (cts/unit)
M894->L:$000452               ; #8 Axis offset (cts)
; Servo IC 2 Registers for 1st ACC-24 Channel 1 (usually for Motor #9)
M901->X:$078201,0,24,S        ; ENC1 24-bit counter position
M902->Y:$078203,8,16,S        ; DAC1 16-bit analog output
M903->X:$078203,0,24,S        ; ENC1 capture/compare position register
M905->Y:$078206,8,16,S        ; ADC1 16-bit analog input
M906->Y:$078200,0,24,U        ; ENC1 time between counts (SCLK cycles)
M910->X:$078200,10,1          ; ENC1 count-write enable control
M911->X:$078200,11,1          ; EQU1 compare flag latch control
M912->X:$078200,12,1          ; EQU1 compare output enable
M913->X:$078200,13,1          ; EQU1 compare invert enable
M914->X:$078200,14,1          ; AENA1/DIR1 Output
M916->X:$078200,16,1          ; EQU1 compare flag
M917->X:$078200,17,1          ; ENC1 position-captured flag
M918->X:$078200,18,1          ; ENC1 Count-error flag
M919->X:$078200,19,1          ; ENC1 3rd channel input status
M920->X:$078200,20,1          ; HMFL1 input status
M921->X:$078200,21,1          ; -LIM1 (positive end) input status
```

```
M922->X:$078200,22,1          ; +LIM1 (negative end) input status
M923->X:$078200,23,1          ; FAULT1 input status
; Motor #9 Status Bits
M930->Y:$0004C0,11,1          ; #9 Stopped-on-position-limit bit
M931->X:$0004B0,21,1          ; #9 Positive-end-limit-set bit
M932->X:$0004B0,22,1          ; #9 Negative-end-limit-set bit
M933->X:$0004B0,13,1          ; #9 Desired-velocity-zero bit
M935->X:$0004B0,15,1          ; #9 Dwell-in-progress bit
M937->X:$0004B0,17,1          ; #9 Running-program bit
M938->X:$0004B0,18,1          ; #9 Open-loop-mode bit
M939->X:$0004B0,19,1          ; #9 Amplifier-enabled status bit
M940->Y:$0004C0,0,1           ; #9 Background in-position bit
M941->Y:$0004C0,1,1           ; #9 Warning-following error bit
M942->Y:$0004C0,2,1           ; #9 Fatal-following-error bit
M943->Y:$0004C0,3,1           ; #9 Amplifier-fault-error bit
M944->Y:$0004C0,13,1          ; #9 Foreground in-position bit
M945->Y:$0004C0,10,1          ; #9 Home-complete bit
M946->Y:$0004C0,6,1           ; #9 Integrated following error fault bit
M947->Y:$0004C0,5,1           ; #9 I2T fault bit
M948->Y:$0004C0,8,1           ; #9 Phasing error fault bit
M949->Y:$0004C0,9,1           ; #9 Phasing search-in-progress bit
; Motor #9 Move Registers
M961->D:$000488               ; #9 Commanded position (1/[Ixx08*32] cts)
M962->D:$00048B               ; #9 Actual position (1/[Ixx08*32] cts)
M963->D:$0004C7               ; #9 Target (end) position (1/[Ixx08*32] cts)
M964->D:$0004CC               ; #9 Position bias (1/[Ixx08*32] cts)
M966->X:$00049D,0,24,S        ; #9 Actual velocity (1/[Ixx09*32] cts/cyc)
M967->D:$00048D               ; #9 Present master pos (1/[Ixx07*32] cts)
M968->X:$0004BF,8,16,S        ; #9 Filter Output (16-bit DAC bits)
M969->D:$000490               ; #9 Compensation correction (1/[Ixx08*32] cts)
M970->D:$0004B4               ; #9 Present phase position (including fraction)
M971->X:$0004B4,24,S          ; #9 Present phase position (counts *Ixx70)
M972->L:$0004D7               ; #9 Variable jog position/distance (cts)
M973->Y:$0004CE,0,24,S        ; #9 Encoder home capture position (cts)
M974->D:$0004EF               ; #9 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M975->X:$0004B9,8,16,S        ; #9 Actual quadrature current
M976->Y:$0004B9,8,16,S        ; #9 Actual direct current
M977->X:$0004BC,8,16,S        ; #9 Quadrature current-loop integrator output
M978->Y:$0004BC,8,16,S        ; #9 Direct current-loop integrator output
M979->X:$0004AE,8,16,S        ; #9 PID internal filter result (16-bit DAC bits)
; Motor #9 Axis Definition Registers
M991->L:$0004CF               ; #9 X/U/A/B/C-Axis scale factor (cts/unit)
M992->L:$0004D0               ; #9 Y/V-Axis scale factor (cts/unit)
M993->L:$0004D1               ; #9 Z/W-Axis scale factor (cts/unit)
M994->L:$0004D2               ; #9 Axis offset (cts)
```

```
; Servo IC 2 Registers for 1st ACC-24 Channel 2 (usually for Motor #10)
M1001->X:$078205,0,24,S      ; ENC2 24-bit counter position
M1002->Y:$078202,8,16,S      ; DAC2 16-bit analog output
M1003->X:$078207,0,24,S      ; ENC2 capture/compare position register
M1005->Y:$078207,8,16,S      ; ADC2 16-bit analog input
M1006->Y:$078204,0,24,U      ; ENC2 time between counts (SCLK cycles)
M1010->X:$078204,10,1        ; ENC2 count-write enable control
M1011->X:$078204,11,1        ; EQU2 compare flag latch control
M1012->X:$078204,12,1        ; EQU2 compare output enable
M1013->X:$078204,13,1        ; EQU2 compare invert enable
M1014->X:$078204,14,1        ; AENA2/DIR2 Output
M1016->X:$078204,16,1        ; EQU2 compare flag
M1017->X:$078204,17,1        ; ENC2 position-captured flag
M1018->X:$078204,18,1        ; ENC2 Count-error flag
M1019->X:$078204,19,1        ; ENC2 3rd channel input status
M1020->X:$078204,20,1        ; HMFL2 input status
M1021->X:$078204,21,1        ; -LIM2 (positive end) input status
M1022->X:$078204,22,1        ; +LIM2 (negative end) input status
M1023->X:$078204,23,1        ; FAULT2 input status
; Motor #10 Status Bits
M1030->Y:$000540,11,1        ; #10 Stopped-on-position-limit bit
M1031->X:$000530,21,1        ; #10 Positive-end-limit-set bit
M1032->X:$000530,22,1        ; #10 Negative-end-limit-set bit
M1033->X:$000530,13,1        ; #10 Desired-velocity-zero bit
M1035->X:$000530,15,1        ; #10 Dwell-in-progress bit
M1037->X:$000530,17,1        ; #10 Running-program bit
M1038->X:$000530,18,1        ; #10 Open-loop-mode bit
M1039->X:$000530,19,1        ; #10 Amplifier-enabled status bit
M1040->Y:$000540,0,1         ; #10 Background in-position bit
M1041->Y:$000540,1,1         ; #10 Warning-following error bit
M1042->Y:$000540,2,1         ; #10 Fatal-following-error bit
M1043->Y:$000540,3,1         ; #10 Amplifier-fault-error bit
M1044->Y:$000540,13,1        ; #10 Foreground in-position bit
M1045->Y:$000540,10,1        ; #10 Home-complete bit
M1046->Y:$000540,6,1         ; #10 Integrated following error fault bit
M1047->Y:$000540,5,1         ; #10 I2T fault bit
M1048->Y:$000540,8,1         ; #10 Phasing error fault bit
M1049->Y:$000540,9,1         ; #10 Phasing search-in-progress bit
; Motor #10 Move Registers
M1061->D:$000508             ; #10 Commanded position (1/[Ixx08*32] cts)
M1062->D:$00050B             ; #10 Actual position (1/[Ixx08*32] cts)
M1063->D:$000547             ; #10 Target (end) position (1/[Ixx08*32] cts)
M1064->D:$00054C             ; #10 Position bias (1/[Ixx08*32] cts)
M1066->X:$00051D,0,24,S      ; #10 Actual velocity (1/[Ixx09*32] cts/cyc)
M1067->D:$00050D             ; #10 Present master pos (1/[Ixx07*32] cts)
M1068->X:$00053F,8,16,S      ; #10 Filter Output (16-bit DAC bits)
M1069->D:$000510             ; #10 Compensation correction (1/[Ixx08*32] cts)
```

```
M1070->D:$000534              ; #10 Present phase position (including fraction)
M1071->X:$000534,24,S         ; #10 Present phase position (counts *Ixx70)
M1072->L:$000557              ; #10 Variable jog position/distance (cts)
M1073->Y:$00054E,0,24,S       ; #10 Encoder home capture position (cts)
M1074->D:$00056F              ; #10 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1075->X:$000539,8,16,S       ; #10 Actual quadrature current
M1076->Y:$000539,8,16,S       ; #10 Actual direct current
M1077->X:$00053C,8,16,S       ; #10 Quadrature current-loop integrator output
M1078->Y:$00053C,8,16,S       ; #10 Direct current-loop integrator output
M1079->X:$00052E,8,16,S       ; #10 PID internal filter result (16-bit DAC bits)
; Motor #10 Axis Definition Registers
M1091->L:$00054F              ; #10 X/U/A/B/C-Axis scale factor (cts/unit)
M1092->L:$000550              ; #10 Y/V-Axis scale factor (cts/unit)
M1093->L:$000551              ; #10 Z/W-Axis scale factor (cts/unit)
M1094->L:$000552              ; #10 Axis offset (cts)
; Servo IC 2 Registers for 1st ACC-24 Channel 3 (usually for Motor #11)
M1101->X:$078209,0,24,S       ; ENC3 24-bit counter position
M1102->Y:$07820B,8,16,S       ; DAC3 16-bit analog output
M1103->X:$07820B,0,24,S       ; ENC3 capture/compare position register
M1105->Y:$07820E,8,16,S       ; ADC3 16-bit analog input
M1106->Y:$078208,0,24,U       ; ENC3 time between counts (SCLK cycles)
M1110->X:$078208,10,1         ; ENC3 count-write enable control
M1111->X:$078208,11,1         ; EQU3 compare flag latch control
M1112->X:$078208,12,1         ; EQU3 compare output enable
M1113->X:$078208,13,1         ; EQU3 compare invert enable
M1114->X:$078208,14,1         ; AENA3/DIR3 Output
M1116->X:$078208,16,1         ; EQU3 compare flag
M1117->X:$078208,17,1         ; ENC3 position-captured flag
M1118->X:$078208,18,1         ; ENC3 Count-error flag
M1119->X:$078208,19,1         ; ENC3 3rd channel input status
M1120->X:$078208,20,1         ; HMFL3 input status
M1121->X:$078208,21,1         ; -LIM3 (positive end) input status
M1122->X:$078208,22,1         ; +LIM3 (negative end) input status
M1123->X:$078208,23,1         ; FAULT3 input status
; Motor #11 Status Bits
M1130->Y:$0005C0,11,1         ; #11 Stopped-on-position-limit bit
M1131->X:$0005B0,21,1         ; #11 Positive-end-limit-set bit
M1132->X:$0005B0,22,1         ; #11 Negative-end-limit-set bit
M1133->X:$0005B0,13,1         ; #11 Desired-velocity-zero bit
M1135->X:$0005B0,15,1         ; #11 Dwell-in-progress bit
M1137->X:$0005B0,17,1         ; #11 Running-program bit
M1138->X:$0005B0,18,1         ; #11 Open-loop-mode bit
M1139->X:$0005B0,19,1         ; #11 Amplifier-enabled status bit
M1140->Y:$0005C0,0,1          ; #11 Background in-position bit
M1141->Y:$0005C0,1,1          ; #11 Warning-following error bit
M1142->Y:$0005C0,2,1          ; #11 Fatal-following-error bit
M1143->Y:$0005C0,3,1          ; #11 Amplifier-fault-error bit
```

```
M1144->Y:$0005C0,13,1        ; #11 Foreground in-position bit
M1145->Y:$0005C0,10,1        ; #11 Home-complete bit
M1146->Y:$0005C0,6,1         ; #11 Integrated following error fault bit
M1147->Y:$0005C0,5,1         ; #11 I2T fault bit
M1148->Y:$0005C0,8,1         ; #11 Phasing error fault bit
M1149->Y:$0005C0,9,1         ; #11 Phasing search-in-progress bit
; Motor #11 Move Registers
M1161->D:$000588             ; #11 Commanded position (1/[Ixx08*32] cts)
M1162->D:$00058B             ; #11 Actual position (1/[Ixx08*32] cts)
M1163->D:$0005C7             ; #11 Target (end) position (1/[Ixx08*32] cts)
M1164->D:$0005CC             ; #11 Position bias (1/[Ixx08*32] cts)
M1166->X:$00059D,0,24,S      ; #11 Actual velocity (1/[Ixx09*32] cts/cyc)
M1167->D:$00058D             ; #11 Present master pos (1/[Ixx07*32] cts)
M1168->X:$0005BF,8,16,S      ; #11 Filter Output (16-bit DAC bits)
M1169->D:$000590             ; #11 Compensation correction (1/[Ixx08*32] cts)
M1170->D:$0005B4             ; #11 Present phase position (including fraction)
M1171->X:$0005B4,24,S        ; #11 Present phase position (counts *Ixx70)
M1172->L:$0005D7             ; #11 Variable jog position/distance (cts)
M1173->Y:$0005CE,0,24,S      ; #11 Encoder home capture position (cts)
M1174->D:$0005EF             ; #11 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1175->X:$0005B9,8,16,S      ; #11 Actual quadrature current
M1176->Y:$0005B9,8,16,S      ; #11 Actual direct current
M1177->X:$0005BC,8,16,S      ; #11 Quadrature current-loop integrator output
M1178->Y:$0005BC,8,16,S      ; #11 Direct current-loop integrator output
M1179->X:$0005AE,8,16,S      ; #11 PID internal filter result (16-bit DAC bits)
; Motor #11 Axis Definition Registers
M1191->L:$0005CF             ; #11 X/U/A/B/C-Axis scale factor (cts/unit)
M1192->L:$0005D0             ; #11 Y/V-Axis scale factor (cts/unit)
M1193->L:$0005D1             ; #11 Z/W-Axis scale factor (cts/unit)
M1194->L:$0005D2             ; #11 Axis offset (cts)
; Servo IC 2 Registers for 1st ACC-24 Channel 4 (usually for Motor #12)
M1201->X:$07820D,0,24,S      ; ENC4 24-bit counter position
M1202->Y:$07820A,8,16,S      ; DAC4 16-bit analog output
M1203->X:$07820F,0,24,S      ; ENC4 capture/compare position register
M1205->Y:$07820F,8,16,S      ; ADC4 16-bit analog input
M1206->Y:$07820C,0,24,U      ; ENC4 time between counts (SCLK cycles)
M1210->X:$07820C,10,1        ; ENC4 count-write enable control
M1211->X:$07820C,11,1        ; EQU4 compare flag latch control
M1212->X:$07820C,12,1        ; EQU4 compare output enable
M1213->X:$07820C,13,1        ; EQU4 compare invert enable
M1214->X:$07820C,14,1        ; AENA4/DIR4 Output
M1216->X:$07820C,16,1        ; EQU4 compare flag
M1217->X:$07820C,17,1        ; ENC4 position-captured flag
M1218->X:$07820C,18,1        ; ENC4 Count-error flag
M1219->X:$07820C,19,1        ; ENC4 3rd channel input status
M1220->X:$07820C,20,1        ; HMFL4 input status
M1221->X:$07820C,21,1        ; -LIM4 (positive end) input status
```

```
M1222->X:$07820C,22,1        ; +LIM4 (negative end) input status
M1223->X:$07820C,23,1        ; FAULT4 input status
; Motor #12 Status Bits
M1230->Y:$000640,11,1        ; #12 Stopped-on-position-limit bit
M1231->X:$000630,21,1        ; #12 Positive-end-limit-set bit
M1232->X:$000630,22,1        ; #12 Negative-end-limit-set bit
M1233->X:$000630,13,1        ; #12 Desired-velocity-zero bit
M1235->X:$000630,15,1        ; #12 Dwell-in-progress bit
M1237->X:$000630,17,1        ; #12 Running-program bit
M1238->X:$000630,18,1        ; #12 Open-loop-mode bit
M1239->X:$000630,19,1        ; #12 Amplifier-enabled status bit
M1240->Y:$000640,0,1         ; #12 Background in-position bit
M1241->Y:$000640,1,1         ; #12 Warning-following error bit
M1242->Y:$000640,2,1         ; #12 Fatal-following-error bit
M1243->Y:$000640,3,1         ; #12 Amplifier-fault-error bit
M1244->Y:$000640,13,1        ; #12 Foreground in-position bit
M1245->Y:$000640,10,1        ; #12 Home-complete bit
M1246->Y:$000640,6,1         ; #12 Integrated following error fault bit
M1247->Y:$000640,5,1         ; #12 I2T fault bit
M1248->Y:$000640,8,1         ; #12 Phasing error fault bit
M1249->Y:$000640,9,1         ; #12 Phasing search-in-progress bit
; Motor #12 Move Registers
M1261->D:$000608             ; #12 Commanded position (1/[Ixx08*32] cts)
M1262->D:$00060B             ; #12 Actual position (1/[Ixx08*32] cts)
M1263->D:$000647             ; #12 Target (end) position (1/[Ixx08*32] cts)
M1264->D:$00064C             ; #12 Position bias (1/[Ixx08*32] cts)
M1266->X:$00061D,0,24,S      ; #12 Actual velocity (1/[Ixx09*32] cts/cyc)
M1267->D:$00060D             ; #12 Present master pos (1/[Ixx07*32] cts)
M1268->X:$00063F,8,16,S      ; #12 Filter Output (16-bit DAC bits)
M1269->D:$000610             ; #12 Compensation correction (1/[Ixx08*32] cts)
M1270->D:$000634             ; #12 Present phase position (including fraction)
M1271->X:$000634,24,S        ; #12 Present phase position (counts *Ixx70)
M1272->L:$000657             ; #12 Variable jog position/distance (cts)
M1273->Y:$00064E,0,24,S      ; #12 Encoder home capture position (cts)
M1274->D:$00066F             ; #12 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1275->X:$000639,8,16,S      ; #12 Actual quadrature current
M1276->Y:$000639,8,16,S      ; #12 Actual direct current
M1277->X:$00063C,8,16,S      ; #12 Quadrature current-loop integrator output
M1278->Y:$00063C,8,16,S      ; #12 Direct current-loop integrator output
M1279->X:$00062E,8,16,S      ; #12 PID internal filter result (16-bit DAC bits)
; Motor #12 Axis Definition Registers
M1291->L:$00064F             ; #12 X/U/A/B/C-Axis scale factor (cts/unit)
M1292->L:$000650             ; #12 Y/V-Axis scale factor (cts/unit)
M1293->L:$000651             ; #12 Z/W-Axis scale factor (cts/unit)
M1294->L:$000652             ; #12 Axis offset (cts)
```

```
; Servo IC 3 Registers for 1st ACC-24 Channel 5 (usually for Motor #13)
M1301->X:$078301,0,24,S      ; ENC5 24-bit counter position
M1302->Y:$078303,8,16,S      ; DAC5 16-bit analog output
M1303->X:$078303,0,24,S      ; ENC5 capture/compare position register
M1305->Y:$078306,8,16,S      ; ADC5 16-bit analog input
M1306->Y:$078300,0,24,U      ; ENC5 time between counts (SCLK cycles)
M1310->X:$078300,10,1        ; ENC5 count-write enable control
M1311->X:$078300,11,1        ; EQU5 compare flag latch control
M1312->X:$078300,12,1        ; EQU5 compare output enable
M1313->X:$078300,13,1        ; EQU5 compare invert enable
M1314->X:$078300,14,1        ; AENA5/DIR5 Output
M1316->X:$078300,16,1        ; EQU5 compare flag
M1317->X:$078300,17,1        ; ENC5 position-captured flag
M1318->X:$078300,18,1        ; ENC5 Count-error flag
M1319->X:$078300,19,1        ; ENC5 3rd channel input status
M1320->X:$078300,20,1        ; HMFL5 input status
M1321->X:$078300,21,1        ; -LIM5 (positive end) input status
M1322->X:$078300,22,1        ; +LIM5 (negative end) input status
M1323->X:$078300,23,1        ; FAULT5 input status
; Motor #13 Status Bits
M1330->Y:$0006C0,11,1        ; #13 Stopped-on-position-limit bit
M1331->X:$0006B0,21,1        ; #13 Positive-end-limit-set bit
M1332->X:$0006B0,22,1        ; #13 Negative-end-limit-set bit
M1333->X:$0006B0,13,1        ; #13 Desired-velocity-zero bit
M1335->X:$0006B0,15,1        ; #13 Dwell-in-progress bit
M1337->X:$0006B0,17,1        ; #13 Running-program bit
M1338->X:$0006B0,18,1        ; #13 Open-loop-mode bit
M1339->X:$0006B0,19,1        ; #13 Amplifier-enabled status bit
M1340->Y:$0006C0,0,1         ; #13 Background in-position bit
M1341->Y:$0006C0,1,1         ; #13 Warning-following error bit
M1342->Y:$0006C0,2,1         ; #13 Fatal-following-error bit
M1343->Y:$0006C0,3,1         ; #13 Amplifier-fault-error bit
M1344->Y:$0006C0,13,1        ; #13 Foreground in-position bit
M1345->Y:$0006C0,10,1        ; #13 Home-complete bit
M1346->Y:$0006C0,6,1         ; #13 Integrated following error fault bit
M1347->Y:$0006C0,5,1         ; #13 I2T fault bit
M1348->Y:$0006C0,8,1         ; #13 Phasing error fault bit
M1349->Y:$0006C0,9,1         ; #13 Phasing search-in-progress bit
; Motor #13 Move Registers
M1361->D:$000688             ; #13 Commanded position (1/[Ixx08*32] cts)
M1362->D:$00068B             ; #13 Actual position (1/[Ixx08*32] cts)
M1363->D:$0006C7             ; #13 Target (end) position (1/[Ixx08*32] cts)
M1364->D:$0006CC             ; #13 Position bias (1/[Ixx08*32] cts)
M1366->X:$00069D,0,24,S      ; #13 Actual velocity (1/[Ixx09*32] cts/cyc)
M1367->D:$00068D             ; #13 Present master pos (1/[Ixx07*32] cts)
M1368->X:$0006BF,8,16,S      ; #13 Filter Output (16-bit DAC bits)
M1369->D:$000690             ; #13 Compensation correction (1/[Ixx08*32] cts)
```

```
M1370->D:$0006B4              ; #13 Present phase position (including fraction)
M1371->X:$0006B4,24,S         ; #13 Present phase position (counts *Ixx70)
M1372->L:$0006D7              ; #13 Variable jog position/distance (cts)
M1373->Y:$0006CE,0,24,S       ; #13 Encoder home capture position (cts)
M1374->D:$0006EF              ; #13 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1375->X:$0006B9,8,16,S       ; #13 Actual quadrature current
M1376->Y:$0006B9,8,16,S       ; #13 Actual direct current
M1377->X:$0006BC,8,16,S       ; #13 Quadrature current-loop integrator output
M1378->Y:$0006BC,8,16,S       ; #13 Direct current-loop integrator output
M1379->X:$0006AE,8,16,S       ; #13 PID internal filter result (16-bit DAC bits)
; Motor #13 Axis Definition Registers
M1391->L:$0006CF              ; #13 X/U/A/B/C-Axis scale factor (cts/unit)
M1392->L:$0006D0              ; #13 Y/V-Axis scale factor (cts/unit)
M1393->L:$0006D1              ; #13 Z/W-Axis scale factor (cts/unit)
M1394->L:$0006D2              ; #13 Axis offset (cts)
; Servo IC 3 Registers for 1st ACC-24 Channel 6 (usually for Motor #14)
M1401->X:$078305,0,24,S       ; ENC6 24-bit counter position
M1402->Y:$078302,8,16,S       ; DAC6 16-bit analog output
M1403->X:$078307,0,24,S       ; ENC6 capture/compare position register
M1405->Y:$078307,8,16,S       ; ADC6 16-bit analog input
M1406->Y:$078304,0,24,U       ; ENC6 time between counts (SCLK cycles)
M1410->X:$078304,10,1         ; ENC6 count-write enable control
M1411->X:$078304,11,1         ; EQU6 compare flag latch control
M1412->X:$078304,12,1         ; EQU6 compare output enable
M1413->X:$078304,13,1         ; EQU6 compare invert enable
M1414->X:$078304,14,1         ; AENA6/DIR6 Output
M1416->X:$078304,16,1         ; EQU6 compare flag
M1417->X:$078304,17,1         ; ENC6 position-captured flag
M1418->X:$078304,18,1         ; ENC6 Count-error flag
M1419->X:$078304,19,1         ; ENC6 3rd channel input status
M1420->X:$078304,20,1         ; HMFL6 input status
M1421->X:$078304,21,1         ; -LIM6 (positive end) input status
M1422->X:$078304,22,1         ; +LIM6 (negative end) input status
M1423->X:$078304,23,1         ; FAULT6 input status
; Motor #14 Status Bits
M1430->Y:$000740,11,1         ; #14 Stopped-on-position-limit bit
M1431->X:$000730,21,1         ; #14 Positive-end-limit-set bit
M1432->X:$000730,22,1         ; #14 Negative-end-limit-set bit
M1433->X:$000730,13,1         ; #14 Desired-velocity-zero bit
M1435->X:$000730,15,1         ; #14 Dwell-in-progress bit
M1437->X:$000730,17,1         ; #14 Running-program bit
M1438->X:$000730,18,1         ; #14 Open-loop-mode bit
M1439->X:$000730,19,1         ; #14 Amplifier-enabled status bit
M1440->Y:$000740,0,1          ; #14 Background in-position bit
M1441->Y:$000740,1,1          ; #14 Warning-following error bit
M1442->Y:$000740,2,1          ; #14 Fatal-following-error bit
M1443->Y:$000740,3,1          ; #14 Amplifier-fault-error bit
```

```
M1444->Y:$000740,13,1        ; #14 Foreground in-position bit
M1445->Y:$000740,10,1        ; #14 Home-complete bit
M1446->Y:$000740,6,1         ; #14 Integrated following error fault bit
M1447->Y:$000740,5,1         ; #14 I2T fault bit
M1448->Y:$000740,8,1         ; #14 Phasing error fault bit
M1449->Y:$000740,9,1         ; #14 Phasing search-in-progress bit
; Motor #14 Move Registers
M1461->D:$000708             ; #14 Commanded position (1/[Ixx08*32] cts)
M1462->D:$00070B             ; #14 Actual position (1/[Ixx08*32] cts)
M1463->D:$000747             ; #14 Target (end) position (1/[Ixx08*32] cts)
M1464->D:$00074C             ; #14 Position bias (1/[Ixx08*32] cts)
M1466->X:$00071D,0,24,S      ; #14 Actual velocity (1/[Ixx09*32] cts/cyc)
M1467->D:$00070D             ; #14 Present master pos (1/[Ixx07*32] cts)
M1468->X:$00073F,8,16,S      ; #14 Filter Output (16-bit DAC bits)
M1469->D:$000710             ; #14 Compensation correction (1/[Ixx08*32] cts)
M1470->D:$000734             ; #14 Present phase position (including fraction)
M1471->X:$000734,24,S        ; #14 Present phase position (counts *Ixx70)
M1472->L:$000757             ; #14 Variable jog position/distance (cts)
M1473->Y:$00074E,0,24,S      ; #14 Encoder home capture position (cts)
M1474->D:$00076F             ; #14 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1475->X:$000739,8,16,S      ; #14 Actual quadrature current
M1476->Y:$000739,8,16,S      ; #14 Actual direct current
M1477->X:$00073C,8,16,S      ; #14 Quadrature current-loop integrator output
M1478->Y:$00073C,8,16,S      ; #14 Direct current-loop integrator output
M1479->X:$00072E,8,16,S      ; #14 PID internal filter result (16-bit DAC bits)
; Motor #14 Axis Definition Registers
M1491->L:$00074F             ; #14 X/U/A/B/C-Axis scale factor (cts/unit)
M1492->L:$000750             ; #14 Y/V-Axis scale factor (cts/unit)
M1493->L:$000751             ; #14 Z/W-Axis scale factor (cts/unit)
M1494->L:$000752             ; #14 Axis offset (cts)
; Servo IC 3 Registers for 1st ACC-24 Channel 7 (usually for Motor #15)
M1501->X:$078309,0,24,S      ; ENC7 24-bit counter position
M1502->Y:$07830B,8,16,S      ; DAC7 16-bit analog output
M1503->X:$07830B,0,24,S      ; ENC7 capture/compare position register
M1505->Y:$07830E,8,16,S      ; ADC7 16-bit analog input
M1506->Y:$078308,0,24,U      ; ENC7 time between counts (SCLK cycles)
M1510->X:$078308,10,1        ; ENC7 count-write enable control
M1511->X:$078308,11,1        ; EQU7 compare flag latch control
M1512->X:$078308,12,1        ; EQU7 compare output enable
M1513->X:$078308,13,1        ; EQU7 compare invert enable
M1514->X:$078308,14,1        ; AENA7/DIR7 Output
M1516->X:$078308,16,1        ; EQU7 compare flag
M1517->X:$078308,17,1        ; ENC7 position-captured flag
M1518->X:$078308,18,1        ; ENC7 Count-error flag
M1519->X:$078308,19,1        ; ENC7 3rd channel input status
M1520->X:$078308,20,1        ; HMFL7 input status
M1521->X:$078308,21,1        ; -LIM7 (positive end) input status
```

```
M1522->X:$078308,22,1        ; +LIM7 (negative end) input status
M1523->X:$078308,23,1        ; FAULT7 input status
; Motor #15 Status Bits
M1530->Y:$0007C0,11,1        ; #15 Stopped-on-position-limit bit
M1531->X:$0007B0,21,1        ; #15 Positive-end-limit-set bit
M1532->X:$0007B0,22,1        ; #15 Negative-end-limit-set bit
M1533->X:$0007B0,13,1        ; #15 Desired-velocity-zero bit
M1535->X:$0007B0,15,1        ; #15 Dwell-in-progress bit
M1537->X:$0007B0,17,1        ; #15 Running-program bit
M1538->X:$0007B0,18,1        ; #15 Open-loop-mode bit
M1539->X:$0007B0,19,1        ; #15 Amplifier-enabled status bit
M1540->Y:$0007C0,0,1         ; #15 Background in-position bit
M1541->Y:$0007C0,1,1         ; #15 Warning-following error bit
M1542->Y:$0007C0,2,1         ; #15 Fatal-following-error bit
M1543->Y:$0007C0,3,1         ; #15 Amplifier-fault-error bit
M1544->Y:$0007C0,13,1        ; #15 Foreground in-position bit
M1545->Y:$0007C0,10,1        ; #15 Home-complete bit
M1546->Y:$0007C0,6,1         ; #15 Integrated following error fault bit
M1547->Y:$0007C0,5,1         ; #15 I2T fault bit
M1548->Y:$0007C0,8,1         ; #15 Phasing error fault bit
M1549->Y:$0007C0,9,1         ; #15 Phasing search-in-progress bit
; Motor #15 Move Registers
M1561->D:$000788             ; #15 Commanded position (1/[Ixx08*32] cts)
M1562->D:$00078B             ; #15 Actual position (1/[Ixx08*32] cts)
M1563->D:$0007C7             ; #15 Target (end) position (1/[Ixx08*32] cts)
M1564->D:$0007CC             ; #15 Position bias (1/[Ixx08*32] cts)
M1566->X:$00079D,0,24,S      ; #15 Actual velocity (1/[Ixx09*32] cts/cyc)
M1567->D:$00078D             ; #15 Present master pos (1/[Ixx07*32] cts)
M1568->X:$0007BF,8,16,S      ; #15 Filter Output (16-bit DAC bits)
M1569->D:$000790             ; #15 Compensation correction (1/[Ixx08*32] cts)
M1570->D:$0007B4             ; #15 Present phase position (including fraction)
M1571->X:$0007B4,24,S        ; #15 Present phase position (counts *Ixx70)
M1572->L:$0007D7             ; #15 Variable jog position/distance (cts)
M1573->Y:$0007CE,0,24,S      ; #15 Encoder home capture position (cts)
M1574->D:$0007EF             ; #15 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1575->X:$0007B9,8,16,S      ; #15 Actual quadrature current
M1576->Y:$0007B9,8,16,S      ; #15 Actual direct current
M1577->X:$0007BC,8,16,S      ; #15 Quadrature current-loop integrator output
M1578->Y:$0007BC,8,16,S      ; #15 Direct current-loop integrator output
M1579->X:$0007AE,8,16,S      ; #15 PID internal filter result (16-bit DAC bits)
; Motor #15 Axis Definition Registers
M1591->L:$0007CF             ; #15 X/U/A/B/C-Axis scale factor (cts/unit)
M1592->L:$0007D0             ; #15 Y/V-Axis scale factor (cts/unit)
M1593->L:$0007D1             ; #15 Z/W-Axis scale factor (cts/unit)
M1594->L:$0007D2             ; #15 Axis offset (cts)
```

```
; Servo IC 3 Registers for 1st ACC-24 Channel 8 (usually for Motor #16)
M1601->X:$07830D,0,24,S      ; ENC8 24-bit counter position
M1602->Y:$07830A,8,16,S      ; DAC8 16-bit analog output
M1603->X:$07830F,0,24,S      ; ENC8 capture/compare position register
M1605->Y:$07830F,8,16,S      ; ADC8 16-bit analog input
M1606->Y:$07830C,0,24,U      ; ENC8 time between counts (SCLK cycles)
M1610->X:$07830C,10,1        ; ENC8 count-write enable control
M1611->X:$07830C,11,1        ; EQU8 compare flag latch control
M1612->X:$07830C,12,1        ; EQU8 compare output enable
M1613->X:$07830C,13,1        ; EQU8 compare invert enable
M1614->X:$07830C,14,1        ; AENA8/DIR8 Output
M1616->X:$07830C,16,1        ; EQU8 compare flag
M1617->X:$07830C,17,1        ; ENC8 position-captured flag
M1618->X:$07830C,18,1        ; ENC8 Count-error flag
M1619->X:$07830C,19,1        ; ENC8 3rd channel input status
M1620->X:$07830C,20,1        ; HMFL8 input status
M1621->X:$07830C,21,1        ; -LIM8 (positive end) input status
M1622->X:$07830C,22,1        ; +LIM8 (negative end) input status
M1623->X:$07830C,23,1        ; FAULT8 input status
; Motor #16 Status Bits
M1630->Y:$000840,11,1        ; #16 Stopped-on-position-limit bit
M1631->X:$000830,21,1        ; #16 Positive-end-limit-set bit
M1632->X:$000830,22,1        ; #16 Negative-end-limit-set bit
M1633->X:$000830,13,1        ; #16 Desired-velocity-zero bit
M1635->X:$000830,15,1        ; #16 Dwell-in-progress bit
M1637->X:$000830,17,1        ; #16 Running-program bit
M1638->X:$000830,18,1        ; #16 Open-loop-mode bit
M1639->X:$000830,19,1        ; #16 Amplifier-enabled status bit
M1640->Y:$000840,0,1         ; #16 Background in-position bit
M1641->Y:$000840,1,1         ; #16 Warning-following error bit
M1642->Y:$000840,2,1         ; #16 Fatal-following-error bit
M1643->Y:$000840,3,1         ; #16 Amplifier-fault-error bit
M1644->Y:$000840,13,1        ; #16 Foreground in-position bit
M1645->Y:$000840,10,1        ; #16 Home-complete bit
M1646->Y:$000840,6,1         ; #16 Integrated following error fault bit
M1647->Y:$000840,5,1         ; #16 I2T fault bit
M1648->Y:$000840,8,1         ; #16 Phasing error fault bit
M1649->Y:$000840,9,1         ; #16 Phasing search-in-progress bit
; Motor #16 Move Registers
M1661->D:$000808             ; #16 Commanded position (1/[Ixx08*32] cts)
M1662->D:$00080B             ; #16 Actual position (1/[Ixx08*32] cts)
M1663->D:$000847             ; #16 Target (end) position (1/[Ixx08*32] cts)
M1664->D:$00084C             ; #16 Position bias (1/[Ixx08*32] cts)
M1666->X:$00081D,0,24,S      ; #16 Actual velocity (1/[Ixx09*32] cts/cyc)
M1667->D:$00080D             ; #16 Present master pos (1/[Ixx07*32] cts)
M1668->X:$00083F,8,16,S      ; #16 Filter Output (16-bit DAC bits)
M1669->D:$000810             ; #16 Compensation correction (1/[Ixx08*32] cts)
```

```
M1670->D:$000834              ; #16 Present phase position (including fraction)
M1671->X:$000834,24,S         ; #16 Present phase position (counts *Ixx70)
M1672->L:$000857              ; #16 Variable jog position/distance (cts)
M1673->Y:$00084E,0,24,S       ; #16 Encoder home capture position (cts)
M1674->D:$00086F              ; #16 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1675->X:$000839,8,16,S       ; #16 Actual quadrature current
M1676->Y:$000839,8,16,S       ; #16 Actual direct current
M1677->X:$00083C,8,16,S       ; #16 Quadrature current-loop integrator output
M1678->Y:$00083C,8,16,S       ; #16 Direct current-loop integrator output
M1679->X:$00082E,8,16,S       ; #16 PID internal filter result (16-bit DAC bits)
; Motor #16 Axis Definition Registers
M1691->L:$00084F              ; #16 X/U/A/B/C-Axis scale factor (cts/unit)
M1692->L:$000850              ; #16 Y/V-Axis scale factor (cts/unit)
M1693->L:$000851              ; #16 Z/W-Axis scale factor (cts/unit)
M1694->L:$000852              ; #16 Axis offset (cts)
; Servo IC 4 Registers for 2nd ACC-24 Channel 1 (usually for Motor #17)
M1701->X:$079201,0,24,S       ; ENC1 24-bit counter position
M1702->Y:$079203,8,16,S       ; DAC1 16-bit analog output
M1703->X:$079203,0,24,S       ; ENC1 capture/compare position register
M1705->Y:$079206,8,16,S       ; ADC1 16-bit analog input
M1706->Y:$079200,0,24,U       ; ENC1 time between counts (SCLK cycles)
M1710->X:$079200,10,1         ; ENC1 count-write enable control
M1711->X:$079200,11,1         ; EQU1 compare flag latch control
M1712->X:$079200,12,1         ; EQU1 compare output enable
M1713->X:$079200,13,1         ; EQU1 compare invert enable
M1714->X:$079200,14,1         ; AENA1/DIR1 Output
M1716->X:$079200,16,1         ; EQU1 compare flag
M1717->X:$079200,17,1         ; ENC1 position-captured flag
M1718->X:$079200,18,1         ; ENC1 Count-error flag
M1719->X:$079200,19,1         ; ENC1 3rd channel input status
M1720->X:$079200,20,1         ; HMFL1 input status
M1721->X:$079200,21,1         ; -LIM1 (positive end) input status
M1722->X:$079200,22,1         ; +LIM1 (negative end) input status
M1723->X:$079200,23,1         ; FAULT1 input status
; Motor #17 Status Bits
M1730->Y:$0008C0,11,1         ; #17 Stopped-on-position-limit bit
M1731->X:$0008B0,21,1         ; #17 Positive-end-limit-set bit
M1732->X:$0008B0,22,1         ; #17 Negative-end-limit-set bit
M1733->X:$0008B0,13,1         ; #17 Desired-velocity-zero bit
M1735->X:$0008B0,15,1         ; #17 Dwell-in-progress bit
M1737->X:$0008B0,17,1         ; #17 Running-program bit
M1738->X:$0008B0,18,1         ; #17 Open-loop-mode bit
M1739->X:$0008B0,19,1         ; #17 Amplifier-enabled status bit
M1740->Y:$0008C0,0,1          ; #17 Background in-position bit
M1741->Y:$0008C0,1,1          ; #17 Warning-following error bit
M1742->Y:$0008C0,2,1          ; #17 Fatal-following-error bit
M1743->Y:$0008C0,3,1          ; #17 Amplifier-fault-error bit
```

```
M1744->Y:$0008C0,13,1        ; #17 Foreground in-position bit
M1745->Y:$0008C0,10,1        ; #17 Home-complete bit
M1746->Y:$0008C0,6,1         ; #17 Integrated following error fault bit
M1747->Y:$0008C0,5,1         ; #17 I2T fault bit
M1748->Y:$0008C0,8,1         ; #17 Phasing error fault bit
M1749->Y:$0008C0,9,1         ; #17 Phasing search-in-progress bit
; Motor #17 Move Registers
M1761->D:$000888             ; #17 Commanded position (1/[Ixx08*32] cts)
M1762->D:$00088B             ; #17 Actual position (1/[Ixx08*32] cts)
M1763->D:$0008C7             ; #17 Target (end) position (1/[Ixx08*32] cts)
M1764->D:$0008CC             ; #17 Position bias (1/[Ixx08*32] cts)
M1766->X:$00089D,0,24,S      ; #17 Actual velocity (1/[Ixx09*32] cts/cyc)
M1767->D:$00088D             ; #17 Present master pos (1/[Ixx07*32] cts)
M1768->X:$0008BF,8,16,S      ; #17 Filter Output (16-bit DAC bits)
M1769->D:$000890             ; #17 Compensation correction (1/[Ixx08*32] cts)
M1770->D:$0008B4             ; #17 Present phase position (including fraction)
M1771->X:$0008B4,24,S        ; #17 Present phase position (counts *Ixx70)
M1772->L:$0008D7             ; #17 Variable jog position/distance (cts)
M1773->Y:$0008CE,0,24,S      ; #17 Encoder home capture position (cts)
M1774->D:$0008EF             ; #17 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1775->X:$0008B9,8,16,S      ; #17 Actual quadrature current
M1776->Y:$0008B9,8,16,S      ; #17 Actual direct current
M1777->X:$0008BC,8,16,S      ; #17 Quadrature current-loop integrator output
M1778->Y:$0008BC,8,16,S      ; #17 Direct current-loop integrator output
M1779->X:$0008AE,8,16,S      ; #17 PID internal filter result (16-bit DAC bits)
; Motor #17 Axis Definition Registers
M1791->L:$0008CF             ; #17 X/U/A/B/C-Axis scale factor (cts/unit)
M1792->L:$0008D0             ; #17 Y/V-Axis scale factor (cts/unit)
M1793->L:$0008D1             ; #17 Z/W-Axis scale factor (cts/unit)
M1794->L:$0008D2             ; #17 Axis offset (cts)
; Servo IC 4 Registers for 2nd ACC-24 Channel 2 (usually for Motor #18)
M1801->X:$079205,0,24,S      ; ENC2 24-bit counter position
M1802->Y:$079202,8,16,S      ; DAC2 16-bit analog output
M1803->X:$079207,0,24,S      ; ENC2 capture/compare position register
M1805->Y:$079207,8,16,S      ; ADC2 16-bit analog input
M1806->Y:$079204,0,24,U      ; ENC2 time between counts (SCLK cycles)
M1810->X:$079204,10,1        ; ENC2 count-write enable control
M1811->X:$079204,11,1        ; EQU2 compare flag latch control
M1812->X:$079204,12,1        ; EQU2 compare output enable
M1813->X:$079204,13,1        ; EQU2 compare invert enable
M1814->X:$079204,14,1        ; AENA2/DIR2 Output
M1816->X:$079204,16,1        ; EQU2 compare flag
M1817->X:$079204,17,1        ; ENC2 position-captured flag
M1818->X:$079204,18,1        ; ENC2 Count-error flag
M1819->X:$079204,19,1        ; ENC2 3rd channel input status
M1820->X:$079204,20,1        ; HMFL2 input status
M1821->X:$079204,21,1        ; -LIM2 (positive end) input status
```

```
M1822->X:$079204,22,1        ; +LIM2 (negative end) input status
M1823->X:$079204,23,1        ; FAULT2 input status
; Motor #18 Status Bits
M1830->Y:$000940,11,1        ; #18 Stopped-on-position-limit bit
M1831->X:$000930,21,1        ; #18 Positive-end-limit-set bit
M1832->X:$000930,22,1        ; #18 Negative-end-limit-set bit
M1833->X:$000930,13,1        ; #18 Desired-velocity-zero bit
M1835->X:$000930,15,1        ; #18 Dwell-in-progress bit
M1837->X:$000930,17,1        ; #18 Running-program bit
M1838->X:$000930,18,1        ; #18 Open-loop-mode bit
M1839->X:$000930,19,1        ; #18 Amplifier-enabled status bit
M1840->Y:$000940,0,1         ; #18 Background in-position bit
M1841->Y:$000940,1,1         ; #18 Warning-following error bit
M1842->Y:$000940,2,1         ; #18 Fatal-following-error bit
M1843->Y:$000940,3,1         ; #18 Amplifier-fault-error bit
M1844->Y:$000940,13,1        ; #18 Foreground in-position bit
M1845->Y:$000940,10,1        ; #18 Home-complete bit
M1846->Y:$000940,6,1         ; #18 Integrated following error fault bit
M1847->Y:$000940,5,1         ; #18 I2T fault bit
M1848->Y:$000940,8,1         ; #18 Phasing error fault bit
M1849->Y:$000940,9,1         ; #18 Phasing search-in-progress bit
; Motor #18 Move Registers
M1861->D:$000908             ; #18 Commanded position (1/[Ixx08*32] cts)
M1862->D:$00090B             ; #18 Actual position (1/[Ixx08*32] cts)
M1863->D:$000947             ; #18 Target (end) position (1/[Ixx08*32] cts)
M1864->D:$00094C             ; #18 Position bias (1/[Ixx08*32] cts)
M1866->X:$00091D,0,24,S      ; #18 Actual velocity (1/[Ixx09*32] cts/cyc)
M1867->D:$00090D             ; #18 Present master pos (1/[Ixx07*32] cts)
M1868->X:$00093F,8,16,S      ; #18 Filter Output (16-bit DAC bits)
M1869->D:$000910             ; #18 Compensation correction (1/[Ixx08*32] cts)
M1870->D:$000934             ; #18 Present phase position (including fraction)
M1871->X:$000934,24,S        ; #18 Present phase position (counts *Ixx70)
M1872->L:$000957             ; #18 Variable jog position/distance (cts)
M1873->Y:$00094E,0,24,S      ; #18 Encoder home capture position (cts)
M1874->D:$00096F             ; #18 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1875->X:$000939,8,16,S      ; #18 Actual quadrature current
M1876->Y:$000939,8,16,S      ; #18 Actual direct current
M1877->X:$00093C,8,16,S      ; #18 Quadrature current-loop integrator output
M1878->Y:$00093C,8,16,S      ; #18 Direct current-loop integrator output
M1879->X:$00092E,8,16,S      ; #18 PID internal filter result (16-bit DAC bits)
; Motor #18 Axis Definition Registers
M1891->L:$00094F             ; #18 X/U/A/B/C-Axis scale factor (cts/unit)
M1892->L:$000950             ; #18 Y/V-Axis scale factor (cts/unit)
M1893->L:$000951             ; #18 Z/W-Axis scale factor (cts/unit)
M1894->L:$000952             ; #18 Axis offset (cts)
```

```
; Servo IC 4 Registers for 2nd ACC-24 Channel 3 (usually for Motor #19)
M1901->X:$079209,0,24,S      ; ENC3 24-bit counter position
M1902->Y:$07920B,8,16,S      ; DAC3 16-bit analog output
M1903->X:$07920B,0,24,S      ; ENC3 capture/compare position register
M1905->Y:$07920E,8,16,S      ; ADC3 16-bit analog input
M1906->Y:$079208,0,24,U      ; ENC3 time between counts (SCLK cycles)
M1910->X:$079208,10,1        ; ENC3 count-write enable control
M1911->X:$079208,11,1        ; EQU3 compare flag latch control
M1912->X:$079208,12,1        ; EQU3 compare output enable
M1913->X:$079208,13,1        ; EQU3 compare invert enable
M1914->X:$079208,14,1        ; AENA3/DIR3 Output
M1916->X:$079208,16,1        ; EQU3 compare flag
M1917->X:$079208,17,1        ; ENC3 position-captured flag
M1918->X:$079208,18,1        ; ENC3 Count-error flag
M1919->X:$079208,19,1        ; ENC3 3rd channel input status
M1920->X:$079208,20,1        ; HMFL3 input status
M1921->X:$079208,21,1        ; -LIM3 (positive end) input status
M1922->X:$079208,22,1        ; +LIM3 (negative end) input status
M1923->X:$079208,23,1        ; FAULT3 input status
; Motor #19 Status Bits
M1930->Y:$0009C0,11,1        ; #19 Stopped-on-position-limit bit
M1931->X:$0009B0,21,1        ; #19 Positive-end-limit-set bit
M1932->X:$0009B0,22,1        ; #19 Negative-end-limit-set bit
M1933->X:$0009B0,13,1        ; #19 Desired-velocity-zero bit
M1935->X:$0009B0,15,1        ; #19 Dwell-in-progress bit
M1937->X:$0009B0,17,1        ; #19 Running-program bit
M1938->X:$0009B0,18,1        ; #19 Open-loop-mode bit
M1939->X:$0009B0,19,1        ; #19 Amplifier-enabled status bit
M1940->Y:$0009C0,0,1         ; #19 Background in-position bit
M1941->Y:$0009C0,1,1         ; #19 Warning-following error bit
M1942->Y:$0009C0,2,1         ; #19 Fatal-following-error bit
M1943->Y:$0009C0,3,1         ; #19 Amplifier-fault-error bit
M1944->Y:$0009C0,13,1        ; #19 Foreground in-position bit
M1945->Y:$0009C0,10,1        ; #19 Home-complete bit
M1946->Y:$0009C0,6,1         ; #19 Integrated following error fault bit
M1947->Y:$0009C0,5,1         ; #19 I2T fault bit
M1948->Y:$0009C0,8,1         ; #19 Phasing error fault bit
M1949->Y:$0009C0,9,1         ; #19 Phasing search-in-progress bit
; Motor #19 Move Registers
M1961->D:$000988             ; #19 Commanded position (1/[Ixx08*32] cts)
M1962->D:$00098B             ; #19 Actual position (1/[Ixx08*32] cts)
M1963->D:$0009C7             ; #19 Target (end) position (1/[Ixx08*32] cts)
M1964->D:$0009CC             ; #19 Position bias (1/[Ixx08*32] cts)
M1966->X:$00099D,0,24,S      ; #19 Actual velocity (1/[Ixx09*32] cts/cyc)
M1967->D:$00098D             ; #19 Present master pos (1/[Ixx07*32] cts)
M1968->X:$0009BF,8,16,S      ; #19 Filter Output (16-bit DAC bits)
M1969->D:$000990             ; #19 Compensation correction (1/[Ixx08*32] cts)
```

```
M1970->D:$0009B4              ; #19 Present phase position (including fraction)
M1971->X:$0009B4,24,S         ; #19 Present phase position (counts *Ixx70)
M1972->L:$0009D7              ; #19 Variable jog position/distance (cts)
M1973->Y:$0009CE,0,24,S       ; #19 Encoder home capture position (cts)
M1974->D:$0009EF              ; #19 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1975->X:$0009B9,8,16,S       ; #19 Actual quadrature current
M1976->Y:$0009B9,8,16,S       ; #19 Actual direct current
M1977->X:$0009BC,8,16,S       ; #19 Quadrature current-loop integrator output
M1978->Y:$0009BC,8,16,S       ; #19 Direct current-loop integrator output
M1979->X:$0009AE,8,16,S       ; #19 PID internal filter result (16-bit DAC bits)
; Motor #19 Axis Definition Registers
M1991->L:$0009CF              ; #19 X/U/A/B/C-Axis scale factor (cts/unit)
M1992->L:$0009D0              ; #19 Y/V-Axis scale factor (cts/unit)
M1993->L:$0009D1              ; #19 Z/W-Axis scale factor (cts/unit)
M1994->L:$0009D2              ; #19 Axis offset (cts)
; Servo IC 4 Registers for 2nd ACC-24 Channel 4 (usually for Motor #20)
M2001->X:$07920D,0,24,S       ; ENC4 24-bit counter position
M2002->Y:$07920A,8,16,S       ; DAC4 16-bit analog output
M2003->X:$07920F,0,24,S       ; ENC4 capture/compare position register
M2004->Y:$000A40,13,1         ; #20 Foreground in-position bit
M2005->Y:$07920F,8,16,S       ; ADC4 16-bit analog input
M2006->Y:$07920C,0,24,U       ; ENC4 time between counts (SCLK cycles)
M2010->X:$07920C,10,1         ; ENC4 count-write enable control
M2011->X:$07920C,11,1         ; EQU4 compare flag latch control
M2012->X:$07920C,12,1         ; EQU4 compare output enable
M2013->X:$07920C,13,1         ; EQU4 compare invert enable
M2014->X:$07920C,14,1         ; AENA4/DIR4 Output
M2016->X:$07920C,16,1         ; EQU4 compare flag
M2017->X:$07920C,17,1         ; ENC4 position-captured flag
M2018->X:$07920C,18,1         ; ENC4 Count-error flag
M2019->X:$07920C,19,1         ; ENC4 3rd channel input status
M2020->X:$07920C,20,1         ; HMFL4 input status
M2021->X:$07920C,21,1         ; -LIM4 (positive end) input status
M2022->X:$07920C,22,1         ; +LIM4 (negative end) input status
M2023->X:$07920C,23,1         ; FAULT4 input status
; Motor #20 Status Bits
M2030->Y:$000A40,11,1         ; #20 Stopped-on-position-limit bit
M2031->X:$000A30,21,1         ; #20 Positive-end-limit-set bit
M2032->X:$000A30,22,1         ; #20 Negative-end-limit-set bit
M2033->X:$000A30,13,1         ; #20 Desired-velocity-zero bit
M2035->X:$000A30,15,1         ; #20 Dwell-in-progress bit
M2037->X:$000A30,17,1         ; #20 Running-program bit
M2038->X:$000A30,18,1         ; #20 Open-loop-mode bit
M2039->X:$000A30,19,1         ; #20 Amplifier-enabled status bit
M2040->Y:$000A40,0,1          ; #20 Background in-position bit
M2041->Y:$000A40,1,1          ; #20 Warning-following error bit
M2042->Y:$000A40,2,1          ; #20 Fatal-following-error bit
```

```
M2043->Y:$000A40,3,1          ; #20 Amplifier-fault-error bit
M2044->Y:$000A40,13,1         ; #20 Foreground in-position bit
M2045->Y:$000A40,10,1         ; #20 Home-complete bit
M2046->Y:$000A40,6,1          ; #20 Integrated following error fault bit
M2047->Y:$000A40,5,1          ; #20 I2T fault bit
M2048->Y:$000A40,8,1          ; #20 Phasing error fault bit
M2049->Y:$000A40,9,1          ; #20 Phasing search-in-progress bit
; Motor #20 Move Registers
M2061->D:$000A08              ; #20 Commanded position (1/[Ixx08*32] cts)
M2062->D:$000A0B              ; #20 Actual position (1/[Ixx08*32] cts)
M2063->D:$000A47              ; #20 Target (end) position (1/[Ixx08*32] cts)
M2064->D:$000A4C              ; #20 Position bias (1/[Ixx08*32] cts)
M2066->X:$000A1D,0,24,S       ; #20 Actual velocity (1/[Ixx09*32] cts/cyc)
M2067->D:$000A0D              ; #20 Present master pos (1/[Ixx07*32] cts)
M2068->X:$000A3F,8,16,S       ; #20 Filter Output (16-bit DAC bits)
M2069->D:$000A10              ; #20 Compensation correction (1/[Ixx08*32] cts)
M2070->D:$000A34              ; #20 Present phase position (including fraction)
M2071->X:$000A34,24,S         ; #20 Present phase position (counts *Ixx70)
M2072->L:$000A57              ; #20 Variable jog position/distance (cts)
M2073->Y:$000A4E,0,24,S       ; #20 Encoder home capture position (cts)
M2074->D:$000A6F              ; #20 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2075->X:$000A39,8,16,S       ; #20 Actual quadrature current
M2076->Y:$000A39,8,16,S       ; #20 Actual direct current
M2077->X:$000A3C,8,16,S       ; #20 Quadrature current-loop integrator output
M2078->Y:$000A3C,8,16,S       ; #20 Direct current-loop integrator output
M2079->X:$000A2E,8,16,S       ; #20 PID internal filter result (16-bit DAC bits)
; Motor #20 Axis Definition Registers
M2091->L:$000A4F              ; #20 X/U/A/B/C-Axis scale factor (cts/unit)
M2092->L:$000A50              ; #20 Y/V-Axis scale factor (cts/unit)
M2093->L:$000A51              ; #20 Z/W-Axis scale factor (cts/unit)
M2094->L:$000A52              ; #20 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 5 (usually for Motor #21)
M2101->X:$079301,0,24,S       ; ENC5 24-bit counter position
M2102->Y:$079303,8,16,S       ; DAC5 16-bit analog output
M2103->X:$079303,0,24,S       ; ENC5 capture/compare position register
M2105->Y:$079306,8,16,S       ; ADC5 16-bit analog input
M2106->Y:$079300,0,24,U       ; ENC5 time between counts (SCLK cycles)
M2110->X:$079300,10,1         ; ENC5 count-write enable control
M2111->X:$079300,11,1         ; EQU5 compare flag latch control
M2112->X:$079300,12,1         ; EQU5 compare output enable
M2113->X:$079300,13,1         ; EQU5 compare invert enable
M2114->X:$079300,14,1         ; AENA5/DIR5 Output
M2116->X:$079300,16,1         ; EQU5 compare flag
M2117->X:$079300,17,1         ; ENC5 position-captured flag
M2118->X:$079300,18,1         ; ENC5 Count-error flag
M2119->X:$079300,19,1         ; ENC5 3rd channel input status
M2120->X:$079300,20,1         ; HMFL5 input status
```

```
M2121->X:$079300,21,1        ; -LIM5 (positive end) input status
M2122->X:$079300,22,1        ; +LIM5 (negative end) input status
M2123->X:$079300,23,1        ; FAULT5 input status
; Motor #21 Status Bits
M2130->Y:$000AC0,11,1        ; #21 Stopped-on-position-limit bit
M2131->X:$000AB0,21,1        ; #21 Positive-end-limit-set bit
M2132->X:$000AB0,22,1        ; #21 Negative-end-limit-set bit
M2133->X:$000AB0,13,1        ; #21 Desired-velocity-zero bit
M2135->X:$000AB0,15,1        ; #21 Dwell-in-progress bit
M2137->X:$000AB0,17,1        ; #21 Running-program bit
M2138->X:$000AB0,18,1        ; #21 Open-loop-mode bit
M2139->X:$000AB0,19,1        ; #21 Amplifier-enabled status bit
M2140->Y:$000AC0,0,1         ; #21 Background in-position bit
M2141->Y:$000AC0,1,1         ; #21 Warning-following error bit
M2142->Y:$000AC0,2,1         ; #21 Fatal-following-error bit
M2143->Y:$000AC0,3,1         ; #21 Amplifier-fault-error bit
M2144->Y:$000AC0,13,1        ; #21 Foreground in-position bit
M2145->Y:$000AC0,10,1        ; #21 Home-complete bit
M2146->Y:$000AC0,6,1         ; #21 Integrated following error fault bit
M2147->Y:$000AC0,5,1         ; #21 I2T fault bit
M2148->Y:$000AC0,8,1         ; #21 Phasing error fault bit
M2149->Y:$000AC0,9,1         ; #21 Phasing search-in-progress bit
; Motor #21 Move Registers
M2161->D:$000A88             ; #21 Commanded position (1/[Ixx08*32] cts)
M2162->D:$000A8B             ; #21 Actual position (1/[Ixx08*32] cts)
M2163->D:$000AC7             ; #21 Target (end) position (1/[Ixx08*32] cts)
M2164->D:$000ACC             ; #21 Position bias (1/[Ixx08*32] cts)
M2166->X:$000A9D,0,24,S      ; #21 Actual velocity (1/[Ixx09*32] cts/cyc)
M2167->D:$000A8D             ; #21 Present master pos (1/[Ixx07*32] cts)
M2168->X:$000ABF,8,16,S      ; #21 Filter Output (16-bit DAC bits)
M2169->D:$000A90             ; #21 Compensation correction (1/[Ixx08*32] cts)
M2170->D:$000AB4             ; #21 Present phase position (including fraction)
M2171->X:$000AB4,24,S        ; #21 Present phase position (counts *Ixx70)
M2172->L:$000AD7             ; #21 Variable jog position/distance (cts)
M2173->Y:$000ACE,0,24,S      ; #21 Encoder home capture position (cts)
M2174->D:$000AEF             ; #21 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2175->X:$000AB9,8,16,S      ; #21 Actual quadrature current
M2176->Y:$000AB9,8,16,S      ; #21 Actual direct current
M2177->X:$000ABC,8,16,S      ; #21 Quadrature current-loop integrator output
M2178->Y:$000ABC,8,16,S      ; #21 Direct current-loop integrator output
M2179->X:$000AAE,8,16,S      ; #21 PID internal filter result (16-bit DAC bits)
; Motor #21 Axis Definition Registers
M2191->L:$000ACF             ; #21 X/U/A/B/C-Axis scale factor (cts/unit)
M2192->L:$000AD0             ; #21 Y/V-Axis scale factor (cts/unit)
M2193->L:$000AD1             ; #21 Z/W-Axis scale factor (cts/unit)
M2194->L:$000AD2             ; #21 Axis offset (cts)
```

```
; Servo IC 5 Registers for 2nd ACC-24 Channel 6 (usually for Motor #22)
M2201->X:$079305,0,24,S      ; ENC6 24-bit counter position
M2202->Y:$079302,8,16,S      ; DAC6 16-bit analog output
M2203->X:$079307,0,24,S      ; ENC6 capture/compare position register
M2205->Y:$079307,8,16,S      ; ADC6 16-bit analog input
M2206->Y:$079304,0,24,U      ; ENC6 time between counts (SCLK cycles)
M2210->X:$079304,10,1        ; ENC6 count-write enable control
M2211->X:$079304,11,1        ; EQU6 compare flag latch control
M2212->X:$079304,12,1        ; EQU6 compare output enable
M2213->X:$079304,13,1        ; EQU6 compare invert enable
M2214->X:$079304,14,1        ; AENA6/DIR6 Output
M2216->X:$079304,16,1        ; EQU6 compare flag
M2217->X:$079304,17,1        ; ENC6 position-captured flag
M2218->X:$079304,18,1        ; ENC6 Count-error flag
M2219->X:$079304,19,1        ; ENC6 3rd channel input status
M2220->X:$079304,20,1        ; HMFL6 input status
M2221->X:$079304,21,1        ; -LIM6 (positive end) input status
M2222->X:$079304,22,1        ; +LIM6 (negative end) input status
M2223->X:$079304,23,1        ; FAULT6 input status
; Motor #22 Status Bits
M2230->Y:$000B40,11,1        ; #22 Stopped-on-position-limit bit
M2231->X:$000B30,21,1        ; #22 Positive-end-limit-set bit
M2232->X:$000B30,22,1        ; #22 Negative-end-limit-set bit
M2233->X:$000B30,13,1        ; #22 Desired-velocity-zero bit
M2235->X:$000B30,15,1        ; #22 Dwell-in-progress bit
M2237->X:$000B30,17,1        ; #22 Running-program bit
M2238->X:$000B30,18,1        ; #22 Open-loop-mode bit
M2239->X:$000B30,19,1        ; #22 Amplifier-enabled status bit
M2240->Y:$000B40,0,1         ; #22 Background in-position bit
M2241->Y:$000B40,1,1         ; #22 Warning-following error bit
M2242->Y:$000B40,2,1         ; #22 Fatal-following-error bit
M2243->Y:$000B40,3,1         ; #22 Amplifier-fault-error bit
M2244->Y:$000B40,13,1        ; #22 Foreground in-position bit
M2245->Y:$000B40,10,1        ; #22 Home-complete bit
M2246->Y:$000B40,6,1         ; #22 Integrated following error fault bit
M2247->Y:$000B40,5,1         ; #22 I2T fault bit
M2248->Y:$000B40,8,1         ; #22 Phasing error fault bit
M2249->Y:$000B40,9,1         ; #22 Phasing search-in-progress bit
; Motor #22 Move Registers
M2261->D:$000B08             ; #22 Commanded position (1/[Ixx08*32] cts)
M2262->D:$000B0B             ; #22 Actual position (1/[Ixx08*32] cts)
M2263->D:$000B47             ; #22 Target (end) position (1/[Ixx08*32] cts)
M2264->D:$000B4C             ; #22 Position bias (1/[Ixx08*32] cts)
M2266->X:$000B1D,0,24,S      ; #22 Actual velocity (1/[Ixx09*32] cts/cyc)
M2267->D:$000B0D             ; #22 Present master pos (1/[Ixx07*32] cts)
M2268->X:$000B3F,8,16,S      ; #22 Filter Output (16-bit DAC bits)
M2269->D:$000B10             ; #22 Compensation correction (1/[Ixx08*32] cts)
```

```
M2270->D:$000B34              ; #22 Present phase position (including fraction)
M2271->X:$000B34,24,S         ; #22 Present phase position (counts *Ixx70)
M2272->L:$000B57              ; #22 Variable jog position/distance (cts)
M2273->Y:$000B4E,0,24,S       ; #22 Encoder home capture position (cts)
M2274->D:$000B6F              ; #22 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2275->X:$000B39,8,16,S       ; #22 Actual quadrature current
M2276->Y:$000B39,8,16,S       ; #22 Actual direct current
M2277->X:$000B3C,8,16,S       ; #22 Quadrature current-loop integrator output
M2278->Y:$000B3C,8,16,S       ; #22 Direct current-loop integrator output
M2279->X:$000B2E,8,16,S       ; #22 PID internal filter result (16-bit DAC bits)
; Motor #22 Axis Definition Registers
M2291->L:$000B4F              ; #22 X/U/A/B/C-Axis scale factor (cts/unit)
M2292->L:$000B50              ; #22 Y/V-Axis scale factor (cts/unit)
M2293->L:$000B51              ; #22 Z/W-Axis scale factor (cts/unit)
M2294->L:$000B52              ; #22 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 7 (usually for Motor #23)
M2301->X:$079309,0,24,S       ; ENC7 24-bit counter position
M2302->Y:$07930B,8,16,S       ; DAC7 16-bit analog output
M2303->X:$07930B,0,24,S       ; ENC7 capture/compare position register
M2305->Y:$07930E,8,16,S       ; ADC7 16-bit analog input
M2306->Y:$079308,0,24,U       ; ENC7 time between counts (SCLK cycles)
M2310->X:$079308,10,1         ; ENC7 count-write enable control
M2311->X:$079308,11,1         ; EQU7 compare flag latch control
M2312->X:$079308,12,1         ; EQU7 compare output enable
M2313->X:$079308,13,1         ; EQU7 compare invert enable
M2314->X:$079308,14,1         ; AENA7/DIR7 Output
M2316->X:$079308,16,1         ; EQU7 compare flag
M2317->X:$079308,17,1         ; ENC7 position-captured flag
M2318->X:$079308,18,1         ; ENC7 Count-error flag
M2319->X:$079308,19,1         ; ENC7 3rd channel input status
M2320->X:$079308,20,1         ; HMFL7 input status
M2321->X:$079308,21,1         ; -LIM7 (positive end) input status
M2322->X:$079308,22,1         ; +LIM7 (negative end) input status
M2323->X:$079308,23,1         ; FAULT7 input status
; Motor #23 Status Bits
M2330->Y:$000BC0,11,1         ; #23 Stopped-on-position-limit bit
M2331->X:$000BB0,21,1         ; #23 Positive-end-limit-set bit
M2332->X:$000BB0,22,1         ; #23 Negative-end-limit-set bit
M2333->X:$000BB0,13,1         ; #23 Desired-velocity-zero bit
M2335->X:$000BB0,15,1         ; #23 Dwell-in-progress bit
M2337->X:$000BB0,17,1         ; #23 Running-program bit
M2338->X:$000BB0,18,1         ; #23 Open-loop-mode bit
M2339->X:$000BB0,19,1         ; #23 Amplifier-enabled status bit
M2340->Y:$000BC0,0,1          ; #23 Background in-position bit
M2341->Y:$000BC0,1,1          ; #23 Warning-following error bit
M2342->Y:$000BC0,2,1          ; #23 Fatal-following-error bit
M2343->Y:$000BC0,3,1          ; #23 Amplifier-fault-error bit
```

```
M2344->Y:$000BC0,13,1        ; #23 Foreground in-position bit
M2345->Y:$000BC0,10,1        ; #23 Home-complete bit
M2346->Y:$000BC0,6,1         ; #23 Integrated following error fault bit
M2347->Y:$000BC0,5,1         ; #23 I2T fault bit
M2348->Y:$000BC0,8,1         ; #23 Phasing error fault bit
M2349->Y:$000BC0,9,1         ; #23 Phasing search-in-progress bit
; Motor #23 Move Registers
M2361->D:$000B88             ; #23 Commanded position (1/[Ixx08*32] cts)
M2362->D:$000B8B             ; #23 Actual position (1/[Ixx08*32] cts)
M2363->D:$000BC7             ; #23 Target (end) position (1/[Ixx08*32] cts)
M2364->D:$000BCC             ; #23 Position bias (1/[Ixx08*32] cts)
M2366->X:$000B9D,0,24,S      ; #23 Actual velocity (1/[Ixx09*32] cts/cyc)
M2367->D:$000B8D             ; #23 Present master pos (1/[Ixx07*32] cts)
M2368->X:$000BBF,8,16,S      ; #23 Filter Output (16-bit DAC bits)
M2369->D:$000B90             ; #23 Compensation correction (1/[Ixx08*32] cts)
M2370->D:$000BB4             ; #23 Present phase position (including fraction)
M2371->X:$000BB4,24,S        ; #23 Present phase position (counts *Ixx70)
M2372->L:$000BD7             ; #23 Variable jog position/distance (cts)
M2373->Y:$000BCE,0,24,S      ; #23 Encoder home capture position (cts)
M2374->D:$000BEF             ; #23 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2375->X:$000BB9,8,16,S      ; #23 Actual quadrature current
M2376->Y:$000BB9,8,16,S      ; #23 Actual direct current
M2377->X:$000BBC,8,16,S      ; #23 Quadrature current-loop integrator output
M2378->Y:$000BBC,8,16,S      ; #23 Direct current-loop integrator output
M2379->X:$000BAE,8,16,S      ; #23 PID internal filter result (16-bit DAC bits)
; Motor #23 Axis Definition Registers
M2391->L:$000BCF             ; #23 X/U/A/B/C-Axis scale factor (cts/unit)
M2392->L:$000BD0             ; #23 Y/V-Axis scale factor (cts/unit)
M2393->L:$000BD1             ; #23 Z/W-Axis scale factor (cts/unit)
M2394->L:$000BD2             ; #23 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 8 (usually for Motor #24)
M2401->X:$07930D,0,24,S      ; ENC8 24-bit counter position
M2402->Y:$07930A,8,16,S      ; DAC8 16-bit analog output
M2403->X:$07930F,0,24,S      ; ENC8 capture/compare position register
M2405->Y:$07930F,8,16,S      ; ADC8 16-bit analog input
M2406->Y:$07930C,0,24,U      ; ENC8 time between counts (SCLK cycles)
M2410->X:$07930C,10,1        ; ENC8 count-write enable control
M2411->X:$07930C,11,1        ; EQU8 compare flag latch control
M2412->X:$07930C,12,1        ; EQU8 compare output enable
M2413->X:$07930C,13,1        ; EQU8 compare invert enable
M2414->X:$07930C,14,1        ; AENA8/DIR8 Output
M2416->X:$07930C,16,1        ; EQU8 compare flag
M2417->X:$07930C,17,1        ; ENC8 position-captured flag
M2418->X:$07930C,18,1        ; ENC8 Count-error flag
M2419->X:$07930C,19,1        ; ENC8 3rd channel input status
M2420->X:$07930C,20,1        ; HMFL8 input status
M2421->X:$07930C,21,1        ; -LIM8 (positive end) input status
```

```
M2422->X:$07930C,22,1        ; +LIM8 (negative end) input status
M2423->X:$07930C,23,1        ; FAULT8 input status
; Motor #24 Status Bits
M2430->Y:$000C40,11,1        ; #24 Stopped-on-position-limit bit
M2431->X:$000C30,21,1        ; #24 Positive-end-limit-set bit
M2432->X:$000C30,22,1        ; #24 Negative-end-limit-set bit
M2433->X:$000C30,13,1        ; #24 Desired-velocity-zero bit
M2435->X:$000C30,15,1        ; #24 Dwell-in-progress bit
M2437->X:$000C30,17,1        ; #24 Running-program bit
M2438->X:$000C30,18,1        ; #24 Open-loop-mode bit
M2439->X:$000C30,19,1        ; #24 Amplifier-enabled status bit
M2440->Y:$000C40,0,1         ; #24 Background in-position bit
M2441->Y:$000C40,1,1         ; #24 Warning-following error bit
M2442->Y:$000C40,2,1         ; #24 Fatal-following-error bit
M2443->Y:$000C40,3,1         ; #24 Amplifier-fault-error bit
M2444->Y:$000C40,13,1        ; #24 Foreground in-position bit
M2445->Y:$000C40,10,1        ; #24 Home-complete bit
M2446->Y:$000C40,6,1         ; #24 Integrated following error fault bit
M2447->Y:$000C40,5,1         ; #24 I2T fault bit
M2448->Y:$000C40,8,1         ; #24 Phasing error fault bit
M2449->Y:$000C40,9,1         ; #24 Phasing search-in-progress bit
; Motor #24 Move Registers
M2461->D:$000C08             ; #24 Commanded position (1/[Ixx08*32] cts)
M2462->D:$000C0B             ; #24 Actual position (1/[Ixx08*32] cts)
M2463->D:$000C47             ; #24 Target (end) position (1/[Ixx08*32] cts)
M2464->D:$000C4C             ; #24 Position bias (1/[Ixx08*32] cts)
M2466->X:$000C1D,0,24,S      ; #24 Actual velocity (1/[Ixx09*32] cts/cyc)
M2467->D:$000C0D             ; #24 Present master pos (1/[Ixx07*32] cts)
M2468->X:$000C3F,8,16,S      ; #24 Filter Output (16-bit DAC bits)
M2469->D:$000C10             ; #24 Compensation correction (1/[Ixx08*32] cts)
M2470->D:$000C34             ; #24 Present phase position (including fraction)
M2471->X:$000C34,24,S        ; #24 Present phase position (counts *Ixx70)
M2472->L:$000C57             ; #24 Variable jog position/distance (cts)
M2473->Y:$000C4E,0,24,S      ; #24 Encoder home capture position (cts)
M2474->D:$000C6F             ; #24 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2475->X:$000C39,8,16,S      ; #24 Actual quadrature current
M2476->Y:$000C39,8,16,S      ; #24 Actual direct current
M2477->X:$000C3C,8,16,S      ; #24 Quadrature current-loop integrator output
M2478->Y:$000C3C,8,16,S      ; #24 Direct current-loop integrator output
M2479->X:$000C2E,8,16,S      ; #24 PID internal filter result (16-bit DAC bits)
; Motor #24 Axis Definition Registers
M2491->L:$000C4F             ; #24 X/U/A/B/C-Axis scale factor (cts/unit)
M2492->L:$000C50             ; #24 Y/V-Axis scale factor (cts/unit)
M2493->L:$000C51             ; #24 Z/W-Axis scale factor (cts/unit)
M2494->L:$000C52             ; #24 Axis offset (cts)
```

```
; Servo IC 6 Registers for 3rd ACC-24 Channel 1 (usually for Motor #25)
M2501->X:$07A201,0,24,S      ; ENC1 24-bit counter position
M2502->Y:$07A203,8,16,S      ; DAC1 16-bit analog output
M2503->X:$07A203,0,24,S      ; ENC1 capture/compare position register
M2505->Y:$07A206,8,16,S      ; ADC1 16-bit analog input
M2506->Y:$07A200,0,24,U      ; ENC1 time between counts (SCLK cycles)
M2510->X:$07A200,10,1        ; ENC1 count-write enable control
M2511->X:$07A200,11,1        ; EQU1 compare flag latch control
M2512->X:$07A200,12,1        ; EQU1 compare output enable
M2513->X:$07A200,13,1        ; EQU1 compare invert enable
M2514->X:$07A200,14,1        ; AENA1/DIR1 Output
M2516->X:$07A200,16,1        ; EQU1 compare flag
M2517->X:$07A200,17,1        ; ENC1 position-captured flag
M2518->X:$07A200,18,1        ; ENC1 Count-error flag
M2519->X:$07A200,19,1        ; ENC1 3rd channel input status
M2520->X:$07A200,20,1        ; HMFL1 input status
M2521->X:$07A200,21,1        ; -LIM1 (positive end) input status
M2522->X:$07A200,22,1        ; +LIM1 (negative end) input status
M2523->X:$07A200,23,1        ; FAULT1 input status
; Motor #25 Status Bits
M2530->Y:$000CC0,11,1        ; #25 Stopped-on-position-limit bit
M2531->X:$000CB0,21,1        ; #25 Positive-end-limit-set bit
M2532->X:$000CB0,22,1        ; #25 Negative-end-limit-set bit
M2533->X:$000CB0,13,1        ; #25 Desired-velocity-zero bit
M2535->X:$000CB0,15,1        ; #25 Dwell-in-progress bit
M2537->X:$000CB0,17,1        ; #25 Running-program bit
M2538->X:$000CB0,18,1        ; #25 Open-loop-mode bit
M2539->X:$000CB0,19,1        ; #25 Amplifier-enabled status bit
M2540->Y:$000CC0,0,1         ; #25 Background in-position bit
M2541->Y:$000CC0,1,1         ; #25 Warning-following error bit
M2542->Y:$000CC0,2,1         ; #25 Fatal-following-error bit
M2543->Y:$000CC0,3,1         ; #25 Amplifier-fault-error bit
M2544->Y:$000CC0,13,1        ; #25 Foreground in-position bit
M2545->Y:$000CC0,10,1        ; #25 Home-complete bit
M2546->Y:$000CC0,6,1         ; #25 Integrated following error fault bit
M2547->Y:$000CC0,5,1         ; #25 I2T fault bit
M2548->Y:$000CC0,8,1         ; #25 Phasing error fault bit
M2549->Y:$000CC0,9,1         ; #25 Phasing search-in-progress bit
; Motor #25 Move Registers
M2561->D:$000C88             ; #25 Commanded position (1/[Ixx08*32] cts)
M2562->D:$000C8B             ; #25 Actual position (1/[Ixx08*32] cts)
M2563->D:$000CC7             ; #25 Target (end) position (1/[Ixx08*32] cts)
M2564->D:$000CCC             ; #25 Position bias (1/[Ixx08*32] cts)
M2566->X:$000C9D,0,24,S      ; #25 Actual velocity (1/[Ixx09*32] cts/cyc)
M2567->D:$000C8D             ; #25 Present master pos (1/[Ixx07*32] cts)
M2568->X:$000CBF,8,16,S      ; #25 Filter Output (16-bit DAC bits)
M2569->D:$000C90             ; #25 Compensation correction (1/[Ixx08*32] cts)
```

```
M2570->D:$000CB4              ; #25 Present phase position (including fraction)
M2571->X:$000CB4,24,S         ; #25 Present phase position (counts *Ixx70)
M2572->L:$000CD7              ; #25 Variable jog position/distance (cts)
M2573->Y:$000CCE,0,24,S       ; #25 Encoder home capture position (cts)
M2574->D:$000CEF              ; #25 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2575->X:$000CB9,8,16,S       ; #25 Actual quadrature current
M2576->Y:$000CB9,8,16,S       ; #25 Actual direct current
M2577->X:$000CBC,8,16,S       ; #25 Quadrature current-loop integrator output
M2578->Y:$000CBC,8,16,S       ; #25 Direct current-loop integrator output
M2579->X:$000CAE,8,16,S       ; #25 PID internal filter result (16-bit DAC bits)
; Motor #25 Axis Definition Registers
M2591->L:$000CCF              ; #25 X/U/A/B/C-Axis scale factor (cts/unit)
M2592->L:$000CD0              ; #25 Y/V-Axis scale factor (cts/unit)
M2593->L:$000CD1              ; #25 Z/W-Axis scale factor (cts/unit)
M2594->L:$000CD2              ; #25 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 2 (usually for Motor #26)
M2601->X:$07A205,0,24,S       ; ENC2 24-bit counter position
M2602->Y:$07A202,8,16,S       ; DAC2 16-bit analog output
M2603->X:$07A207,0,24,S       ; ENC2 capture/compare position register
M2605->Y:$07A207,8,16,S       ; ADC2 16-bit analog input
M2606->Y:$07A204,0,24,U       ; ENC2 time between counts (SCLK cycles)
M2610->X:$07A204,10,1         ; ENC2 count-write enable control
M2611->X:$07A204,11,1         ; EQU2 compare flag latch control
M2612->X:$07A204,12,1         ; EQU2 compare output enable
M2613->X:$07A204,13,1         ; EQU2 compare invert enable
M2614->X:$07A204,14,1         ; AENA2/DIR2 Output
M2616->X:$07A204,16,1         ; EQU2 compare flag
M2617->X:$07A204,17,1         ; ENC2 position-captured flag
M2618->X:$07A204,18,1         ; ENC2 Count-error flag
M2619->X:$07A204,19,1         ; ENC2 3rd channel input status
M2620->X:$07A204,20,1         ; HMFL2 input status
M2621->X:$07A204,21,1         ; -LIM2 (positive end) input status
M2622->X:$07A204,22,1         ; +LIM2 (negative end) input status
M2623->X:$07A204,23,1         ; FAULT2 input status
; Motor #26 Status Bits
M2630->Y:$000D40,11,1         ; #26 Stopped-on-position-limit bit
M2631->X:$000D30,21,1         ; #26 Positive-end-limit-set bit
M2632->X:$000D30,22,1         ; #26 Negative-end-limit-set bit
M2633->X:$000D30,13,1         ; #26 Desired-velocity-zero bit
M2635->X:$000D30,15,1         ; #26 Dwell-in-progress bit
M2637->X:$000D30,17,1         ; #26 Running-program bit
M2638->X:$000D30,18,1         ; #26 Open-loop-mode bit
M2639->X:$000D30,19,1         ; #26 Amplifier-enabled status bit
M2640->Y:$000D40,0,1          ; #26 Background in-position bit
M2641->Y:$000D40,1,1          ; #26 Warning-following error bit
M2642->Y:$000D40,2,1          ; #26 Fatal-following-error bit
M2643->Y:$000D40,3,1          ; #26 Amplifier-fault-error bit
```

```
M2644->Y:$000D40,13,1        ; #26 Foreground in-position bit
M2645->Y:$000D40,10,1        ; #26 Home-complete bit
M2646->Y:$000D40,6,1         ; #26 Integrated following error fault bit
M2647->Y:$000D40,5,1         ; #26 I2T fault bit
M2648->Y:$000D40,8,1         ; #26 Phasing error fault bit
M2649->Y:$000D40,9,1         ; #26 Phasing search-in-progress bit
; Motor #26 Move Registers
M2661->D:$000D08             ; #26 Commanded position (1/[Ixx08*32] cts)
M2662->D:$000D0B             ; #26 Actual position (1/[Ixx08*32] cts)
M2663->D:$000D47             ; #26 Target (end) position (1/[Ixx08*32] cts)
M2664->D:$000D4C             ; #26 Position bias (1/[Ixx08*32] cts)
M2666->X:$000D1D,0,24,S      ; #26 Actual velocity (1/[Ixx09*32] cts/cyc)
M2667->D:$000D0D             ; #26 Present master pos (1/[Ixx07*32] cts)
M2668->X:$000D3F,8,16,S      ; #26 Filter Output (16-bit DAC bits)
M2669->D:$000D10             ; #26 Compensation correction (1/[Ixx08*32] cts)
M2670->D:$000D34             ; #26 Present phase position (including fraction)
M2671->X:$000D34,24,S        ; #26 Present phase position (counts *Ixx70)
M2672->L:$000D57             ; #26 Variable jog position/distance (cts)
M2673->Y:$000D4E,0,24,S      ; #26 Encoder home capture position (cts)
M2674->D:$000D6F             ; #26 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2675->X:$000D39,8,16,S      ; #26 Actual quadrature current
M2676->Y:$000D39,8,16,S      ; #26 Actual direct current
M2677->X:$000D3C,8,16,S      ; #26 Quadrature current-loop integrator output
M2678->Y:$000D3C,8,16,S      ; #26 Direct current-loop integrator output
M2679->X:$000D2E,8,16,S      ; #26 PID internal filter result (16-bit DAC bits)
; Motor #26 Axis Definition Registers
M2691->L:$000D4F             ; #26 X/U/A/B/C-Axis scale factor (cts/unit)
M2692->L:$000D50             ; #26 Y/V-Axis scale factor (cts/unit)
M2693->L:$000D51             ; #26 Z/W-Axis scale factor (cts/unit)
M2694->L:$000D52             ; #26 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 3 (usually for Motor #27)
M2701->X:$07A209,0,24,S      ; ENC3 24-bit counter position
M2702->Y:$07A20B,8,16,S      ; DAC3 16-bit analog output
M2703->X:$07A20B,0,24,S      ; ENC3 capture/compare position register
M2705->Y:$07A20E,8,16,S      ; ADC3 16-bit analog input
M2706->Y:$07A208,0,24,U      ; ENC3 time between counts (SCLK cycles)
M2710->X:$07A208,10,1        ; ENC3 count-write enable control
M2711->X:$07A208,11,1        ; EQU3 compare flag latch control
M2712->X:$07A208,12,1        ; EQU3 compare output enable
M2713->X:$07A208,13,1        ; EQU3 compare invert enable
M2714->X:$07A208,14,1        ; AENA3/DIR3 Output
M2716->X:$07A208,16,1        ; EQU3 compare flag
M2717->X:$07A208,17,1        ; ENC3 position-captured flag
M2718->X:$07A208,18,1        ; ENC3 Count-error flag
M2719->X:$07A208,19,1        ; ENC3 3rd channel input status
M2720->X:$07A208,20,1        ; HMFL3 input status
M2721->X:$07A208,21,1        ; -LIM3 (positive end) input status
```

```
M2722->X:$07A208,22,1        ; +LIM3 (negative end) input status
M2723->X:$07A208,23,1        ; FAULT3 input status
; Motor #27 Status Bits
M2730->Y:$000DC0,11,1        ; #27 Stopped-on-position-limit bit
M2731->X:$000DB0,21,1        ; #27 Positive-end-limit-set bit
M2732->X:$000DB0,22,1        ; #27 Negative-end-limit-set bit
M2733->X:$000DB0,13,1        ; #27 Desired-velocity-zero bit
M2735->X:$000DB0,15,1        ; #27 Dwell-in-progress bit
M2737->X:$000DB0,17,1        ; #27 Running-program bit
M2738->X:$000DB0,18,1        ; #27 Open-loop-mode bit
M2739->X:$000DB0,19,1        ; #27 Amplifier-enabled status bit
M2740->Y:$000DC0,0,1         ; #27 Background in-position bit
M2741->Y:$000DC0,1,1         ; #27 Warning-following error bit
M2742->Y:$000DC0,2,1         ; #27 Fatal-following-error bit
M2743->Y:$000DC0,3,1         ; #27 Amplifier-fault-error bit
M2744->Y:$000DC0,13,1        ; #27 Foreground in-position bit
M2745->Y:$000DC0,10,1        ; #27 Home-complete bit
M2746->Y:$000DC0,6,1         ; #27 Integrated following error fault bit
M2747->Y:$000DC0,5,1         ; #27 I2T fault bit
M2748->Y:$000DC0,8,1         ; #27 Phasing error fault bit
M2749->Y:$000DC0,9,1         ; #27 Phasing search-in-progress bit
; Motor #27 Move Registers
M2761->D:$000D88             ; #27 Commanded position (1/[Ixx08*32] cts)
M2762->D:$000D8B             ; #27 Actual position (1/[Ixx08*32] cts)
M2763->D:$000DC7             ; #27 Target (end) position (1/[Ixx08*32] cts)
M2764->D:$000DCC             ; #27 Position bias (1/[Ixx08*32] cts)
M2766->X:$000D9D,0,24,S      ; #27 Actual velocity (1/[Ixx09*32] cts/cyc)
M2767->D:$000D8D             ; #27 Present master pos (1/[Ixx07*32] cts)
M2768->X:$000DBF,8,16,S      ; #27 Filter Output (16-bit DAC bits)
M2769->D:$000D90             ; #27 Compensation correction (1/[Ixx08*32] cts)
M2770->D:$000DB4             ; #27 Present phase position (including fraction)
M2771->X:$000DB4,24,S        ; #27 Present phase position (counts *Ixx70)
M2772->L:$000DD7             ; #27 Variable jog position/distance (cts)
M2773->Y:$000DCE,0,24,S      ; #27 Encoder home capture position (cts)
M2774->D:$000DEF             ; #27 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2775->X:$000DB9,8,16,S      ; #27 Actual quadrature current
M2776->Y:$000DB9,8,16,S      ; #27 Actual direct current
M2777->X:$000DBC,8,16,S      ; #27 Quadrature current-loop integrator output
M2778->Y:$000DBC,8,16,S      ; #27 Direct current-loop integrator output
M2779->X:$000DAE,8,16,S      ; #27 PID internal filter result (16-bit DAC bits)
; Motor #27 Axis Definition Registers
M2791->L:$000DCF             ; #27 X/U/A/B/C-Axis scale factor (cts/unit)
M2792->L:$000DD0             ; #27 Y/V-Axis scale factor (cts/unit)
M2793->L:$000DD1             ; #27 Z/W-Axis scale factor (cts/unit)
M2794->L:$000DD2             ; #27 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 4 (usually for Motor #28)
M2801->X:$07A20D,0,24,S      ; ENC4 24-bit counter position
```

```
M2802->Y:$07A20A,8,16,S     ; DAC4 16-bit analog output
M2803->X:$07A20F,0,24,S     ; ENC4 capture/compare position register
M2805->Y:$07A20F,8,16,S     ; ADC4 16-bit analog input
M2806->Y:$07A20C,0,24,U     ; ENC4 time between counts (SCLK cycles)
M2810->X:$07A20C,10,1       ; ENC4 count-write enable control
M2811->X:$07A20C,11,1       ; EQU4 compare flag latch control
M2812->X:$07A20C,12,1       ; EQU4 compare output enable
M2813->X:$07A20C,13,1       ; EQU4 compare invert enable
M2814->X:$07A20C,14,1       ; AENA4/DIR4 Output
M2816->X:$07A20C,16,1       ; EQU4 compare flag
M2817->X:$07A20C,17,1       ; ENC4 position-captured flag
M2818->X:$07A20C,18,1       ; ENC4 Count-error flag
M2819->X:$07A20C,19,1       ; ENC4 3rd channel input status
M2820->X:$07A20C,20,1       ; HMFL4 input status
M2821->X:$07A20C,21,1       ; -LIM4 (positive end) input status
M2822->X:$07A20C,22,1       ; +LIM4 (negative end) input status
M2823->X:$07A20C,23,1       ; FAULT4 input status
; Motor #28 Status Bits
M2830->Y:$000E40,11,1       ; #28 Stopped-on-position-limit bit
M2831->X:$000E30,21,1       ; #28 Positive-end-limit-set bit
M2832->X:$000E30,22,1       ; #28 Negative-end-limit-set bit
M2833->X:$000E30,13,1       ; #28 Desired-velocity-zero bit
M2835->X:$000E30,15,1       ; #28 Dwell-in-progress bit
M2837->X:$000E30,17,1       ; #28 Running-program bit
M2838->X:$000E30,18,1       ; #28 Open-loop-mode bit
M2839->X:$000E30,19,1       ; #28 Amplifier-enabled status bit
M2840->Y:$000E40,0,1        ; #28 Background in-position bit
M2841->Y:$000E40,1,1        ; #28 Warning-following error bit
M2842->Y:$000E40,2,1        ; #28 Fatal-following-error bit
M2843->Y:$000E40,3,1        ; #28 Amplifier-fault-error bit
M2844->Y:$000E40,13,1       ; #28 Foreground in-position bit
M2845->Y:$000E40,10,1       ; #28 Home-complete bit
M2846->Y:$000E40,6,1        ; #28 Integrated following error fault bit
M2847->Y:$000E40,5,1        ; #28 I2T fault bit
M2848->Y:$000E40,8,1        ; #28 Phasing error fault bit
M2849->Y:$000E40,9,1        ; #28 Phasing search-in-progress bit
; Motor #28 Move Registers
M2861->D:$000E08            ; #28 Commanded position (1/[Ixx08*32] cts)
M2862->D:$000E0B            ; #28 Actual position (1/[Ixx08*32] cts)
M2863->D:$000E47            ; #28 Target (end) position (1/[Ixx08*32] cts)
M2864->D:$000E4C            ; #28 Position bias (1/[Ixx08*32] cts)
M2866->X:$000E1D,0,24,S     ; #28 Actual velocity (1/[Ixx09*32] cts/cyc)
M2867->D:$000E0D            ; #28 Present master pos (1/[Ixx07*32] cts)
M2868->X:$000E3F,8,16,S     ; #28 Filter Output (16-bit DAC bits)
M2869->D:$000E10            ; #28 Compensation correction (1/[Ixx08*32] cts)
M2870->D:$000E34            ; #28 Present phase position (including fraction)
M2871->X:$000E34,24,S       ; #28 Present phase position (counts *Ixx70)
```

```
M2872->L:$000E57              ; #28 Variable jog position/distance (cts)
M2873->Y:$000E4E,0,24,S       ; #28 Encoder home capture position (cts)
M2874->D:$000E6F              ; #28 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2875->X:$000E39,8,16,S       ; #28 Actual quadrature current
M2876->Y:$000E39,8,16,S       ; #28 Actual direct current
M2877->X:$000E3C,8,16,S       ; #28 Quadrature current-loop integrator output
M2878->Y:$000E3C,8,16,S       ; #28 Direct current-loop integrator output
M2879->X:$000E2E,8,16,S       ; #28 PID internal filter result (16-bit DAC bits)
; Motor #28 Axis Definition Registers
M2891->L:$000E4F              ; #28 X/U/A/B/C-Axis scale factor (cts/unit)
M2892->L:$000E50              ; #28 Y/V-Axis scale factor (cts/unit)
M2893->L:$000E51              ; #28 Z/W-Axis scale factor (cts/unit)
M2894->L:$000E52              ; #28 Axis offset (cts)
; Servo IC 7 Registers for 3rd ACC-24 Channel 5 (usually for Motor #29)
M2901->X:$07A301,0,24,S       ; ENC5 24-bit counter position
M2902->Y:$07A303,8,16,S       ; DAC5 16-bit analog output
M2903->X:$07A303,0,24,S       ; ENC5 capture/compare position register
M2905->Y:$07A306,8,16,S       ; ADC5 16-bit analog input
M2906->Y:$07A300,0,24,U       ; ENC5 time between counts (SCLK cycles)
M2910->X:$07A300,10,1         ; ENC5 count-write enable control
M2911->X:$07A300,11,1         ; EQU5 compare flag latch control
M2912->X:$07A300,12,1         ; EQU5 compare output enable
M2913->X:$07A300,13,1         ; EQU5 compare invert enable
M2914->X:$07A300,14,1         ; AENA5/DIR5 Output
M2916->X:$07A300,16,1         ; EQU5 compare flag
M2917->X:$07A300,17,1         ; ENC5 position-captured flag
M2918->X:$07A300,18,1         ; ENC5 Count-error flag
M2919->X:$07A300,19,1         ; ENC5 3rd channel input status
M2920->X:$07A300,20,1         ; HMFL5 input status
M2921->X:$07A300,21,1         ; -LIM5 (positive end) input status
M2922->X:$07A300,22,1         ; +LIM5 (negative end) input status
M2923->X:$07A300,23,1         ; FAULT5 input status
; Motor #29 Status Bits
M2930->Y:$000EC0,11,1         ; #29 Stopped-on-position-limit bit
M2931->X:$000EB0,21,1         ; #29 Positive-end-limit-set bit
M2932->X:$000EB0,22,1         ; #29 Negative-end-limit-set bit
M2933->X:$000EB0,13,1         ; #29 Desired-velocity-zero bit
M2935->X:$000EB0,15,1         ; #29 Dwell-in-progress bit
M2937->X:$000EB0,17,1         ; #29 Running-program bit
M2938->X:$000EB0,18,1         ; #29 Open-loop-mode bit
M2939->X:$000EB0,19,1         ; #29 Amplifier-enabled status bit
M2940->Y:$000EC0,0,1          ; #29 Background in-position bit
M2941->Y:$000EC0,1,1          ; #29 Warning-following error bit
M2942->Y:$000EC0,2,1          ; #29 Fatal-following-error bit
M2943->Y:$000EC0,3,1          ; #29 Amplifier-fault-error bit
M2944->Y:$000EC0,13,1         ; #29 Foreground in-position bit
M2945->Y:$000EC0,10,1         ; #29 Home-complete bit
```

```
M2946->Y:$000EC0,6,1          ; #29 Integrated following error fault bit
M2947->Y:$000EC0,5,1          ; #29 I2T fault bit
M2948->Y:$000EC0,8,1          ; #29 Phasing error fault bit
M2949->Y:$000EC0,9,1          ; #29 Phasing search-in-progress bit
; Motor #29 Move Registers
M2961->D:$000E88              ; #29 Commanded position (1/[Ixx08*32] cts)
M2962->D:$000E8B              ; #29 Actual position (1/[Ixx08*32] cts)
M2963->D:$000EC7              ; #29 Target (end) position (1/[Ixx08*32] cts)
M2964->D:$000ECC              ; #29 Position bias (1/[Ixx08*32] cts)
M2966->X:$000E9D,0,24,S       ; #29 Actual velocity (1/[Ixx09*32] cts/cyc)
M2967->D:$000E8D              ; #29 Present master pos (1/[Ixx07*32] cts)
M2968->X:$000EBF,8,16,S       ; #29 Filter Output (16-bit DAC bits)
M2969->D:$000E90              ; #29 Compensation correction (1/[Ixx08*32] cts)
M2970->D:$000EB4              ; #29 Present phase position (including fraction)
M2971->X:$000EB4,24,S         ; #29 Present phase position (counts *Ixx70)
M2972->L:$000ED7              ; #29 Variable jog position/distance (cts)
M2973->Y:$000ECE,0,24,S       ; #29 Encoder home capture position (cts)
M2974->D:$000EEF              ; #29 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2975->X:$000EB9,8,16,S       ; #29 Actual quadrature current
M2976->Y:$000EB9,8,16,S       ; #29 Actual direct current
M2977->X:$000EBC,8,16,S       ; #29 Quadrature current-loop integrator output
M2978->Y:$000EBC,8,16,S       ; #29 Direct current-loop integrator output
M2979->X:$000EAE,8,16,S       ; #29 PID internal filter result (16-bit DAC bits)
; Motor #29 Axis Definition Registers
M2991->L:$000ECF              ; #29 X/U/A/B/C-Axis scale factor (cts/unit)
M2992->L:$000ED0              ; #29 Y/V-Axis scale factor (cts/unit)
M2993->L:$000ED1              ; #29 Z/W-Axis scale factor (cts/unit)
M2994->L:$000ED2              ; #29 Axis offset (cts)

; Servo IC 7 Registers for 3rd ACC-24 Channel 6 (usually for Motor #30)
M3001->X:$07A305,0,24,S       ; ENC6 24-bit counter position
M3002->Y:$07A302,8,16,S       ; DAC6 16-bit analog output
M3003->X:$07A307,0,24,S       ; ENC6 capture/compare position register
M3005->Y:$07A307,8,16,S       ; ADC6 16-bit analog input
M3006->Y:$07A304,0,24,U       ; ENC6 time between counts (SCLK cycles)
M3010->X:$07A304,10,1         ; ENC6 count-write enable control
M3011->X:$07A304,11,1         ; EQU6 compare flag latch control
M3012->X:$07A304,12,1         ; EQU6 compare output enable
M3013->X:$07A304,13,1         ; EQU6 compare invert enable
M3014->X:$07A304,14,1         ; AENA6/DIR6 Output
M3016->X:$07A304,16,1         ; EQU6 compare flag
M3017->X:$07A304,17,1         ; ENC6 position-captured flag
M3018->X:$07A304,18,1         ; ENC6 Count-error flag
M3019->X:$07A304,19,1         ; ENC6 3rd channel input status
M3020->X:$07A304,20,1         ; HMFL6 input status
M3021->X:$07A304,21,1         ; -LIM6 (positive end) input status
M3022->X:$07A304,22,1         ; +LIM6 (negative end) input status
M3023->X:$07A304,23,1         ; FAULT6 input status
```

```
; Motor #30 Status Bits
M3030->Y:$000F40,11,1        ; #30 Stopped-on-position-limit bit
M3031->X:$000F30,21,1        ; #30 Positive-end-limit-set bit
M3032->X:$000F30,22,1        ; #30 Negative-end-limit-set bit
M3033->X:$000F30,13,1        ; #30 Desired-velocity-zero bit
M3035->X:$000F30,15,1        ; #30 Dwell-in-progress bit
M3037->X:$000F30,17,1        ; #30 Running-program bit
M3038->X:$000F30,18,1        ; #30 Open-loop-mode bit
M3039->X:$000F30,19,1        ; #30 Amplifier-enabled status bit
M3040->Y:$000F40,0,1         ; #30 Background in-position bit
M3041->Y:$000F40,1,1         ; #30 Warning-following error bit
M3042->Y:$000F40,2,1         ; #30 Fatal-following-error bit
M3043->Y:$000F40,3,1         ; #30 Amplifier-fault-error bit
M3044->Y:$000F40,13,1        ; #30 Foreground in-position bit
M3045->Y:$000F40,10,1        ; #30 Home-complete bit
M3046->Y:$000F40,6,1         ; #30 Integrated following error fault bit
M3047->Y:$000F40,5,1         ; #30 I2T fault bit
M3048->Y:$000F40,8,1         ; #30 Phasing error fault bit
M3049->Y:$000F40,9,1         ; #30 Phasing search-in-progress bit
; Motor #30 Move Registers
M3061->D:$000F08             ; #30 Commanded position (1/[Ixx08*32] cts)
M3062->D:$000F0B             ; #30 Actual position (1/[Ixx08*32] cts)
M3063->D:$000F47             ; #30 Target (end) position (1/[Ixx08*32] cts)
M3064->D:$000F4C             ; #30 Position bias (1/[Ixx08*32] cts)
M3066->X:$000F1D,0,24,S      ; #30 Actual velocity (1/[Ixx09*32] cts/cyc)
M3067->D:$000F0D             ; #30 Present master pos (1/[Ixx07*32] cts)
M3068->X:$000F3F,8,16,S      ; #30 Filter Output (16-bit DAC bits)
M3069->D:$000F10             ; #30 Compensation correction (1/[Ixx08*32] cts)
M3070->D:$000F34             ; #30 Present phase position (including fraction)
M3071->X:$000F34,24,S        ; #30 Present phase position (counts *Ixx70)
M3072->L:$000F57             ; #30 Variable jog position/distance (cts)
M3073->Y:$000F4E,0,24,S      ; #30 Encoder home capture position (cts)
M3074->D:$000F6F             ; #30 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3075->X:$000F39,8,16,S      ; #30 Actual quadrature current
M3076->Y:$000F39,8,16,S      ; #30 Actual direct current
M3077->X:$000F3C,8,16,S      ; #30 Quadrature current-loop integrator output
M3078->Y:$000F3C,8,16,S      ; #30 Direct current-loop integrator output
M3079->X:$000F2E,8,16,S      ; #30 PID internal filter result (16-bit DAC bits)
; Motor #30 Axis Definition Registers
M3091->L:$000F4F             ; #30 X/U/A/B/C-Axis scale factor (cts/unit)
M3092->L:$000F50             ; #30 Y/V-Axis scale factor (cts/unit)
M3093->L:$000F51             ; #30 Z/W-Axis scale factor (cts/unit)
M3094->L:$000F52             ; #30 Axis offset (cts)
```

```
; Servo IC 7 Registers for 3rd ACC-24 Channel 7 (usually for Motor #31)
M3101->X:$07A309,0,24,S      ; ENC7 24-bit counter position
M3102->Y:$07A30B,8,16,S      ; DAC7 16-bit analog output
M3103->X:$07A30B,0,24,S      ; ENC7 capture/compare position register
M3105->Y:$07A30E,8,16,S      ; ADC7 16-bit analog input
M3106->Y:$07A308,0,24,U      ; ENC7 time between counts (SCLK cycles)
M3110->X:$07A308,10,1        ; ENC7 count-write enable control
M3111->X:$07A308,11,1        ; EQU7 compare flag latch control
M3112->X:$07A308,12,1        ; EQU7 compare output enable
M3113->X:$07A308,13,1        ; EQU7 compare invert enable
M3114->X:$07A308,14,1        ; AENA7/DIR7 Output
M3116->X:$07A308,16,1        ; EQU7 compare flag
M3117->X:$07A308,17,1        ; ENC7 position-captured flag
M3118->X:$07A308,18,1        ; ENC7 Count-error flag
M3119->X:$07A308,19,1        ; ENC7 3rd channel input status
M3120->X:$07A308,20,1        ; HMFL7 input status
M3121->X:$07A308,21,1        ; -LIM7 (positive end) input status
M3122->X:$07A308,22,1        ; +LIM7 (negative end) input status
M3123->X:$07A308,23,1        ; FAULT7 input status
; Motor #31 Status Bits
M3130->Y:$000FC0,11,1        ; #31 Stopped-on-position-limit bit
M3131->X:$000FB0,21,1        ; #31 Positive-end-limit-set bit
M3132->X:$000FB0,22,1        ; #31 Negative-end-limit-set bit
M3133->X:$000FB0,13,1        ; #31 Desired-velocity-zero bit
M3135->X:$000FB0,15,1        ; #31 Dwell-in-progress bit
M3137->X:$000FB0,17,1        ; #31 Running-program bit
M3138->X:$000FB0,18,1        ; #31 Open-loop-mode bit
M3139->X:$000FB0,19,1        ; #31 Amplifier-enabled status bit
M3140->Y:$000FC0,0,1         ; #31 Background in-position bit
M3141->Y:$000FC0,1,1         ; #31 Warning-following error bit
M3142->Y:$000FC0,2,1         ; #31 Fatal-following-error bit
M3143->Y:$000FC0,3,1         ; #31 Amplifier-fault-error bit
M3144->Y:$000FC0,13,1        ; #31 Foreground in-position bit
M3145->Y:$000FC0,10,1        ; #31 Home-complete bit
M3146->Y:$000FC0,6,1         ; #31 Integrated following error fault bit
M3147->Y:$000FC0,5,1         ; #31 I2T fault bit
M3148->Y:$000FC0,8,1         ; #31 Phasing error fault bit
M3149->Y:$000FC0,9,1         ; #31 Phasing search-in-progress bit
; Motor #31 Move Registers
M3161->D:$000F88             ; #31 Commanded position (1/[Ixx08*32] cts)
M3162->D:$000F8B             ; #31 Actual position (1/[Ixx08*32] cts)
M3163->D:$000FC7             ; #31 Target (end) position (1/[Ixx08*32] cts)
M3164->D:$000FCC             ; #31 Position bias (1/[Ixx08*32] cts)
M3166->X:$000F9D,0,24,S      ; #31 Actual velocity (1/[Ixx09*32] cts/cyc)
M3167->D:$000F8D             ; #31 Present master pos (1/[Ixx07*32] cts)
M3168->X:$000FBF,8,16,S      ; #31 Filter Output (16-bit DAC bits)
M3169->D:$000F90             ; #31 Compensation correction (1/[Ixx08*32] cts)
```

```
M3170->D:$000FB4              ; #31 Present phase position (including fraction)
M3171->X:$000FB4,24,S         ; #31 Present phase position (counts *Ixx70)
M3172->L:$000FD7              ; #31 Variable jog position/distance (cts)
M3173->Y:$000FCE,0,24,S       ; #31 Encoder home capture position (cts)
M3174->D:$000FEF              ; #31 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3175->X:$000FB9,8,16,S       ; #31 Actual quadrature current
M3176->Y:$000FB9,8,16,S       ; #31 Actual direct current
M3177->X:$000FBC,8,16,S       ; #31 Quadrature current-loop integrator output
M3178->Y:$000FBC,8,16,S       ; #31 Direct current-loop integrator output
M3179->X:$000FAE,8,16,S       ; #31 PID internal filter result (16-bit DAC bits)
; Motor #31 Axis Definition Registers
M3191->L:$000FCF              ; #31 X/U/A/B/C-Axis scale factor (cts/unit)
M3192->L:$000FD0              ; #31 Y/V-Axis scale factor (cts/unit)
M3193->L:$000FD1              ; #31 Z/W-Axis scale factor (cts/unit)
M3194->L:$000FD2              ; #31 Axis offset (cts)
; Servo IC 7 Registers for 3rd ACC-24 Channel 8 (usually for Motor #32)
M3201->X:$07A30D,0,24,S       ; ENC8 24-bit counter position
M3202->Y:$07A30A,8,16,S       ; DAC8 16-bit analog output
M3203->X:$07A30F,0,24,S       ; ENC8 capture/compare position register
M3205->Y:$07A30F,8,16,S       ; ADC8 16-bit analog input
M3206->Y:$07A30C,0,24,U       ; ENC8 time between counts (SCLK cycles)
M3210->X:$07A30C,10,1         ; ENC8 count-write enable control
M3211->X:$07A30C,11,1         ; EQU8 compare flag latch control
M3212->X:$07A30C,12,1         ; EQU8 compare output enable
M3213->X:$07A30C,13,1         ; EQU8 compare invert enable
M3214->X:$07A30C,14,1         ; AENA8/DIR8 Output
M3216->X:$07A30C,16,1         ; EQU8 compare flag
M3217->X:$07A30C,17,1         ; ENC8 position-captured flag
M3218->X:$07A30C,18,1         ; ENC8 Count-error flag
M3219->X:$07A30C,19,1         ; ENC8 3rd channel input status
M3220->X:$07A30C,20,1         ; HMFL8 input status
M3221->X:$07A30C,21,1         ; -LIM8 (positive end) input status
M3222->X:$07A30C,22,1         ; +LIM8 (negative end) input status
M3223->X:$07A30C,23,1         ; FAULT8 input status
; Motor #32 Status Bits
M3230->Y:$001040,11,1         ; #32 Stopped-on-position-limit bit
M3231->X:$001030,21,1         ; #32 Positive-end-limit-set bit
M3232->X:$001030,22,1         ; #32 Negative-end-limit-set bit
M3233->X:$001030,13,1         ; #32 Desired-velocity-zero bit
M3235->X:$001030,15,1         ; #32 Dwell-in-progress bit
M3237->X:$001030,17,1         ; #32 Running-program bit
M3238->X:$001030,18,1         ; #32 Open-loop-mode bit
M3239->X:$001030,19,1         ; #32 Amplifier-enabled status bit
M3240->Y:$001040,0,1          ; #32 Background in-position bit
M3241->Y:$001040,1,1          ; #32 Warning-following error bit
M3242->Y:$001040,2,1          ; #32 Fatal-following-error bit
M3243->Y:$001040,3,1          ; #32 Amplifier-fault-error bit
```

```
M3244->Y:$001040,13,1        ; #32 Foreground in-position bit
M3245->Y:$001040,10,1        ; #32 Home-complete bit
M3246->Y:$001040,6,1         ; #32 Integrated following error fault bit
M3247->Y:$001040,5,1         ; #32 I2T fault bit
M3248->Y:$001040,8,1         ; #32 Phasing error fault bit
M3249->Y:$001040,9,1         ; #32 Phasing search-in-progress bit
; Motor #32 Move Registers
M3261->D:$001008             ; #32 Commanded position (1/[Ixx08*32] cts)
M3262->D:$00100B             ; #32 Actual position (1/[Ixx08*32] cts)
M3263->D:$001047             ; #32 Target (end) position (1/[Ixx08*32] cts)
M3264->D:$00104C             ; #32 Position bias (1/[Ixx08*32] cts)
M3266->X:$00101D,0,24,S      ; #32 Actual velocity (1/[Ixx09*32] cts/cyc)
M3267->D:$00100D             ; #32 Present master pos (1/[Ixx07*32] cts)
M3268->X:$00103F,8,16,S      ; #32 Filter Output (16-bit DAC bits)
M3269->D:$001010             ; #32 Compensation correction (1/[Ixx08*32] cts)
M3270->D:$001034             ; #32 Present phase position (including fraction)
M3271->X:$001034,24,S        ; #32 Present phase position (counts *Ixx70)
M3272->L:$001057             ; #32 Variable jog position/distance (cts)
M3273->Y:$00104E,0,24,S      ; #32 Encoder home capture position (cts)
M3274->D:$00106F             ; #32 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3275->X:$001039,8,16,S      ; #32 Actual quadrature current
M3276->Y:$001039,8,16,S      ; #32 Actual direct current
M3277->X:$00103C,8,16,S      ; #32 Quadrature current-loop integrator output
M3278->Y:$00103C,8,16,S      ; #32 Direct current-loop integrator output
M3279->X:$00102E,8,16,S      ; #32 PID internal filter result (16-bit DAC bits)
; Motor #32 Axis Definition Registers
M3291->L:$00104F             ; #32 X/U/A/B/C-Axis scale factor (cts/unit)
M3292->L:$001050             ; #32 Y/V-Axis scale factor (cts/unit)
M3293->L:$001051             ; #32 Z/W-Axis scale factor (cts/unit)
M3294->L:$001052             ; #32 Axis offset (cts)
; De-multiplexed ADC values from Opt. 12, ACC-36
M5061->Y:$003400,12,12,U     ; Demuxed low ADC register from I5061
M5062->Y:$003402,12,12,U     ; Demuxed low ADC register from I5062
M5063->Y:$003404,12,12,U     ; Demuxed low ADC register from I5063
M5064->Y:$003406,12,12,U     ; Demuxed low ADC register from I5064
M5065->Y:$003408,12,12,U     ; Demuxed low ADC register from I5065
M5066->Y:$00340A,12,12,U     ; Demuxed low ADC register from I5066
M5067->Y:$00340C,12,12,U     ; Demuxed low ADC register from I5067
M5068->Y:$00340E,12,12,U     ; Demuxed low ADC register from I5068
M5069->Y:$003410,12,12,U     ; Demuxed low ADC register from I5069
M5070->Y:$003412,12,12,U     ; Demuxed low ADC register from I5070
M5071->Y:$003414,12,12,U     ; Demuxed low ADC register from I5071
M5072->Y:$003416,12,12,U     ; Demuxed low ADC register from I5072
M5073->Y:$003418,12,12,U     ; Demuxed low ADC register from I5073
M5074->Y:$00341A,12,12,U     ; Demuxed low ADC register from I5074
M5075->Y:$00341C,12,12,U     ; Demuxed low ADC register from I5075
M5076->Y:$00341E,12,12,U     ; Demuxed low ADC register from I5076
```

```
M5081->Y:$003401,12,12,U     ; Demuxed high ADC register from I5061
M5082->Y:$003403,12,12,U     ; Demuxed high ADC register from I5062
M5083->Y:$003405,12,12,U     ; Demuxed high ADC register from I5063
M5084->Y:$003407,12,12,U     ; Demuxed high ADC register from I5064
M5085->Y:$003409,12,12,U     ; Demuxed high ADC register from I5065
M5086->Y:$00340B,12,12,U     ; Demuxed high ADC register from I5066
M5087->Y:$00340D,12,12,U     ; Demuxed high ADC register from I5067
M5088->Y:$00340F,12,12,U     ; Demuxed high ADC register from I5068
M5089->Y:$003411,12,12,U     ; Demuxed high ADC register from I5069
M5090->Y:$003413,12,12,U     ; Demuxed high ADC register from I5070
M5091->Y:$003415,12,12,U     ; Demuxed high ADC register from I5071
M5092->Y:$003417,12,12,U     ; Demuxed high ADC register from I5072
M5093->Y:$003419,12,12,U     ; Demuxed high ADC register from I5073
M5094->Y:$00341B,12,12,U     ; Demuxed high ADC register from I5074
M5095->Y:$00341D,12,12,U     ; Demuxed high ADC register from I5075
M5096->Y:$00341F,12,12,U     ; Demuxed high ADC register from I5076
; Coordinate System 1 (&1) timers
M5111->X:$002015,0,24,S      ; &1 Isx11 timer (for synchronous assignment)
M5112->Y:$002015,0,24,S      ; &1 Isx12 timer (for synchronous assignment)
; Coordinate System 1 (&1) end-of-calculated-move positions
M5141->L:$002041             ; &1 A-axis target position (engineering units)
M5142->L:$002042             ; &1 B-axis target position (engineering units)
M5143->L:$002043             ; &1 C-axis target position (engineering units)
M5144->L:$002044             ; &1 U-axis target position (engineering units)
M5145->L:$002045             ; &1 V-axis target position (engineering units)
M5146->L:$002046             ; &1 W-axis target position (engineering units)
M5147->L:$002047             ; &1 X-axis target position (engineering units)
M5148->L:$002048             ; &1 Y-axis target position (engineering units)
M5149->L:$002049             ; &1 Z-axis target position (engineering units)
; Coordinate System 1 (&1) Status Bits
M5180->X:$002040,0,1         ; &1 Program-running bit
M5181->Y:$00203F,21,1        ; &1 Circle-radius-error bit
M5182->Y:$00203F,22,1        ; &1 Run-time-error bit
M5184->X:$002040,0,4         ; &1 Continuous motion request
M5187->Y:$00203F,17,1        ; &1 In-position bit (AND of motors)
M5188->Y:$00203F,18,1        ; &1 Warning-following-error bit (OR)
M5189->Y:$00203F,19,1        ; &1 Fatal-following-error bit (OR)
M5190->Y:$00203F,20,1        ; &1 Amp-fault-error bit (OR of motors)
; Coordinate System 1 (&1) Variables
M5197->X:$002000,0,24,S      ; &1 Host commanded time base (I10 units)
M5198->X:$002002,0,24,S      ; &1 Present time base (I10 units)
; Coordinate System 2 (&2) timers
M5211->X:$002115,0,24,S      ; &2 Isx11 timer (for synchronous assignment)
M5212->Y:$002115,0,24,S      ; &2 Isx12 timer (for synchronous assignment)
```

```
; Coordinate System 2 (&2) end-of-calculated-move positions
M5241->L:$002141              ; &2 A-axis target position (engineering units)
M5242->L:$002142              ; &2 B-axis target position (engineering units)
M5243->L:$002143              ; &2 C-axis target position (engineering units)
M5244->L:$002144              ; &2 U-axis target position (engineering units)
M5245->L:$002145              ; &2 V-axis target position (engineering units)
M5246->L:$002146              ; &2 W-axis target position (engineering units)
M5247->L:$002147              ; &2 X-axis target position (engineering units)
M5248->L:$002148              ; &2 Y-axis target position (engineering units)
M5249->L:$002149              ; &2 Z-axis target position (engineering units)
; Coordinate System 2 (&2) Status Bits
M5280->X:$002140,0,1          ; &2 Program-running bit
M5281->Y:$00213F,21,1         ; &2 Circle-radius-error bit
M5282->Y:$00213F,22,1         ; &2 Run-time-error bit
M5284->X:$002140,0,4          ; &2 Continuous motion request
M5287->Y:$00213F,17,1         ; &2 In-position bit (AND of motors)
M5288->Y:$00213F,18,1         ; &2 Warning-following-error bit (OR)
M5289->Y:$00213F,19,1         ; &2 Fatal-following-error bit (OR)
M5290->Y:$00213F,20,1         ; &2 Amp-fault-error bit (OR of motors)
; Coordinate System 2 (&2) Variables
M5297->X:$002100,0,24,S       ; &2 Host commanded time base (I10 units)
M5298->X:$002102,0,24,S       ; &2 Present time base (I10 units)
; Coordinate System 3 (&3) timers
M5311->X:$002215,0,24,S       ; &3 Isx11 timer (for synchronous assignment)
M5312->Y:$002215,0,24,S       ; &3 Isx12 timer (for synchronous assignment)
; Coordinate System 3 (&3) end-of-calculated-move positions
M5341->L:$002241              ; &3 A-axis target position (engineering units)
M5342->L:$002242              ; &3 B-axis target position (engineering units)
M5343->L:$002243              ; &3 C-axis target position (engineering units)
M5344->L:$002244              ; &3 U-axis target position (engineering units)
M5345->L:$002245              ; &3 V-axis target position (engineering units)
M5346->L:$002246              ; &3 W-axis target position (engineering units)
M5347->L:$002247              ; &3 X-axis target position (engineering units)
M5348->L:$002248              ; &3 Y-axis target position (engineering units)
M5349->L:$002249              ; &3 Z-axis target position (engineering units)
; Coordinate System 3 (&3) Status Bits
M5380->X:$002240,0,1          ; &3 Program-running bit
M5381->Y:$00223F,21,1         ; &3 Circle-radius-error bit
M5382->Y:$00223F,22,1         ; &3 Run-time-error bit
M5384->X:$002240,0,4          ; &3 Continuous motion request
M5387->Y:$00223F,17,1         ; &3 In-position bit (AND of motors)
M5388->Y:$00223F,18,1         ; &3 Warning-following-error bit (OR)
M5389->Y:$00223F,19,1         ; &3 Fatal-following-error bit (OR)
M5390->Y:$00223F,20,1         ; &3 Amp-fault-error bit (OR of motors)
```

```
; Coordinate System 3 (&3) Variables
M5397->X:$002200,0,24,S      ; &3 Host commanded time base (I10 units)
M5398->X:$002202,0,24,S      ; &3 Present time base (I10 units)
; Coordinate System 4 (&4) timers
M5411->X:$002315,0,24,S      ; &4 Isx11 timer (for synchronous assignment)
M5412->Y:$002315,0,24,S      ; &4 Isx12 timer (for synchronous assignment)
; Coordinate System 4 (&4) end-of-calculated-move positions
M5441->L:$002341             ; &4 A-axis target position (engineering units)
M5442->L:$002342             ; &4 B-axis target position (engineering units)
M5443->L:$002343             ; &4 C-axis target position (engineering units)
M5444->L:$002344             ; &4 U-axis target position (engineering units)
M5445->L:$002345             ; &4 V-axis target position (engineering units)
M5446->L:$002346             ; &4 W-axis target position (engineering units)
M5447->L:$002347             ; &4 X-axis target position (engineering units)
M5448->L:$002348             ; &4 Y-axis target position (engineering units)
M5449->L:$002349             ; &4 Z-axis target position (engineering units)
; Coordinate System 4 (&4) Status Bits
M5480->X:$002340,0,1         ; &4 Program-running bit
M5481->Y:$00233F,21,1        ; &4 Circle-radius-error bit
M5482->Y:$00233F,22,1        ; &4 Run-time-error bit
M5484->X:$002340,0,4         ; &4 Continuous motion request
M5487->Y:$00233F,17,1        ; &4 In-position bit (AND of motors)
M5488->Y:$00233F,18,1        ; &4 Warning-following-error bit (OR)
M5489->Y:$00233F,19,1        ; &4 Fatal-following-error bit (OR)
M5490->Y:$00233F,20,1        ; &4 Amp-fault-error bit (OR of motors)
; Coordinate System 4 (&4) Variables
M5497->X:$002300,0,24,S      ; &4 Host commanded time base (I10 units)
M5498->X:$002302,0,24,S      ; &4 Present time base (I10 units)
; Coordinate System 5 (&5) timers
M5511->X:$002415,0,24,S      ; &5 Isx11 timer (for synchronous assignment)
M5512->Y:$002415,0,24,S      ; &5 Isx12 timer (for synchronous assignment)
; Coordinate System 5 (&5) end-of-calculated-move positions
M5541->L:$002441             ; &5 A-axis target position (engineering units)
M5542->L:$002442             ; &5 B-axis target position (engineering units)
M5543->L:$002443             ; &5 C-axis target position (engineering units)
M5544->L:$002444             ; &5 U-axis target position (engineering units)
M5545->L:$002445             ; &5 V-axis target position (engineering units)
M5546->L:$002446             ; &5 W-axis target position (engineering units)
M5547->L:$002447             ; &5 X-axis target position (engineering units)
M5548->L:$002448             ; &5 Y-axis target position (engineering units)
M5549->L:$002449             ; &5 Z-axis target position (engineering units)
; Coordinate System 5 (&5) Status Bits
M5580->X:$002440,0,1         ; &5 Program-running bit
M5581->Y:$00243F,21,1        ; &5 Circle-radius-error bit
M5582->Y:$00243F,22,1        ; &5 Run-time-error bit
M5584->X:$002440,0,4         ; &5 Continuous motion request
```

```
M5587->Y:$00243F,17,1          ; &5 In-position bit (AND of motors)
M5588->Y:$00243F,18,1          ; &5 Warning-following-error bit (OR)
M5589->Y:$00243F,19,1          ; &5 Fatal-following-error bit (OR)
M5590->Y:$00243F,20,1          ; &5 Amp-fault-error bit (OR of motors)
; Coordinate System 5 (&5) Variables
M5597->X:$002400,0,24,S        ; &5 Host commanded time base (I10 units)
M5598->X:$002402,0,24,S        ; &5 Present time base (I10 units)
; Coordinate System 6 (&6) timers
M5611->X:$002515,0,24,S        ; &6 Isx11 timer (for synchronous assignment)
M5612->Y:$002515,0,24,S        ; &6 Isx12 timer (for synchronous assignment)
; Coordinate System 6 (&6) end-of-calculated-move positions
M5641->L:$002541               ; &6 A-axis target position (engineering units)
M5642->L:$002542               ; &6 B-axis target position (engineering units)
M5643->L:$002543               ; &6 C-axis target position (engineering units)
M5644->L:$002544               ; &6 U-axis target position (engineering units)
M5645->L:$002545               ; &6 V-axis target position (engineering units)
M5646->L:$002546               ; &6 W-axis target position (engineering units)
M5647->L:$002547               ; &6 X-axis target position (engineering units)
M5648->L:$002548               ; &6 Y-axis target position (engineering units)
M5649->L:$002549               ; &6 Z-axis target position (engineering units)
; Coordinate System 6 (&6) Status Bits
M5680->X:$002540,0,1           ; &6 Program-running bit
M5681->Y:$00253F,21,1          ; &6 Circle-radius-error bit
M5682->Y:$00253F,22,1          ; &6 Run-time-error bit
M5684->X:$002540,0,4           ; &6 Continuous motion request
M5687->Y:$00253F,17,1          ; &6 In-position bit (AND of motors)
M5688->Y:$00253F,18,1          ; &6 Warning-following-error bit (OR)
M5689->Y:$00253F,19,1          ; &6 Fatal-following-error bit (OR)
M5690->Y:$00253F,20,1          ; &6 Amp-fault-error bit (OR of motors)
; Coordinate System 6 (&6) Variables
M5697->X:$002500,0,24,S        ; &6 Host commanded time base (I10 units)
M5698->X:$002502,0,24,S        ; &6 Present time base (I10 units)
; Coordinate System 7 (&7) timers
M5711->X:$002615,0,24,S        ; &7 Isx11 timer (for synchronous assignment)
M5712->Y:$002615,0,24,S        ; &7 Isx12 timer (for synchronous assignment)
; Coordinate System 7 (&7) end-of-calculated-move positions
M5741->L:$002641               ; &7 A-axis target position (engineering units)
M5742->L:$002642               ; &7 B-axis target position (engineering units)
M5743->L:$002643               ; &7 C-axis target position (engineering units)
M5744->L:$002644               ; &7 U-axis target position (engineering units)
M5745->L:$002645               ; &7 V-axis target position (engineering units)
M5746->L:$002646               ; &7 W-axis target position (engineering units)
M5747->L:$002647               ; &7 X-axis target position (engineering units)
M5748->L:$002648               ; &7 Y-axis target position (engineering units)
M5749->L:$002649               ; &7 Z-axis target position (engineering units)
```

```
; Coordinate System 7 (&7) Status Bits
M5780->X:$002640,0,1          ; &7 Program-running bit
M5781->Y:$00263F,21,1         ; &7 Circle-radius-error bit
M5782->Y:$00263F,22,1         ; &7 Run-time-error bit
M5784->X:$002640,0,4          ; &7 Continuous motion request
M5787->Y:$00263F,17,1         ; &7 In-position bit (AND of motors)
M5788->Y:$00263F,18,1         ; &7 Warning-following-error bit (OR)
M5789->Y:$00263F,19,1         ; &7 Fatal-following-error bit (OR)
M5790->Y:$00263F,20,1         ; &7 Amp-fault-error bit (OR of motors)
; Coordinate System 7 (&7) Variables
M5797->X:$002600,0,24,S       ; &7 Host commanded time base (I10 units)
M5798->X:$002602,0,24,S       ; &7 Present time base (I10 units)
; Coordinate System 8 (&8) timers
M5811->X:$002715,0,24,S       ; &8 Isx11 timer (for synchronous assignment)
M5812->Y:$002715,0,24,S       ; &8 Isx12 timer (for synchronous assignment)
; Coordinate System 8 (&8) end-of-calculated-move positions
M5841->L:$002741              ; &8 A-axis target position (engineering units)
M5842->L:$002742              ; &8 B-axis target position (engineering units)
M5843->L:$002743              ; &8 C-axis target position (engineering units)
M5844->L:$002744              ; &8 U-axis target position (engineering units)
M5845->L:$002745              ; &8 V-axis target position (engineering units)
M5846->L:$002746              ; &8 W-axis target position (engineering units)
M5847->L:$002747              ; &8 X-axis target position (engineering units)
M5848->L:$002748              ; &8 Y-axis target position (engineering units)
M5849->L:$002749              ; &8 Z-axis target position (engineering units)
; Coordinate System 8 (&8) Status Bits
M5880->X:$002740,0,1          ; &8 Program-running bit
M5881->Y:$00273F,21,1         ; &8 Circle-radius-error bit
M5882->Y:$00273F,22,1         ; &8 Run-time-error bit
M5884->X:$002740,0,4          ; &8 Continuous motion request
M5887->Y:$00273F,17,1         ; &8 In-position bit (AND of motors)
M5888->Y:$00273F,18,1         ; &8 Warning-following-error bit (OR)
M5889->Y:$00273F,19,1         ; &8 Fatal-following-error bit (OR)
M5890->Y:$00273F,20,1         ; &8 Amp-fault-error bit (OR of motors)
; Coordinate System 8 (&8) Variables
M5897->X:$002700,0,24,S       ; &8 Host commanded time base (I10 units)
M5898->X:$002702,0,24,S       ; &8 Present time base (I10 units)
; Coordinate System 9 (&9) timers
M5911->X:$002815,0,24,S       ; &9 Isx11 timer (for synchronous assignment)
M5912->Y:$002815,0,24,S       ; &9 Isx12 timer (for synchronous assignment)
; Coordinate System 9 (&9) end-of-calculated-move positions
M5941->L:$002841              ; &9 A-axis target position (engineering units)
M5942->L:$002842              ; &9 B-axis target position (engineering units)
M5943->L:$002843              ; &9 C-axis target position (engineering units)
M5944->L:$002844              ; &9 U-axis target position (engineering units)
M5945->L:$002845              ; &9 V-axis target position (engineering units)
```

```
M5946->L:$002846              ; &9 W-axis target position (engineering units)
M5947->L:$002847              ; &9 X-axis target position (engineering units)
M5948->L:$002848              ; &9 Y-axis target position (engineering units)
M5949->L:$002849              ; &9 Z-axis target position (engineering units)
; Coordinate System 1 (&1) Status Bits
M5980->X:$002840,0,1          ; &9 Program-running bit
M5981->Y:$00283F,21,1         ; &9 Circle-radius-error bit
M5982->Y:$00283F,22,1         ; &9 Run-time-error bit
M5984->X:$002840,0,4          ; &9 Continuous motion request
M5987->Y:$00283F,17,1         ; &9 In-position bit (AND of motors)
M5988->Y:$00283F,18,1         ; &9 Warning-following-error bit (OR)
M5989->Y:$00283F,19,1         ; &9 Fatal-following-error bit (OR)
M5990->Y:$00283F,20,1         ; &9 Amp-fault-error bit (OR of motors)
; Coordinate System 1 (&1) Variables
M5997->X:$002800,0,24,S       ; &9 Host commanded time base (I10 units)
M5998->X:$002802,0,24,S       ; &9 Present time base (I10 units)
; Coordinate System 10 (&10) timers
M6011->X:$002915,0,24,S       ; &10 Isx11 timer (for synchronous assignment)
M6012->Y:$002915,0,24,S       ; &10 Isx12 timer (for synchronous assignment)
; Coordinate System 10 (&10) end-of-calculated-move positions
M6041->L:$002941              ; &10 A-axis target position (engineering units)
M6042->L:$002942              ; &10 B-axis target position (engineering units)
M6043->L:$002943              ; &10 C-axis target position (engineering units)
M6044->L:$002944              ; &10 U-axis target position (engineering units)
M6045->L:$002945              ; &10 V-axis target position (engineering units)
M6046->L:$002946              ; &10 W-axis target position (engineering units)
M6047->L:$002947              ; &10 X-axis target position (engineering units)
M6048->L:$002948              ; &10 Y-axis target position (engineering units)
M6049->L:$002949              ; &10 Z-axis target position (engineering units)
; Coordinate System 10 (&10) Status Bits
M6080->X:$002940,0,1          ; &10 Program-running bit
M6081->Y:$00293F,21,1         ; &10 Circle-radius-error bit
M6082->Y:$00293F,22,1         ; &10 Run-time-error bit
M6084->X:$002940,0,4          ; &10 Continuous motion request
M6087->Y:$00293F,17,1         ; &10 In-position bit (AND of motors)
M6088->Y:$00293F,18,1         ; &10 Warning-following-error bit (OR)
M6089->Y:$00293F,19,1         ; &10 Fatal-following-error bit (OR)
M6090->Y:$00293F,20,1         ; &10 Amp-fault-error bit (OR of motors)
; Coordinate System 10 (&10) Variables
M6097->X:$002900,0,24,S       ; &10 Host commanded time base (I10 units)
M6098->X:$002902,0,24,S       ; &10 Present time base (I10 units)
; Coordinate System 11 (&11) timers
M6111->X:$002A15,0,24,S       ; &11 Isx11 timer (for synchronous assignment)
M6112->Y:$002A15,0,24,S       ; &11 Isx12 timer (for synchronous assignment)
```

```
; Coordinate System 11 (&11) end-of-calculated-move positions
M6141->L:$002A41              ; &11 A-axis target position (engineering units)
M6142->L:$002A42              ; &11 B-axis target position (engineering units)
M6143->L:$002A43              ; &11 C-axis target position (engineering units)
M6144->L:$002A44              ; &11 U-axis target position (engineering units)
M6145->L:$002A45              ; &11 V-axis target position (engineering units)
M6146->L:$002A46              ; &11 W-axis target position (engineering units)
M6147->L:$002A47              ; &11 X-axis target position (engineering units)
M6148->L:$002A48              ; &11 Y-axis target position (engineering units)
M6149->L:$002A49              ; &11 Z-axis target position (engineering units)
; Coordinate System 11 (&11) Status Bits
M6180->X:$002A40,0,1          ; &11 Program-running bit
M6181->Y:$002A3F,21,1         ; &11 Circle-radius-error bit
M6182->Y:$002A3F,22,1         ; &11 Run-time-error bit
M6184->X:$002A40,0,4          ; &11 Continuous motion request
M6187->Y:$002A3F,17,1         ; &11 In-position bit (AND of motors)
M6188->Y:$002A3F,18,1         ; &11 Warning-following-error bit (OR)
M6189->Y:$002A3F,19,1         ; &11 Fatal-following-error bit (OR)
M6190->Y:$002A3F,20,1         ; &11 Amp-fault-error bit (OR of motors)
; Coordinate System 11 (&11) Variables
M6197->X:$002A00,0,24,S       ; &11 Host commanded time base (I10 units)
M6198->X:$002A02,0,24,S       ; &11 Present time base (I10 units)
; Coordinate System 12 (&12) timers
M6211->X:$002B15,0,24,S       ; &12 Isx11 timer (for synchronous assignment)
M6212->Y:$002B15,0,24,S       ; &12 Isx12 timer (for synchronous assignment)
; Coordinate System 12 (&12) end-of-calculated-move positions
M6241->L:$002B41              ; &12 A-axis target position (engineering units)
M6242->L:$002B42              ; &12 B-axis target position (engineering units)
M6243->L:$002B43              ; &12 C-axis target position (engineering units)
M6244->L:$002B44              ; &12 U-axis target position (engineering units)
M6245->L:$002B45              ; &12 V-axis target position (engineering units)
M6246->L:$002B46              ; &12 W-axis target position (engineering units)
M6247->L:$002B47              ; &12 X-axis target position (engineering units)
M6248->L:$002B48              ; &12 Y-axis target position (engineering units)
M6249->L:$002B49              ; &12 Z-axis target position (engineering units)
; Coordinate System 12 (&12) Status Bits
M6280->X:$002B40,0,1          ; &12 Program-running bit
M6281->Y:$002B3F,21,1         ; &12 Circle-radius-error bit
M6282->Y:$002B3F,22,1         ; &12 Run-time-error bit
M6284->X:$002B40,0,4          ; &12 Continuous motion request
M6287->Y:$002B3F,17,1         ; &12 In-position bit (AND of motors)
M6288->Y:$002B3F,18,1         ; &12 Warning-following-error bit (OR)
M6289->Y:$002B3F,19,1         ; &12 Fatal-following-error bit (OR)
M6290->Y:$002B3F,20,1         ; &12 Amp-fault-error bit (OR of motors)
```

```
; Coordinate System 12 (&12) Variables
M6297->X:$002B00,0,24,S      ; &12 Host commanded time base (I10 units)
M6298->X:$002B02,0,24,S      ; &12 Present time base (I10 units)
; Coordinate System 13 (&13) timers
M6311->X:$002C15,0,24,S      ; &13 Isx11 timer (for synchronous assignment)
M6312->Y:$002C15,0,24,S      ; &13 Isx12 timer (for synchronous assignment)
; Coordinate System 13 (&13) end-of-calculated-move positions
M6341->L:$002C41             ; &13 A-axis target position (engineering units)
M6342->L:$002C42             ; &13 B-axis target position (engineering units)
M6343->L:$002C43             ; &13 C-axis target position (engineering units)
M6344->L:$002C44             ; &13 U-axis target position (engineering units)
M6345->L:$002C45             ; &13 V-axis target position (engineering units)
M6346->L:$002C46             ; &13 W-axis target position (engineering units)
M6347->L:$002C47             ; &13 X-axis target position (engineering units)
M6348->L:$002C48             ; &13 Y-axis target position (engineering units)
M6349->L:$002C49             ; &13 Z-axis target position (engineering units)
; Coordinate System 13 (&13) Status Bits
M6380->X:$002C40,0,1         ; &13 Program-running bit
M6381->Y:$002C3F,21,1        ; &13 Circle-radius-error bit
M6382->Y:$002C3F,22,1        ; &13 Run-time-error bit
M6384->X:$002C40,0,4         ; &13 Continuous motion request
M6387->Y:$002C3F,17,1        ; &13 In-position bit (AND of motors)
M6388->Y:$002C3F,18,1        ; &13 Warning-following-error bit (OR)
M6389->Y:$002C3F,19,1        ; &13 Fatal-following-error bit (OR)
M6390->Y:$002C3F,20,1        ; &13 Amp-fault-error bit (OR of motors)
; Coordinate System 13 (&13) Variables
M6397->X:$002C00,0,24,S      ; &13 Host commanded time base (I10 units)
M6398->X:$002C02,0,24,S      ; &13 Present time base (I10 units)
; Coordinate System 14 (&14) timers
M6411->X:$002D15,0,24,S      ; &14 Isx11 timer (for synchronous assignment)
M6412->Y:$002D15,0,24,S      ; &14 Isx12 timer (for synchronous assignment)
; Coordinate System 14 (&14) end-of-calculated-move positions
M6441->L:$002D41             ; &14 A-axis target position (engineering units)
M6442->L:$002D42             ; &14 B-axis target position (engineering units)
M6443->L:$002D43             ; &14 C-axis target position (engineering units)
M6444->L:$002D44             ; &14 U-axis target position (engineering units)
M6445->L:$002D45             ; &14 V-axis target position (engineering units)
M6446->L:$002D46             ; &14 W-axis target position (engineering units)
M6447->L:$002D47             ; &14 X-axis target position (engineering units)
M6448->L:$002D48             ; &14 Y-axis target position (engineering units)
M6449->L:$002D49             ; &14 Z-axis target position (engineering units)
; Coordinate System 14 (&14) Status Bits
M6480->X:$002D40,0,1         ; &14 Program-running bit
M6481->Y:$002D3F,21,1        ; &14 Circle-radius-error bit
M6482->Y:$002D3F,22,1        ; &14 Run-time-error bit
M6484->X:$002D40,0,4         ; &14 Continuous motion request
```

```
M6487->Y:$002D3F,17,1          ; &14 In-position bit (AND of motors)
M6488->Y:$002D3F,18,1          ; &14 Warning-following-error bit (OR)
M6489->Y:$002D3F,19,1          ; &14 Fatal-following-error bit (OR)
M6490->Y:$002D3F,20,1          ; &14 Amp-fault-error bit (OR of motors)
; Coordinate System 14 (&14) Variables
M6497->X:$002D00,0,24,S        ; &14 Host commanded time base (I10 units)
M6498->X:$002D02,0,24,S        ; &14 Present time base (I10 units)
; Coordinate System 15 (&15) timers
M6511->X:$002E15,0,24,S        ; &15 Isx11 timer (for synchronous assignment)
M6512->Y:$002E15,0,24,S        ; &15 Isx12 timer (for synchronous assignment)
; Coordinate System 15 (&15) end-of-calculated-move positions
M6541->L:$002E41               ; &15 A-axis target position (engineering units)
M6542->L:$002E42               ; &15 B-axis target position (engineering units)
M6543->L:$002E43               ; &15 C-axis target position (engineering units)
M6544->L:$002E44               ; &15 U-axis target position (engineering units)
M6545->L:$002E45               ; &15 V-axis target position (engineering units)
M6546->L:$002E46               ; &15 W-axis target position (engineering units)
M6547->L:$002E47               ; &15 X-axis target position (engineering units)
M6548->L:$002E48               ; &15 Y-axis target position (engineering units)
M6549->L:$002E49               ; &15 Z-axis target position (engineering units)
; Coordinate System 15 (&15) Status Bits
M6580->X:$002E40,0,1           ; &15 Program-running bit
M6581->Y:$002E3F,21,1          ; &15 Circle-radius-error bit
M6582->Y:$002E3F,22,1          ; &15 Run-time-error bit
M6584->X:$002E40,0,4           ; &15 Continuous motion request
M6587->Y:$002E3F,17,1          ; &15 In-position bit (AND of motors)
M6588->Y:$002E3F,18,1          ; &15 Warning-following-error bit (OR)
M6589->Y:$002E3F,19,1          ; &15 Fatal-following-error bit (OR)
M6590->Y:$002E3F,20,1          ; &15 Amp-fault-error bit (OR of motors)
; Coordinate System 15 (&15) Variables
M6597->X:$002E00,0,24,S        ; &15 Host commanded time base (I10 units)
M6598->X:$002E02,0,24,S        ; &15 Present time base (I10 units)
; Coordinate System 16 (&16) timers
M6611->X:$002F15,0,24,S        ; &16 Isx11 timer (for synchronous assignment)
M6612->Y:$002F15,0,24,S        ; &16 Isx12 timer (for synchronous assignment)
; Coordinate System 16 (&16) end-of-calculated-move positions
M6641->L:$002F41               ; &16 A-axis target position (engineering units)
M6642->L:$002F42               ; &16 B-axis target position (engineering units)
M6643->L:$002F43               ; &16 C-axis target position (engineering units)
M6644->L:$002F44               ; &16 U-axis target position (engineering units)
M6645->L:$002F45               ; &16 V-axis target position (engineering units)
M6646->L:$002F46               ; &16 W-axis target position (engineering units)
M6647->L:$002F47               ; &16 X-axis target position (engineering units)
M6648->L:$002F48               ; &16 Y-axis target position (engineering units)
M6649->L:$002F49               ; &16 Z-axis target position (engineering units)
```

```
; Coordinate System 16 (&16) Status Bits
M6680->X:$002F40,0,1        ; &16 Program-running bit
M6681->Y:$002F3F,21,1       ; &16 Circle-radius-error bit
M6682->Y:$002F3F,22,1       ; &16 Run-time-error bit
M6684->X:$002F40,0,4        ; &16 Continuous motion request
M6687->Y:$002F3F,17,1       ; &16 In-position bit (AND of motors)
M6688->Y:$002F3F,18,1       ; &16 Warning-following-error bit (OR)
M6689->Y:$002F3F,19,1       ; &16 Fatal-following-error bit (OR)
M6690->Y:$002F3F,20,1       ; &16 Amp-fault-error bit (OR of motors)
; Coordinate System 16 (&16) Variables
M6697->X:$002F00,0,24,S     ; &16 Host commanded time base (I10 units)
M6698->X:$002F02,0,24,S     ; &16 Present time base (I10 units)
; Accessory 14 I/O M-Variables (1st ACC-14)
M7000->Y:$078A00,0,1        ; MI/O0
M7001->Y:$078A00,1,1        ; MI/O1
M7002->Y:$078A00,2,1        ; MI/O2
M7003->Y:$078A00,3,1        ; MI/O3
M7004->Y:$078A00,4,1        ; MI/O4
M7005->Y:$078A00,5,1        ; MI/O5
M7006->Y:$078A00,6,1        ; MI/O6
M7007->Y:$078A00,7,1        ; MI/O7
M7008->Y:$078A00,8,1        ; MI/O8
M7009->Y:$078A00,9,1        ; MI/O9
M7010->Y:$078A00,10,1       ; MI/O10
M7011->Y:$078A00,11,1       ; MI/O11
M7012->Y:$078A00,12,1       ; MI/O12
M7013->Y:$078A00,13,1       ; MI/O13
M7014->Y:$078A00,14,1       ; MI/O14
M7015->Y:$078A00,15,1       ; MI/O15
M7016->Y:$078A00,16,1       ; MI/O16
M7017->Y:$078A00,17,1       ; MI/O17
M7018->Y:$078A00,18,1       ; MI/O18
M7019->Y:$078A00,19,1       ; MI/O19
M7020->Y:$078A00,20,1       ; MI/O20
M7021->Y:$078A00,21,1       ; MI/O21
M7022->Y:$078A00,22,1       ; MI/O22
M7023->Y:$078A00,23,1       ; MI/O23
M7024->Y:$078A01,0,1        ; MI/O24
M7025->Y:$078A01,1,1        ; MI/O25
M7026->Y:$078A01,2,1        ; MI/O26
M7027->Y:$078A01,3,1        ; MI/O27
M7028->Y:$078A01,4,1        ; MI/O28
M7029->Y:$078A01,5,1        ; MI/O29
M7030->Y:$078A01,6,1        ; MI/O30
M7031->Y:$078A01,7,1        ; MI/O31
M7032->Y:$078A01,8,1        ; MI/O32
M7033->Y:$078A01,9,1        ; MI/O33
```

```
M7034->Y:$078A01,10,1        ; MI/O34
M7035->Y:$078A01,11,1        ; MI/O35
M7036->Y:$078A01,12,1        ; MI/O36
M7037->Y:$078A01,13,1        ; MI/O37
M7038->Y:$078A01,14,1        ; MI/O38
M7039->Y:$078A01,15,1        ; MI/O39
M7040->Y:$078A01,16,1        ; MI/O40
M7041->Y:$078A01,17,1        ; MI/O41
M7042->Y:$078A01,18,1        ; MI/O42
M7043->Y:$078A01,19,1        ; MI/O43
M7044->Y:$078A01,20,1        ; MI/O44
M7045->Y:$078A01,21,1        ; MI/O45
M7046->Y:$078A01,22,1        ; MI/O46
M7047->Y:$078A01,23,1        ; MI/O47
; Encoder Conversion Table Result Registers (M8xxx matches I8xxx)
M8000->X:$003501,0,24,S      ; Line 0 result from conversion table
M8001->X:$003502,0,24,S      ; Line 1 result from conversion table
M8002->X:$003503,0,24,S      ; Line 2 result from conversion table
M8003->X:$003504,0,24,S      ; Line 3 result from conversion table
M8004->X:$003505,0,24,S      ; Line 4 result from conversion table
M8005->X:$003506,0,24,S      ; Line 5 result from conversion table
M8006->X:$003507,0,24,S      ; Line 6 result from conversion table
M8007->X:$003508,0,24,S      ; Line 7 result from conversion table
M8008->X:$003509,0,24,S      ; Line 8 result from conversion table
M8009->X:$00350A,0,24,S      ; Line 9 result from conversion table
M8010->X:$00350B,0,24,S      ; Line 10 result from conversion table
M8011->X:$00350C,0,24,S      ; Line 11 result from conversion table
M8012->X:$00350D,0,24,S      ; Line 12 result from conversion table
M8013->X:$00350E,0,24,S      ; Line 13 result from conversion table
M8014->X:$00350F,0,24,S      ; Line 14 result from conversion table
M8015->X:$003510,0,24,S      ; Line 15 result from conversion table
M8016->X:$003511,0,24,S      ; Line 16 result from conversion table
M8017->X:$003512,0,24,S      ; Line 17 result from conversion table
M8018->X:$003513,0,24,S      ; Line 18 result from conversion table
M8019->X:$003514,0,24,S      ; Line 19 result from conversion table
M8020->X:$003515,0,24,S      ; Line 20 result from conversion table
M8021->X:$003516,0,24,S      ; Line 21 result from conversion table
M8022->X:$003517,0,24,S      ; Line 22 result from conversion table
M8023->X:$003518,0,24,S      ; Line 23 result from conversion table
M8024->X:$003519,0,24,S      ; Line 24 result from conversion table
M8025->X:$00351A,0,24,S      ; Line 25 result from conversion table
M8026->X:$00351B,0,24,S      ; Line 26 result from conversion table
M8027->X:$00351C,0,24,S      ; Line 27 result from conversion table
M8028->X:$00351D,0,24,S      ; Line 28 result from conversion table
M8029->X:$00351E,0,24,S      ; Line 29 result from conversion table
M8030->X:$00351F,0,24,S      ; Line 30 result from conversion table
M8031->X:$003520,0,24,S      ; Line 31 result from conversion table
```

```
M8032->X:$003521,0,24,S      ; Line 32 result from conversion table
M8033->X:$003522,0,24,S      ; Line 33 result from conversion table
M8034->X:$003523,0,24,S      ; Line 34 result from conversion table
M8035->X:$003524,0,24,S      ; Line 35 result from conversion table
M8036->X:$003525,0,24,S      ; Line 36 result from conversion table
M8037->X:$003526,0,24,S      ; Line 37 result from conversion table
M8038->X:$003527,0,24,S      ; Line 38 result from conversion table
M8039->X:$003528,0,24,S      ; Line 39 result from conversion table
M8040->X:$003529,0,24,S      ; Line 40 result from conversion table
M8041->X:$00352A,0,24,S      ; Line 41 result from conversion table
M8042->X:$00352B,0,24,S      ; Line 42 result from conversion table
M8043->X:$00352C,0,24,S      ; Line 43 result from conversion table
M8044->X:$00352D,0,24,S      ; Line 44 result from conversion table
M8045->X:$00352E,0,24,S      ; Line 45 result from conversion table
M8046->X:$00352F,0,24,S      ; Line 46 result from conversion table
M8047->X:$003530,0,24,S      ; Line 47 result from conversion table
M8048->X:$003531,0,24,S      ; Line 48 result from conversion table
M8049->X:$003532,0,24,S      ; Line 49 result from conversion table
M8050->X:$003533,0,24,S      ; Line 50 result from conversion table
M8051->X:$003534,0,24,S      ; Line 51 result from conversion table
M8052->X:$003535,0,24,S      ; Line 52 result from conversion table
M8053->X:$003536,0,24,S      ; Line 53 result from conversion table
M8054->X:$003537,0,24,S      ; Line 54 result from conversion table
M8055->X:$003538,0,24,S      ; Line 55 result from conversion table
M8056->X:$003539,0,24,S      ; Line 56 result from conversion table
M8057->X:$00353A,0,24,S      ; Line 57 result from conversion table
M8058->X:$00353B,0,24,S      ; Line 58 result from conversion table
M8059->X:$00353C,0,24,S      ; Line 59 result from conversion table
M8060->X:$00353D,0,24,S      ; Line 60 result from conversion table
M8061->X:$00353E,0,24,S      ; Line 61 result from conversion table
M8062->X:$00353F,0,24,S      ; Line 62 result from conversion table
M8063->X:$003540,0,24,S      ; Line 63 result from conversion table
```

TURBO PMAC2 SUGGESTED M-VARIABLE DEFINITIONS

; This file contains suggested definitions for M-variables on the Turbo PMAC2. For the UMAC Turbo (3U-Rack Turbo PMAC2), there is a separate set of suggested M-variables. Note that these are only suggestions; the user is free to make whatever definitions are desired.

```
; Clear existing definitions
CLOSE                       ; Make sure no buffer is open
M0..8191->*                 ; All M-variables are now self-referenced
; JI/O Port M-variables
M0->Y:$078400,0             ; I/O00 Data Line; J3 Pin 1
M1->Y:$078400,1             ; I/O01 Data Line; J3 Pin 2
M2->Y:$078400,2             ; I/O02 Data Line; J3 Pin 3
M3->Y:$078400,3             ; I/O03 Data Line; J3 Pin 4
M4->Y:$078400,4             ; I/O04 Data Line; J3 Pin 5
M5->Y:$078400,5             ; I/O05 Data Line; J3 Pin 6
M6->Y:$078400,6             ; I/O06 Data Line; J3 Pin 7
M7->Y:$078400,7             ; I/O07 Data Line; J3 Pin 8
M8->Y:$078400,8             ; I/O08 Data Line; J3 Pin 9
M9->Y:$078400,9             ; I/O09 Data Line; J3 Pin 10
M10->Y:$078400,10           ; I/O10 Data Line; J3 Pin 11
M11->Y:$078400,11           ; I/O11 Data Line; J3 Pin 12
M12->Y:$078400,12           ; I/O12 Data Line; J3 Pin 13
M13->Y:$078400,13           ; I/O13 Data Line; J3 Pin 14
M14->Y:$078400,14           ; I/O14 Data Line; J3 Pin 15
M15->Y:$078400,15           ; I/O15 Data Line; J3 Pin 16
M16->Y:$078400,16           ; I/O16 Data Line; J3 Pin 17
M17->Y:$078400,17           ; I/O17 Data Line; J3 Pin 18
M18->Y:$078400,18           ; I/O18 Data Line; J3 Pin 19
M19->Y:$078400,19           ; I/O19 Data Line; J3 Pin 20
M20->Y:$078400,20           ; I/O20 Data Line; J3 Pin 21
M21->Y:$078400,21           ; I/O21 Data Line; J3 Pin 22
M22->Y:$078400,22           ; I/O22 Data Line; J3 Pin 23
M23->Y:$078400,23           ; I/O23 Data Line; J3 Pin 24
M24->Y:$078401,0            ; I/O24 Data Line; J3 Pin 25
M25->Y:$078401,1            ; I/O25 Data Line; J3 Pin 26
M26->Y:$078401,2            ; I/O26 Data Line; J3 Pin 27
M27->Y:$078401,3            ; I/O27 Data Line; J3 Pin 28
M28->Y:$078401,4            ; I/O28 Data Line; J3 Pin 29
M29->Y:$078401,5            ; I/O29 Data Line; J3 Pin 30
M30->Y:$078401,6            ; I/O30 Data Line; J3 Pin 31
M31->Y:$078401,7            ; I/O31 Data Line; J3 Pin 32
M32->X:$078400,0,8          ; Direction control for I/O00 to I/O07
M33->Y:$070800,0            ; Buffer direction control for I/O00 to I/O07
M34->X:$078400,8,8          ; Direction control for I/O08 to I/O15
M35->Y:$070800,1            ; Buffer direction control for I/O08 to I/O15
M36->X:$078400,16,8         ; Direction control for I/O16 to I/O23
M37->Y:$070800,2            ; Buffer direction control for I/O16 to I/O23
M38->X:$078401,0,8          ; Direction control for I/O24 to I/O31
```

```
M39->Y:$070800,3                ; Buffer direction control for I/O24 to I/O31
; JTHW Thumbwheel Multiplexer Port M-variables
M40->Y:$078402,8                ; SEL0 Line; J2 Pin 4
M41->Y:$078402,9                ; SEL1 Line; J2 Pin 6
M42->Y:$078402,10               ; SEL2 Line; J2 Pin 8
M43->Y:$078402,11               ; SEL3 Line; J2 Pin 10
M44->Y:$078402,12               ; SEL4 Line; J2 Pin 12
M45->Y:$078402,13               ; SEL5 Line; J2 Pin 14
M46->Y:$078402,14               ; SEL6 Line; J2 Pin 16
M47->Y:$078402,15               ; SEL7 Line; J2 Pin 18
M48->Y:$078402,8,8,U            ; SEL0-7 Lines treated as a byte
M50->Y:$078402,0                ; DAT0 Line; J2 Pin 3
M51->Y:$078402,1                ; DAT1 Line; J2 Pin 5
M52->Y:$078402,2                ; DAT2 Line; J2 Pin 7
M53->Y:$078402,3                ; DAT3 Line; J2 Pin 9
M54->Y:$078402,4                ; DAT4 Line; J2 Pin 11
M55->Y:$078402,5                ; DAT5 Line; J2 Pin 13
M56->Y:$078402,6                ; DAT6 Line; J2 Pin 15
M57->Y:$078402,7                ; DAT7 Line; J2 Pin 17
M58->Y:$078402,0,8,U            ; DAT0-7 Lines treated as a byte
M60->X:$078402,0,8              ; Direction control for DAT0 to DAT7
M61->Y:$070800,4                ; Buffer direction control for DAT0 to DAT7, PCbus
M61->Y:$070802,0                ; Buffer direction control for DAT0 to DAT7, VMEbus
M62->X:$078400,8,8              ; Direction control for SEL0 to SEL7
M63->Y:$070800,5                ; Buffer direction control for SEL0 to SEL7, PCbus
M63->Y:$070802,1                ; Buffer direction control for SEL0 to SEL7, VMEbus
; Miscellaneous global registers
M70->X:$FFFF8C,0,24             ; Time between phase interrupts (CPU cycles/2)
M71->X:$000037,0,24             ; Time for phase tasks (CPU cycles/2)
M72->Y:$000037,0,24             ; Time for servo tasks (CPU cycles/2)
M73->X:$00000B,0,24             ; Time for RTI tasks (CPU cycles/2)
M80->X:$000025,0,24             ; Minimum watchdog timer count
M81->X:$000024,0,24             ; Pointer to display buffer
M82->Y:$001080,0,8              ; First character of display buffer
M83->X:$000006,12,1             ; Firmware checksum error bit
M84->X:$000006,13,1             ; Any memory checksum error bit
M85->X:$000006,5,1              ; MACRO auxiliary communications error bit
M86->X:$000006,6,1              ; ACC-34 serial parity error bit
; VME/DPRAM active setup registers
M90->X:$070006,0,8      ; VME Active Address Modifier (Bits 0-7; from I90)
M91->X:$070007,0,8      ; VME Active Address Modifier Don't Care Bits
                        ; (Bits 0-7; from I91)
M92->X:$070008,0,8      ; VME Active Base Address Bits A31-A24 (Bits 0-7; from I92)
M93->X:$070009,0,8      ; VME Active Mailbox Base Address Bits A23-A16
                        ; ISA Active DPRAM Base Address bits A23-A16
                        ; (Bits 0-7; from I93)
M94->X:$07000A,0,8      ; VME Active Mailbox Base Address Bits A15-A08
```

```
                        ; ISA Active DPRAM Base Address bits A15-A14, Enable &
                        ; Bank Select
                        ; (Bits 0-7; from I94)
M95->X:$07000B,0,8      ; VME Active Interrupt Level (Bits 0-7; from I95)
M96->X:$07000C,0,8      ; VME Active Interrupt Vector (Bits 0-7; from I96)
M97->X:$07000D,0,8      ; VME Active DPRAM Base Address Bits A23-A20
                        ; (Bits 0-7; from I97)
M98->X:$07000E,0,8      ; VME Active DPRAM Enable State (Bits 0-7; from I98)
M99->X:$07000F,0,8      ; VME Active Address Width Control (Bits 0-7; from I99)
; Servo cycle counter (read only) -- counts up once per servo cycle
M100->X:$000000,0,24,S      ; 24-bit servo cycle counter
; Servo IC 0 Registers for PMAC2 Channel 1 (usually for Motor #1)
M101->X:$078001,0,24,S      ; ENC1 24-bit counter position
M102->Y:$078002,8,16,S      ; OUT1A command value; DAC or PWM
M103->X:$078003,0,24,S      ; ENC1 captured position
M104->Y:$078003,8,16,S      ; OUT1B command value; DAC or PWM
M105->Y:$078005,8,16,S      ; ADC1A input value
M106->Y:$078006,8,16,S      ; ADC1B input value
M107->Y:$078004,8,16,S      ; OUT1C command value; PFM or PWM
M108->Y:$078007,0,24,S      ; ENC1 compare A position
M109->X:$078007,0,24,S      ; ENC1 compare B position
M110->X:$078006,0,24,S      ; ENC1 compare autoincrement value
M111->X:$078005,11          ; ENC1 compare initial state write enable
M112->X:$078005,12          ; ENC1 compare initial state
M114->X:$078005,14          ; AENA1 output status
M115->X:$078000,19          ; USER1 flag input status
M116->X:$078000,9           ; ENC1 compare output value
M117->X:$078000,11          ; ENC1 capture flag
M118->X:$078000,8           ; ENC1 count error flag
M119->X:$078000,14          ; CHC1 input status
M120->X:$078000,16          ; HMFL1 flag input status
M121->X:$078000,17          ; PLIM1 flag input status
M122->X:$078000,18          ; MLIM1 flag input status
M123->X:$078000,15          ; FAULT1 flag input status
M124->X:$078000,20          ; Channel 1 W flag input status
M125->X:$078000,21          ; Channel 1 V flag input status
M126->X:$078000,22          ; Channel 1 U flag input status
M127->X:$078000,23          ; Channel 1 T flag input status
M128->X:$078000,20,4        ; Channel 1 TUVW inputs as 4-bit value
; Motor #1 Status Bits
M130->Y:$0000C0,11,1        ; #1 Stopped-on-position-limit bit
M131->X:$0000B0,21,1        ; #1 Positive-end-limit-set bit
M132->X:$0000B0,22,1        ; #1 Negative-end-limit-set bit
M133->X:$0000B0,13,1        ; #1 Desired-velocity-zero bit
M135->X:$0000B0,15,1        ; #1 Dwell-in-progress bit
M137->X:$0000B0,17,1        ; #1 Running-program bit
M138->X:$0000B0,18,1        ; #1 Open-loop-mode bit
```

```
M139->X:$0000B0,19,1          ; #1 Amplifier-enabled status bit
M140->Y:$0000C0,0,1           ; #1 Background in-position bit
M141->Y:$0000C0,1,1           ; #1 Warning-following error bit
M142->Y:$0000C0,2,1           ; #1 Fatal-following-error bit
M143->Y:$0000C0,3,1           ; #1 Amplifier-fault-error bit
M144->Y:$0000C0,13,1          ; #1 Foreground in-position bit
M145->Y:$0000C0,10,1          ; #1 Home-complete bit
M146->Y:$0000C0,6,1           ; #1 Integrated following error fault bit
M147->Y:$0000C0,5,1           ; #1 I2T fault bit
M148->Y:$0000C0,8,1           ; #1 Phasing error fault bit
M149->Y:$0000C0,9,1           ; #1 Phasing search-in-progress bit
; MACRO IC 0 Node 0 Flag Registers (usually used for Motor #1)
M150->X:$003440,0,24          ; MACRO IC 0 Node 0 flag status register
M151->Y:$003440,0,24          ; MACRO IC 0 Node 0 flag command register
M153->X:$003440,20,4          ; MACRO IC 0 Node 0 TUVW flags
M154->Y:$003440,14,1          ; MACRO IC 0 Node 0 amplifier enable flag
M155->X:$003440,15,1          ; MACRO IC 0 Node 0 node/amplifier fault flag
M156->X:$003440,16,1          ; MACRO IC 0 Node 0 home flag
M157->X:$003440,17,1          ; MACRO IC 0 Node 0 positive limit flag
M158->X:$003440,18,1          ; MACRO IC 0 Node 0 negative limit flag
M159->X:$003440,19,1          ; MACRO IC 0 Node 0 user flag
; Motor #1 Move Registers
M161->D:$000088               ; #1 Commanded position (1/[Ixx08*32] cts)
M162->D:$00008B               ; #1 Actual position (1/[Ixx08*32] cts)
M163->D:$0000C7               ; #1 Target (end) position (1/[Ixx08*32] cts)
M164->D:$0000CC               ; #1 Position bias (1/[Ixx08*32] cts)
M166->X:$00009D,0,24,S        ; #1 Actual velocity (1/[Ixx09*32] cts/cyc)
M167->D:$00008D               ; #1 Present master pos (1/[Ixx07*32] cts)
M168->X:$0000BF,8,16,S        ; #1 Filter Output (16-bit DAC bits)
M169->D:$000090               ; #1 Compensation correction (1/[Ixx08*32] cts)
M170->D:$0000B4               ; #1 Present phase position (including fraction)
M171->X:$0000B4,24,S          ; #1 Present phase position (counts *Ixx70)
M172->L:$0000D7               ; #1 Variable jog position/distance (cts)
M173->Y:$0000CE,0,24,S        ; #1 Encoder home capture position (cts)
M174->D:$0000EF               ; #1 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M175->X:$0000B9,8,16,S        ; #1 Actual quadrature current
M176->Y:$0000B9,8,16,S        ; #1 Actual direct current
M177->X:$0000BC,8,16,S        ; #1 Quadrature current-loop integrator output
M178->Y:$0000BC,8,16,S        ; #1 Direct current-loop integrator output
M179->X:$0000AE,8,16,S        ; #1 PID internal filter result (16-bit DAC bits)
M188->Y:$078001,0,12,U        ; IC 0 Ch 1 Compare A fractional count
M189->Y:$078000,0,12,U        ; IC 0 Ch 1 Compare B fractional count
; Motor #1 Axis Definition Registers
M191->L:$0000CF               ; #1 X/U/A/B/C-Axis scale factor (cts/unit)
M192->L:$0000D0               ; #1 Y/V-Axis scale factor (cts/unit)
M193->L:$0000D1               ; #1 Z/W-Axis scale factor (cts/unit)
M194->L:$0000D2               ; #1 Axis offset (cts)
```

```
; Servo IC 0 Registers for PMAC2 Channel 2 (usually for Motor #2)
M201->X:$078009,0,24,S      ; ENC2 24-bit counter position
M202->Y:$07800A,8,16,S      ; OUT2A command value; DAC or PWM
M203->X:$07800B,0,24,S      ; ENC2 captured position
M204->Y:$07800B,8,16,S      ; OUT2B command value; DAC or PWM
M205->Y:$07800D,8,16,S      ; ADC2A input value
M206->Y:$07800E,8,16,S      ; ADC2B input value
M207->Y:$07800C,8,16,S      ; OUT2C command value; PFM or PWM
M208->Y:$07800F,0,24,S      ; ENC2 compare A position
M209->X:$07800F,0,24,S      ; ENC2 compare B position
M210->X:$07800E,0,24,S      ; ENC2 compare autoincrement value
M211->X:$07800D,11          ; ENC2 compare initial state write enable
M212->X:$07800D,12          ; ENC2 compare initial state
M214->X:$07800D,14          ; AENA2 output status
M215->X:$078008,19          ; USER2 flag input status
M216->X:$078008,9           ; ENC2 compare output value
M217->X:$078008,11          ; ENC2 capture flag
M218->X:$078008,8           ; ENC2 count error flag
M219->X:$078008,14          ; CHC2 input status
M220->X:$078008,16          ; HMFL2 flag input status
M221->X:$078008,17          ; PLIM2 flag input status
M222->X:$078008,18          ; MLIM2 flag input status
M223->X:$078008,15          ; FAULT2 flag input status
M224->X:$078008,20          ; Channel 2 W flag input status
M225->X:$078008,21          ; Channel 2 V flag input status
M226->X:$078008,22          ; Channel 2 U flag input status
M227->X:$078008,23          ; Channel 2 T flag input status
M228->X:$078008,20,4        ; Channel 2 TUVW inputs as 4-bit value
; Motor #2 Status Bits
M230->Y:$000140,11,1        ; #2 Stopped-on-position-limit bit
M231->X:$000130,21,1        ; #2 Positive-end-limit-set bit
M232->X:$000130,22,1        ; #2 Negative-end-limit-set bit
M233->X:$000130,13,1        ; #2 Desired-velocity-zero bit
M235->X:$000130,15,1        ; #2 Dwell-in-progress bit
M237->X:$000130,17,1        ; #2 Running-program bit
M238->X:$000130,18,1        ; #2 Open-loop-mode bit
M239->X:$000130,19,1        ; #2 Amplifier-enabled status bit
M240->Y:$000140,0,1         ; #2 Background in-position bit
M241->Y:$000140,1,1         ; #2 Warning-following error bit
M242->Y:$000140,2,1         ; #2 Fatal-following-error bit
M243->Y:$000140,3,1         ; #2 Amplifier-fault-error bit
M244->Y:$000140,13,1        ; #2 Foreground in-position bit
M245->Y:$000140,10,1        ; #2 Home-complete bit
M246->Y:$000140,6,1         ; #2 Integrated following error fault bit
M247->Y:$000140,5,1         ; #2 I2T fault bit
M248->Y:$000140,8,1         ; #2 Phasing error fault bit
M249->Y:$000140,9,1         ; #2 Phasing search-in-progress bit
```

```
; MACRO IC 0 Node 1 Flag Registers (usually used for Motor #2)
M250->X:$003441,0,24          ; MACRO IC 0 Node 1 flag status register
M251->Y:$003441,0,24          ; MACRO IC 0 Node 1 flag command register
M253->X:$003441,20,4          ; MACRO IC 0 Node 1 TUVW flags
M254->Y:$003441,14,1          ; MACRO IC 0 Node 1 amplifier enable flag
M255->X:$003441,15,1          ; MACRO IC 0 Node 1 node/amplifier fault flag
M256->X:$003441,16,1          ; MACRO IC 0 Node 1 home flag
M257->X:$003441,17,1          ; MACRO IC 0 Node 1 positive limit flag
M258->X:$003441,18,1          ; MACRO IC 0 Node 1 negative limit flag
M259->X:$003441,19,1          ; MACRO IC 0 Node 1 user flag
; Motor #2 Move Registers
M261->D:$000108               ; #2 Commanded position (1/[Ixx08*32] cts)
M262->D:$00010B               ; #2 Actual position (1/[Ixx08*32] cts)
M263->D:$000147               ; #2 Target (end) position (1/[Ixx08*32] cts)
M264->D:$00014C               ; #2 Position bias (1/[Ixx08*32] cts)
M266->X:$00011D,0,24,S        ; #2 Actual velocity (1/[Ixx09*32] cts/cyc)
M267->D:$00010D               ; #2 Present master pos (1/[Ixx07*32] cts)
M268->X:$00013F,8,16,S        ; #2 Filter Output (16-bit DAC bits)
M269->D:$000110               ; #2 Compensation correction (1/[Ixx08*32] cts)
M270->D:$000134               ; #2 Present phase position (including fraction)
M271->X:$000134,24,S          ; #2 Present phase position (counts *Ixx70)
M272->L:$000157               ; #2 Variable jog position/distance (cts)
M273->Y:$00014E,0,24,S        ; #2 Encoder home capture position (cts)
M274->D:$00016F               ; #2 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M275->X:$000139,8,16,S        ; #2 Actual quadrature current
M276->Y:$000139,8,16,S        ; #2 Actual direct current
M277->X:$00013C,8,16,S        ; #2 Quadrature current-loop integrator output
M278->Y:$00013C,8,16,S        ; #2 Direct current-loop integrator output
M279->X:$00012E,8,16,S        ; #2 PID internal filter result (16-bit DAC bits)
M288->Y:$078009,0,12,U        ; IC 0 Ch 2 Compare A fractional count
M289->Y:$078008,0,12,U        ; IC 0 Ch 2 Compare B fractional count
; Motor #2 Axis Definition Registers
M291->L:$00014F               ; #2 X/U/A/B/C-Axis scale factor (cts/unit)
M292->L:$000150               ; #2 Y/V-Axis scale factor (cts/unit)
M293->L:$000151               ; #2 Z/W-Axis scale factor (cts/unit)
M294->L:$000152               ; #2 Axis offset (cts)
; Servo IC 0 Registers for PMAC2 Channel 3 (usually for Motor #3)
M301->X:$078011,0,24,S        ; ENC3 24-bit counter position
M302->Y:$078012,8,16,S        ; OUT3A command value; DAC or PWM
M303->X:$078013,0,24,S        ; ENC3 captured position
M304->Y:$078013,8,16,S        ; OUT3B command value; DAC or PWM
M305->Y:$078015,8,16,S        ; ADC3A input value
M306->Y:$078016,8,16,S        ; ADC3B input value
M307->Y:$078014,8,16,S        ; OUT3C command value; PFM or PWM
M308->Y:$078017,0,24,S        ; ENC3 compare A position
M309->X:$078017,0,24,S        ; ENC3 compare B position
M310->X:$078016,0,24,S        ; ENC3 compare autoincrement value
```

```
M311->X:$078015,11          ; ENC3 compare initial state write enable
M312->X:$078015,12          ; ENC3 compare initial state
M314->X:$078015,14          ; AENA3 output status
M315->X:$078010,19          ; USER3 flag input status
M316->X:$078010,9           ; ENC3 compare output value
M317->X:$078010,11          ; ENC3 capture flag
M318->X:$078010,8           ; ENC3 count error flag
M319->X:$078010,14          ; CHC3 input status
M320->X:$078010,16          ; HMFL3 flag input status
M321->X:$078010,17          ; PLIM3 flag input status
M322->X:$078010,18          ; MLIM3 flag input status
M323->X:$078010,15          ; FAULT3 flag input status
M324->X:$078010,20          ; Channel 3 W flag input status
M325->X:$078010,21          ; Channel 3 V flag input status
M326->X:$078010,22          ; Channel 3 U flag input status
M327->X:$078010,23          ; Channel 3 T flag input status
M328->X:$078010,20,4        ; Channel 3 TUVW inputs as 4-bit value
; Motor #3 Status Bits
M330->Y:$0001C0,11,1        ; #3 Stopped-on-position-limit bit
M331->X:$0001B0,21,1        ; #3 Positive-end-limit-set bit
M332->X:$0001B0,22,1        ; #3 Negative-end-limit-set bit
M333->X:$0001B0,13,1        ; #3 Desired-velocity-zero bit
M335->X:$0001B0,15,1        ; #3 Dwell-in-progress bit
M337->X:$0001B0,17,1        ; #3 Running-program bit
M338->X:$0001B0,18,1        ; #3 Open-loop-mode bit
M339->X:$0001B0,19,1        ; #3 Amplifier-enabled status bit
M340->Y:$0001C0,0,1         ; #3 Background in-position bit
M341->Y:$0001C0,1,1         ; #3 Warning-following error bit
M342->Y:$0001C0,2,1         ; #3 Fatal-following-error bit
M343->Y:$0001C0,3,1         ; #3 Amplifier-fault-error bit
M344->Y:$0001C0,13,1        ; #3 Foreground in-position bit
M345->Y:$0001C0,10,1        ; #3 Home-complete bit
M346->Y:$0001C0,6,1         ; #3 Integrated following error fault bit
M347->Y:$0001C0,5,1         ; #3 I2T fault bit
M348->Y:$0001C0,8,1         ; #3 Phasing error fault bit
M349->Y:$0001C0,9,1         ; #3 Phasing search-in-progress bit
; MACRO IC 0 Node 4 Flag Registers (usually used for Motor #3)
M350->X:$003444,0,24        ; MACRO IC 0 Node 4 flag status register
M351->Y:$003444,0,24        ; MACRO IC 0 Node 4 flag command register
M353->X:$003444,20,4        ; MACRO IC 0 Node 4 TUVW flags
M354->Y:$003444,14,1        ; MACRO IC 0 Node 4 amplifier enable flag
M355->X:$003444,15,1        ; MACRO IC 0 Node 4 node/amplifier fault flag
M356->X:$003444,16,1        ; MACRO IC 0 Node 4 home flag
M357->X:$003444,17,1        ; MACRO IC 0 Node 4 positive limit flag
M358->X:$003444,18,1        ; MACRO IC 0 Node 4 negative limit flag
M359->X:$003444,19,1        ; MACRO IC 0 Node 4 user flag
```

```
; Motor #3 Move Registers
M361->D:$000188              ; #3 Commanded position (1/[Ixx08*32] cts)
M362->D:$00018B              ; #3 Actual position (1/[Ixx08*32] cts)
M363->D:$0001C7              ; #3 Target (end) position (1/[Ixx08*32] cts)
M364->D:$0001CC              ; #3 Position bias (1/[Ixx08*32] cts)
M366->X:$00019D,0,24,S       ; #3 Actual velocity (1/[Ixx09*32] cts/cyc)
M367->D:$00018D              ; #3 Present master pos (1/[Ixx07*32] cts)
M368->X:$0001BF,8,16,S       ; #3 Filter Output (16-bit DAC bits)
M369->D:$000190              ; #3 Compensation correction (1/[Ixx08*32] cts)
M370->D:$0001B4              ; #3 Present phase position (including fraction)
M371->X:$0001B4,24,S         ; #3 Present phase position (counts *Ixx70)
M372->L:$0001D7              ; #3 Variable jog position/distance (cts)
M373->Y:$0001CE,0,24,S       ; #3 Encoder home capture position (cts)
M374->D:$0001EF              ; #3 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M375->X:$0001B9,8,16,S       ; #3 Actual quadrature current
M376->Y:$0001B9,8,16,S       ; #3 Actual direct current
M377->X:$0001BC,8,16,S       ; #3 Quadrature current-loop integrator output
M378->Y:$0001BC,8,16,S       ; #3 Direct current-loop integrator output
M379->X:$0001AE,8,16,S       ; #3 PID internal filter result (16-bit DAC bits)
M388->Y:$078011,0,12,U       ; IC 0 Ch 3 Compare A fractional count
M389->Y:$078010,0,12,U       ; IC 0 Ch 3 Compare B fractional count
; Motor #3 Axis Definition Registers
M391->L:$0001CF              ; #3 X/U/A/B/C-Axis scale factor (cts/unit)
M392->L:$0001D0              ; #3 Y/V-Axis scale factor (cts/unit)
M393->L:$0001D1              ; #3 Z/W-Axis scale factor (cts/unit)
M394->L:$0001D2              ; #3 Axis offset (cts)
; Servo IC 0 Registers for PMAC2 Channel 4 (usually for Motor #4)
M401->X:$078019,0,24,S       ; ENC4 24-bit counter position
M402->Y:$07801A,8,16,S       ; OUT4A command value; DAC or PWM
M403->X:$07801B,0,24,S       ; ENC4 captured position
M404->Y:$07801B,8,16,S       ; OUT4B command value; DAC or PWM
M405->Y:$07801D,8,16,S       ; ADC4A input value
M406->Y:$07801E,8,16,S       ; ADC4B input value
M407->Y:$07801C,8,16,S       ; OUT4C command value; PFM or PWM
M408->Y:$07801F,0,24,S       ; ENC4 compare A position
M409->X:$07801F,0,24,S       ; ENC4 compare B position
M410->X:$07801E,0,24,S       ; ENC4 compare autoincrement value
M411->X:$07801D,11           ; ENC4 compare initial state write enable
M412->X:$07801D,12           ; ENC4 compare initial state
M414->X:$07801D,14           ; AENA4 output status
M415->X:$078018,19           ; USER4 flag input status
M416->X:$078018,9            ; ENC4 compare output value
M417->X:$078018,11           ; ENC4 capture flag
M418->X:$078018,8            ; ENC4 count error flag
M419->X:$078018,14           ; HMFL4 flag input status
M420->X:$078018,16           ; CHC4 input status
M421->X:$078018,17           ; PLIM4 flag input status
```

```
M422->X:$078018,18          ; MLIM4 flag input status
M423->X:$078018,15          ; FAULT4 flag input status
M424->X:$078018,20          ; Channel 4 W flag input status
M425->X:$078018,21          ; Channel 4 V flag input status
M426->X:$078018,22          ; Channel 4 U flag input status
M427->X:$078018,23          ; Channel 4 T flag input status
M428->X:$078018,20,4        ; Channel 4 TUVW inputs as 4-bit value
; Motor #4 Status Bits
M430->Y:$000240,11,1        ; #4 Stopped-on-position-limit bit
M431->X:$000230,21,1        ; #4 Positive-end-limit-set bit
M432->X:$000230,22,1        ; #4 Negative-end-limit-set bit
M433->X:$000230,13,1        ; #4 Desired-velocity-zero bit
M435->X:$000230,15,1        ; #4 Dwell-in-progress bit
M437->X:$000230,17,1        ; #4 Running-program bit
M438->X:$000230,18,1        ; #4 Open-loop-mode bit
M439->X:$000230,19,1        ; #4 Amplifier-enabled status bit
M440->Y:$000240,0,1         ; #4 Background in-position bit
M441->Y:$000240,1,1         ; #4 Warning-following error bit
M442->Y:$000240,2,1         ; #4 Fatal-following-error bit
M443->Y:$000240,3,1         ; #4 Amplifier-fault-error bit
M444->Y:$000240,13,1        ; #4 Foreground in-position bit
M445->Y:$000240,10,1        ; #4 Home-complete bit
M446->Y:$000240,6,1         ; #4 Integrated following error fault bit
M447->Y:$000240,5,1         ; #4 I2T fault bit
M448->Y:$000240,8,1         ; #4 Phasing error fault bit
M449->Y:$000240,9,1         ; #4 Phasing search-in-progress bit
; MACRO IC 0 Node 5 Flag Registers (usually used for Motor #4)
M450->X:$003445,0,24        ; MACRO IC 0 Node 5 flag status register
M451->Y:$003445,0,24        ; MACRO IC 0 Node 5 flag command register
M453->X:$003445,20,4        ; MACRO IC 0 Node 5 TUVW flags
M454->Y:$003445,14,1        ; MACRO IC 0 Node 5 amplifier enable flag
M455->X:$003445,15,1        ; MACRO IC 0 Node 5 node/amplifier fault flag
M456->X:$003445,16,1        ; MACRO IC 0 Node 5 home flag
M457->X:$003445,17,1        ; MACRO IC 0 Node 5 positive limit flag
M458->X:$003445,18,1        ; MACRO IC 0 Node 5 negative limit flag
M459->X:$003445,19,1        ; MACRO IC 0 Node 5 user flag
; Motor #4 Move Registers
M461->D:$000208             ; #4 Commanded position (1/[Ixx08*32] cts)
M462->D:$00020B             ; #4 Actual position (1/[Ixx08*32] cts)
M463->D:$000247             ; #4 Target (end) position (1/[Ixx08*32] cts)
M464->D:$00024C             ; #4 Position bias (1/[Ixx08*32] cts)
M466->X:$00021D,0,24,S      ; #4 Actual velocity (1/[Ixx09*32] cts/cyc)
M467->D:$00020D             ; #4 Present master pos (1/[Ixx07*32] cts)
M468->X:$00023F,8,16,S      ; #4 Filter Output (16-bit DAC bits)
M469->D:$000210             ; #4 Compensation correction (1/[Ixx08*32] cts)
M470->D:$000234             ; #4 Present phase position (including fraction)
M471->X:$000234,24,S        ; #4 Present phase position (counts *Ixx70)
```

```
M472->L:$000257              ; #4 Variable jog position/distance (cts)
M473->Y:$00024E,0,24,S       ; #4 Encoder home capture position (cts)
M474->D:$00026F              ; #4 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M475->X:$000239,8,16,S       ; #4 Actual quadrature current
M476->Y:$000239,8,16,S       ; #4 Actual direct current
M477->X:$00023C,8,16,S       ; #4 Quadrature current-loop integrator output
M478->Y:$00023C,8,16,S       ; #4 Direct current-loop integrator output
M479->X:$00022E,8,16,S       ; #4 PID internal filter result (16-bit DAC bits)
M488->Y:$078019,0,12,U       ; IC 0 Ch 4 Compare A fractional count
M489->Y:$078018,0,12,U       ; IC 0 Ch 4 Compare B fractional count
; Motor #4 Axis Definition Registers
M491->L:$00024F              ; #4 X/U/A/B/C-Axis scale factor (cts/unit)
M492->L:$000250              ; #4 Y/V-Axis scale factor (cts/unit)
M493->L:$000251              ; #4 Z/W-Axis scale factor (cts/unit)
M494->L:$000252              ; #4 Axis offset (cts)
; Servo IC 1 Registers for PMAC2 Channel 5 (usually for Motor #5)
M501->X:$078101,0,24,S       ; ENC5 24-bit counter position
M502->Y:$078102,8,16,S       ; OUT5A command value; DAC or PWM
M503->X:$078103,0,24,S       ; ENC5 captured position
M504->Y:$078103,8,16,S       ; OUT5B command value; DAC or PWM
M505->Y:$078105,8,16,S       ; ADC5A input value
M506->Y:$078106,8,16,S       ; ADC5B input value
M507->Y:$078104,8,16,S       ; OUT5C command value; PFM or PWM
M508->Y:$078107,0,24,S       ; ENC5 compare A position
M509->X:$078107,0,24,S       ; ENC5 compare B position
M510->X:$078106,0,24,S       ; ENC5 compare autoincrement value
M511->X:$078105,11           ; ENC5 compare initial state write enable
M512->X:$078105,12           ; ENC5 compare initial state
M514->X:$078105,14           ; AENA5 output status
M515->X:$078100,19           ; USER5 flag input status
M516->X:$078100,9            ; ENC5 compare output value
M517->X:$078100,11           ; ENC5 capture flag
M518->X:$078100,8            ; ENC5 count error flag
M519->X:$078100,14           ; CHC5 input status
M520->X:$078100,16           ; HMFL5 flag input status
M521->X:$078100,17           ; PLIM5 flag input status
M522->X:$078100,18           ; MLIM5 flag input status
M523->X:$078100,15           ; FAULT5 flag input status
M524->X:$078100,20           ; Channel 5 W flag input status
M525->X:$078100,21           ; Channel 5 V flag input status
M526->X:$078100,22           ; Channel 5 U flag input status
M527->X:$078100,23           ; Channel 5 T flag input status
M528->X:$078100,20,4         ; Channel 5 TUVW inputs as 4-bit value
; Motor #5 Status Bits
M530->Y:$0002C0,11,1         ; #5 Stopped-on-position-limit bit
M531->X:$0002B0,21,1         ; #5 Positive-end-limit-set bit
M532->X:$0002B0,22,1         ; #5 Negative-end-limit-set bit
```

```
M533->X:$0002B0,13,1          ; #5 Desired-velocity-zero bit
M535->X:$0002B0,15,1          ; #5 Dwell-in-progress bit
M537->X:$0002B0,17,1          ; #5 Running-program bit
M538->X:$0002B0,18,1          ; #5 Open-loop-mode bit
M539->X:$0002B0,19,1          ; #5 Amplifier-enabled status bit
M540->Y:$0002C0,0,1           ; #5 Background in-position bit
M541->Y:$0002C0,1,1           ; #5 Warning-following error bit
M542->Y:$0002C0,2,1           ; #5 Fatal-following-error bit
M543->Y:$0002C0,3,1           ; #5 Amplifier-fault-error bit
M544->Y:$0002C0,13,1          ; #5 Foreground in-position bit
M545->Y:$0002C0,10,1          ; #5 Home-complete bit
M546->Y:$0002C0,6,1           ; #5 Integrated following error fault bit
M547->Y:$0002C0,5,1           ; #5 I2T fault bit
M548->Y:$0002C0,8,1           ; #5 Phasing error fault bit
M549->Y:$0002C0,9,1           ; #5 Phasing search-in-progress bit
; MACRO IC 0 Node 8 Flag Registers (usually used for Motor #5)
M550->X:$003448,0,24          ; MACRO IC 0 Node 8 flag status register
M551->Y:$003448,0,24          ; MACRO IC 0 Node 8 flag command register
M553->X:$003448,20,4          ; MACRO IC 0 Node 8 TUVW flags
M554->Y:$003448,14,1          ; MACRO IC 0 Node 8 amplifier enable flag
M555->X:$003448,15,1          ; MACRO IC 0 Node 8 node/amplifier fault flag
M556->X:$003448,16,1          ; MACRO IC 0 Node 8 home flag
M557->X:$003448,17,1          ; MACRO IC 0 Node 8 positive limit flag
M558->X:$003448,18,1          ; MACRO IC 0 Node 8 negative limit flag
M559->X:$003448,19,1          ; MACRO IC 0 Node 8 user flag
; Motor #5 Move Registers
M561->D:$000288               ; #5 Commanded position (1/[Ixx08*32] cts)
M562->D:$00028B               ; #5 Actual position (1/[Ixx08*32] cts)
M563->D:$0002C7               ; #5 Target (end) position (1/[Ixx08*32] cts)
M564->D:$0002CC               ; #5 Position bias (1/[Ixx08*32] cts)
M566->X:$00029D,0,24,S        ; #5 Actual velocity (1/[Ixx09*32] cts/cyc)
M567->D:$00028D               ; #5 Present master pos (1/[Ixx07*32] cts)
M568->X:$0002BF,8,16,S        ; #5 Filter Output (16-bit DAC bits)
M569->D:$000290               ; #5 Compensation correction (1/[Ixx08*32] cts)
M570->D:$0002B4               ; #5 Present phase position (including fraction)
M571->X:$0002B4,24,S          ; #5 Present phase position (counts *Ixx70)
M572->L:$0002D7               ; #5 Variable jog position/distance (cts)
M573->Y:$0002CE,0,24,S        ; #5 Encoder home capture position (cts)
M574->D:$0002EF               ; #5 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M575->X:$0002B9,8,16,S        ; #5 Actual quadrature current
M576->Y:$0002B9,8,16,S        ; #5 Actual direct current
M577->X:$0002BC,8,16,S        ; #5 Quadrature current-loop integrator output
M578->Y:$0002BC,8,16,S        ; #5 Direct current-loop integrator output
M579->X:$0002AE,8,16,S        ; #5 PID internal filter result (16-bit DAC bits)
M588->Y:$078101,0,12,U        ; IC 1 Ch 1 Compare A fractional count
M589->Y:$078100,0,12,U        ; IC 1 Ch 1 Compare B fractional count
```

```
; Motor #5 Axis Definition Registers
M591->L:$0002CF              ; #5 X/U/A/B/C-Axis scale factor (cts/unit)
M592->L:$0002D0              ; #5 Y/V-Axis scale factor (cts/unit)
M593->L:$0002D1              ; #5 Z/W-Axis scale factor (cts/unit)
M594->L:$0002D2              ; #5 Axis offset (cts)
; Servo IC 1 Registers for PMAC2 Channel 6 (usually for Motor #6)
M601->X:$078109,0,24,S       ; ENC6 24-bit counter position
M602->Y:$07810A,8,16,S       ; OUT6A command value; DAC or PWM
M603->X:$07810B,0,24,S       ; ENC6 captured position
M604->Y:$07810B,8,16,S       ; OUT6B command value; DAC or PWM
M605->Y:$07810D,8,16,S       ; ADC6A input value
M606->Y:$07810E,8,16,S       ; ADC6B input value
M607->Y:$07810C,8,16,S       ; OUT6C command value; PFM or PWM
M608->Y:$07810F,0,24,S       ; ENC6 compare A position
M609->X:$07810F,0,24,S       ; ENC6 compare B position
M610->X:$07810E,0,24,S       ; ENC6 compare autoincrement value
M611->X:$07810D,11           ; ENC6 compare initial state write enable
M612->X:$07810D,12           ; ENC6 compare initial state
M614->X:$07810D,14           ; AENA6 output status
M615->X:$078108,19           ; USER6 flag input status
M616->X:$078108,9            ; ENC6 compare output value
M617->X:$078108,11           ; ENC6 capture flag
M618->X:$078108,8            ; ENC6 count error flag
M619->X:$078108,14           ; CHC6 input status
M620->X:$078108,16           ; HMFL6 flag input status
M621->X:$078108,17           ; PLIM6 flag input status
M622->X:$078108,18           ; MLIM6 flag input status
M623->X:$078108,15           ; FAULT6 flag input status
M624->X:$078108,20           ; Channel 6 W flag input status
M625->X:$078108,21           ; Channel 6 V flag input status
M626->X:$078108,22           ; Channel 6 U flag input status
M627->X:$078108,23           ; Channel 6 T flag input status
M628->X:$078108,20,4         ; Channel 6 TUVW inputs as 4-bit value
; Motor #6 Status Bits
M630->Y:$000340,11,1         ; #6 Stopped-on-position-limit bit
M631->X:$000330,21,1         ; #6 Positive-end-limit-set bit
M632->X:$000330,22,1         ; #6 Negative-end-limit-set bit
M633->X:$000330,13,1         ; #6 Desired-velocity-zero bit
M635->X:$000330,15,1         ; #6 Dwell-in-progress bit
M637->X:$000330,17,1         ; #6 Running-program bit
M638->X:$000330,18,1         ; #6 Open-loop-mode bit
M639->X:$000330,19,1         ; #6 Amplifier-enabled status bit
M640->Y:$000340,0,1          ; #6 Background in-position bit
M641->Y:$000340,1,1          ; #6 Warning-following error bit
M642->Y:$000340,2,1          ; #6 Fatal-following-error bit
M643->Y:$000340,3,1          ; #6 Amplifier-fault-error bit
M644->Y:$000340,13,1         ; #6 Foreground in-position bit
```

```
M645->Y:$000340,10,1          ; #6 Home-complete bit
M646->Y:$000340,6,1           ; #6 Integrated following error fault bit
M647->Y:$000340,5,1           ; #6 I2T fault bit
M648->Y:$000340,8,1           ; #6 Phasing error fault bit
M649->Y:$000340,9,1           ; #6 Phasing search-in-progress bit
; MACRO IC 0 Node 9 Flag Registers (usually used for Motor #6)
M650->X:$003449,0,24          ; MACRO IC 0 Node 9 flag status register
M651->Y:$003449,0,24          ; MACRO IC 0 Node 9 flag command register
M653->X:$003449,20,4          ; MACRO IC 0 Node 9 TUVW flags
M654->Y:$003449,14,1          ; MACRO IC 0 Node 9 amplifier enable flag
M655->X:$003449,15,1          ; MACRO IC 0 Node 9 node/amplifier fault flag
M656->X:$003449,16,1          ; MACRO IC 0 Node 9 home flag
M657->X:$003449,17,1          ; MACRO IC 0 Node 9 positive limit flag
M658->X:$003449,18,1          ; MACRO IC 0 Node 9 negative limit flag
M659->X:$003449,19,1          ; MACRO IC 0 Node 9 user flag
; Motor #6 Move Registers
M661->D:$000308               ; #6 Commanded position (1/[Ixx08*32] cts)
M662->D:$00030B               ; #6 Actual position (1/[Ixx08*32] cts)
M663->D:$000347               ; #6 Target (end) position (1/[Ixx08*32] cts)
M664->D:$00034C               ; #6 Position bias (1/[Ixx08*32] cts)
M666->X:$00031D,0,24,S        ; #6 Actual velocity (1/[Ixx09*32] cts/cyc)
M667->D:$00030D               ; #6 Present master pos (1/[Ixx07*32] cts)
M668->X:$00033F,8,16,S        ; #6 Filter Output (16-bit DAC bits)
M669->D:$000310               ; #6 Compensation correction (1/[Ixx08*32] cts)
M670->D:$000334               ; #6 Present phase position (including fraction)
M671->X:$000334,24,S          ; #6 Present phase position (counts *Ixx70)
M672->L:$000357               ; #6 Variable jog position/distance (cts)
M673->Y:$00034E,0,24,S        ; #6 Encoder home capture position (cts)
M674->D:$00036F               ; #6 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M675->X:$000339,8,16,S        ; #6 Actual quadrature current
M676->Y:$000339,8,16,S        ; #6 Actual direct current
M677->X:$00033C,8,16,S        ; #6 Quadrature current-loop integrator output
M678->Y:$00033C,8,16,S        ; #6 Direct current-loop integrator output
M679->X:$00032E,8,16,S        ; #6 PID internal filter result (16-bit DAC bits)
M688->Y:$078109,0,12,U        ; IC 1 Ch 2 Compare A fractional count
M689->Y:$078108,0,12,U        ; IC 1 Ch 2 Compare B fractional count
; Motor #6 Axis Definition Registers
M691->L:$00034F               ; #6 X/U/A/B/C-Axis scale factor (cts/unit)
M692->L:$000350               ; #6 Y/V-Axis scale factor (cts/unit)
M693->L:$000351               ; #6 Z/W-Axis scale factor (cts/unit)
M694->L:$000352               ; #6 Axis offset (cts)
; Servo IC 1 Registers for PMAC2 Channel 7 (usually for Motor #7)
M701->X:$078111,0,24,S        ; ENC7 24-bit counter position
M702->Y:$078112,8,16,S        ; OUT7A command value; DAC or PWM
M703->X:$078113,0,24,S        ; ENC7 captured position
M704->Y:$078113,8,16,S        ; OUT7B command value; DAC or PWM
M705->Y:$078115,8,16,S        ; ADC7A input value
```

```
M706->Y:$078116,8,16,S      ; ADC7B input value
M707->Y:$078114,8,16,S      ; OUT7C command value; PFM or PWM
M708->Y:$078117,0,24,S      ; ENC7 compare A position
M709->X:$078117,0,24,S      ; ENC7 compare B position
M710->X:$078116,0,24,S      ; ENC7 compare autoincrement value
M711->X:$078115,11          ; ENC7 compare initial state write enable
M712->X:$078115,12          ; ENC7 compare initial state
M714->X:$078115,14          ; AENA7 output status
M715->X:$078110,19          ; CHC7 input status
M716->X:$078110,9           ; ENC7 compare output value
M717->X:$078110,11          ; ENC7 capture flag
M718->X:$078110,8           ; ENC7 count error flag
M719->X:$078110,14          ; CHC7 input status
M720->X:$078110,16          ; HMFL7 flag input status
M721->X:$078110,17          ; PLIM7 flag input status
M722->X:$078110,18          ; MLIM7 flag input status
M723->X:$078110,15          ; FAULT7 flag input status
M724->X:$078110,20          ; Channel 7 W flag input status
M725->X:$078110,21          ; Channel 7 V flag input status
M726->X:$078110,22          ; Channel 7 U flag input status
M727->X:$078110,23          ; Channel 7 T flag input status
M728->X:$078110,20,4        ; Channel 7 TUVW inputs as 4-bit value
; Motor #7 Status Bits
M730->Y:$0003C0,11,1        ; #7 Stopped-on-position-limit bit
M731->X:$0003B0,21,1        ; #7 Positive-end-limit-set bit
M732->X:$0003B0,22,1        ; #7 Negative-end-limit-set bit
M733->X:$0003B0,13,1        ; #7 Desired-velocity-zero bit
M735->X:$0003B0,15,1        ; #7 Dwell-in-progress bit
M737->X:$0003B0,17,1        ; #7 Running-program bit
M738->X:$0003B0,18,1        ; #7 Open-loop-mode bit
M739->X:$0003B0,19,1        ; #7 Amplifier-enabled status bit
M740->Y:$0003C0,0,1         ; #7 Background in-position bit
M741->Y:$0003C0,1,1         ; #7 Warning-following error bit
M742->Y:$0003C0,2,1         ; #7 Fatal-following-error bit
M743->Y:$0003C0,3,1         ; #7 Amplifier-fault-error bit
M744->Y:$0003C0,13,1        ; #7 Foreground in-position bit
M745->Y:$0003C0,10,1        ; #7 Home-complete bit
M746->Y:$0003C0,6,1         ; #7 Integrated following error fault bit
M747->Y:$0003C0,5,1         ; #7 I2T fault bit
M748->Y:$0003C0,8,1         ; #7 Phasing error fault bit
M749->Y:$0003C0,9,1         ; #7 Phasing search-in-progress bit
; MACRO IC 0 Node 12 Flag Registers (usually used for Motor #7)
M750->X:$00344C,0,24        ; MACRO IC 0 Node 12 flag status register
M751->Y:$00344C,0,24        ; MACRO IC 0 Node 12 flag command register
M753->X:$00344C,20,4        ; MACRO IC 0 Node 12 TUVW flags
M754->Y:$00344C,14,1        ; MACRO IC 0 Node 12 amplifier enable flag
M755->X:$00344C,15,1        ; MACRO IC 0 Node 12 node/amplifier fault flag
```

```
M756->X:$00344C,16,1        ; MACRO IC 0 Node 12 home flag
M757->X:$00344C,17,1        ; MACRO IC 0 Node 12 positive limit flag
M758->X:$00344C,18,1        ; MACRO IC 0 Node 12 negative limit flag
M759->X:$00344C,19,1        ; MACRO IC 0 Node 12 user flag
; Motor #7 Move Registers
M761->D:$000388             ; #7 Commanded position (1/[Ixx08*32] cts)
M762->D:$00038B             ; #7 Actual position (1/[Ixx08*32] cts)
M763->D:$0003C7             ; #7 Target (end) position (1/[Ixx08*32] cts)
M764->D:$0003CC             ; #7 Position bias (1/[Ixx08*32] cts)
M766->X:$00039D,0,24,S      ; #7 Actual velocity (1/[Ixx09*32] cts/cyc)
M767->D:$00038D             ; #7 Present master pos (1/[Ixx07*32] cts)
M768->X:$0003BF,8,16,S      ; #7 Filter Output (16-bit DAC bits)
M769->D:$000390             ; #7 Compensation correction (1/[Ixx08*32] cts)
M770->D:$0003B4             ; #7 Present phase position (including fraction)
M771->X:$0003B4,24,S        ; #7 Present phase position (counts *Ixx70)
M772->L:$0003D7             ; #7 Variable jog position/distance (cts)
M773->Y:$0003CE,0,24,S      ; #7 Encoder home capture position (cts)
M774->D:$0003EF             ; #7 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M775->X:$0003B9,8,16,S      ; #7 Actual quadrature current
M776->Y:$0003B9,8,16,S      ; #7 Actual direct current
M777->X:$0003BC,8,16,S      ; #7 Quadrature current-loop integrator output
M778->Y:$0003BC,8,16,S      ; #7 Direct current-loop integrator output
M779->X:$0003AE,8,16,S      ; #7 PID internal filter result (16-bit DAC bits)
M788->Y:$078111,0,12,U      ; IC 1 Ch 3 Compare A fractional count
M789->Y:$078110,0,12,U      ; IC 1 Ch 3 Compare B fractional count
; Motor #7 Axis Definition Registers
M791->L:$0003CF             ; #7 X/U/A/B/C-Axis scale factor (cts/unit)
M792->L:$0003D0             ; #7 Y/V-Axis scale factor (cts/unit)
M793->L:$0003D1             ; #7 Z/W-Axis scale factor (cts/unit)
M794->L:$0003D2             ; #7 Axis offset (cts)
; Servo IC 1 Registers for PMAC2 Channel 8 (usually for Motor #8)
M801->X:$078119,0,24,S      ; ENC8 24-bit counter position
M802->Y:$07811A,8,16,S      ; OUT8A command value; DAC or PWM
M803->X:$07811B,0,24,S      ; ENC8 captured position
M804->Y:$07811B,8,16,S      ; OUT8B command value; DAC or PWM
M805->Y:$07811D,8,16,S      ; ADC8A input value
M806->Y:$07811E,8,16,S      ; ADC8B input value
M807->Y:$07811C,8,16,S      ; OUT8C command value; PFM or PWM
M808->Y:$07811F,0,24,S      ; ENC8 compare A position
M809->X:$07811F,0,24,S      ; ENC8 compare B position
M810->X:$07811E,0,24,S      ; ENC8 compare autoincrement value
M811->X:$07811D,11          ; ENC8 compare initial state write enable
M812->X:$07811D,12          ; ENC8 compare initial state
M814->X:$07811D,14          ; AENA8 output status
M815->X:$078118,19          ; USER8 flag input status
M816->X:$078118,9           ; ENC8 compare output value
M817->X:$078118,11          ; ENC8 capture flag
```

```
M818->X:$078118,8          ; ENC8 count error flag
M819->X:$078118,14         ; CHC8 input status
M820->X:$078118,16         ; HMFL8 flag input status
M821->X:$078118,17         ; PLIM8 flag input status
M822->X:$078118,18         ; MLIM8 flag input status
M823->X:$078118,15         ; FAULT8 flag input status
M824->X:$078118,20         ; Channel 8 W flag input status
M825->X:$078118,21         ; Channel 8 V flag input status
M826->X:$078118,22         ; Channel 8 U flag input status
M827->X:$078118,23         ; Channel 8 T flag input status
M828->X:$078118,20,4       ; Channel 8 TUVW inputs as 4-bit value
; Motor #8 Status Bits
M830->Y:$000440,11,1       ; #8 Stopped-on-position-limit bit
M831->X:$000430,21,1       ; #8 Positive-end-limit-set bit
M832->X:$000430,22,1       ; #8 Negative-end-limit-set bit
M833->X:$000430,13,1       ; #8 Desired-velocity-zero bit
M835->X:$000430,15,1       ; #8 Dwell-in-progress bit
M837->X:$000430,17,1       ; #8 Running-program bit
M838->X:$000430,18,1       ; #8 Open-loop-mode bit
M839->X:$000430,19,1       ; #8 Amplifier-enabled status bit
M840->Y:$000440,0,1        ; #8 Background in-position bit
M841->Y:$000440,1,1        ; #8 Warning-following error bit
M842->Y:$000440,2,1        ; #8 Fatal-following-error bit
M843->Y:$000440,3,1        ; #8 Amplifier-fault-error bit
M844->Y:$000440,13,1       ; #8 Foreground in-position bit
M845->Y:$000440,10,1       ; #8 Home-complete bit
M846->Y:$000440,6,1        ; #8 Integrated following error fault bit
M847->Y:$000440,5,1        ; #8 I2T fault bit
M848->Y:$000440,8,1        ; #8 Phasing error fault bit
M849->Y:$000440,9,1        ; #8 Phasing search-in-progress bit
; MACRO IC 0 Node 13 Flag Registers (usually used for Motor #8)
M850->X:$00344D,0,24       ; MACRO IC 0 Node 13 flag status register
M851->Y:$00344D,0,24       ; MACRO IC 0 Node 13 flag command register
M853->X:$00344D,20,4       ; MACRO IC 0 Node 13 TUVW flags
M854->Y:$00344D,14,1       ; MACRO IC 0 Node 13 amplifier enable flag
M855->X:$00344D,15,1       ; MACRO IC 0 Node 13 node/amplifier fault flag
M856->X:$00344D,16,1       ; MACRO IC 0 Node 13 home flag
M857->X:$00344D,17,1       ; MACRO IC 0 Node 13 positive limit flag
M858->X:$00344D,18,1       ; MACRO IC 0 Node 13 negative limit flag
M859->X:$00344D,19,1       ; MACRO IC 0 Node 13 user flag
; Motor #8 Move Registers
M861->D:$000408            ; #8 Commanded position (1/[Ixx08*32] cts)
M862->D:$00040B            ; #8 Actual position (1/[Ixx08*32] cts)
M863->D:$000447            ; #8 Target (end) position (1/[Ixx08*32] cts)
M864->D:$00044C            ; #8 Position bias (1/[Ixx08*32] cts)
M866->X:$00041D,0,24,S     ; #8 Actual velocity (1/[Ixx09*32] cts/cyc)
M867->D:$00040D            ; #8 Present master pos (1/[Ixx07*32] cts)
```

```
M868->X:$00043F,8,16,S      ; #8 Filter Output (16-bit DAC bits)
M869->D:$000410             ; #8 Compensation correction (1/[Ixx08*32] cts)
M870->D:$000434             ; #8 Present phase position (including fraction)
M871->X:$000434,24,S        ; #8 Present phase position (counts *Ixx70)
M872->L:$000457             ; #8 Variable jog position/distance (cts)
M873->Y:$00044E,0,24,S      ; #8 Encoder home capture position (cts)
M874->D:$00046F             ; #8 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M875->X:$000439,8,16,S      ; #8 Actual quadrature current
M876->Y:$000439,8,16,S      ; #8 Actual direct current
M877->X:$00043C,8,16,S      ; #8 Quadrature current-loop integrator output
M878->Y:$00043C,8,16,S      ; #8 Direct current-loop integrator output
M879->X:$00042E,8,16,S      ; #8 PID internal filter result (16-bit DAC bits)
M888->Y:$078119,0,12,U      ; IC 1 Ch 4 Compare A fractional count
M889->Y:$078118,0,12,U      ; IC 1 Ch 4 Compare B fractional count
; Motor #8 Axis Definition Registers
M891->L:$00044F             ; #8 X/U/A/B/C-Axis scale factor (cts/unit)
M892->L:$000450             ; #8 Y/V-Axis scale factor (cts/unit)
M893->L:$000451             ; #8 Z/W-Axis scale factor (cts/unit)
M894->L:$000452             ; #8 Axis offset (cts)
; Servo IC 2 Registers for 1st ACC-24 Channel 1 (usually for Motor #9)
M901->X:$078201,0,24,S      ; ENC1 24-bit counter position
M902->Y:$078202,8,16,S      ; OUT1A command value; DAC or PWM
M903->X:$078203,0,24,S      ; ENC1 captured position
M904->Y:$078203,8,16,S      ; OUT1B command value; DAC or PWM
M905->Y:$078205,8,16,S      ; ADC1A input value
M906->Y:$078206,8,16,S      ; ADC1B input value
M907->Y:$078204,8,16,S      ; OUT1C command value; PFM or PWM
M908->Y:$078207,0,24,S      ; ENC1 compare A position
M909->X:$078207,0,24,S      ; ENC1 compare B position
M910->X:$078206,0,24,S      ; ENC1 compare autoincrement value
M911->X:$078205,11          ; ENC1 compare initial state write enable
M912->X:$078205,12          ; ENC1 compare initial state
M914->X:$078205,14          ; AENA1 output status
M915->X:$078200,19          ; USER1 flag input status
M916->X:$078200,9           ; ENC1 compare output value
M917->X:$078200,11          ; ENC1 capture flag
M918->X:$078200,8           ; ENC1 count error flag
M919->X:$078200,14          ; CHC1 input status
M920->X:$078200,16          ; HMFL1 flag input status
M921->X:$078200,17          ; PLIM1 flag input status
M922->X:$078200,18          ; MLIM1 flag input status
M923->X:$078200,15          ; FAULT1 flag input status
M924->X:$078200,20          ; Channel 1 W flag input status
M925->X:$078200,21          ; Channel 1 V flag input status
M926->X:$078200,22          ; Channel 1 U flag input status
M927->X:$078200,23          ; Channel 1 T flag input status
M928->X:$078200,20,4        ; Channel 1 TUVW inputs as 4-bit value
```

```
; Motor #9 Status Bits
M930->Y:$0004C0,11,1          ; #9 Stopped-on-position-limit bit
M931->X:$0004B0,21,1          ; #9 Positive-end-limit-set bit
M932->X:$0004B0,22,1          ; #9 Negative-end-limit-set bit
M933->X:$0004B0,13,1          ; #9 Desired-velocity-zero bit
M935->X:$0004B0,15,1          ; #9 Dwell-in-progress bit
M937->X:$0004B0,17,1          ; #9 Running-program bit
M938->X:$0004B0,18,1          ; #9 Open-loop-mode bit
M939->X:$0004B0,19,1          ; #9 Amplifier-enabled status bit
M940->Y:$0004C0,0,1           ; #9 Background in-position bit
M941->Y:$0004C0,1,1           ; #9 Warning-following error bit
M942->Y:$0004C0,2,1           ; #9 Fatal-following-error bit
M943->Y:$0004C0,3,1           ; #9 Amplifier-fault-error bit
M944->Y:$0004C0,13,1          ; #9 Foreground in-position bit
M945->Y:$0004C0,10,1          ; #9 Home-complete bit
M946->Y:$0004C0,6,1           ; #9 Integrated following error fault bit
M947->Y:$0004C0,5,1           ; #9 I2T fault bit
M948->Y:$0004C0,8,1           ; #9 Phasing error fault bit
M949->Y:$0004C0,9,1           ; #9 Phasing search-in-progress bit
; MACRO IC 1 Node 0 Flag Registers (usually used for Motor #9)
M950->X:$003450,0,24          ; MACRO IC 1 Node 0 flag status register
M951->Y:$003450,0,24          ; MACRO IC 1 Node 0 flag command register
M953->X:$003450,20,4          ; MACRO IC 1 Node 0 TUVW flags
M954->Y:$003450,14,1          ; MACRO IC 1 Node 0 amplifier enable flag
M955->X:$003450,15,1          ; MACRO IC 1 Node 0 node/amplifier fault flag
M956->X:$003450,16,1          ; MACRO IC 1 Node 0 home flag
M957->X:$003450,17,1          ; MACRO IC 1 Node 0 positive limit flag
M958->X:$003450,18,1          ; MACRO IC 1 Node 0 negative limit flag
M959->X:$003450,19,1          ; MACRO IC 1 Node 0 user flag
; Motor #9 Move Registers
M961->D:$000488               ; #9 Commanded position (1/[Ixx08*32] cts)
M962->D:$00048B               ; #9 Actual position (1/[Ixx08*32] cts)
M963->D:$0004C7               ; #9 Target (end) position (1/[Ixx08*32] cts)
M964->D:$0004CC               ; #9 Position bias (1/[Ixx08*32] cts)
M966->X:$00049D,0,24,S        ; #9 Actual velocity (1/[Ixx09*32] cts/cyc)
M967->D:$00048D               ; #9 Present master pos (1/[Ixx07*32] cts)
M968->X:$0004BF,8,16,S        ; #9 Filter Output (16-bit DAC bits)
M969->D:$000490               ; #9 Compensation correction (1/[Ixx08*32] cts)
M970->D:$0004B4               ; #9 Present phase position (including fraction)
M971->X:$0004B4,24,S          ; #9 Present phase position (counts *Ixx70)
M972->L:$0004D7               ; #9 Variable jog position/distance (cts)
M973->Y:$0004CE,0,24,S        ; #9 Encoder home capture position (cts)
M974->D:$0004EF               ; #9 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M975->X:$0004B9,8,16,S        ; #9 Actual quadrature current
M976->Y:$0004B9,8,16,S        ; #9 Actual direct current
M977->X:$0004BC,8,16,S        ; #9 Quadrature current-loop integrator output
M978->Y:$0004BC,8,16,S        ; #9 Direct current-loop integrator output
```

```
M979->X:$0004AE,8,16,S      ; #9 PID internal filter result (16-bit DAC bits)
M988->Y:$078201,0,12,U      ; IC 2 Ch 1 Compare A fractional count
M989->Y:$078200,0,12,U      ; IC 2 Ch 1 Compare B fractional count
; Motor #9 Axis Definition Registers
M991->L:$0004CF             ; #9 X/U/A/B/C-Axis scale factor (cts/unit)
M992->L:$0004D0             ; #9 Y/V-Axis scale factor (cts/unit)
M993->L:$0004D1             ; #9 Z/W-Axis scale factor (cts/unit)
M994->L:$0004D2             ; #9 Axis offset (cts)
; Servo IC 2 Registers for 1st ACC-24 Channel 2 (usually for Motor #10)
M1001->X:$078209,0,24,S     ; ENC2 24-bit counter position
M1002->Y:$07820A,8,16,S     ; OUT2A command value; DAC or PWM
M1003->X:$07820B,0,24,S     ; ENC2 captured position
M1004->Y:$07820B,8,16,S     ; OUT2B command value; DAC or PWM
M1005->Y:$07820D,8,16,S     ; ADC2A input value
M1006->Y:$07820E,8,16,S     ; ADC2B input value
M1007->Y:$07820C,8,16,S     ; OUT2C command value; PFM or PWM
M1008->Y:$07820F,0,24,S     ; ENC2 compare A position
M1009->X:$07820F,0,24,S     ; ENC2 compare B position
M1010->X:$07820E,0,24,S     ; ENC2 compare autoincrement value
M1011->X:$07820D,11         ; ENC2 compare initial state write enable
M1012->X:$07820D,12         ; ENC2 compare initial state
M1014->X:$07820D,14         ; AENA2 output status
M1015->X:$078208,19         ; USER2 flag input status
M1016->X:$078208,9          ; ENC2 compare output value
M1017->X:$078208,11         ; ENC2 capture flag
M1018->X:$078208,8          ; ENC2 count error flag
M1019->X:$078208,14         ; CHC2 input status
M1020->X:$078208,16         ; HMFL2 flag input status
M1021->X:$078208,17         ; PLIM2 flag input status
M1022->X:$078208,18         ; MLIM2 flag input status
M1023->X:$078208,15         ; FAULT2 flag input status
M1024->X:$078208,20         ; Channel 2 W flag input status
M1025->X:$078208,21         ; Channel 2 V flag input status
M1026->X:$078208,22         ; Channel 2 U flag input status
M1027->X:$078208,23         ; Channel 2 T flag input status
M1028->X:$078208,20,4       ; Channel 2 TUVW inputs as 4-bit value
; Motor #10 Status Bits
M1030->Y:$000540,11,1       ; #10 Stopped-on-position-limit bit
M1031->X:$000530,21,1       ; #10 Positive-end-limit-set bit
M1032->X:$000530,22,1       ; #10 Negative-end-limit-set bit
M1033->X:$000530,13,1       ; #10 Desired-velocity-zero bit
M1035->X:$000530,15,1       ; #10 Dwell-in-progress bit
M1037->X:$000530,17,1       ; #10 Running-program bit
M1038->X:$000530,18,1       ; #10 Open-loop-mode bit
M1039->X:$000530,19,1       ; #10 Amplifier-enabled status bit
M1040->Y:$000540,0,1        ; #10 Background in-position bit
M1041->Y:$000540,1,1        ; #10 Warning-following error bit
```

```
M1042->Y:$000540,2,1          ; #10 Fatal-following-error bit
M1043->Y:$000540,3,1          ; #10 Amplifier-fault-error bit
M1044->Y:$000540,13,1         ; #10 Foreground in-position bit
M1045->Y:$000540,10,1         ; #10 Home-complete bit
M1046->Y:$000540,6,1          ; #10 Integrated following error fault bit
M1047->Y:$000540,5,1          ; #10 I2T fault bit
M1048->Y:$000540,8,1          ; #10 Phasing error fault bit
M1049->Y:$000540,9,1          ; #10 Phasing search-in-progress bit
; MACRO IC 1 Node 1 Flag Registers (usually used for Motor #10)
M1050->X:$003451,0,24         ; MACRO IC 1 Node 1 flag status register
M1051->Y:$003451,0,24         ; MACRO IC 1 Node 1 flag command register
M1053->X:$003451,20,4         ; MACRO IC 1 Node 1 TUVW flags
M1054->Y:$003451,14,1         ; MACRO IC 1 Node 1 amplifier enable flag
M1055->X:$003451,15,1         ; MACRO IC 1 Node 1 node/amplifier fault flag
M1056->X:$003451,16,1         ; MACRO IC 1 Node 1 home flag
M1057->X:$003451,17,1         ; MACRO IC 1 Node 1 positive limit flag
M1058->X:$003451,18,1         ; MACRO IC 1 Node 1 negative limit flag
M1059->X:$003451,19,1         ; MACRO IC 1 Node 1 user flag
; Motor #10 Move Registers
M1061->D:$000508              ; #10 Commanded position (1/[Ixx08*32] cts)
M1062->D:$00050B              ; #10 Actual position (1/[Ixx08*32] cts)
M1063->D:$000547              ; #10 Target (end) position (1/[Ixx08*32] cts)
M1064->D:$00054C              ; #10 Position bias (1/[Ixx08*32] cts)
M1066->X:$00051D,0,24,S       ; #10 Actual velocity (1/[Ixx09*32] cts/cyc)
M1067->D:$00050D              ; #10 Present master pos (1/[Ixx07*32] cts)
M1068->X:$00053F,8,16,S       ; #10 Filter Output (16-bit DAC bits)
M1069->D:$000510              ; #10 Compensation correction (1/[Ixx08*32] cts)
M1070->D:$000534              ; #10 Present phase position (including fraction)
M1071->X:$000534,24,S         ; #10 Present phase position (counts *Ixx70)
M1072->L:$000557              ; #10 Variable jog position/distance (cts)
M1073->Y:$00054E,0,24,S       ; #10 Encoder home capture position (cts)
M1074->D:$00056F              ; #10 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1075->X:$000539,8,16,S       ; #10 Actual quadrature current
M1076->Y:$000539,8,16,S       ; #10 Actual direct current
M1077->X:$00053C,8,16,S       ; #10 Quadrature current-loop integrator output
M1078->Y:$00053C,8,16,S       ; #10 Direct current-loop integrator output
M1079->X:$00052E,8,16,S       ; #10 PID internal filter result (16-bit DAC bits)
M1088->Y:$078209,0,12,U       ; IC 2 Ch 2 Compare A fractional count
M1089->Y:$078208,0,12,U       ; IC 2 Ch 2 Compare A fractional count
; Motor #10 Axis Definition Registers
M1091->L:$00054F              ; #10 X/U/A/B/C-Axis scale factor (cts/unit)
M1092->L:$000550              ; #10 Y/V-Axis scale factor (cts/unit)
M1093->L:$000551              ; #10 Z/W-Axis scale factor (cts/unit)
M1094->L:$000552              ; #10 Axis offset (cts)
```

; Servo IC 2 Registers for 1st ACC-24 Channel 3 (usually for Motor #11)

```
M1101->X:$078211,0,24,S      ; ENC3 24-bit counter position
M1102->Y:$078212,8,16,S      ; OUT3A command value; DAC or PWM
M1103->X:$078213,0,24,S      ; ENC3 captured position
M1104->Y:$078213,8,16,S      ; OUT3B command value; DAC or PWM
M1105->Y:$078215,8,16,S      ; ADC3A input value
M1106->Y:$078216,8,16,S      ; ADC3B input value
M1107->Y:$078214,8,16,S      ; OUT3C command value; PFM or PWM
M1108->Y:$078217,0,24,S      ; ENC3 compare A position
M1109->X:$078217,0,24,S      ; ENC3 compare B position
M1110->X:$078216,0,24,S      ; ENC3 compare autoincrement value
M1111->X:$078215,11          ; ENC3 compare initial state write enable
M1112->X:$078215,12          ; ENC3 compare initial state
M1114->X:$078215,14          ; AENA3 output status
M1115->X:$078210,19          ; USER3 flag input status
M1116->X:$078210,9           ; ENC3 compare output value
M1117->X:$078210,11          ; ENC3 capture flag
M1118->X:$078210,8           ; ENC3 count error flag
M1119->X:$078210,14          ; CHC3 input status
M1120->X:$078210,16          ; HMFL3 flag input status
M1121->X:$078210,17          ; PLIM3 flag input status
M1122->X:$078210,18          ; MLIM3 flag input status
M1123->X:$078210,15          ; FAULT3 flag input status
M1124->X:$078210,20          ; Channel 3 W flag input status
M1125->X:$078210,21          ; Channel 3 V flag input status
M1126->X:$078210,22          ; Channel 3 U flag input status
M1127->X:$078210,23          ; Channel 3 T flag input status
M1128->X:$078210,20,4        ; Channel 3 TUVW inputs as 4-bit value
```

; Motor #11 Status Bits

```
M1130->Y:$0005C0,11,1        ; #11 Stopped-on-position-limit bit
M1131->X:$0005B0,21,1        ; #11 Positive-end-limit-set bit
M1132->X:$0005B0,22,1        ; #11 Negative-end-limit-set bit
M1133->X:$0005B0,13,1        ; #11 Desired-velocity-zero bit
M1135->X:$0005B0,15,1        ; #11 Dwell-in-progress bit
M1137->X:$0005B0,17,1        ; #11 Running-program bit
M1138->X:$0005B0,18,1        ; #11 Open-loop-mode bit
M1139->X:$0005B0,19,1        ; #11 Amplifier-enabled status bit
M1140->Y:$0005C0,0,1         ; #11 Background in-position bit
M1141->Y:$0005C0,1,1         ; #11 Warning-following error bit
M1142->Y:$0005C0,2,1         ; #11 Fatal-following-error bit
M1143->Y:$0005C0,3,1         ; #11 Amplifier-fault-error bit
M1144->Y:$0005C0,13,1        ; #11 Foreground in-position bit
M1145->Y:$0005C0,10,1        ; #11 Home-complete bit
M1146->Y:$0005C0,6,1         ; #11 Integrated following error fault bit
M1147->Y:$0005C0,5,1         ; #11 I2T fault bit
M1148->Y:$0005C0,8,1         ; #11 Phasing error fault bit
M1149->Y:$0005C0,9,1         ; #11 Phasing search-in-progress bit
```

```
; MACRO IC 1 Node 4 Flag Registers (usually used for Motor #11)
M1150->X:$003454,0,24        ; MACRO IC 1 Node 4 flag status register
M1151->Y:$003454,0,24        ; MACRO IC 1 Node 4 flag command register
M1153->X:$003454,20,4        ; MACRO IC 1 Node 4 TUVW flags
M1154->Y:$003454,14,1        ; MACRO IC 1 Node 4 amplifier enable flag
M1155->X:$003454,15,1        ; MACRO IC 1 Node 4 node/amplifier fault flag
M1156->X:$003454,16,1        ; MACRO IC 1 Node 4 home flag
M1157->X:$003454,17,1        ; MACRO IC 1 Node 4 positive limit flag
M1158->X:$003454,18,1        ; MACRO IC 1 Node 4 negative limit flag
M1159->X:$003454,19,1        ; MACRO IC 1 Node 4 user flag
; Motor #11 Move Registers
M1161->D:$000588             ; #11 Commanded position (1/[Ixx08*32] cts)
M1162->D:$00058B             ; #11 Actual position (1/[Ixx08*32] cts)
M1163->D:$0005C7             ; #11 Target (end) position (1/[Ixx08*32] cts)
M1164->D:$0005CC             ; #11 Position bias (1/[Ixx08*32] cts)
M1166->X:$00059D,0,24,S      ; #11 Actual velocity (1/[Ixx09*32] cts/cyc)
M1167->D:$00058D             ; #11 Present master pos (1/[Ixx07*32] cts)
M1168->X:$0005BF,8,16,S      ; #11 Filter Output (16-bit DAC bits)
M1169->D:$000590             ; #11 Compensation correction (1/[Ixx08*32] cts)
M1170->D:$0005B4             ; #11 Present phase position (including fraction)
M1171->X:$0005B4,24,S        ; #11 Present phase position (counts *Ixx70)
M1172->L:$0005D7             ; #11 Variable jog position/distance (cts)
M1173->Y:$0005CE,0,24,S      ; #11 Encoder home capture position (cts)
M1174->D:$0005EF             ; #11 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1175->X:$0005B9,8,16,S      ; #11 Actual quadrature current
M1176->Y:$0005B9,8,16,S      ; #11 Actual direct current
M1177->X:$0005BC,8,16,S      ; #11 Quadrature current-loop integrator output
M1178->Y:$0005BC,8,16,S      ; #11 Direct current-loop integrator output
M1179->X:$0005AE,8,16,S      ; #11 PID internal filter result (16-bit DAC bits)
M1188->Y:$078211,0,12,U      ; IC 2 Ch 3 Compare A fractional count
M1189->Y:$078210,0,12,U      ; IC 2 Ch 3 Compare A fractional count
; Motor #11 Axis Definition Registers
M1191->L:$0005CF             ; #11 X/U/A/B/C-Axis scale factor (cts/unit)
M1192->L:$0005D0             ; #11 Y/V-Axis scale factor (cts/unit)
M1193->L:$0005D1             ; #11 Z/W-Axis scale factor (cts/unit)
M1194->L:$0005D2             ; #11 Axis offset (cts)
; Servo IC 2 Registers for 1st ACC-24 Channel 4 (usually for Motor #12)
M1201->X:$078219,0,24,S      ; ENC4 24-bit counter position
M1202->Y:$07821A,8,16,S      ; OUT4A command value; DAC or PWM
M1203->X:$07821B,0,24,S      ; ENC4 captured position
M1204->Y:$07821B,8,16,S      ; OUT4B command value; DAC or PWM
M1205->Y:$07821D,8,16,S      ; ADC4A input value
M1206->Y:$07821E,8,16,S      ; ADC4B input value
M1207->Y:$07821C,8,16,S      ; OUT4C command value; PFM or PWM
M1208->Y:$07821F,0,24,S      ; ENC4 compare A position
M1209->X:$07821F,0,24,S      ; ENC4 compare B position
M1210->X:$07821E,0,24,S      ; ENC4 compare autoincrement value
```

```
M1211->X:$07821D,11          ; ENC4 compare initial state write enable
M1212->X:$07821D,12          ; ENC4 compare initial state
M1214->X:$07821D,14          ; AENA4 output status
M1215->X:$078218,19          ; USER4 flag input status
M1216->X:$078218,9           ; ENC4 compare output value
M1217->X:$078218,11          ; ENC4 capture flag
M1218->X:$078218,8           ; ENC4 count error flag
M1219->X:$078218,14          ; HMFL4 flag input status
M1220->X:$078218,16          ; CHC4 input status
M1221->X:$078218,17          ; PLIM4 flag input status
M1222->X:$078218,18          ; MLIM4 flag input status
M1223->X:$078218,15          ; FAULT4 flag input status
M1224->X:$078218,20          ; Channel 4 W flag input status
M1225->X:$078218,21          ; Channel 4 V flag input status
M1226->X:$078218,22          ; Channel 4 U flag input status
M1227->X:$078218,23          ; Channel 4 T flag input status
M1228->X:$078218,20,4        ; Channel 4 TUVW inputs as 4-bit value
; Motor #12 Status Bits
M1230->Y:$000640,11,1        ; #12 Stopped-on-position-limit bit
M1231->X:$000630,21,1        ; #12 Positive-end-limit-set bit
M1232->X:$000630,22,1        ; #12 Negative-end-limit-set bit
M1233->X:$000630,13,1        ; #12 Desired-velocity-zero bit
M1235->X:$000630,15,1        ; #12 Dwell-in-progress bit
M1237->X:$000630,17,1        ; #12 Running-program bit
M1238->X:$000630,18,1        ; #12 Open-loop-mode bit
M1239->X:$000630,19,1        ; #12 Amplifier-enabled status bit
M1240->Y:$000640,0,1         ; #12 Background in-position bit
M1241->Y:$000640,1,1         ; #12 Warning-following error bit
M1242->Y:$000640,2,1         ; #12 Fatal-following-error bit
M1243->Y:$000640,3,1         ; #12 Amplifier-fault-error bit
M1244->Y:$000640,13,1        ; #12 Foreground in-position bit
M1245->Y:$000640,10,1        ; #12 Home-complete bit
M1246->Y:$000640,6,1         ; #12 Integrated following error fault bit
M1247->Y:$000640,5,1         ; #12 I2T fault bit
M1248->Y:$000640,8,1         ; #12 Phasing error fault bit
M1249->Y:$000640,9,1         ; #12 Phasing search-in-progress bit
; MACRO IC 1 Node 5 Flag Registers (usually used for Motor #12)
M1250->X:$003455,0,24        ; MACRO IC 1 Node 5 flag status register
M1251->Y:$003455,0,24        ; MACRO IC 1 Node 5 flag command register
M1253->X:$003455,20,4        ; MACRO IC 1 Node 5 TUVW flags
M1254->Y:$003455,14,1        ; MACRO IC 1 Node 5 amplifier enable flag
M1255->X:$003455,15,1        ; MACRO IC 1 Node 5 node/amplifier fault flag
M1256->X:$003455,16,1        ; MACRO IC 1 Node 5 home flag
M1257->X:$003455,17,1        ; MACRO IC 1 Node 5 positive limit flag
M1258->X:$003455,18,1        ; MACRO IC 1 Node 5 negative limit flag
M1259->X:$003455,19,1        ; MACRO IC 1 Node 5 user flag
```

```
; Motor #12 Move Registers
M1261->D:$000608              ; #12 Commanded position (1/[Ixx08*32] cts)
M1262->D:$00060B              ; #12 Actual position (1/[Ixx08*32] cts)
M1263->D:$000647              ; #12 Target (end) position (1/[Ixx08*32] cts)
M1264->D:$00064C              ; #12 Position bias (1/[Ixx08*32] cts)
M1266->X:$00061D,0,24,S       ; #12 Actual velocity (1/[Ixx09*32] cts/cyc)
M1267->D:$00060D              ; #12 Present master pos (1/[Ixx07*32] cts)
M1268->X:$00063F,8,16,S       ; #12 Filter Output (16-bit DAC bits)
M1269->D:$000610              ; #12 Compensation correction (1/[Ixx08*32] cts)
M1270->D:$000634              ; #12 Present phase position (including fraction)
M1271->X:$000634,24,S         ; #12 Present phase position (counts *Ixx70)
M1272->L:$000657              ; #12 Variable jog position/distance (cts)
M1273->Y:$00064E,0,24,S       ; #12 Encoder home capture position (cts)
M1274->D:$00066F              ; #12 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1275->X:$000639,8,16,S       ; #12 Actual quadrature current
M1276->Y:$000639,8,16,S       ; #12 Actual direct current
M1277->X:$00063C,8,16,S       ; #12 Quadrature current-loop integrator output
M1278->Y:$00063C,8,16,S       ; #12 Direct current-loop integrator output
M1279->X:$00062E,8,16,S       ; #12 PID internal filter result (16-bit DAC bits)
M1288->Y:$078219,0,12,U       ; IC 2 Ch 4 Compare A fractional count
M1289->Y:$078218,0,12,U       ; IC 2 Ch 4 Compare A fractional count
; Motor #12 Axis Definition Registers
M1291->L:$00064F              ; #12 X/U/A/B/C-Axis scale factor (cts/unit)
M1292->L:$000650              ; #12 Y/V-Axis scale factor (cts/unit)
M1293->L:$000651              ; #12 Z/W-Axis scale factor (cts/unit)
M1294->L:$000652              ; #12 Axis offset (cts)
; Servo IC 3 Registers for 1st ACC-24 Channel 5 (usually for Motor #13)
M1301->X:$078301,0,24,S       ; ENC5 24-bit counter position
M1302->Y:$078302,8,16,S       ; OUT5A command value; DAC or PWM
M1303->X:$078303,0,24,S       ; ENC5 captured position
M1304->Y:$078303,8,16,S       ; OUT5B command value; DAC or PWM
M1305->Y:$078305,8,16,S       ; ADC5A input value
M1306->Y:$078306,8,16,S       ; ADC5B input value
M1307->Y:$078304,8,16,S       ; OUT5C command value; PFM or PWM
M1308->Y:$078307,0,24,S       ; ENC5 compare A position
M1309->X:$078307,0,24,S       ; ENC5 compare B position
M1310->X:$078306,0,24,S       ; ENC5 compare autoincrement value
M1311->X:$078305,11           ; ENC5 compare initial state write enable
M1312->X:$078305,12           ; ENC5 compare initial state
M1314->X:$078305,14           ; AENA5 output status
M1315->X:$078300,19           ; USER5 flag input status
M1316->X:$078300,9            ; ENC5 compare output value
M1317->X:$078300,11           ; ENC5 capture flag
M1318->X:$078300,8            ; ENC5 count error flag
M1319->X:$078300,14           ; CHC5 input status
M1320->X:$078300,16           ; HMFL5 flag input status
M1321->X:$078300,17           ; PLIM5 flag input status
```

```
M1322->X:$078300,18          ; MLIM5 flag input status
M1323->X:$078300,15          ; FAULT5 flag input status
M1324->X:$078300,20          ; Channel 5 W flag input status
M1325->X:$078300,21          ; Channel 5 V flag input status
M1326->X:$078300,22          ; Channel 5 U flag input status
M1327->X:$078300,23          ; Channel 5 T flag input status
M1328->X:$078300,20,4        ; Channel 5 TUVW inputs as 4-bit value
; Motor #13 Status Bits
M1330->Y:$0006C0,11,1        ; #13 Stopped-on-position-limit bit
M1331->X:$0006B0,21,1        ; #13 Positive-end-limit-set bit
M1332->X:$0006B0,22,1        ; #13 Negative-end-limit-set bit
M1333->X:$0006B0,13,1        ; #13 Desired-velocity-zero bit
M1335->X:$0006B0,15,1        ; #13 Dwell-in-progress bit
M1337->X:$0006B0,17,1        ; #13 Running-program bit
M1338->X:$0006B0,18,1        ; #13 Open-loop-mode bit
M1339->X:$0006B0,19,1        ; #13 Amplifier-enabled status bit
M1340->Y:$0006C0,0,1         ; #13 Background in-position bit
M1341->Y:$0006C0,1,1         ; #13 Warning-following error bit
M1342->Y:$0006C0,2,1         ; #13 Fatal-following-error bit
M1343->Y:$0006C0,3,1         ; #13 Amplifier-fault-error bit
M1344->Y:$0006C0,13,1        ; #13 Foreground in-position bit
M1345->Y:$0006C0,10,1        ; #13 Home-complete bit
M1346->Y:$0006C0,6,1         ; #13 Integrated following error fault bit
M1347->Y:$0006C0,5,1         ; #13 I2T fault bit
M1348->Y:$0006C0,8,1         ; #13 Phasing error fault bit
M1349->Y:$0006C0,9,1         ; #13 Phasing search-in-progress bit
; MACRO IC 1 Node 8 Flag Registers (usually used for Motor #13)
M1350->X:$003458,0,24        ; MACRO IC 1 Node 8 flag status register
M1351->Y:$003458,0,24        ; MACRO IC 1 Node 8 flag command register
M1353->X:$003458,20,4        ; MACRO IC 1 Node 8 TUVW flags
M1354->Y:$003458,14,1        ; MACRO IC 1 Node 8 amplifier enable flag
M1355->X:$003458,15,1        ; MACRO IC 1 Node 8 node/amplifier fault flag
M1356->X:$003458,16,1        ; MACRO IC 1 Node 8 home flag
M1357->X:$003458,17,1        ; MACRO IC 1 Node 8 positive limit flag
M1358->X:$003458,18,1        ; MACRO IC 1 Node 8 negative limit flag
M1359->X:$003458,19,1        ; MACRO IC 1 Node 8 user flag
; Motor #13 Move Registers
M1361->D:$000688             ; #13 Commanded position (1/[Ixx08*32] cts)
M1362->D:$00068B             ; #13 Actual position (1/[Ixx08*32] cts)
M1363->D:$0006C7             ; #13 Target (end) position (1/[Ixx08*32] cts)
M1364->D:$0006CC             ; #13 Position bias (1/[Ixx08*32] cts)
M1366->X:$00069D,0,24,S      ; #13 Actual velocity (1/[Ixx09*32] cts/cyc)
M1367->D:$00068D             ; #13 Present master pos (1/[Ixx07*32] cts)
M1368->X:$0006BF,8,16,S      ; #13 Filter Output (16-bit DAC bits)
M1369->D:$000690             ; #13 Compensation correction (1/[Ixx08*32] cts)
M1370->D:$0006B4             ; #13 Present phase position (including fraction)
M1371->X:$0006B4,24,S        ; #13 Present phase position (counts *Ixx70)
```

```
M1372->L:$0006D7              ; #13 Variable jog position/distance (cts)
M1373->Y:$0006CE,0,24,S       ; #13 Encoder home capture position (cts)
M1374->D:$0006EF              ; #13 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1375->X:$0006B9,8,16,S       ; #13 Actual quadrature current
M1376->Y:$0006B9,8,16,S       ; #13 Actual direct current
M1377->X:$0006BC,8,16,S       ; #13 Quadrature current-loop integrator output
M1378->Y:$0006BC,8,16,S       ; #13 Direct current-loop integrator output
M1379->X:$0006AE,8,16,S       ; #13 PID internal filter result (16-bit DAC bits)
M1388->Y:$078301,0,12,U       ; IC 3 Ch 1 Compare A fractional count
M1389->Y:$078300,0,12,U       ; IC 3 Ch 1 Compare A fractional count
; Motor #13 Axis Definition Registers
M1391->L:$0006CF              ; #13 X/U/A/B/C-Axis scale factor (cts/unit)
M1392->L:$0006D0              ; #13 Y/V-Axis scale factor (cts/unit)
M1393->L:$0006D1              ; #13 Z/W-Axis scale factor (cts/unit)
M1394->L:$0006D2              ; #13 Axis offset (cts)
; Servo IC 3 Registers for 1st ACC-24 Channel 6 (usually for Motor #14)
M1401->X:$078309,0,24,S       ; ENC6 24-bit counter position
M1402->Y:$07830A,8,16,S       ; OUT6A command value; DAC or PWM
M1403->X:$07830B,0,24,S       ; ENC6 captured position
M1404->Y:$07830B,8,16,S       ; OUT6B command value; DAC or PWM
M1405->Y:$07830D,8,16,S       ; ADC6A input value
M1406->Y:$07830E,8,16,S       ; ADC6B input value
M1407->Y:$07830C,8,16,S       ; OUT6C command value; PFM or PWM
M1408->Y:$07830F,0,24,S       ; ENC6 compare A position
M1409->X:$07830F,0,24,S       ; ENC6 compare B position
M1410->X:$07830E,0,24,S       ; ENC6 compare autoincrement value
M1411->X:$07830D,11           ; ENC6 compare initial state write enable
M1412->X:$07830D,12           ; ENC6 compare initial state
M1414->X:$07830D,14           ; AENA6 output status
M1415->X:$078308,19           ; USER6 flag input status
M1416->X:$078308,9            ; ENC6 compare output value
M1417->X:$078308,11           ; ENC6 capture flag
M1418->X:$078308,8            ; ENC6 count error flag
M1419->X:$078308,14           ; CHC6 input status
M1420->X:$078308,16           ; HMFL6 flag input status
M1421->X:$078308,17           ; PLIM6 flag input status
M1422->X:$078308,18           ; MLIM6 flag input status
M1423->X:$078308,15           ; FAULT6 flag input status
M1424->X:$078308,20           ; Channel 6 W flag input status
M1425->X:$078308,21           ; Channel 6 V flag input status
M1426->X:$078308,22           ; Channel 6 U flag input status
M1427->X:$078308,23           ; Channel 6 T flag input status
M1428->X:$078308,20,4         ; Channel 6 TUVW inputs as 4-bit value
; Motor #14 Status Bits
M1430->Y:$000740,11,1         ; #14 Stopped-on-position-limit bit
M1431->X:$000730,21,1         ; #14 Positive-end-limit-set bit
M1432->X:$000730,22,1         ; #14 Negative-end-limit-set bit
```

```
M1433->X:$000730,13,1        ; #14 Desired-velocity-zero bit
M1435->X:$000730,15,1        ; #14 Dwell-in-progress bit
M1437->X:$000730,17,1        ; #14 Running-program bit
M1438->X:$000730,18,1        ; #14 Open-loop-mode bit
M1439->X:$000730,19,1        ; #14 Amplifier-enabled status bit
M1440->Y:$000740,0,1         ; #14 Background in-position bit
M1441->Y:$000740,1,1         ; #14 Warning-following error bit
M1442->Y:$000740,2,1         ; #14 Fatal-following-error bit
M1443->Y:$000740,3,1         ; #14 Amplifier-fault-error bit
M1444->Y:$000740,13,1        ; #14 Foreground in-position bit
M1445->Y:$000740,10,1        ; #14 Home-complete bit
M1446->Y:$000740,6,1         ; #14 Integrated following error fault bit
M1447->Y:$000740,5,1         ; #14 I2T fault bit
M1448->Y:$000740,8,1         ; #14 Phasing error fault bit
M1449->Y:$000740,9,1         ; #14 Phasing search-in-progress bit
; MACRO IC 1 Node 9 Flag Registers (usually used for Motor #14)
M1450->X:$003459,0,24        ; MACRO IC 1 Node 9 flag status register
M1451->Y:$003459,0,24        ; MACRO IC 1 Node 9 flag command register
M1453->X:$003459,20,4        ; MACRO IC 1 Node 9 TUVW flags
M1454->Y:$003459,14,1        ; MACRO IC 1 Node 9 amplifier enable flag
M1455->X:$003459,15,1        ; MACRO IC 1 Node 9 node/amplifier fault flag
M1456->X:$003459,16,1        ; MACRO IC 1 Node 9 home flag
M1457->X:$003459,17,1        ; MACRO IC 1 Node 9 positive limit flag
M1458->X:$003459,18,1        ; MACRO IC 1 Node 9 negative limit flag
M1459->X:$003459,19,1        ; MACRO IC 1 Node 9 user flag
; Motor #14 Move Registers
M1461->D:$000708             ; #14 Commanded position (1/[Ixx08*32] cts)
M1462->D:$00070B             ; #14 Actual position (1/[Ixx08*32] cts)
M1463->D:$000747             ; #14 Target (end) position (1/[Ixx08*32] cts)
M1464->D:$00074C             ; #14 Position bias (1/[Ixx08*32] cts)
M1466->X:$00071D,0,24,S      ; #14 Actual velocity (1/[Ixx09*32] cts/cyc)
M1467->D:$00070D             ; #14 Present master pos (1/[Ixx07*32] cts)
M1468->X:$00073F,8,16,S      ; #14 Filter Output (16-bit DAC bits)
M1469->D:$000710             ; #14 Compensation correction (1/[Ixx08*32] cts)
M1470->D:$000734             ; #14 Present phase position (including fraction)
M1471->X:$000734,24,S        ; #14 Present phase position (counts *Ixx70)
M1472->L:$000757             ; #14 Variable jog position/distance (cts)
M1473->Y:$00074E,0,24,S      ; #14 Encoder home capture position (cts)
M1474->D:$00076F             ; #14 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1475->X:$000739,8,16,S      ; #14 Actual quadrature current
M1476->Y:$000739,8,16,S      ; #14 Actual direct current
M1477->X:$00073C,8,16,S      ; #14 Quadrature current-loop integrator output
M1478->Y:$00073C,8,16,S      ; #14 Direct current-loop integrator output
M1479->X:$00072E,8,16,S      ; #14 PID internal filter result (16-bit DAC bits)
M1488->Y:$078309,0,12,U      ; IC 3 Ch 2 Compare A fractional count
M1489->Y:$078308,0,12,U      ; IC 3 Ch 2 Compare A fractional count
```

```
; Motor #14 Axis Definition Registers
M1491->L:$00074F              ; #14 X/U/A/B/C-Axis scale factor (cts/unit)
M1492->L:$000750              ; #14 Y/V-Axis scale factor (cts/unit)
M1493->L:$000751              ; #14 Z/W-Axis scale factor (cts/unit)
M1494->L:$000752              ; #14 Axis offset (cts)
; Servo IC 3 Registers for 1st ACC-24 Channel 7 (usually for Motor #15)
M1501->X:$078311,0,24,S       ; ENC7 24-bit counter position
M1502->Y:$078312,8,16,S       ; OUT7A command value; DAC or PWM
M1503->X:$078313,0,24,S       ; ENC7 captured position
M1504->Y:$078313,8,16,S       ; OUT7B command value; DAC or PWM
M1505->Y:$078315,8,16,S       ; ADC7A input value
M1506->Y:$078316,8,16,S       ; ADC7B input value
M1507->Y:$078314,8,16,S       ; OUT7C command value; PFM or PWM
M1508->Y:$078317,0,24,S       ; ENC7 compare A position
M1509->X:$078317,0,24,S       ; ENC7 compare B position
M1510->X:$078316,0,24,S       ; ENC7 compare autoincrement value
M1511->X:$078315,11           ; ENC7 compare initial state write enable
M1512->X:$078315,12           ; ENC7 compare initial state
M1514->X:$078315,14           ; AENA7 output status
M1515->X:$078310,19           ; CHC7 input status
M1516->X:$078310,9            ; ENC7 compare output value
M1517->X:$078310,11           ; ENC7 capture flag
M1518->X:$078310,8            ; ENC7 count error flag
M1519->X:$078310,14           ; CHC7 input status
M1520->X:$078310,16           ; HMFL7 flag input status
M1521->X:$078310,17           ; PLIM7 flag input status
M1522->X:$078310,18           ; MLIM7 flag input status
M1523->X:$078310,15           ; FAULT7 flag input status
M1524->X:$078310,20           ; Channel 7 W flag input status
M1525->X:$078310,21           ; Channel 7 V flag input status
M1526->X:$078310,22           ; Channel 7 U flag input status
M1527->X:$078310,23           ; Channel 7 T flag input status
M1528->X:$078310,20,4         ; Channel 7 TUVW inputs as 4-bit value
; Motor #15 Status Bits
M1530->Y:$0007C0,11,1         ; #15 Stopped-on-position-limit bit
M1531->X:$0007B0,21,1         ; #15 Positive-end-limit-set bit
M1532->X:$0007B0,22,1         ; #15 Negative-end-limit-set bit
M1533->X:$0007B0,13,1         ; #15 Desired-velocity-zero bit
M1535->X:$0007B0,15,1         ; #15 Dwell-in-progress bit
M1537->X:$0007B0,17,1         ; #15 Running-program bit
M1538->X:$0007B0,18,1         ; #15 Open-loop-mode bit
M1539->X:$0007B0,19,1         ; #15 Amplifier-enabled status bit
M1540->Y:$0007C0,0,1          ; #15 Background in-position bit
M1541->Y:$0007C0,1,1          ; #15 Warning-following error bit
M1542->Y:$0007C0,2,1          ; #15 Fatal-following-error bit
M1543->Y:$0007C0,3,1          ; #15 Amplifier-fault-error bit
M1544->Y:$0007C0,13,1         ; #15 Foreground in-position bit
```

```
M1545->Y:$0007C0,10,1        ; #15 Home-complete bit
M1546->Y:$0007C0,6,1         ; #15 Integrated following error fault bit
M1547->Y:$0007C0,5,1         ; #15 I2T fault bit
M1548->Y:$0007C0,8,1         ; #15 Phasing error fault bit
M1549->Y:$0007C0,9,1         ; #15 Phasing search-in-progress bit
; MACRO IC 1 Node 12 Flag Registers (usually used for Motor #15)
M1550->X:$00345C,0,24        ; MACRO IC 1 Node 12 flag status register
M1551->Y:$00345C,0,24        ; MACRO IC 1 Node 12 flag command register
M1553->X:$00345C,20,4        ; MACRO IC 1 Node 12 TUVW flags
M1554->Y:$00345C,14,1        ; MACRO IC 1 Node 12 amplifier enable flag
M1555->X:$00345C,15,1        ; MACRO IC 1 Node 12 node/amplifier fault flag
M1556->X:$00345C,16,1        ; MACRO IC 1 Node 12 home flag
M1557->X:$00345C,17,1        ; MACRO IC 1 Node 12 positive limit flag
M1558->X:$00345C,18,1        ; MACRO IC 1 Node 12 negative limit flag
M1559->X:$00345C,19,1        ; MACRO IC 1 Node 12 user flag
; Motor #15 Move Registers
M1561->D:$000788             ; #15 Commanded position (1/[Ixx08*32] cts)
M1562->D:$00078B             ; #15 Actual position (1/[Ixx08*32] cts)
M1563->D:$0007C7             ; #15 Target (end) position (1/[Ixx08*32] cts)
M1564->D:$0007CC             ; #15 Position bias (1/[Ixx08*32] cts)
M1566->X:$00079D,0,24,S      ; #15 Actual velocity (1/[Ixx09*32] cts/cyc)
M1567->D:$00078D             ; #15 Present master pos (1/[Ixx07*32] cts)
M1568->X:$0007BF,8,16,S      ; #15 Filter Output (16-bit DAC bits)
M1569->D:$000790             ; #15 Compensation correction (1/[Ixx08*32] cts)
M1570->D:$0007B4             ; #15 Present phase position (including fraction)
M1571->X:$0007B4,24,S        ; #15 Present phase position (counts *Ixx70)
M1572->L:$0007D7             ; #15 Variable jog position/distance (cts)
M1573->Y:$0007CE,0,24,S      ; #15 Encoder home capture position (cts)
M1574->D:$0007EF             ; #15 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1575->X:$0007B9,8,16,S      ; #15 Actual quadrature current
M1576->Y:$0007B9,8,16,S      ; #15 Actual direct current
M1577->X:$0007BC,8,16,S      ; #15 Quadrature current-loop integrator output
M1578->Y:$0007BC,8,16,S      ; #15 Direct current-loop integrator output
M1579->X:$0007AE,8,16,S      ; #15 PID internal filter result (16-bit DAC bits)
M1588->Y:$078311,0,12,U      ; IC 3 Ch 3 Compare A fractional count
M1589->Y:$078310,0,12,U      ; IC 3 Ch 3 Compare A fractional count
; Motor #15 Axis Definition Registers
M1591->L:$0007CF             ; #15 X/U/A/B/C-Axis scale factor (cts/unit)
M1592->L:$0007D0             ; #15 Y/V-Axis scale factor (cts/unit)
M1593->L:$0007D1             ; #15 Z/W-Axis scale factor (cts/unit)
M1594->L:$0007D2             ; #15 Axis offset (cts)
; Servo IC 3 Registers for 1st ACC-24 Channel 8 (usually for Motor #16)
M1601->X:$078319,0,24,S      ; ENC8 24-bit counter position
M1602->Y:$07831A,8,16,S      ; OUT8A command value; DAC or PWM
M1603->X:$07831B,0,24,S      ; ENC8 captured position
M1604->Y:$07831B,8,16,S      ; OUT8B command value; DAC or PWM
M1605->Y:$07831D,8,16,S      ; ADC8A input value
```

```
M1606->Y:$07831E,8,16,S      ; ADC8B input value
M1607->Y:$07831C,8,16,S      ; OUT8C command value; PFM or PWM
M1608->Y:$07831F,0,24,S      ; ENC8 compare A position
M1609->X:$07831F,0,24,S      ; ENC8 compare B position
M1610->X:$07831E,0,24,S      ; ENC8 compare autoincrement value
M1611->X:$07831D,11          ; ENC8 compare initial state write enable
M1612->X:$07831D,12          ; ENC8 compare initial state
M1614->X:$07831D,14          ; AENA8 output status
M1615->X:$078318,19          ; USER8 flag input status
M1616->X:$078318,9           ; ENC8 compare output value
M1617->X:$078318,11          ; ENC8 capture flag
M1618->X:$078318,8           ; ENC8 count error flag
M1619->X:$078318,14          ; CHC8 input status
M1620->X:$078318,16          ; HMFL8 flag input status
M1621->X:$078318,17          ; PLIM8 flag input status
M1622->X:$078318,18          ; MLIM8 flag input status
M1623->X:$078318,15          ; FAULT8 flag input status
M1624->X:$078318,20          ; Channel 8 W flag input status
M1625->X:$078318,21          ; Channel 8 V flag input status
M1626->X:$078318,22          ; Channel 8 U flag input status
M1627->X:$078318,23          ; Channel 8 T flag input status
M1628->X:$078318,20,4        ; Channel 8 TUVW inputs as 4-bit value
; Motor #16 Status Bits
M1630->Y:$000840,11,1        ; #16 Stopped-on-position-limit bit
M1631->X:$000830,21,1        ; #16 Positive-end-limit-set bit
M1632->X:$000830,22,1        ; #16 Negative-end-limit-set bit
M1633->X:$000830,13,1        ; #16 Desired-velocity-zero bit
M1635->X:$000830,15,1        ; #16 Dwell-in-progress bit
M1637->X:$000830,17,1        ; #16 Running-program bit
M1638->X:$000830,18,1        ; #16 Open-loop-mode bit
M1639->X:$000830,19,1        ; #16 Amplifier-enabled status bit
M1640->Y:$000840,0,1         ; #16 Background in-position bit
M1641->Y:$000840,1,1         ; #16 Warning-following error bit
M1642->Y:$000840,2,1         ; #16 Fatal-following-error bit
M1643->Y:$000840,3,1         ; #16 Amplifier-fault-error bit
M1644->Y:$000840,13,1        ; #16 Foreground in-position bit
M1645->Y:$000840,10,1        ; #16 Home-complete bit
M1646->Y:$000840,6,1         ; #16 Integrated following error fault bit
M1647->Y:$000840,5,1         ; #16 I2T fault bit
M1648->Y:$000840,8,1         ; #16 Phasing error fault bit
; MACRO IC 1 Node 13 Flag Registers (usually used for Motor #16)
M1650->X:$00345D,0,24        ; MACRO IC 1 Node 13 flag status register
M1651->Y:$00345D,0,24        ; MACRO IC 1 Node 13 flag command register
M1653->X:$00345D,20,4        ; MACRO IC 1 Node 13 TUVW flags
M1654->Y:$00345D,14,1        ; MACRO IC 1 Node 13 amplifier enable flag
M1655->X:$00345D,15,1        ; MACRO IC 1 Node 13 node/amplifier fault flag
M1656->X:$00345D,16,1        ; MACRO IC 1 Node 13 home flag
```

```
M1657->X:$00345D,17,1        ; MACRO IC 1 Node 13 positive limit flag
M1658->X:$00345D,18,1        ; MACRO IC 1 Node 13 negative limit flag
M1659->X:$00345D,19,1        ; MACRO IC 1 Node 13 user flag
; Motor #16 Move Registers
M1661->D:$000808             ; #16 Commanded position (1/[Ixx08*32] cts)
M1662->D:$00080B             ; #16 Actual position (1/[Ixx08*32] cts)
M1663->D:$000847             ; #16 Target (end) position (1/[Ixx08*32] cts)
M1664->D:$00084C             ; #16 Position bias (1/[Ixx08*32] cts)
M1666->X:$00081D,0,24,S      ; #16 Actual velocity (1/[Ixx09*32] cts/cyc)
M1667->D:$00080D             ; #16 Present master pos (1/[Ixx07*32] cts)
M1668->X:$00083F,8,16,S      ; #16 Filter Output (16-bit DAC bits)
M1669->D:$000810             ; #16 Compensation correction (1/[Ixx08*32] cts)
M1670->D:$000834             ; #16 Present phase position (including fraction)
M1671->X:$000834,24,S        ; #16 Present phase position (counts *Ixx70)
M1672->L:$000857             ; #16 Variable jog position/distance (cts)
M1673->Y:$00084E,0,24,S      ; #16 Encoder home capture position (cts)
M1674->D:$00086F             ; #16 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1675->X:$000839,8,16,S      ; #16 Actual quadrature current
M1676->Y:$000839,8,16,S      ; #16 Actual direct current
M1677->X:$00083C,8,16,S      ; #16 Quadrature current-loop integrator output
M1678->Y:$00083C,8,16,S      ; #16 Direct current-loop integrator output
M1679->X:$00082E,8,16,S      ; #16 PID internal filter result (16-bit DAC bits)
M1688->Y:$078319,0,12,U      ; IC 3 Ch 4 Compare A fractional count
M1689->Y:$078318,0,12,U      ; IC 3 Ch 4 Compare A fractional count
; Motor #16 Axis Definition Registers
M1691->L:$00084F             ; #16 X/U/A/B/C-Axis scale factor (cts/unit)
M1692->L:$000850             ; #16 Y/V-Axis scale factor (cts/unit)
M1693->L:$000851             ; #16 Z/W-Axis scale factor (cts/unit)
M1694->L:$000852             ; #16 Axis offset (cts)
; Servo IC 4 Registers for 2nd ACC-24 Channel 1 (usually for Motor #17)
M1701->X:$079201,0,24,S      ; ENC1 24-bit counter position
M1702->Y:$079202,8,16,S      ; OUT1A command value; DAC or PWM
M1703->X:$079203,0,24,S      ; ENC1 captured position
M1704->Y:$079203,8,16,S      ; OUT1B command value; DAC or PWM
M1705->Y:$079205,8,16,S      ; ADC1A input value
M1706->Y:$079206,8,16,S      ; ADC1B input value
M1707->Y:$079204,8,16,S      ; OUT1C command value; PFM or PWM
M1708->Y:$079207,0,24,S      ; ENC1 compare A position
M1709->X:$079207,0,24,S      ; ENC1 compare B position
M1710->X:$079206,0,24,S      ; ENC1 compare autoincrement value
M1711->X:$079205,11          ; ENC1 compare initial state write enable
M1712->X:$079205,12          ; ENC1 compare initial state
M1714->X:$079205,14          ; AENA1 output status
M1715->X:$079200,19          ; USER1 flag input status
M1716->X:$079200,9           ; ENC1 compare output value
M1717->X:$079200,11          ; ENC1 capture flag
M1718->X:$079200,8           ; ENC1 count error flag
```

```
M1719->X:$079200,14          ; CHC1 input status
M1720->X:$079200,16          ; HMFL1 flag input status
M1721->X:$079200,17          ; PLIM1 flag input status
M1722->X:$079200,18          ; MLIM1 flag input status
M1723->X:$079200,15          ; FAULT1 flag input status
M1724->X:$079200,20          ; Channel 1 W flag input status
M1725->X:$079200,21          ; Channel 1 V flag input status
M1726->X:$079200,22          ; Channel 1 U flag input status
M1727->X:$079200,23          ; Channel 1 T flag input status
M1728->X:$079200,20,4        ; Channel 1 TUVW inputs as 4-bit value
; Motor #17 Status Bits
M1730->Y:$0008C0,11,1        ; #17 Stopped-on-position-limit bit
M1731->X:$0008B0,21,1        ; #17 Positive-end-limit-set bit
M1732->X:$0008B0,22,1        ; #17 Negative-end-limit-set bit
M1733->X:$0008B0,13,1        ; #17 Desired-velocity-zero bit
M1735->X:$0008B0,15,1        ; #17 Dwell-in-progress bit
M1737->X:$0008B0,17,1        ; #17 Running-program bit
M1738->X:$0008B0,18,1        ; #17 Open-loop-mode bit
M1739->X:$0008B0,19,1        ; #17 Amplifier-enabled status bit
M1740->Y:$0008C0,0,1         ; #17 Background in-position bit
M1741->Y:$0008C0,1,1         ; #17 Warning-following error bit
M1742->Y:$0008C0,2,1         ; #17 Fatal-following-error bit
M1743->Y:$0008C0,3,1         ; #17 Amplifier-fault-error bit
M1744->Y:$0008C0,13,1        ; #17 Foreground in-position bit
M1745->Y:$0008C0,10,1        ; #17 Home-complete bit
M1746->Y:$0008C0,6,1         ; #17 Integrated following error fault bit
M1747->Y:$0008C0,5,1         ; #17 I2T fault bit
M1748->Y:$0008C0,8,1         ; #17 Phasing error fault bit
M1749->Y:$0008C0,9,1         ; #17 Phasing search-in-progress bit
; MACRO IC 2 Node 0 Flag Registers (usually used for Motor #17)
M1750->X:$003460,0,24        ; MACRO IC 2 Node 0 flag status register
M1751->Y:$003460,0,24        ; MACRO IC 2 Node 0 flag command register
M1753->X:$003460,20,4        ; MACRO IC 2 Node 0 TUVW flags
M1754->Y:$003460,14,1        ; MACRO IC 2 Node 0 amplifier enable flag
M1755->X:$003460,15,1        ; MACRO IC 2 Node 0 node/amplifier fault flag
M1756->X:$003460,16,1        ; MACRO IC 2 Node 0 home flag
M1757->X:$003460,17,1        ; MACRO IC 2 Node 0 positive limit flag
M1758->X:$003460,18,1        ; MACRO IC 2 Node 0 negative limit flag
M1759->X:$003460,19,1        ; MACRO IC 2 Node 0 user flag
; Motor #17 Move Registers
M1761->D:$000888             ; #17 Commanded position (1/[Ixx08*32] cts)
M1762->D:$00088B             ; #17 Actual position (1/[Ixx08*32] cts)
M1763->D:$0008C7             ; #17 Target (end) position (1/[Ixx08*32] cts)
M1764->D:$0008CC             ; #17 Position bias (1/[Ixx08*32] cts)
M1766->X:$00089D,0,24,S      ; #17 Actual velocity (1/[Ixx09*32] cts/cyc)
M1767->D:$00088D             ; #17 Present master pos (1/[Ixx07*32] cts)
M1768->X:$0008BF,8,16,S      ; #17 Filter Output (16-bit DAC bits)
```

```
M1769->D:$000890            ; #17 Compensation correction (1/[Ixx08*32] cts)
M1770->D:$0008B4            ; #17 Present phase position (including fraction)
M1771->X:$0008B4,24,S       ; #17 Present phase position (counts *Ixx70)
M1772->L:$0008D7            ; #17 Variable jog position/distance (cts)
M1773->Y:$0008CE,0,24,S     ; #17 Encoder home capture position (cts)
M1774->D:$0008EF            ; #17 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1775->X:$0008B9,8,16,S     ; #17 Actual quadrature current
M1776->Y:$0008B9,8,16,S     ; #17 Actual direct current
M1777->X:$0008BC,8,16,S     ; #17 Quadrature current-loop integrator output
M1778->Y:$0008BC,8,16,S     ; #17 Direct current-loop integrator output
M1779->X:$0008AE,8,16,S     ; #17 PID internal filter result (16-bit DAC bits)
M1788->Y:$079201,0,12,U     ; IC 4 Ch 1 Compare A fractional count
M1789->Y:$079200,0,12,U     ; IC 4 Ch 1 Compare A fractional count
; Motor #17 Axis Definition Registers
M1791->L:$0008CF            ; #17 X/U/A/B/C-Axis scale factor (cts/unit)
M1792->L:$0008D0            ; #17 Y/V-Axis scale factor (cts/unit)
M1793->L:$0008D1            ; #17 Z/W-Axis scale factor (cts/unit)
M1794->L:$0008D2            ; #17 Axis offset (cts)
; Servo IC 4 Registers for 2nd ACC-24 Channel 2 (usually for Motor #18)
M1801->X:$079209,0,24,S     ; ENC2 24-bit counter position
M1802->Y:$07920A,8,16,S     ; OUT2A command value; DAC or PWM
M1803->X:$07920B,0,24,S     ; ENC2 captured position
M1804->Y:$07920B,8,16,S     ; OUT2B command value; DAC or PWM
M1805->Y:$07920D,8,16,S     ; ADC2A input value
M1806->Y:$07920E,8,16,S     ; ADC2B input value
M1807->Y:$07920C,8,16,S     ; OUT2C command value; PFM or PWM
M1808->Y:$07920F,0,24,S     ; ENC2 compare A position
M1809->X:$07920F,0,24,S     ; ENC2 compare B position
M1810->X:$07920E,0,24,S     ; ENC2 compare autoincrement value
M1811->X:$07920D,11         ; ENC2 compare initial state write enable
M1812->X:$07920D,12         ; ENC2 compare initial state
M1814->X:$07920D,14         ; AENA2 output status
M1815->X:$079208,19         ; USER2 flag input status
M1816->X:$079208,9          ; ENC2 compare output value
M1817->X:$079208,11         ; ENC2 capture flag
M1818->X:$079208,8          ; ENC2 count error flag
M1819->X:$079208,14         ; CHC2 input status
M1820->X:$079208,16         ; HMFL2 flag input status
M1821->X:$079208,17         ; PLIM2 flag input status
M1822->X:$079208,18         ; MLIM2 flag input status
M1823->X:$079208,15         ; FAULT2 flag input status
M1824->X:$079208,20         ; Channel 2 W flag input status
M1825->X:$079208,21         ; Channel 2 V flag input status
M1826->X:$079208,22         ; Channel 2 U flag input status
M1827->X:$079208,23         ; Channel 2 T flag input status
M1828->X:$079208,20,4       ; Channel 2 TUVW inputs as 4-bit value
```

```
; Motor #18 Status Bits
M1830->Y:$000940,11,1        ; #18 Stopped-on-position-limit bit
M1831->X:$000930,21,1        ; #18 Positive-end-limit-set bit
M1832->X:$000930,22,1        ; #18 Negative-end-limit-set bit
M1833->X:$000930,13,1        ; #18 Desired-velocity-zero bit
M1835->X:$000930,15,1        ; #18 Dwell-in-progress bit
M1837->X:$000930,17,1        ; #18 Running-program bit
M1838->X:$000930,18,1        ; #18 Open-loop-mode bit
M1839->X:$000930,19,1        ; #18 Amplifier-enabled status bit
M1840->Y:$000940,0,1         ; #18 Background in-position bit
M1841->Y:$000940,1,1         ; #18 Warning-following error bit
M1842->Y:$000940,2,1         ; #18 Fatal-following-error bit
M1843->Y:$000940,3,1         ; #18 Amplifier-fault-error bit
M1844->Y:$000940,13,1        ; #18 Foreground in-position bit
M1845->Y:$000940,10,1        ; #18 Home-complete bit
M1846->Y:$000940,6,1         ; #18 Integrated following error fault bit
M1847->Y:$000940,5,1         ; #18 I2T fault bit
M1848->Y:$000940,8,1         ; #18 Phasing error fault bit
M1849->Y:$000940,9,1         ; #18 Phasing search-in-progress bit
; MACRO IC 2 Node 1 Flag Registers (usually used for Motor #18)
M1850->X:$003461,0,24        ; MACRO IC 2 Node 1 flag status register
M1851->Y:$003461,0,24        ; MACRO IC 2 Node 1 flag command register
M1853->X:$003461,20,4        ; MACRO IC 2 Node 1 TUVW flags
M1854->Y:$003461,14,1        ; MACRO IC 2 Node 1 amplifier enable flag
M1855->X:$003461,15,1        ; MACRO IC 2 Node 1 node/amplifier fault flag
M1856->X:$003461,16,1        ; MACRO IC 2 Node 1 home flag
M1857->X:$003461,17,1        ; MACRO IC 2 Node 1 positive limit flag
M1858->X:$003461,18,1        ; MACRO IC 2 Node 1 negative limit flag
M1859->X:$003461,19,1        ; MACRO IC 2 Node 1 user flag
; Motor #18 Move Registers
M1861->D:$000908             ; #18 Commanded position (1/[Ixx08*32] cts)
M1862->D:$00090B             ; #18 Actual position (1/[Ixx08*32] cts)
M1863->D:$000947             ; #18 Target (end) position (1/[Ixx08*32] cts)
M1864->D:$00094C             ; #18 Position bias (1/[Ixx08*32] cts)
M1866->X:$00091D,0,24,S      ; #18 Actual velocity (1/[Ixx09*32] cts/cyc)
M1867->D:$00090D             ; #18 Present master pos (1/[Ixx07*32] cts)
M1868->X:$00093F,8,16,S      ; #18 Filter Output (16-bit DAC bits)
M1869->D:$000910             ; #18 Compensation correction (1/[Ixx08*32] cts)
M1870->D:$000934             ; #18 Present phase position (including fraction)
M1871->X:$000934,24,S        ; #18 Present phase position (counts *Ixx70)
M1872->L:$000957             ; #18 Variable jog position/distance (cts)
M1873->Y:$00094E,0,24,S      ; #18 Encoder home capture position (cts)
M1874->D:$00096F             ; #18 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1875->X:$000939,8,16,S      ; #18 Actual quadrature current
M1876->Y:$000939,8,16,S      ; #18 Actual direct current
M1877->X:$00093C,8,16,S      ; #18 Quadrature current-loop integrator output
M1878->Y:$00093C,8,16,S      ; #18 Direct current-loop integrator output
```

```
M1879->X:$00092E,8,16,S      ; #18 PID internal filter result (16-bit DAC bits)
M1888->Y:$079209,0,12,U      ; IC 4 Ch 2 Compare A fractional count
M1889->Y:$079208,0,12,U      ; IC 4 Ch 2 Compare A fractional count
; Motor #18 Axis Definition Registers
M1891->L:$00094F             ; #18 X/U/A/B/C-Axis scale factor (cts/unit)
M1892->L:$000950             ; #18 Y/V-Axis scale factor (cts/unit)
M1893->L:$000951             ; #18 Z/W-Axis scale factor (cts/unit)
M1894->L:$000952             ; #18 Axis offset (cts)
; Servo IC 4 Registers for 2nd ACC-24 Channel 3 (usually for Motor #19)
M1901->X:$079211,0,24,S      ; ENC3 24-bit counter position
M1902->Y:$079212,8,16,S      ; OUT3A command value; DAC or PWM
M1903->X:$079213,0,24,S      ; ENC3 captured position
M1904->Y:$079213,8,16,S      ; OUT3B command value; DAC or PWM
M1905->Y:$079215,8,16,S      ; ADC3A input value
M1906->Y:$079216,8,16,S      ; ADC3B input value
M1907->Y:$079214,8,16,S      ; OUT3C command value; PFM or PWM
M1908->Y:$079217,0,24,S      ; ENC3 compare A position
M1909->X:$079217,0,24,S      ; ENC3 compare B position
M1910->X:$079216,0,24,S      ; ENC3 compare autoincrement value
M1911->X:$079215,11          ; ENC3 compare initial state write enable
M1912->X:$079215,12          ; ENC3 compare initial state
M1914->X:$079215,14          ; AENA3 output status
M1915->X:$079210,19          ; USER3 flag input status
M1916->X:$079210,9           ; ENC3 compare output value
M1917->X:$079210,11          ; ENC3 capture flag
M1918->X:$079210,8           ; ENC3 count error flag
M1919->X:$079210,14          ; CHC3 input status
M1920->X:$079210,16          ; HMFL3 flag input status
M1921->X:$079210,17          ; PLIM3 flag input status
M1922->X:$079210,18          ; MLIM3 flag input status
M1923->X:$079210,15          ; FAULT3 flag input status
M1924->X:$079210,20          ; Channel 3 W flag input status
M1925->X:$079210,21          ; Channel 3 V flag input status
M1926->X:$079210,22          ; Channel 3 U flag input status
M1927->X:$079210,23          ; Channel 3 T flag input status
M1928->X:$079210,20,4        ; Channel 3 TUVW inputs as 4-bit value
; Motor #19 Status Bits
M1930->Y:$0009C0,11,1        ; #19 Stopped-on-position-limit bit
M1931->X:$0009B0,21,1        ; #19 Positive-end-limit-set bit
M1932->X:$0009B0,22,1        ; #19 Negative-end-limit-set bit
M1933->X:$0009B0,13,1        ; #19 Desired-velocity-zero bit
M1935->X:$0009B0,15,1        ; #19 Dwell-in-progress bit
M1937->X:$0009B0,17,1        ; #19 Running-program bit
M1938->X:$0009B0,18,1        ; #19 Open-loop-mode bit
M1939->X:$0009B0,19,1        ; #19 Amplifier-enabled status bit
M1940->Y:$0009C0,0,1         ; #19 Background in-position bit
M1941->Y:$0009C0,1,1         ; #19 Warning-following error bit
```

```
M1942->Y:$0009C0,2,1          ; #19 Fatal-following-error bit
M1943->Y:$0009C0,3,1          ; #19 Amplifier-fault-error bit
M1944->Y:$0009C0,13,1         ; #19 Foreground in-position bit
M1945->Y:$0009C0,10,1         ; #19 Home-complete bit
M1946->Y:$0009C0,6,1          ; #19 Integrated following error fault bit
M1947->Y:$0009C0,5,1          ; #19 I2T fault bit
M1948->Y:$0009C0,8,1          ; #19 Phasing error fault bit
M1949->Y:$0009C0,9,1          ; #19 Phasing search-in-progress bit
; MACRO IC 2 Node 4 Flag Registers (usually used for Motor #19)
M1950->X:$003464,0,24         ; MACRO IC 2 Node 4 flag status register
M1951->Y:$003464,0,24         ; MACRO IC 2 Node 4 flag command register
M1953->X:$003464,20,4         ; MACRO IC 2 Node 4 TUVW flags
M1954->Y:$003464,14,1         ; MACRO IC 2 Node 4 amplifier enable flag
M1955->X:$003464,15,1         ; MACRO IC 2 Node 4 node/amplifier fault flag
M1956->X:$003464,16,1         ; MACRO IC 2 Node 4 home flag
M1957->X:$003464,17,1         ; MACRO IC 2 Node 4 positive limit flag
M1958->X:$003464,18,1         ; MACRO IC 2 Node 4 negative limit flag
M1959->X:$003464,19,1         ; MACRO IC 2 Node 4 user flag
; Motor #19 Move Registers
M1961->D:$000988              ; #19 Commanded position (1/[Ixx08*32] cts)
M1962->D:$00098B              ; #19 Actual position (1/[Ixx08*32] cts)
M1963->D:$0009C7              ; #19 Target (end) position (1/[Ixx08*32] cts)
M1964->D:$0009CC              ; #19 Position bias (1/[Ixx08*32] cts)
M1966->X:$00099D,0,24,S       ; #19 Actual velocity (1/[Ixx09*32] cts/cyc)
M1967->D:$00098D              ; #19 Present master pos (1/[Ixx07*32] cts)
M1968->X:$0009BF,8,16,S       ; #19 Filter Output (16-bit DAC bits)
M1969->D:$000990              ; #19 Compensation correction (1/[Ixx08*32] cts)
M1970->D:$0009B4              ; #19 Present phase position (including fraction)
M1971->X:$0009B4,24,S         ; #19 Present phase position (counts *Ixx70)
M1972->L:$0009D7              ; #19 Variable jog position/distance (cts)
M1973->Y:$0009CE,0,24,S       ; #19 Encoder home capture position (cts)
M1974->D:$0009EF              ; #19 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1975->X:$0009B9,8,16,S       ; #19 Actual quadrature current
M1976->Y:$0009B9,8,16,S       ; #19 Actual direct current
M1977->X:$0009BC,8,16,S       ; #19 Quadrature current-loop integrator output
M1978->Y:$0009BC,8,16,S       ; #19 Direct current-loop integrator output
M1979->X:$0009AE,8,16,S       ; #19 PID internal filter result (16-bit DAC bits)
M1988->Y:$079211,0,12,U       ; IC 4 Ch 3 Compare A fractional count
M1989->Y:$079210,0,12,U       ; IC 4 Ch 3 Compare A fractional count
; Motor #19 Axis Definition Registers
M1991->L:$0009CF              ; #19 X/U/A/B/C-Axis scale factor (cts/unit)
M1992->L:$0009D0              ; #19 Y/V-Axis scale factor (cts/unit)
M1993->L:$0009D1              ; #19 Z/W-Axis scale factor (cts/unit)
M1994->L:$0009D2              ; #19 Axis offset (cts)
```

```
; Servo IC 4 Registers for 2nd ACC-24 Channel 4 (usually for Motor #20)
M2001->X:$079219,0,24,S      ; ENC4 24-bit counter position
M2002->Y:$07921A,8,16,S      ; OUT4A command value; DAC or PWM
M2003->X:$07921B,0,24,S      ; ENC4 captured position
M2004->Y:$07921B,8,16,S      ; OUT4B command value; DAC or PWM
M2005->Y:$07921D,8,16,S      ; ADC4A input value
M2006->Y:$07921E,8,16,S      ; ADC4B input value
M2007->Y:$07921C,8,16,S      ; OUT4C command value; PFM or PWM
M2008->Y:$07921F,0,24,S      ; ENC4 compare A position
M2009->X:$07921F,0,24,S      ; ENC4 compare B position
M2010->X:$07921E,0,24,S      ; ENC4 compare autoincrement value
M2011->X:$07921D,11          ; ENC4 compare initial state write enable
M2012->X:$07921D,12          ; ENC4 compare initial state
M2014->X:$07921D,14          ; AENA4 output status
M2015->X:$079218,19          ; USER4 flag input status
M2016->X:$079218,9           ; ENC4 compare output value
M2017->X:$079218,11          ; ENC4 capture flag
M2018->X:$079218,8           ; ENC4 count error flag
M2019->X:$079218,14          ; HMFL4 flag input status
M2020->X:$079218,16          ; CHC4 input status
M2021->X:$079218,17          ; PLIM4 flag input status
M2022->X:$079218,18          ; MLIM4 flag input status
M2023->X:$079218,15          ; FAULT4 flag input status
M2024->X:$079218,20          ; Channel 4 W flag input status
M2025->X:$079218,21          ; Channel 4 V flag input status
M2026->X:$079218,22          ; Channel 4 U flag input status
M2027->X:$079218,23          ; Channel 4 T flag input status
M2028->X:$079218,20,4        ; Channel 4 TUVW inputs as 4-bit value
; Motor #20 Status Bits
M2030->Y:$000A40,11,1        ; #20 Stopped-on-position-limit bit
M2031->X:$000A30,21,1        ; #20 Positive-end-limit-set bit
M2032->X:$000A30,22,1        ; #20 Negative-end-limit-set bit
M2033->X:$000A30,13,1        ; #20 Desired-velocity-zero bit
M2035->X:$000A30,15,1        ; #20 Dwell-in-progress bit
M2037->X:$000A30,17,1        ; #20 Running-program bit
M2038->X:$000A30,18,1        ; #20 Open-loop-mode bit
M2039->X:$000A30,19,1        ; #20 Amplifier-enabled status bit
M2040->Y:$000A40,0,1         ; #20 Background in-position bit
M2041->Y:$000A40,1,1         ; #20 Warning-following error bit
M2042->Y:$000A40,2,1         ; #20 Fatal-following-error bit
M2043->Y:$000A40,3,1         ; #20 Amplifier-fault-error bit
M2044->Y:$000A40,13,1        ; #20 Foreground in-position bit
M2045->Y:$000A40,10,1        ; #20 Home-complete bit
M2046->Y:$000A40,6,1         ; #20 Integrated following error fault bit
M2047->Y:$000A40,5,1         ; #20 I2T fault bit
M2048->Y:$000A40,8,1         ; #20 Phasing error fault bit
M2049->Y:$000A40,9,1         ; #20 Phasing search-in-progress bit
```

```
; MACRO IC 2 Node 5 Flag Registers (usually used for Motor #20)
M2050->X:$003465,0,24        ; MACRO IC 2 Node 5 flag status register
M2051->Y:$003465,0,24        ; MACRO IC 2 Node 5 flag command register
M2053->X:$003465,20,4        ; MACRO IC 2 Node 5 TUVW flags
M2054->Y:$003465,14,1        ; MACRO IC 2 Node 5 amplifier enable flag
M2055->X:$003465,15,1        ; MACRO IC 2 Node 5 node/amplifier fault flag
M2056->X:$003465,16,1        ; MACRO IC 2 Node 5 home flag
M2057->X:$003465,17,1        ; MACRO IC 2 Node 5 positive limit flag
M2058->X:$003465,18,1        ; MACRO IC 2 Node 5 negative limit flag
M2059->X:$003465,19,1        ; MACRO IC 2 Node 5 user flag
; Motor #20 Move Registers
M2061->D:$000A08             ; #20 Commanded position (1/[Ixx08*32] cts)
M2062->D:$000A0B             ; #20 Actual position (1/[Ixx08*32] cts)
M2063->D:$000A47             ; #20 Target (end) position (1/[Ixx08*32] cts)
M2064->D:$000A4C             ; #20 Position bias (1/[Ixx08*32] cts)
M2066->X:$000A1D,0,24,S      ; #20 Actual velocity (1/[Ixx09*32] cts/cyc)
M2067->D:$000A0D             ; #20 Present master pos (1/[Ixx07*32] cts)
M2068->X:$000A3F,8,16,S      ; #20 Filter Output (16-bit DAC bits)
M2069->D:$000A10             ; #20 Compensation correction (1/[Ixx08*32] cts)
M2070->D:$000A34             ; #20 Present phase position (including fraction)
M2071->X:$000A34,24,S        ; #20 Present phase position (counts *Ixx70)
M2072->L:$000A57             ; #20 Variable jog position/distance (cts)
M2073->Y:$000A4E,0,24,S      ; #20 Encoder home capture position (cts)
M2074->D:$000A6F             ; #20 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2075->X:$000A39,8,16,S      ; #20 Actual quadrature current
M2076->Y:$000A39,8,16,S      ; #20 Actual direct current
M2077->X:$000A3C,8,16,S      ; #20 Quadrature current-loop integrator output
M2078->Y:$000A3C,8,16,S      ; #20 Direct current-loop integrator output
M2079->X:$000A2E,8,16,S      ; #20 PID internal filter result (16-bit DAC bits)
M2088->Y:$079219,0,12,U      ; IC 4 Ch 4 Compare A fractional count
M2089->Y:$079218,0,12,U      ; IC 4 Ch 4 Compare A fractional count
; Motor #20 Axis Definition Registers
M2091->L:$000A4F             ; #20 X/U/A/B/C-Axis scale factor (cts/unit)
M2092->L:$000A50             ; #20 Y/V-Axis scale factor (cts/unit)
M2093->L:$000A51             ; #20 Z/W-Axis scale factor (cts/unit)
M2094->L:$000A52             ; #20 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 5 (usually for Motor #21)
M2101->X:$079301,0,24,S      ; ENC5 24-bit counter position
M2102->Y:$079302,8,16,S      ; OUT5A command value; DAC or PWM
M2103->X:$079303,0,24,S      ; ENC5 captured position
M2104->Y:$079303,8,16,S      ; OUT5B command value; DAC or PWM
M2105->Y:$079305,8,16,S      ; ADC5A input value
M2106->Y:$079306,8,16,S      ; ADC5B input value
M2107->Y:$079304,8,16,S      ; OUT5C command value; PFM or PWM
M2108->Y:$079307,0,24,S      ; ENC5 compare A position
M2109->X:$079307,0,24,S      ; ENC5 compare B position
M2110->X:$079306,0,24,S      ; ENC5 compare autoincrement value
```

```
M2111->X:$079305,11          ; ENC5 compare initial state write enable
M2112->X:$079305,12          ; ENC5 compare initial state
M2114->X:$079305,14          ; AENA5 output status
M2115->X:$079300,19          ; USER5 flag input status
M2116->X:$079300,9           ; ENC5 compare output value
M2117->X:$079300,11          ; ENC5 capture flag
M2118->X:$079300,8           ; ENC5 count error flag
M2119->X:$079300,14          ; CHC5 input status
M2120->X:$079300,16          ; HMFL5 flag input status
M2121->X:$079300,17          ; PLIM5 flag input status
M2122->X:$079300,18          ; MLIM5 flag input status
M2123->X:$079300,15          ; FAULT5 flag input status
M2124->X:$079300,20          ; Channel 5 W flag input status
M2125->X:$079300,21          ; Channel 5 V flag input status
M2126->X:$079300,22          ; Channel 5 U flag input status
M2127->X:$079300,23          ; Channel 5 T flag input status
M2128->X:$079300,20,4        ; Channel 5 TUVW inputs as 4-bit value
; Motor #21 Status Bits
M2130->Y:$000AC0,11,1        ; #21 Stopped-on-position-limit bit
M2131->X:$000AB0,21,1        ; #21 Positive-end-limit-set bit
M2132->X:$000AB0,22,1        ; #21 Negative-end-limit-set bit
M2133->X:$000AB0,13,1        ; #21 Desired-velocity-zero bit
M2135->X:$000AB0,15,1        ; #21 Dwell-in-progress bit
M2137->X:$000AB0,17,1        ; #21 Running-program bit
M2138->X:$000AB0,18,1        ; #21 Open-loop-mode bit
M2139->X:$000AB0,19,1        ; #21 Amplifier-enabled status bit
M2140->Y:$000AC0,0,1         ; #21 Background in-position bit
M2141->Y:$000AC0,1,1         ; #21 Warning-following error bit
M2142->Y:$000AC0,2,1         ; #21 Fatal-following-error bit
M2143->Y:$000AC0,3,1         ; #21 Amplifier-fault-error bit
M2144->Y:$000AC0,13,1        ; #21 Foreground in-position bit
M2145->Y:$000AC0,10,1        ; #21 Home-complete bit
M2146->Y:$000AC0,6,1         ; #21 Integrated following error fault bit
M2147->Y:$000AC0,5,1         ; #21 I2T fault bit
M2148->Y:$000AC0,8,1         ; #21 Phasing error fault bit
M2149->Y:$000AC0,9,1         ; #21 Phasing search-in-progress bit
; MACRO IC 2 Node 8 Flag Registers (usually used for Motor #21)
M2150->X:$003468,0,24        ; MACRO IC 2 Node 8 flag status register
M2151->Y:$003468,0,24        ; MACRO IC 2 Node 8 flag command register
M2153->X:$003468,20,4        ; MACRO IC 2 Node 8 TUVW flags
M2154->Y:$003468,14,1        ; MACRO IC 2 Node 8 amplifier enable flag
M2155->X:$003468,15,1        ; MACRO IC 2 Node 8 node/amplifier fault flag
M2156->X:$003468,16,1        ; MACRO IC 2 Node 8 home flag
M2157->X:$003468,17,1        ; MACRO IC 2 Node 8 positive limit flag
M2158->X:$003468,18,1        ; MACRO IC 2 Node 8 negative limit flag
M2159->X:$003468,19,1        ; MACRO IC 2 Node 8 user flag
```

```
; Motor #21 Move Registers
M2161->D:$000A88              ; #21 Commanded position (1/[Ixx08*32] cts)
M2162->D:$000A8B              ; #21 Actual position (1/[Ixx08*32] cts)
M2163->D:$000AC7              ; #21 Target (end) position (1/[Ixx08*32] cts)
M2164->D:$000ACC              ; #21 Position bias (1/[Ixx08*32] cts)
M2166->X:$000A9D,0,24,S       ; #21 Actual velocity (1/[Ixx09*32] cts/cyc)
M2167->D:$000A8D              ; #21 Present master pos (1/[Ixx07*32] cts)
M2168->X:$000ABF,8,16,S       ; #21 Filter Output (16-bit DAC bits)
M2169->D:$000A90              ; #21 Compensation correction (1/[Ixx08*32] cts)
M2170->D:$000AB4              ; #21 Present phase position (including fraction)
M2171->X:$000AB4,24,S         ; #21 Present phase position (counts *Ixx70)
M2172->L:$000AD7              ; #21 Variable jog position/distance (cts)
M2173->Y:$000ACE,0,24,S       ; #21 Encoder home capture position (cts)
M2174->D:$000AEF              ; #21 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2175->X:$000AB9,8,16,S       ; #21 Actual quadrature current
M2176->Y:$000AB9,8,16,S       ; #21 Actual direct current
M2177->X:$000ABC,8,16,S       ; #21 Quadrature current-loop integrator output
M2178->Y:$000ABC,8,16,S       ; #21 Direct current-loop integrator output
M2179->X:$000AAE,8,16,S       ; #21 PID internal filter result (16-bit DAC bits)
M2188->Y:$079301,0,12,U       ; IC 5 Ch 1 Compare A fractional count
M2189->Y:$079300,0,12,U       ; IC 5 Ch 1 Compare A fractional count
; Motor #21 Axis Definition Registers
M2191->L:$000ACF              ; #21 X/U/A/B/C-Axis scale factor (cts/unit)
M2192->L:$000AD0              ; #21 Y/V-Axis scale factor (cts/unit)
M2193->L:$000AD1              ; #21 Z/W-Axis scale factor (cts/unit)
M2194->L:$000AD2              ; #21 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 6 (usually for Motor #22)
M2201->X:$079309,0,24,S       ; ENC6 24-bit counter position
M2202->Y:$07930A,8,16,S       ; OUT6A command value; DAC or PWM
M2203->X:$07930B,0,24,S       ; ENC6 captured position
M2204->Y:$07930B,8,16,S       ; OUT6B command value; DAC or PWM
M2205->Y:$07930D,8,16,S       ; ADC6A input value
M2206->Y:$07930E,8,16,S       ; ADC6B input value
M2207->Y:$07930C,8,16,S       ; OUT6C command value; PFM or PWM
M2208->Y:$07930F,0,24,S       ; ENC6 compare A position
M2209->X:$07930F,0,24,S       ; ENC6 compare B position
M2210->X:$07930E,0,24,S       ; ENC6 compare autoincrement value
M2211->X:$07930D,11           ; ENC6 compare initial state write enable
M2212->X:$07930D,12           ; ENC6 compare initial state
M2214->X:$07930D,14           ; AENA6 output status
M2215->X:$079308,19           ; USER6 flag input status
M2216->X:$079308,9            ; ENC6 compare output value
M2217->X:$079308,11           ; ENC6 capture flag
M2218->X:$079308,8            ; ENC6 count error flag
M2219->X:$079308,14           ; CHC6 input status
M2220->X:$079308,16           ; HMFL6 flag input status
M2221->X:$079308,17           ; PLIM6 flag input status
```

```
M2222->X:$079308,18          ; MLIM6 flag input status
M2223->X:$079308,15          ; FAULT6 flag input status
M2224->X:$079308,20          ; Channel 6 W flag input status
M2225->X:$079308,21          ; Channel 6 V flag input status
M2226->X:$079308,22          ; Channel 6 U flag input status
M2227->X:$079308,23          ; Channel 6 T flag input status
M2228->X:$079308,20,4        ; Channel 6 TUVW inputs as 4-bit value
; Motor #22 Status Bits
M2230->Y:$000B40,11,1        ; #22 Stopped-on-position-limit bit
M2231->X:$000B30,21,1        ; #22 Positive-end-limit-set bit
M2232->X:$000B30,22,1        ; #22 Negative-end-limit-set bit
M2233->X:$000B30,13,1        ; #22 Desired-velocity-zero bit
M2235->X:$000B30,15,1        ; #22 Dwell-in-progress bit
M2237->X:$000B30,17,1        ; #22 Running-program bit
M2238->X:$000B30,18,1        ; #22 Open-loop-mode bit
M2239->X:$000B30,19,1        ; #22 Amplifier-enabled status bit
M2240->Y:$000B40,0,1         ; #22 Background in-position bit
M2241->Y:$000B40,1,1         ; #22 Warning-following error bit
M2242->Y:$000B40,2,1         ; #22 Fatal-following-error bit
M2243->Y:$000B40,3,1         ; #22 Amplifier-fault-error bit
M2244->Y:$000B40,13,1        ; #22 Foreground in-position bit
M2245->Y:$000B40,10,1        ; #22 Home-complete bit
M2246->Y:$000B40,6,1         ; #22 Integrated following error fault bit
M2247->Y:$000B40,5,1         ; #22 I2T fault bit
M2248->Y:$000B40,8,1         ; #22 Phasing error fault bit
M2249->Y:$000B40,9,1         ; #22 Phasing search-in-progress bit
; MACRO IC 2 Node 9 Flag Registers (usually used for Motor #22)
M2250->X:$003469,0,24        ; MACRO IC 2 Node 9 flag status register
M2251->Y:$003469,0,24        ; MACRO IC 2 Node 9 flag command register
M2253->X:$003469,20,4        ; MACRO IC 2 Node 9 TUVW flags
M2254->Y:$003469,14,1        ; MACRO IC 2 Node 9 amplifier enable flag
M2255->X:$003469,15,1        ; MACRO IC 2 Node 9 node/amplifier fault flag
M2256->X:$003469,16,1        ; MACRO IC 2 Node 9 home flag
M2257->X:$003469,17,1        ; MACRO IC 2 Node 9 positive limit flag
M2258->X:$003469,18,1        ; MACRO IC 2 Node 9 negative limit flag
M2259->X:$003469,19,1        ; MACRO IC 2 Node 9 user flag
; Motor #22 Move Registers
M2261->D:$000B08             ; #22 Commanded position (1/[Ixx08*32] cts)
M2262->D:$000B0B             ; #22 Actual position (1/[Ixx08*32] cts)
M2263->D:$000B47             ; #22 Target (end) position (1/[Ixx08*32] cts)
M2264->D:$000B4C             ; #22 Position bias (1/[Ixx08*32] cts)
M2266->X:$000B1D,0,24,S      ; #22 Actual velocity (1/[Ixx09*32] cts/cyc)
M2267->D:$000B0D             ; #22 Present master pos (1/[Ixx07*32] cts)
M2268->X:$000B3F,8,16,S      ; #22 Filter Output (16-bit DAC bits)
M2269->D:$000B10             ; #22 Compensation correction (1/[Ixx08*32] cts)
M2270->D:$000B34             ; #22 Present phase position (including fraction)
M2271->X:$000B34,24,S        ; #22 Present phase position (counts *Ixx70)
```

```
M2272->L:$000B57              ; #22 Variable jog position/distance (cts)
M2273->Y:$000B4E,0,24,S       ; #22 Encoder home capture position (cts)
M2274->D:$000B6F              ; #22 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2275->X:$000B39,8,16,S       ; #22 Actual quadrature current
M2276->Y:$000B39,8,16,S       ; #22 Actual direct current
M2277->X:$000B3C,8,16,S       ; #22 Quadrature current-loop integrator output
M2278->Y:$000B3C,8,16,S       ; #22 Direct current-loop integrator output
M2279->X:$000B2E,8,16,S       ; #22 PID internal filter result (16-bit DAC bits)
M2288->Y:$079309,0,12,U       ; IC 5 Ch 2 Compare A fractional count
M2289->Y:$079308,0,12,U       ; IC 5 Ch 2 Compare A fractional count
; Motor #22 Axis Definition Registers
M2291->L:$000B4F              ; #22 X/U/A/B/C-Axis scale factor (cts/unit)
M2292->L:$000B50              ; #22 Y/V-Axis scale factor (cts/unit)
M2293->L:$000B51              ; #22 Z/W-Axis scale factor (cts/unit)
M2294->L:$000B52              ; #22 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 7 (usually for Motor #23)
M2301->X:$079311,0,24,S       ; ENC7 24-bit counter position
M2302->Y:$079312,8,16,S       ; OUT7A command value; DAC or PWM
M2303->X:$079313,0,24,S       ; ENC7 captured position
M2304->Y:$079313,8,16,S       ; OUT7B command value; DAC or PWM
M2305->Y:$079315,8,16,S       ; ADC7A input value
M2306->Y:$079316,8,16,S       ; ADC7B input value
M2307->Y:$079314,8,16,S       ; OUT7C command value; PFM or PWM
M2308->Y:$079317,0,24,S       ; ENC7 compare A position
M2309->X:$079317,0,24,S       ; ENC7 compare B position
M2310->X:$079316,0,24,S       ; ENC7 compare autoincrement value
M2311->X:$079315,11           ; ENC7 compare initial state write enable
M2312->X:$079315,12           ; ENC7 compare initial state
M2314->X:$079315,14           ; AENA7 output status
M2315->X:$079310,19           ; CHC7 input status
M2316->X:$079310,9            ; ENC7 compare output value
M2317->X:$079310,11           ; ENC7 capture flag
M2318->X:$079310,8            ; ENC7 count error flag
M2319->X:$079310,14           ; CHC7 input status
M2320->X:$079310,16           ; HMFL7 flag input status
M2321->X:$079310,17           ; PLIM7 flag input status
M2322->X:$079310,18           ; MLIM7 flag input status
M2323->X:$079310,15           ; FAULT7 flag input status
M2324->X:$079310,20           ; Channel 7 W flag input status
M2325->X:$079310,21           ; Channel 7 V flag input status
M2326->X:$079310,22           ; Channel 7 U flag input status
M2327->X:$079310,23           ; Channel 7 T flag input status
M2328->X:$079310,20,4         ; Channel 7 TUVW inputs as 4-bit value
; Motor #23 Status Bits
M2330->Y:$000BC0,11,1         ; #23 Stopped-on-position-limit bit
M2331->X:$000BB0,21,1         ; #23 Positive-end-limit-set bit
M2332->X:$000BB0,22,1         ; #23 Negative-end-limit-set bit
```

```
M2333->X:$000BB0,13,1        ; #23 Desired-velocity-zero bit
M2335->X:$000BB0,15,1        ; #23 Dwell-in-progress bit
M2337->X:$000BB0,17,1        ; #23 Running-program bit
M2338->X:$000BB0,18,1        ; #23 Open-loop-mode bit
M2339->X:$000BB0,19,1        ; #23 Amplifier-enabled status bit
M2340->Y:$000BC0,0,1         ; #23 Background in-position bit
M2341->Y:$000BC0,1,1         ; #23 Warning-following error bit
M2342->Y:$000BC0,2,1         ; #23 Fatal-following-error bit
M2343->Y:$000BC0,3,1         ; #23 Amplifier-fault-error bit
M2344->Y:$000BC0,13,1        ; #23 Foreground in-position bit
M2345->Y:$000BC0,10,1        ; #23 Home-complete bit
M2346->Y:$000BC0,6,1         ; #23 Integrated following error fault bit
M2347->Y:$000BC0,5,1         ; #23 I2T fault bit
M2348->Y:$000BC0,8,1         ; #23 Phasing error fault bit
M2349->Y:$000BC0,9,1         ; #23 Phasing search-in-progress bit
; MACRO IC 2 Node 12 Flag Registers (usually used for Motor #23)
M2350->X:$00346C,0,24        ; MACRO IC 2 Node 12 flag status register
M2351->Y:$00346C,0,24        ; MACRO IC 2 Node 12 flag command register
M2353->X:$00346C,20,4        ; MACRO IC 2 Node 12 TUVW flags
M2354->Y:$00346C,14,1        ; MACRO IC 2 Node 12 amplifier enable flag
M2355->X:$00346C,15,1        ; MACRO IC 2 Node 12 node/amplifier fault flag
M2356->X:$00346C,16,1        ; MACRO IC 2 Node 12 home flag
M2357->X:$00346C,17,1        ; MACRO IC 2 Node 12 positive limit flag
M2358->X:$00346C,18,1        ; MACRO IC 2 Node 12 negative limit flag
M2359->X:$00346C,19,1        ; MACRO IC 2 Node 12 user flag
; Motor #23 Move Registers
M2361->D:$000B88             ; #23 Commanded position (1/[Ixx08*32] cts)
M2362->D:$000B8B             ; #23 Actual position (1/[Ixx08*32] cts)
M2363->D:$000BC7             ; #23 Target (end) position (1/[Ixx08*32] cts)
M2364->D:$000BCC             ; #23 Position bias (1/[Ixx08*32] cts)
M2366->X:$000B9D,0,24,S      ; #23 Actual velocity (1/[Ixx09*32] cts/cyc)
M2367->D:$000B8D             ; #23 Present master pos (1/[Ixx07*32] cts)
M2368->X:$000BBF,8,16,S      ; #23 Filter Output (16-bit DAC bits)
M2369->D:$000B90             ; #23 Compensation correction (1/[Ixx08*32] cts)
M2370->D:$000BB4             ; #23 Present phase position (including fraction)
M2371->X:$000BB4,24,S        ; #23 Present phase position (counts *Ixx70)
M2372->L:$000BD7             ; #23 Variable jog position/distance (cts)
M2373->Y:$000BCE,0,24,S      ; #23 Encoder home capture position (cts)
M2374->D:$000BEF             ; #23 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2375->X:$000BB9,8,16,S      ; #23 Actual quadrature current
M2376->Y:$000BB9,8,16,S      ; #23 Actual direct current
M2377->X:$000BBC,8,16,S      ; #23 Quadrature current-loop integrator output
M2378->Y:$000BBC,8,16,S      ; #23 Direct current-loop integrator output
M2379->X:$000BAE,8,16,S      ; #23 PID internal filter result (16-bit DAC bits)
M2388->Y:$079311,0,12,U      ; IC 5 Ch 3 Compare A fractional count
M2389->Y:$079310,0,12,U      ; IC 5 Ch 3 Compare A fractional count
```

```
; Motor #23 Axis Definition Registers
M2391->L:$000BCF              ; #23 X/U/A/B/C-Axis scale factor (cts/unit)
M2392->L:$000BD0              ; #23 Y/V-Axis scale factor (cts/unit)
M2393->L:$000BD1              ; #23 Z/W-Axis scale factor (cts/unit)
M2394->L:$000BD2              ; #23 Axis offset (cts)
; Servo IC 5 Registers for 2nd ACC-24 Channel 8 (usually for Motor #24)
M2401->X:$079319,0,24,S       ; ENC8 24-bit counter position
M2402->Y:$07931A,8,16,S       ; OUT8A command value; DAC or PWM
M2403->X:$07931B,0,24,S       ; ENC8 captured position
M2404->Y:$07931B,8,16,S       ; OUT8B command value; DAC or PWM
M2405->Y:$07931D,8,16,S       ; ADC8A input value
M2406->Y:$07931E,8,16,S       ; ADC8B input value
M2407->Y:$07931C,8,16,S       ; OUT8C command value; PFM or PWM
M2408->Y:$07931F,0,24,S       ; ENC8 compare A position
M2409->X:$07931F,0,24,S       ; ENC8 compare B position
M2410->X:$07931E,0,24,S       ; ENC8 compare autoincrement value
M2411->X:$07931D,11           ; ENC8 compare initial state write enable
M2412->X:$07931D,12           ; ENC8 compare initial state
M2414->X:$07931D,14           ; AENA8 output status
M2415->X:$079318,19           ; USER8 flag input status
M2416->X:$079318,9            ; ENC8 compare output value
M2417->X:$079318,11           ; ENC8 capture flag
M2418->X:$079318,8            ; ENC8 count error flag
M2419->X:$079318,14           ; CHC8 input status
M2420->X:$079318,16           ; HMFL8 flag input status
M2421->X:$079318,17           ; PLIM8 flag input status
M2422->X:$079318,18           ; MLIM8 flag input status
M2423->X:$079318,15           ; FAULT8 flag input status
M2424->X:$079318,20           ; Channel 8 W flag input status
M2425->X:$079318,21           ; Channel 8 V flag input status
M2426->X:$079318,22           ; Channel 8 U flag input status
M2427->X:$079318,23           ; Channel 8 T flag input status
M2428->X:$079318,20,4         ; Channel 8 TUVW inputs as 4-bit value
; Motor #24 Status Bits
M2430->Y:$000C40,11,1         ; #24 Stopped-on-position-limit bit
M2431->X:$000C30,21,1         ; #24 Positive-end-limit-set bit
M2432->X:$000C30,22,1         ; #24 Negative-end-limit-set bit
M2433->X:$000C30,13,1         ; #24 Desired-velocity-zero bit
M2435->X:$000C30,15,1         ; #24 Dwell-in-progress bit
M2437->X:$000C30,17,1         ; #24 Running-program bit
M2438->X:$000C30,18,1         ; #24 Open-loop-mode bit
M2439->X:$000C30,19,1         ; #24 Amplifier-enabled status bit
M2440->Y:$000C40,0,1          ; #24 Background in-position bit
M2441->Y:$000C40,1,1          ; #24 Warning-following error bit
M2442->Y:$000C40,2,1          ; #24 Fatal-following-error bit
M2443->Y:$000C40,3,1          ; #24 Amplifier-fault-error bit
M2444->Y:$000C40,13,1         ; #24 Foreground in-position bit
```

```
M2445->Y:$000C40,10,1          ; #24 Home-complete bit
M2446->Y:$000C40,6,1           ; #24 Integrated following error fault bit
M2447->Y:$000C40,5,1           ; #24 I2T fault bit
M2448->Y:$000C40,8,1           ; #24 Phasing error fault bit
M2449->Y:$000C40,9,1           ; #24 Phasing search-in-progress bit
; MACRO IC 2 Node 13 Flag Registers (usually used for Motor #24)
M2450->X:$00346D,0,24          ; MACRO IC 2 Node 13 flag status register
M2451->Y:$00346D,0,24          ; MACRO IC 2 Node 13 flag command register
M2453->X:$00346D,20,4          ; MACRO IC 2 Node 13 TUVW flags
M2454->Y:$00346D,14,1          ; MACRO IC 2 Node 13 amplifier enable flag
M2455->X:$00346D,15,1          ; MACRO IC 2 Node 13 node/amplifier fault flag
M2456->X:$00346D,16,1          ; MACRO IC 2 Node 13 home flag
M2457->X:$00346D,17,1          ; MACRO IC 2 Node 13 positive limit flag
M2458->X:$00346D,18,1          ; MACRO IC 2 Node 13 negative limit flag
M2459->X:$00346D,19,1          ; MACRO IC 2 Node 13 user flag
; Motor #24 Move Registers
M2461->D:$000C08               ; #24 Commanded position (1/[Ixx08*32] cts)
M2462->D:$000C0B               ; #24 Actual position (1/[Ixx08*32] cts)
M2463->D:$000C47               ; #24 Target (end) position (1/[Ixx08*32] cts)
M2464->D:$000C4C               ; #24 Position bias (1/[Ixx08*32] cts)
M2466->X:$000C1D,0,24,S        ; #24 Actual velocity (1/[Ixx09*32] cts/cyc)
M2467->D:$000C0D               ; #24 Present master pos (1/[Ixx07*32] cts)
M2468->X:$000C3F,8,16,S        ; #24 Filter Output (16-bit DAC bits)
M2469->D:$000C10               ; #24 Compensation correction (1/[Ixx08*32] cts)
M2470->D:$000C34               ; #24 Present phase position (including fraction)
M2471->X:$000C34,24,S          ; #24 Present phase position (counts *Ixx70)
M2472->L:$000C57               ; #24 Variable jog position/distance (cts)
M2473->Y:$000C4E,0,24,S        ; #24 Encoder home capture position (cts)
M2474->D:$000C6F               ; #24 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2475->X:$000C39,8,16,S        ; #24 Actual quadrature current
M2476->Y:$000C39,8,16,S        ; #24 Actual direct current
M2477->X:$000C3C,8,16,S        ; #24 Quadrature current-loop integrator output
M2478->Y:$000C3C,8,16,S        ; #24 Direct current-loop integrator output
M2479->X:$000C2E,8,16,S        ; #24 PID internal filter result (16-bit DAC bits)
M2488->Y:$079319,0,12,U        ; IC 5 Ch 4 Compare A fractional count
M2489->Y:$079318,0,12,U        ; IC 5 Ch 4 Compare A fractional count
; Motor #24 Axis Definition Registers
M2491->L:$000C4F               ; #24 X/U/A/B/C-Axis scale factor (cts/unit)
M2492->L:$000C50               ; #24 Y/V-Axis scale factor (cts/unit)
M2493->L:$000C51               ; #24 Z/W-Axis scale factor (cts/unit)
M2494->L:$000C52               ; #24 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 1 (usually for Motor #25)
M2501->X:$07A201,0,24,S        ; ENC1 24-bit counter position
M2502->Y:$07A202,8,16,S        ; OUT1A command value; DAC or PWM
M2503->X:$07A203,0,24,S        ; ENC1 captured position
M2504->Y:$07A203,8,16,S        ; OUT1B command value; DAC or PWM
M2505->Y:$07A205,8,16,S        ; ADC1A input value
```

```
M2506->Y:$07A206,8,16,S      ; ADC1B input value
M2507->Y:$07A204,8,16,S      ; OUT1C command value; PFM or PWM
M2508->Y:$07A207,0,24,S      ; ENC1 compare A position
M2509->X:$07A207,0,24,S      ; ENC1 compare B position
M2510->X:$07A206,0,24,S      ; ENC1 compare autoincrement value
M2511->X:$07A205,11          ; ENC1 compare initial state write enable
M2512->X:$07A205,12          ; ENC1 compare initial state
M2514->X:$07A205,14          ; AENA1 output status
M2515->X:$07A200,19          ; USER1 flag input status
M2516->X:$07A200,9           ; ENC1 compare output value
M2517->X:$07A200,11          ; ENC1 capture flag
M2518->X:$07A200,8           ; ENC1 count error flag
M2519->X:$07A200,14          ; CHC1 input status
M2520->X:$07A200,16          ; HMFL1 flag input status
M2521->X:$07A200,17          ; PLIM1 flag input status
M2522->X:$07A200,18          ; MLIM1 flag input status
M2523->X:$07A200,15          ; FAULT1 flag input status
M2524->X:$07A200,20          ; Channel 1 W flag input status
M2525->X:$07A200,21          ; Channel 1 V flag input status
M2526->X:$07A200,22          ; Channel 1 U flag input status
M2527->X:$07A200,23          ; Channel 1 T flag input status
M2528->X:$07A200,20,4        ; Channel 1 TUVW inputs as 4-bit value
; Motor #25 Status Bits
M2530->Y:$000CC0,11,1        ; #25 Stopped-on-position-limit bit
M2531->X:$000CB0,21,1        ; #25 Positive-end-limit-set bit
M2532->X:$000CB0,22,1        ; #25 Negative-end-limit-set bit
M2533->X:$000CB0,13,1        ; #25 Desired-velocity-zero bit
M2535->X:$000CB0,15,1        ; #25 Dwell-in-progress bit
M2537->X:$000CB0,17,1        ; #25 Running-program bit
M2538->X:$000CB0,18,1        ; #25 Open-loop-mode bit
M2539->X:$000CB0,19,1        ; #25 Amplifier-enabled status bit
M2540->Y:$000CC0,0,1         ; #25 Background in-position bit
M2541->Y:$000CC0,1,1         ; #25 Warning-following error bit
M2542->Y:$000CC0,2,1         ; #25 Fatal-following-error bit
M2543->Y:$000CC0,3,1         ; #25 Amplifier-fault-error bit
M2544->Y:$000CC0,13,1        ; #25 Foreground in-position bit
M2545->Y:$000CC0,10,1        ; #25 Home-complete bit
M2546->Y:$000CC0,6,1         ; #25 Integrated following error fault bit
M2547->Y:$000CC0,5,1         ; #25 I2T fault bit
M2548->Y:$000CC0,8,1         ; #25 Phasing error fault bit
M2549->Y:$000CC0,9,1         ; #25 Phasing search-in-progress bit
; MACRO IC 3 Node 0 Flag Registers (usually used for Motor #25)
M2550->X:$003470,0,24        ; MACRO IC 3 Node 0 flag status register
M2551->Y:$003470,0,24        ; MACRO IC 3 Node 0 flag command register
M2553->X:$003470,20,4        ; MACRO IC 3 Node 0 TUVW flags
M2554->Y:$003470,14,1        ; MACRO IC 3 Node 0 amplifier enable flag
M2555->X:$003470,15,1        ; MACRO IC 3 Node 0 node/amplifier fault flag
```

```
M2556->X:$003470,16,1        ; MACRO IC 3 Node 0 home flag
M2557->X:$003470,17,1        ; MACRO IC 3 Node 0 positive limit flag
M2558->X:$003470,18,1        ; MACRO IC 3 Node 0 negative limit flag
M2559->X:$003470,19,1        ; MACRO IC 3 Node 0 user flag
; Motor #25 Move Registers
M2561->D:$000C88             ; #25 Commanded position (1/[Ixx08*32] cts)
M2562->D:$000C8B             ; #25 Actual position (1/[Ixx08*32] cts)
M2563->D:$000CC7             ; #25 Target (end) position (1/[Ixx08*32] cts)
M2564->D:$000CCC             ; #25 Position bias (1/[Ixx08*32] cts)
M2566->X:$000C9D,0,24,S      ; #25 Actual velocity (1/[Ixx09*32] cts/cyc)
M2567->D:$000C8D             ; #25 Present master pos (1/[Ixx07*32] cts)
M2568->X:$000CBF,8,16,S      ; #25 Filter Output (16-bit DAC bits)
M2569->D:$000C90             ; #25 Compensation correction (1/[Ixx08*32] cts)
M2570->D:$000CB4             ; #25 Present phase position (including fraction)
M2571->X:$000CB4,24,S        ; #25 Present phase position (counts *Ixx70)
M2572->L:$000CD7             ; #25 Variable jog position/distance (cts)
M2573->Y:$000CCE,0,24,S      ; #25 Encoder home capture position (cts)
M2574->D:$000CEF             ; #25 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2575->X:$000CB9,8,16,S      ; #25 Actual quadrature current
M2576->Y:$000CB9,8,16,S      ; #25 Actual direct current
M2577->X:$000CBC,8,16,S      ; #25 Quadrature current-loop integrator output
M2578->Y:$000CBC,8,16,S      ; #25 Direct current-loop integrator output
M2579->X:$000CAE,8,16,S      ; #25 PID internal filter result (16-bit DAC bits)
M2588->Y:$07A201,0,12,U      ; IC 6 Ch 1 Compare A fractional count
M2589->Y:$07A200,0,12,U      ; IC 6 Ch 1 Compare A fractional count
; Motor #25 Axis Definition Registers
M2591->L:$000CCF             ; #25 X/U/A/B/C-Axis scale factor (cts/unit)
M2592->L:$000CD0             ; #25 Y/V-Axis scale factor (cts/unit)
M2593->L:$000CD1             ; #25 Z/W-Axis scale factor (cts/unit)
M2594->L:$000CD2             ; #25 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 2 (usually for Motor #26)
M2601->X:$07A209,0,24,S      ; ENC2 24-bit counter position
M2602->Y:$07A20A,8,16,S      ; OUT2A command value; DAC or PWM
M2603->X:$07A20B,0,24,S      ; ENC2 captured position
M2604->Y:$07A20B,8,16,S      ; OUT2B command value; DAC or PWM
M2605->Y:$07A20D,8,16,S      ; ADC2A input value
M2606->Y:$07A20E,8,16,S      ; ADC2B input value
M2607->Y:$07A20C,8,16,S      ; OUT2C command value; PFM or PWM
M2608->Y:$07A20F,0,24,S      ; ENC2 compare A position
M2609->X:$07A20F,0,24,S      ; ENC2 compare B position
M2610->X:$07A20E,0,24,S      ; ENC2 compare autoincrement value
M2611->X:$07A20D,11          ; ENC2 compare initial state write enable
M2612->X:$07A20D,12          ; ENC2 compare initial state
M2614->X:$07A20D,14          ; AENA2 output status
M2615->X:$07A208,19          ; USER2 flag input status
M2616->X:$07A208,9           ; ENC2 compare output value
M2617->X:$07A208,11          ; ENC2 capture flag
```

```
M2618->X:$07A208,8          ; ENC2 count error flag
M2619->X:$07A208,14         ; CHC2 input status
M2620->X:$07A208,16         ; HMFL2 flag input status
M2621->X:$07A208,17         ; PLIM2 flag input status
M2622->X:$07A208,18         ; MLIM2 flag input status
M2623->X:$07A208,15         ; FAULT2 flag input status
M2624->X:$07A208,20         ; Channel 2 W flag input status
M2625->X:$07A208,21         ; Channel 2 V flag input status
M2626->X:$07A208,22         ; Channel 2 U flag input status
M2627->X:$07A208,23         ; Channel 2 T flag input status
M2628->X:$07A208,20,4       ; Channel 2 TUVW inputs as 4-bit value
; Motor #26 Status Bits
M2630->Y:$000D40,11,1       ; #26 Stopped-on-position-limit bit
M2631->X:$000D30,21,1       ; #26 Positive-end-limit-set bit
M2632->X:$000D30,22,1       ; #26 Negative-end-limit-set bit
M2633->X:$000D30,13,1       ; #26 Desired-velocity-zero bit
M2635->X:$000D30,15,1       ; #26 Dwell-in-progress bit
M2637->X:$000D30,17,1       ; #26 Running-program bit
M2638->X:$000D30,18,1       ; #26 Open-loop-mode bit
M2639->X:$000D30,19,1       ; #26 Amplifier-enabled status bit
M2640->Y:$000D40,0,1        ; #26 Background in-position bit
M2641->Y:$000D40,1,1        ; #26 Warning-following error bit
M2642->Y:$000D40,2,1        ; #26 Fatal-following-error bit
M2643->Y:$000D40,3,1        ; #26 Amplifier-fault-error bit
M2644->Y:$000D40,13,1       ; #26 Foreground in-position bit
M2645->Y:$000D40,10,1       ; #26 Home-complete bit
M2646->Y:$000D40,6,1        ; #26 Integrated following error fault bit
M2647->Y:$000D40,5,1        ; #26 I2T fault bit
M2648->Y:$000D40,8,1        ; #26 Phasing error fault bit
M2649->Y:$000D40,9,1        ; #26 Phasing search-in-progress bit
; MACRO IC 3 Node 1 Flag Registers (usually used for Motor #26)
M2650->X:$003471,0,24       ; MACRO IC 3 Node 1 flag status register
M2651->Y:$003471,0,24       ; MACRO IC 3 Node 1 flag command register
M2653->X:$003471,20,4       ; MACRO IC 3 Node 1 TUVW flags
M2654->Y:$003471,14,1       ; MACRO IC 3 Node 1 amplifier enable flag
M2655->X:$003471,15,1       ; MACRO IC 3 Node 1 node/amplifier fault flag
M2656->X:$003471,16,1       ; MACRO IC 3 Node 1 home flag
M2657->X:$003471,17,1       ; MACRO IC 3 Node 1 positive limit flag
M2658->X:$003471,18,1       ; MACRO IC 3 Node 1 negative limit flag
M2659->X:$003471,19,1       ; MACRO IC 3 Node 1 user flag
; Motor #26 Move Registers
M2661->D:$000D08            ; #26 Commanded position (1/[Ixx08*32] cts)
M2662->D:$000D0B            ; #26 Actual position (1/[Ixx08*32] cts)
M2663->D:$000D47            ; #26 Target (end) position (1/[Ixx08*32] cts)
M2664->D:$000D4C            ; #26 Position bias (1/[Ixx08*32] cts)
M2666->X:$000D1D,0,24,S     ; #26 Actual velocity (1/[Ixx09*32] cts/cyc)
M2667->D:$000D0D            ; #26 Present master pos (1/[Ixx07*32] cts)
```

```
M2668->X:$000D3F,8,16,S     ; #26 Filter Output (16-bit DAC bits)
M2669->D:$000D10            ; #26 Compensation correction (1/[Ixx08*32] cts)
M2670->D:$000D34            ; #26 Present phase position (including fraction)
M2671->X:$000D34,24,S       ; #26 Present phase position (counts *Ixx70)
M2672->L:$000D57            ; #26 Variable jog position/distance (cts)
M2673->Y:$000D4E,0,24,S     ; #26 Encoder home capture position (cts)
M2674->D:$000D6F            ; #26 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2675->X:$000D39,8,16,S     ; #26 Actual quadrature current
M2676->Y:$000D39,8,16,S     ; #26 Actual direct current
M2677->X:$000D3C,8,16,S     ; #26 Quadrature current-loop integrator output
M2678->Y:$000D3C,8,16,S     ; #26 Direct current-loop integrator output
M2679->X:$000D2E,8,16,S     ; #26 PID internal filter result (16-bit DAC bits)
M2688->Y:$07A209,0,12,U     ; IC 6 Ch 2 Compare A fractional count
M2689->Y:$07A208,0,12,U     ; IC 6 Ch 2 Compare A fractional count
; Motor #26 Axis Definition Registers
M2691->L:$000D4F            ; #26 X/U/A/B/C-Axis scale factor (cts/unit)
M2692->L:$000D50            ; #26 Y/V-Axis scale factor (cts/unit)
M2693->L:$000D51            ; #26 Z/W-Axis scale factor (cts/unit)
M2694->L:$000D52            ; #26 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 3 (usually for Motor #27)
M2701->X:$07A211,0,24,S     ; ENC3 24-bit counter position
M2702->Y:$07A212,8,16,S     ; OUT3A command value; DAC or PWM
M2703->X:$07A213,0,24,S     ; ENC3 captured position
M2704->Y:$07A213,8,16,S     ; OUT3B command value; DAC or PWM
M2705->Y:$07A215,8,16,S     ; ADC3A input value
M2706->Y:$07A216,8,16,S     ; ADC3B input value
M2707->Y:$07A214,8,16,S     ; OUT3C command value; PFM or PWM
M2708->Y:$07A217,0,24,S     ; ENC3 compare A position
M2709->X:$07A217,0,24,S     ; ENC3 compare B position
M2710->X:$07A216,0,24,S     ; ENC3 compare autoincrement value
M2711->X:$07A215,11         ; ENC3 compare initial state write enable
M2712->X:$07A215,12         ; ENC3 compare initial state
M2714->X:$07A215,14         ; AENA3 output status
M2715->X:$07A210,19         ; USER3 flag input status
M2716->X:$07A210,9          ; ENC3 compare output value
M2717->X:$07A210,11         ; ENC3 capture flag
M2718->X:$07A210,8          ; ENC3 count error flag
M2719->X:$07A210,14         ; CHC3 input status
M2720->X:$07A210,16         ; HMFL3 flag input status
M2721->X:$07A210,17         ; PLIM3 flag input status
M2722->X:$07A210,18         ; MLIM3 flag input status
M2723->X:$07A210,15         ; FAULT3 flag input status
M2724->X:$07A210,20         ; Channel 3 W flag input status
M2725->X:$07A210,21         ; Channel 3 V flag input status
M2726->X:$07A210,22         ; Channel 3 U flag input status
M2727->X:$07A210,23         ; Channel 3 T flag input status
M2728->X:$07A210,20,4       ; Channel 3 TUVW inputs as 4-bit value
```

```
; Motor #27 Status Bits
M2730->Y:$000DC0,11,1        ; #27 Stopped-on-position-limit bit
M2731->X:$000DB0,21,1        ; #27 Positive-end-limit-set bit
M2732->X:$000DB0,22,1        ; #27 Negative-end-limit-set bit
M2733->X:$000DB0,13,1        ; #27 Desired-velocity-zero bit
M2735->X:$000DB0,15,1        ; #27 Dwell-in-progress bit
M2737->X:$000DB0,17,1        ; #27 Running-program bit
M2738->X:$000DB0,18,1        ; #27 Open-loop-mode bit
M2739->X:$000DB0,19,1        ; #27 Amplifier-enabled status bit
M2740->Y:$000DC0,0,1         ; #27 Background in-position bit
M2741->Y:$000DC0,1,1         ; #27 Warning-following error bit
M2742->Y:$000DC0,2,1         ; #27 Fatal-following-error bit
M2743->Y:$000DC0,3,1         ; #27 Amplifier-fault-error bit
M2744->Y:$000DC0,13,1        ; #27 Foreground in-position bit
M2745->Y:$000DC0,10,1        ; #27 Home-complete bit
M2746->Y:$000DC0,6,1         ; #27 Integrated following error fault bit
M2747->Y:$000DC0,5,1         ; #27 I2T fault bit
M2748->Y:$000DC0,8,1         ; #27 Phasing error fault bit
M2749->Y:$000DC0,9,1         ; #27 Phasing search-in-progress bit
; MACRO IC 3 Node 4 Flag Registers (usually used for Motor #27)
M2750->X:$003474,0,24        ; MACRO IC 3 Node 4 flag status register
M2751->Y:$003474,0,24        ; MACRO IC 3 Node 4 flag command register
M2753->X:$003474,20,4        ; MACRO IC 3 Node 4 TUVW flags
M2754->Y:$003474,14,1        ; MACRO IC 3 Node 4 amplifier enable flag
M2755->X:$003474,15,1        ; MACRO IC 3 Node 4 node/amplifier fault flag
M2756->X:$003474,16,1        ; MACRO IC 3 Node 4 home flag
M2757->X:$003474,17,1        ; MACRO IC 3 Node 4 positive limit flag
M2758->X:$003474,18,1        ; MACRO IC 3 Node 4 negative limit flag
M2759->X:$003474,19,1        ; MACRO IC 3 Node 4 user flag
; Motor #27 Move Registers
M2761->D:$000D88             ; #27 Commanded position (1/[Ixx08*32] cts)
M2762->D:$000D8B             ; #27 Actual position (1/[Ixx08*32] cts)
M2763->D:$000DC7             ; #27 Target (end) position (1/[Ixx08*32] cts)
M2764->D:$000DCC             ; #27 Position bias (1/[Ixx08*32] cts)
M2766->X:$000D9D,0,24,S      ; #27 Actual velocity (1/[Ixx09*32] cts/cyc)
M2767->D:$000D8D             ; #27 Present master pos (1/[Ixx07*32] cts)
M2768->X:$000DBF,8,16,S      ; #27 Filter Output (16-bit DAC bits)
M2769->D:$000D90             ; #27 Compensation correction (1/[Ixx08*32] cts)
M2770->D:$000DB4             ; #27 Present phase position (including fraction)
M2771->X:$000DB4,24,S        ; #27 Present phase position (counts *Ixx70)
M2772->L:$000DD7             ; #27 Variable jog position/distance (cts)
M2773->Y:$000DCE,0,24,S      ; #27 Encoder home capture position (cts)
M2774->D:$000DEF             ; #27 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2775->X:$000DB9,8,16,S      ; #27 Actual quadrature current
M2776->Y:$000DB9,8,16,S      ; #27 Actual direct current
M2777->X:$000DBC,8,16,S      ; #27 Quadrature current-loop integrator output
M2778->Y:$000DBC,8,16,S      ; #27 Direct current-loop integrator output
```

```
M2779->X:$000DAE,8,16,S     ; #27 PID internal filter result (16-bit DAC bits)
M2788->Y:$07A211,0,12,U     ; IC 6 Ch 3 Compare A fractional count
M2789->Y:$07A210,0,12,U     ; IC 6 Ch 3 Compare A fractional count
; Motor #27 Axis Definition Registers
M2791->L:$000DCF            ; #27 X/U/A/B/C-Axis scale factor (cts/unit)
M2792->L:$000DD0            ; #27 Y/V-Axis scale factor (cts/unit)
M2793->L:$000DD1            ; #27 Z/W-Axis scale factor (cts/unit)
M2794->L:$000DD2            ; #27 Axis offset (cts)
; Servo IC 6 Registers for 3rd ACC-24 Channel 4 (usually for Motor #28)
M2801->X:$07A219,0,24,S     ; ENC4 24-bit counter position
M2802->Y:$07A21A,8,16,S     ; OUT4A command value; DAC or PWM
M2803->X:$07A21B,0,24,S     ; ENC4 captured position
M2804->Y:$07A21B,8,16,S     ; OUT4B command value; DAC or PWM
M2805->Y:$07A21D,8,16,S     ; ADC4A input value
M2806->Y:$07A21E,8,16,S     ; ADC4B input value
M2807->Y:$07A21C,8,16,S     ; OUT4C command value; PFM or PWM
M2808->Y:$07A21F,0,24,S     ; ENC4 compare A position
M2809->X:$07A21F,0,24,S     ; ENC4 compare B position
M2810->X:$07A21E,0,24,S     ; ENC4 compare autoincrement value
M2811->X:$07A21D,11         ; ENC4 compare initial state write enable
M2812->X:$07A21D,12         ; ENC4 compare initial state
M2814->X:$07A21D,14         ; AENA4 output status
M2815->X:$07A218,19         ; USER4 flag input status
M2816->X:$07A218,9          ; ENC4 compare output value
M2817->X:$07A218,11         ; ENC4 capture flag
M2818->X:$07A218,8          ; ENC4 count error flag
M2819->X:$07A218,14         ; HMFL4 flag input status
M2820->X:$07A218,16         ; CHC4 input status
M2821->X:$07A218,17         ; PLIM4 flag input status
M2822->X:$07A218,18         ; MLIM4 flag input status
M2823->X:$07A218,15         ; FAULT4 flag input status
M2824->X:$07A218,20         ; Channel 4 W flag input status
M2825->X:$07A218,21         ; Channel 4 V flag input status
M2826->X:$07A218,22         ; Channel 4 U flag input status
M2827->X:$07A218,23         ; Channel 4 T flag input status
M2828->X:$07A218,20,4       ; Channel 4 TUVW inputs as 4-bit value
; Motor #28 Status Bits
M2830->Y:$000E40,11,1       ; #28 Stopped-on-position-limit bit
M2831->X:$000E30,21,1       ; #28 Positive-end-limit-set bit
M2832->X:$000E30,22,1       ; #28 Negative-end-limit-set bit
M2833->X:$000E30,13,1       ; #28 Desired-velocity-zero bit
M2835->X:$000E30,15,1       ; #28 Dwell-in-progress bit
M2837->X:$000E30,17,1       ; #28 Running-program bit
M2838->X:$000E30,18,1       ; #28 Open-loop-mode bit
M2839->X:$000E30,19,1       ; #28 Amplifier-enabled status bit
M2840->Y:$000E40,0,1        ; #28 Background in-position bit
M2841->Y:$000E40,1,1        ; #28 Warning-following error bit
```

```
M2842->Y:$000E40,2,1          ; #28 Fatal-following-error bit
M2843->Y:$000E40,3,1          ; #28 Amplifier-fault-error bit
M2844->Y:$000E40,13,1         ; #28 Foreground in-position bit
M2845->Y:$000E40,10,1         ; #28 Home-complete bit
M2846->Y:$000E40,6,1          ; #28 Integrated following error fault bit
M2847->Y:$000E40,5,1          ; #28 I2T fault bit
M2848->Y:$000E40,8,1          ; #28 Phasing error fault bit
M2849->Y:$000E40,9,1          ; #28 Phasing search-in-progress bit
; MACRO IC 3 Node 5 Flag Registers (usually used for Motor #28)
M2850->X:$003475,0,24         ; MACRO IC 3 Node 5 flag status register
M2851->Y:$003475,0,24         ; MACRO IC 3 Node 5 flag command register
M2853->X:$003475,20,4         ; MACRO IC 3 Node 5 TUVW flags
M2854->Y:$003475,14,1         ; MACRO IC 3 Node 5 amplifier enable flag
M2855->X:$003475,15,1         ; MACRO IC 3 Node 5 node/amplifier fault flag
M2856->X:$003475,16,1         ; MACRO IC 3 Node 5 home flag
M2857->X:$003475,17,1         ; MACRO IC 3 Node 5 positive limit flag
M2858->X:$003475,18,1         ; MACRO IC 3 Node 5 negative limit flag
M2859->X:$003475,19,1         ; MACRO IC 3 Node 5 user flag
; Motor #28 Move Registers
M2861->D:$000E08              ; #28 Commanded position (1/[Ixx08*32] cts)
M2862->D:$000E0B              ; #28 Actual position (1/[Ixx08*32] cts)
M2863->D:$000E47              ; #28 Target (end) position (1/[Ixx08*32] cts)
M2864->D:$000E4C              ; #28 Position bias (1/[Ixx08*32] cts)
M2866->X:$000E1D,0,24,S       ; #28 Actual velocity (1/[Ixx09*32] cts/cyc)
M2867->D:$000E0D              ; #28 Present master pos (1/[Ixx07*32] cts)
M2868->X:$000E3F,8,16,S       ; #28 Filter Output (16-bit DAC bits)
M2869->D:$000E10              ; #28 Compensation correction (1/[Ixx08*32] cts)
M2870->D:$000E34              ; #28 Present phase position (including fraction)
M2871->X:$000E34,24,S         ; #28 Present phase position (counts *Ixx70)
M2872->L:$000E57              ; #28 Variable jog position/distance (cts)
M2873->Y:$000E4E,0,24,S       ; #28 Encoder home capture position (cts)
M2874->D:$000E6F              ; #28 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2875->X:$000E39,8,16,S       ; #28 Actual quadrature current
M2876->Y:$000E39,8,16,S       ; #28 Actual direct current
M2877->X:$000E3C,8,16,S       ; #28 Quadrature current-loop integrator output
M2878->Y:$000E3C,8,16,S       ; #28 Direct current-loop integrator output
M2879->X:$000E2E,8,16,S       ; #28 PID internal filter result (16-bit DAC bits)
M2888->Y:$07A219,0,12,U       ; IC 6 Ch 4 Compare A fractional count
M2889->Y:$07A218,0,12,U       ; IC 6 Ch 4 Compare A fractional count
; Motor #28 Axis Definition Registers
M2891->L:$000E4F              ; #28 X/U/A/B/C-Axis scale factor (cts/unit)
M2892->L:$000E50              ; #28 Y/V-Axis scale factor (cts/unit)
M2893->L:$000E51              ; #28 Z/W-Axis scale factor (cts/unit)
M2894->L:$000E52              ; #28 Axis offset (cts)
```

```
; Servo IC 7 Registers for 3rd ACC-24 Channel 5 (usually for Motor #29)
M2901->X:$07A301,0,24,S      ; ENC5 24-bit counter position
M2902->Y:$07A302,8,16,S      ; OUT5A command value; DAC or PWM
M2903->X:$07A303,0,24,S      ; ENC5 captured position
M2904->Y:$07A303,8,16,S      ; OUT5B command value; DAC or PWM
M2905->Y:$07A305,8,16,S      ; ADC5A input value
M2906->Y:$07A306,8,16,S      ; ADC5B input value
M2907->Y:$07A304,8,16,S      ; OUT5C command value; PFM or PWM
M2908->Y:$07A307,0,24,S      ; ENC5 compare A position
M2909->X:$07A307,0,24,S      ; ENC5 compare B position
M2910->X:$07A306,0,24,S      ; ENC5 compare autoincrement value
M2911->X:$07A305,11          ; ENC5 compare initial state write enable
M2912->X:$07A305,12          ; ENC5 compare initial state
M2914->X:$07A305,14          ; AENA5 output status
M2915->X:$07A300,19          ; USER5 flag input status
M2916->X:$07A300,9           ; ENC5 compare output value
M2917->X:$07A300,11          ; ENC5 capture flag
M2918->X:$07A300,8           ; ENC5 count error flag
M2919->X:$07A300,14          ; CHC5 input status
M2920->X:$07A300,16          ; HMFL5 flag input status
M2921->X:$07A300,17          ; PLIM5 flag input status
M2922->X:$07A300,18          ; MLIM5 flag input status
M2923->X:$07A300,15          ; FAULT5 flag input status
M2924->X:$07A300,20          ; Channel 5 W flag input status
M2925->X:$07A300,21          ; Channel 5 V flag input status
M2926->X:$07A300,22          ; Channel 5 U flag input status
M2927->X:$07A300,23          ; Channel 5 T flag input status
M2928->X:$07A300,20,4        ; Channel 5 TUVW inputs as 4-bit value
; Motor #29 Status Bits
M2930->Y:$000EC0,11,1        ; #29 Stopped-on-position-limit bit
M2931->X:$000EB0,21,1        ; #29 Positive-end-limit-set bit
M2932->X:$000EB0,22,1        ; #29 Negative-end-limit-set bit
M2933->X:$000EB0,13,1        ; #29 Desired-velocity-zero bit
M2935->X:$000EB0,15,1        ; #29 Dwell-in-progress bit
M2937->X:$000EB0,17,1        ; #29 Running-program bit
M2938->X:$000EB0,18,1        ; #29 Open-loop-mode bit
M2939->X:$000EB0,19,1        ; #29 Amplifier-enabled status bit
M2940->Y:$000EC0,0,1         ; #29 Background in-position bit
M2941->Y:$000EC0,1,1         ; #29 Warning-following error bit
M2942->Y:$000EC0,2,1         ; #29 Fatal-following-error bit
M2943->Y:$000EC0,3,1         ; #29 Amplifier-fault-error bit
M2944->Y:$000EC0,13,1        ; #29 Foreground in-position bit
M2945->Y:$000EC0,10,1        ; #29 Home-complete bit
M2946->Y:$000EC0,6,1         ; #29 Integrated following error fault bit
M2947->Y:$000EC0,5,1         ; #29 I2T fault bit
M2948->Y:$000EC0,8,1         ; #29 Phasing error fault bit
M2949->Y:$000EC0,9,1         ; #29 Phasing search-in-progress bit
```

```
; MACRO IC 3 Node 8 Flag Registers (usually used for Motor #29)
M2950->X:$003478,0,24          ; MACRO IC 3 Node 8 flag status register
M2951->Y:$003478,0,24          ; MACRO IC 3 Node 8 flag command register
M2953->X:$003478,20,4          ; MACRO IC 3 Node 8 TUVW flags
M2954->Y:$003478,14,1          ; MACRO IC 3 Node 8 amplifier enable flag
M2955->X:$003478,15,1          ; MACRO IC 3 Node 8 node/amplifier fault flag
M2956->X:$003478,16,1          ; MACRO IC 3 Node 8 home flag
M2957->X:$003478,17,1          ; MACRO IC 3 Node 8 positive limit flag
M2958->X:$003478,18,1          ; MACRO IC 3 Node 8 negative limit flag
M2959->X:$003478,19,1          ; MACRO IC 3 Node 8 user flag
; Motor #29 Move Registers
M2961->D:$000E88               ; #29 Commanded position (1/[Ixx08*32] cts)
M2962->D:$000E8B               ; #29 Actual position (1/[Ixx08*32] cts)
M2963->D:$000EC7               ; #29 Target (end) position (1/[Ixx08*32] cts)
M2964->D:$000ECC               ; #29 Position bias (1/[Ixx08*32] cts)
M2966->X:$000E9D,0,24,S        ; #29 Actual velocity (1/[Ixx09*32] cts/cyc)
M2967->D:$000E8D               ; #29 Present master pos (1/[Ixx07*32] cts)
M2968->X:$000EBF,8,16,S        ; #29 Filter Output (16-bit DAC bits)
M2969->D:$000E90               ; #29 Compensation correction (1/[Ixx08*32] cts)
M2970->D:$000EB4               ; #29 Present phase position (including fraction)
M2971->X:$000EB4,24,S          ; #29 Present phase position (counts *Ixx70)
M2972->L:$000ED7               ; #29 Variable jog position/distance (cts)
M2973->Y:$000ECE,0,24,S        ; #29 Encoder home capture position (cts)
M2974->D:$000EEF               ; #29 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2975->X:$000EB9,8,16,S        ; #29 Actual quadrature current
M2976->Y:$000EB9,8,16,S        ; #29 Actual direct current
M2977->X:$000EBC,8,16,S        ; #29 Quadrature current-loop integrator output
M2978->Y:$000EBC,8,16,S        ; #29 Direct current-loop integrator output
M2979->X:$000EAE,8,16,S        ; #29 PID internal filter result (16-bit DAC bits)
M2988->Y:$07A301,0,12,U        ; IC 7 Ch 1 Compare A fractional count
M2989->Y:$07A300,0,12,U        ; IC 7 Ch 1 Compare A fractional count
; Motor #29 Axis Definition Registers
M2991->L:$000ECF               ; #29 X/U/A/B/C-Axis scale factor (cts/unit)
M2992->L:$000ED0               ; #29 Y/V-Axis scale factor (cts/unit)
M2993->L:$000ED1               ; #29 Z/W-Axis scale factor (cts/unit)
M2994->L:$000ED2               ; #29 Axis offset (cts)
; Servo IC 7 Registers for 3rd ACC-24 Channel 6 (usually for Motor #30)
M3001->X:$07A309,0,24,S        ; ENC6 24-bit counter position
M3002->Y:$07A30A,8,16,S        ; OUT6A command value; DAC or PWM
M3003->X:$07A30B,0,24,S        ; ENC6 captured position
M3004->Y:$07A30B,8,16,S        ; OUT6B command value; DAC or PWM
M3005->Y:$07A30D,8,16,S        ; ADC6A input value
M3006->Y:$07A30E,8,16,S        ; ADC6B input value
M3007->Y:$07A30C,8,16,S        ; OUT6C command value; PFM or PWM
M3008->Y:$07A30F,0,24,S        ; ENC6 compare A position
M3009->X:$07A30F,0,24,S        ; ENC6 compare B position
M3010->X:$07A30E,0,24,S        ; ENC6 compare autoincrement value
```

```
M3011->X:$07A30D,11          ; ENC6 compare initial state write enable
M3012->X:$07A30D,12          ; ENC6 compare initial state
M3014->X:$07A30D,14          ; AENA6 output status
M3015->X:$07A308,19          ; USER6 flag input status
M3016->X:$07A308,9           ; ENC6 compare output value
M3017->X:$07A308,11          ; ENC6 capture flag
M3018->X:$07A308,8           ; ENC6 count error flag
M3019->X:$07A308,14          ; CHC6 input status
M3020->X:$07A308,16          ; HMFL6 flag input status
M3021->X:$07A308,17          ; PLIM6 flag input status
M3022->X:$07A308,18          ; MLIM6 flag input status
M3023->X:$07A308,15          ; FAULT6 flag input status
M3024->X:$07A308,20          ; Channel 6 W flag input status
M3025->X:$07A308,21          ; Channel 6 V flag input status
M3026->X:$07A308,22          ; Channel 6 U flag input status
M3027->X:$07A308,23          ; Channel 6 T flag input status
M3028->X:$07A308,20,4        ; Channel 6 TUVW inputs as 4-bit value
; Motor #30 Status Bits
M3030->Y:$000F40,11,1        ; #30 Stopped-on-position-limit bit
M3031->X:$000F30,21,1        ; #30 Positive-end-limit-set bit
M3032->X:$000F30,22,1        ; #30 Negative-end-limit-set bit
M3033->X:$000F30,13,1        ; #30 Desired-velocity-zero bit
M3035->X:$000F30,15,1        ; #30 Dwell-in-progress bit
M3037->X:$000F30,17,1        ; #30 Running-program bit
M3038->X:$000F30,18,1        ; #30 Open-loop-mode bit
M3039->X:$000F30,19,1        ; #30 Amplifier-enabled status bit
M3040->Y:$000F40,0,1         ; #30 Background in-position bit
M3041->Y:$000F40,1,1         ; #30 Warning-following error bit
M3042->Y:$000F40,2,1         ; #30 Fatal-following-error bit
M3043->Y:$000F40,3,1         ; #30 Amplifier-fault-error bit
M3044->Y:$000F40,13,1        ; #30 Foreground in-position bit
M3045->Y:$000F40,10,1        ; #30 Home-complete bit
M3046->Y:$000F40,6,1         ; #30 Integrated following error fault bit
M3047->Y:$000F40,5,1         ; #30 I2T fault bit
M3048->Y:$000F40,8,1         ; #30 Phasing error fault bit
M3049->Y:$000F40,9,1         ; #30 Phasing search-in-progress bit
; MACRO IC 3 Node 9 Flag Registers (usually used for Motor #30)
M3050->X:$003479,0,24        ; MACRO IC 3 Node 9 flag status register
M3051->Y:$003479,0,24        ; MACRO IC 3 Node 9 flag command register
M3053->X:$003479,20,4        ; MACRO IC 3 Node 9 TUVW flags
M3054->Y:$003479,14,1        ; MACRO IC 3 Node 9 amplifier enable flag
M3055->X:$003479,15,1        ; MACRO IC 3 Node 9 node/amplifier fault flag
M3056->X:$003479,16,1        ; MACRO IC 3 Node 9 home flag
M3057->X:$003479,17,1        ; MACRO IC 3 Node 9 positive limit flag
M3058->X:$003479,18,1        ; MACRO IC 3 Node 9 negative limit flag
M3059->X:$003479,19,1        ; MACRO IC 3 Node 9 user flag
```

```
; Motor #30 Move Registers
M3061->D:$000F08              ; #30 Commanded position (1/[Ixx08*32] cts)
M3062->D:$000F0B              ; #30 Actual position (1/[Ixx08*32] cts)
M3063->D:$000F47              ; #30 Target (end) position (1/[Ixx08*32] cts)
M3064->D:$000F4C              ; #30 Position bias (1/[Ixx08*32] cts)
M3066->X:$000F1D,0,24,S       ; #30 Actual velocity (1/[Ixx09*32] cts/cyc)
M3067->D:$000F0D              ; #30 Present master pos (1/[Ixx07*32] cts)
M3068->X:$000F3F,8,16,S       ; #30 Filter Output (16-bit DAC bits)
M3069->D:$000F10              ; #30 Compensation correction (1/[Ixx08*32] cts)
M3070->D:$000F34              ; #30 Present phase position (including fraction)
M3071->X:$000F34,24,S         ; #30 Present phase position (counts *Ixx70)
M3072->L:$000F57              ; #30 Variable jog position/distance (cts)
M3073->Y:$000F4E,0,24,S       ; #30 Encoder home capture position (cts)
M3074->D:$000F6F              ; #30 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3075->X:$000F39,8,16,S       ; #30 Actual quadrature current
M3076->Y:$000F39,8,16,S       ; #30 Actual direct current
M3077->X:$000F3C,8,16,S       ; #30 Quadrature current-loop integrator output
M3078->Y:$000F3C,8,16,S       ; #30 Direct current-loop integrator output
M3079->X:$000F2E,8,16,S       ; #30 PID internal filter result (16-bit DAC bits)
M3088->Y:$07A309,0,12,U       ; IC 7 Ch 2 Compare A fractional count
M3089->Y:$07A308,0,12,U       ; IC 7 Ch 2 Compare A fractional count
; Motor #30 Axis Definition Registers
M3091->L:$000F4F              ; #30 X/U/A/B/C-Axis scale factor (cts/unit)
M3092->L:$000F50              ; #30 Y/V-Axis scale factor (cts/unit)
M3093->L:$000F51              ; #30 Z/W-Axis scale factor (cts/unit)
M3094->L:$000F52              ; #30 Axis offset (cts)
; Servo IC 7 Registers for 3rd ACC-24 Channel 7 (usually for Motor #31)
M3101->X:$07A311,0,24,S       ; ENC7 24-bit counter position
M3102->Y:$07A312,8,16,S       ; OUT7A command value; DAC or PWM
M3103->X:$07A313,0,24,S       ; ENC7 captured position
M3104->Y:$07A313,8,16,S       ; OUT7B command value; DAC or PWM
M3105->Y:$07A315,8,16,S       ; ADC7A input value
M3106->Y:$07A316,8,16,S       ; ADC7B input value
M3107->Y:$07A314,8,16,S       ; OUT7C command value; PFM or PWM
M3108->Y:$07A317,0,24,S       ; ENC7 compare A position
M3109->X:$07A317,0,24,S       ; ENC7 compare B position
M3110->X:$07A316,0,24,S       ; ENC7 compare autoincrement value
M3111->X:$07A315,11           ; ENC7 compare initial state write enable
M3112->X:$07A315,12           ; ENC7 compare initial state
M3114->X:$07A315,14           ; AENA7 output status
M3115->X:$07A310,19           ; CHC7 input status
M3116->X:$07A310,9            ; ENC7 compare output value
M3117->X:$07A310,11           ; ENC7 capture flag
M3118->X:$07A310,8            ; ENC7 count error flag
M3119->X:$07A310,14           ; CHC7 input status
M3120->X:$07A310,16           ; HMFL7 flag input status
M3121->X:$07A310,17           ; PLIM7 flag input status
```

```
M3122->X:$07A310,18          ; MLIM7 flag input status
M3123->X:$07A310,15          ; FAULT7 flag input status
M3124->X:$07A310,20          ; Channel 7 W flag input status
M3125->X:$07A310,21          ; Channel 7 V flag input status
M3126->X:$07A310,22          ; Channel 7 U flag input status
M3127->X:$07A310,23          ; Channel 7 T flag input status
M3128->X:$07A310,20,4        ; Channel 7 TUVW inputs as 4-bit value
; Motor #31 Status Bits
M3130->Y:$000FC0,11,1        ; #31 Stopped-on-position-limit bit
M3131->X:$000FB0,21,1        ; #31 Positive-end-limit-set bit
M3132->X:$000FB0,22,1        ; #31 Negative-end-limit-set bit
M3133->X:$000FB0,13,1        ; #31 Desired-velocity-zero bit
M3135->X:$000FB0,15,1        ; #31 Dwell-in-progress bit
M3137->X:$000FB0,17,1        ; #31 Running-program bit
M3138->X:$000FB0,18,1        ; #31 Open-loop-mode bit
M3139->X:$000FB0,19,1        ; #31 Amplifier-enabled status bit
M3140->Y:$000FC0,0,1         ; #31 Background in-position bit
M3141->Y:$000FC0,1,1         ; #31 Warning-following error bit
M3142->Y:$000FC0,2,1         ; #31 Fatal-following-error bit
M3143->Y:$000FC0,3,1         ; #31 Amplifier-fault-error bit
M3144->Y:$000FC0,13,1        ; #31 Foreground in-position bit
M3145->Y:$000FC0,10,1        ; #31 Home-complete bit
M3146->Y:$000FC0,6,1         ; #31 Integrated following error fault bit
M3147->Y:$000FC0,5,1         ; #31 I2T fault bit
M3148->Y:$000FC0,8,1         ; #31 Phasing error fault bit
M3149->Y:$000FC0,9,1         ; #31 Phasing search-in-progress bit
; MACRO IC 3 Node 12 Flag Registers (usually used for Motor #31)
M3150->X:$00347C,0,24        ; MACRO IC 3 Node 12 flag status register
M3151->Y:$00347C,0,24        ; MACRO IC 3 Node 12 flag command register
M3153->X:$00347C,20,4        ; MACRO IC 3 Node 12 TUVW flags
M3154->Y:$00347C,14,1        ; MACRO IC 3 Node 12 amplifier enable flag
M3155->X:$00347C,15,1        ; MACRO IC 3 Node 12 node/amplifier fault flag
M3156->X:$00347C,16,1        ; MACRO IC 3 Node 12 home flag
M3157->X:$00347C,17,1        ; MACRO IC 3 Node 12 positive limit flag
M3158->X:$00347C,18,1        ; MACRO IC 3 Node 12 negative limit flag
M3159->X:$00347C,19,1        ; MACRO IC 3 Node 12 user flag
; Motor #31 Move Registers
M3161->D:$000F88             ; #31 Commanded position (1/[Ixx08*32] cts)
M3162->D:$000F8B             ; #31 Actual position (1/[Ixx08*32] cts)
M3163->D:$000FC7             ; #31 Target (end) position (1/[Ixx08*32] cts)
M3164->D:$000FCC             ; #31 Position bias (1/[Ixx08*32] cts)
M3166->X:$000F9D,0,24,S      ; #31 Actual velocity (1/[Ixx09*32] cts/cyc)
M3167->D:$000F8D             ; #31 Present master pos (1/[Ixx07*32] cts)
M3168->X:$000FBF,8,16,S      ; #31 Filter Output (16-bit DAC bits)
M3169->D:$000F90             ; #31 Compensation correction (1/[Ixx08*32] cts)
M3170->D:$000FB4             ; #31 Present phase position (including fraction)
M3171->X:$000FB4,24,S        ; #31 Present phase position (counts *Ixx70)
```

```
M3172->L:$000FD7              ; #31 Variable jog position/distance (cts)
M3173->Y:$000FCE,0,24,S       ; #31 Encoder home capture position (cts)
M3174->D:$000FEF              ; #31 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3175->X:$000FB9,8,16,S       ; #31 Actual quadrature current
M3176->Y:$000FB9,8,16,S       ; #31 Actual direct current
M3177->X:$000FBC,8,16,S       ; #31 Quadrature current-loop integrator output
M3178->Y:$000FBC,8,16,S       ; #31 Direct current-loop integrator output
M3179->X:$000FAE,8,16,S       ; #31 PID internal filter result (16-bit DAC bits)
M3188->Y:$07A311,0,12,U       ; IC 7 Ch 3 Compare A fractional count
M3189->Y:$07A310,0,12,U       ; IC 7 Ch 3 Compare A fractional count
; Motor #31 Axis Definition Registers
M3191->L:$000FCF              ; #31 X/U/A/B/C-Axis scale factor (cts/unit)
M3192->L:$000FD0              ; #31 Y/V-Axis scale factor (cts/unit)
M3193->L:$000FD1              ; #31 Z/W-Axis scale factor (cts/unit)
M3194->L:$000FD2              ; #31 Axis offset (cts)
; Servo IC 7 Registers for 3rd ACC-24 Channel 8 (usually for Motor #32)
M3201->X:$07A319,0,24,S       ; ENC8 24-bit counter position
M3202->Y:$07A31A,8,16,S       ; OUT8A command value; DAC or PWM
M3203->X:$07A31B,0,24,S       ; ENC8 captured position
M3204->Y:$07A31B,8,16,S       ; OUT8B command value; DAC or PWM
M3205->Y:$07A31D,8,16,S       ; ADC8A input value
M3206->Y:$07A31E,8,16,S       ; ADC8B input value
M3207->Y:$07A31C,8,16,S       ; OUT8C command value; PFM or PWM
M3208->Y:$07A31F,0,24,S       ; ENC8 compare A position
M3209->X:$07A31F,0,24,S       ; ENC8 compare B position
M3210->X:$07A31E,0,24,S       ; ENC8 compare autoincrement value
M3211->X:$07A31D,11           ; ENC8 compare initial state write enable
M3212->X:$07A31D,12           ; ENC8 compare initial state
M3214->X:$07A31D,14           ; AENA8 output status
M3215->X:$07A318,19           ; USER8 flag input status
M3216->X:$07A318,9            ; ENC8 compare output value
M3217->X:$07A318,11           ; ENC8 capture flag
M3218->X:$07A318,8            ; ENC8 count error flag
M3219->X:$07A318,14           ; CHC8 input status
M3220->X:$07A318,16           ; HMFL8 flag input status
M3221->X:$07A318,17           ; PLIM8 flag input status
M3222->X:$07A318,18           ; MLIM8 flag input status
M3223->X:$07A318,15           ; FAULT8 flag input status
M3224->X:$07A318,20           ; Channel 8 W flag input status
M3225->X:$07A318,21           ; Channel 8 V flag input status
M3226->X:$07A318,22           ; Channel 8 U flag input status
M3227->X:$07A318,23           ; Channel 8 T flag input status
M3228->X:$07A318,20,4         ; Channel 8 TUVW inputs as 4-bit value
; Motor #32 Status Bits
M3230->Y:$001040,11,1         ; #32 Stopped-on-position-limit bit
M3231->X:$001030,21,1         ; #32 Positive-end-limit-set bit
M3232->X:$001030,22,1         ; #32 Negative-end-limit-set bit
```

```
M3233->X:$001030,13,1        ; #32 Desired-velocity-zero bit
M3235->X:$001030,15,1        ; #32 Dwell-in-progress bit
M3237->X:$001030,17,1        ; #32 Running-program bit
M3238->X:$001030,18,1        ; #32 Open-loop-mode bit
M3239->X:$001030,19,1        ; #32 Amplifier-enabled status bit
M3240->Y:$001040,0,1         ; #32 Background in-position bit
M3241->Y:$001040,1,1         ; #32 Warning-following error bit
M3242->Y:$001040,2,1         ; #32 Fatal-following-error bit
M3243->Y:$001040,3,1         ; #32 Amplifier-fault-error bit
M3244->Y:$001040,13,1        ; #32 Foreground in-position bit
M3245->Y:$001040,10,1        ; #32 Home-complete bit
M3246->Y:$001040,6,1         ; #32 Integrated following error fault bit
M3247->Y:$001040,5,1         ; #32 I2T fault bit
M3248->Y:$001040,8,1         ; #32 Phasing error fault bit
M3249->Y:$001040,9,1         ; #32 Phasing search-in-progress bit
; MACRO IC 3 Node 13 Flag Registers (usually used for Motor #32)
M3250->X:$00347D,0,24        ; MACRO IC 3 Node 13 flag status register
M3251->Y:$00347D,0,24        ; MACRO IC 3 Node 13 flag command register
M3253->X:$00347D,20,4        ; MACRO IC 3 Node 13 TUVW flags
M3254->Y:$00347D,14,1        ; MACRO IC 3 Node 13 amplifier enable flag
M3255->X:$00347D,15,1        ; MACRO IC 3 Node 13 node/amplifier fault flag
M3256->X:$00347D,16,1        ; MACRO IC 3 Node 13 home flag
M3257->X:$00347D,17,1        ; MACRO IC 3 Node 13 positive limit flag
M3258->X:$00347D,18,1        ; MACRO IC 3 Node 13 negative limit flag
M3259->X:$00347D,19,1        ; MACRO IC 3 Node 13 user flag
; Motor #32 Move Registers
M3261->D:$001008             ; #32 Commanded position (1/[Ixx08*32] cts)
M3262->D:$00100B             ; #32 Actual position (1/[Ixx08*32] cts)
M3263->D:$001047             ; #32 Target (end) position (1/[Ixx08*32] cts)
M3264->D:$00104C             ; #32 Position bias (1/[Ixx08*32] cts)
M3266->X:$00101D,0,24,S      ; #32 Actual velocity (1/[Ixx09*32] cts/cyc)
M3267->D:$00100D             ; #32 Present master pos (1/[Ixx07*32] cts)
M3268->X:$00103F,8,16,S      ; #32 Filter Output (16-bit DAC bits)
M3269->D:$001010             ; #32 Compensation correction (1/[Ixx08*32] cts)
M3270->D:$001034             ; #32 Present phase position (including fraction)
M3271->X:$001034,24,S        ; #32 Present phase position (counts *Ixx70)
M3272->L:$001057             ; #32 Variable jog position/distance (cts)
M3273->Y:$00104E,0,24,S      ; #32 Encoder home capture position (cts)
M3274->D:$00106F             ; #32 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3275->X:$001039,8,16,S      ; #32 Actual quadrature current
M3276->Y:$001039,8,16,S      ; #32 Actual direct current
M3277->X:$00103C,8,16,S      ; #32 Quadrature current-loop integrator output
M3278->Y:$00103C,8,16,S      ; #32 Direct current-loop integrator output
M3279->X:$00102E,8,16,S      ; #32 PID internal filter result (16-bit DAC bits)
M3288->Y:$07A319,0,12,U      ; IC 7 Ch 4 Compare A fractional count
M3289->Y:$07A318,0,12,U      ; IC 7 Ch 4 Compare A fractional count
```

```
; Motor #32 Axis Definition Registers
M3291->L:$00104F              ; #32 X/U/A/B/C-Axis scale factor (cts/unit)
M3292->L:$001050              ; #32 Y/V-Axis scale factor (cts/unit)
M3293->L:$001051              ; #32 Z/W-Axis scale factor (cts/unit)
M3294->L:$001052              ; #32 Axis offset (cts)
; De-multiplexed ADC values from Opt. 12, ACC-36
M5061->Y:$003400,12,12,U      ; Demuxed low ADC register from I5061
M5062->Y:$003402,12,12,U      ; Demuxed low ADC register from I5062
M5063->Y:$003404,12,12,U      ; Demuxed low ADC register from I5063
M5064->Y:$003406,12,12,U      ; Demuxed low ADC register from I5064
M5065->Y:$003408,12,12,U      ; Demuxed low ADC register from I5065
M5066->Y:$00340A,12,12,U      ; Demuxed low ADC register from I5066
M5067->Y:$00340C,12,12,U      ; Demuxed low ADC register from I5067
M5068->Y:$00340E,12,12,U      ; Demuxed low ADC register from I5068
M5069->Y:$003410,12,12,U      ; Demuxed low ADC register from I5069
M5070->Y:$003412,12,12,U      ; Demuxed low ADC register from I5070
M5071->Y:$003414,12,12,U      ; Demuxed low ADC register from I5071
M5072->Y:$003416,12,12,U      ; Demuxed low ADC register from I5072
M5073->Y:$003418,12,12,U      ; Demuxed low ADC register from I5073
M5074->Y:$00341A,12,12,U      ; Demuxed low ADC register from I5074
M5075->Y:$00341C,12,12,U      ; Demuxed low ADC register from I5075
M5076->Y:$00341E,12,12,U      ; Demuxed low ADC register from I5076
M5081->Y:$003401,12,12,U      ; Demuxed high ADC register from I5061
M5082->Y:$003403,12,12,U      ; Demuxed high ADC register from I5062
M5083->Y:$003405,12,12,U      ; Demuxed high ADC register from I5063
M5084->Y:$003407,12,12,U      ; Demuxed high ADC register from I5064
M5085->Y:$003409,12,12,U      ; Demuxed high ADC register from I5065
M5086->Y:$00340B,12,12,U      ; Demuxed high ADC register from I5066
M5087->Y:$00340D,12,12,U      ; Demuxed high ADC register from I5067
M5088->Y:$00340F,12,12,U      ; Demuxed high ADC register from I5068
M5089->Y:$003411,12,12,U      ; Demuxed high ADC register from I5069
M5090->Y:$003413,12,12,U      ; Demuxed high ADC register from I5070
M5091->Y:$003415,12,12,U      ; Demuxed high ADC register from I5071
M5092->Y:$003417,12,12,U      ; Demuxed high ADC register from I5072
M5093->Y:$003419,12,12,U      ; Demuxed high ADC register from I5073
M5094->Y:$00341B,12,12,U      ; Demuxed high ADC register from I5074
M5095->Y:$00341D,12,12,U      ; Demuxed high ADC register from I5075
M5096->Y:$00341F,12,12,U      ; Demuxed high ADC register from I5076
; Coordinate System 1 (&1) timers
M5111->X:$002015,0,24,S       ; &1 Isx11 timer (for synchronous assignment)
M5112->Y:$002015,0,24,S       ; &1 Isx12 timer (for synchronous assignment)
; Coordinate System 1 (&1) end-of-calculated-move positions
M5141->L:$002041              ; &1 A-axis target position (engineering units)
M5142->L:$002042              ; &1 B-axis target position (engineering units)
M5143->L:$002043              ; &1 C-axis target position (engineering units)
M5144->L:$002044              ; &1 U-axis target position (engineering units)
M5145->L:$002045              ; &1 V-axis target position (engineering units)
```

```
M5146->L:$002046              ; &1 W-axis target position (engineering units)
M5147->L:$002047              ; &1 X-axis target position (engineering units)
M5148->L:$002048              ; &1 Y-axis target position (engineering units)
M5149->L:$002049              ; &1 Z-axis target position (engineering units)
; Coordinate System 1 (&1) Status Bits
M5180->X:$002040,0,1          ; &1 Program-running bit
M5181->Y:$00203F,21,1         ; &1 Circle-radius-error bit
M5182->Y:$00203F,22,1         ; &1 Run-time-error bit
M5184->X:$002040,0,4          ; &1 Continuous motion request
M5187->Y:$00203F,17,1         ; &1 In-position bit (AND of motors)
M5188->Y:$00203F,18,1         ; &1 Warning-following-error bit (OR)
M5189->Y:$00203F,19,1         ; &1 Fatal-following-error bit (OR)
M5190->Y:$00203F,20,1         ; &1 Amp-fault-error bit (OR of motors)
; Coordinate System 1 (&1) Variables
M5197->X:$002000,0,24,S       ; &1 Host commanded time base (I10 units)
M5198->X:$002002,0,24,S       ; &1 Present time base (I10 units)
; Coordinate System 2 (&2) timers
M5211->X:$002115,0,24,S       ; &2 Isx11 timer (for synchronous assignment)
M5212->Y:$002115,0,24,S       ; &2 Isx12 timer (for synchronous assignment)
; Coordinate System 2 (&2) end-of-calculated-move positions
M5241->L:$002141              ; &2 A-axis target position (engineering units)
M5242->L:$002142              ; &2 B-axis target position (engineering units)
M5243->L:$002143              ; &2 C-axis target position (engineering units)
M5244->L:$002144              ; &2 U-axis target position (engineering units)
M5245->L:$002145              ; &2 V-axis target position (engineering units)
M5246->L:$002146              ; &2 W-axis target position (engineering units)
M5247->L:$002147              ; &2 X-axis target position (engineering units)
M5248->L:$002148              ; &2 Y-axis target position (engineering units)
M5249->L:$002149              ; &2 Z-axis target position (engineering units)
; Coordinate System 2 (&2) Status Bits
M5280->X:$002140,0,1          ; &2 Program-running bit
M5281->Y:$00213F,21,1         ; &2 Circle-radius-error bit
M5282->Y:$00213F,22,1         ; &2 Run-time-error bit
M5284->X:$002140,0,4          ; &2 Continuous motion request
M5287->Y:$00213F,17,1         ; &2 In-position bit (AND of motors)
M5288->Y:$00213F,18,1         ; &2 Warning-following-error bit (OR)
M5289->Y:$00213F,19,1         ; &2 Fatal-following-error bit (OR)
M5290->Y:$00213F,20,1         ; &2 Amp-fault-error bit (OR of motors)
; Coordinate System 2 (&2) Variables
M5297->X:$002100,0,24,S       ; &2 Host commanded time base (I10 units)
M5298->X:$002102,0,24,S       ; &2 Present time base (I10 units)
; Coordinate System 3 (&3) timers
M5311->X:$002215,0,24,S       ; &3 Isx11 timer (for synchronous assignment)
M5312->Y:$002215,0,24,S       ; &3 Isx12 timer (for synchronous assignment)
```

```
; Coordinate System 3 (&3) end-of-calculated-move positions
M5341->L:$002241              ; &3 A-axis target position (engineering units)
M5342->L:$002242              ; &3 B-axis target position (engineering units)
M5343->L:$002243              ; &3 C-axis target position (engineering units)
M5344->L:$002244              ; &3 U-axis target position (engineering units)
M5345->L:$002245              ; &3 V-axis target position (engineering units)
M5346->L:$002246              ; &3 W-axis target position (engineering units)
M5347->L:$002247              ; &3 X-axis target position (engineering units)
M5348->L:$002248              ; &3 Y-axis target position (engineering units)
M5349->L:$002249              ; &3 Z-axis target position (engineering units)
; Coordinate System 3 (&3) Status Bits
M5380->X:$002240,0,1          ; &3 Program-running bit
M5381->Y:$00223F,21,1         ; &3 Circle-radius-error bit
M5382->Y:$00223F,22,1         ; &3 Run-time-error bit
M5384->X:$002240,0,4          ; &3 Continuous motion request
M5387->Y:$00223F,17,1         ; &3 In-position bit (AND of motors)
M5388->Y:$00223F,18,1         ; &3 Warning-following-error bit (OR)
M5389->Y:$00223F,19,1         ; &3 Fatal-following-error bit (OR)
M5390->Y:$00223F,20,1         ; &3 Amp-fault-error bit (OR of motors)
; Coordinate System 3 (&3) Variables
M5397->X:$002200,0,24,S       ; &3 Host commanded time base (I10 units)
M5398->X:$002202,0,24,S       ; &3 Present time base (I10 units)
; Coordinate System 4 (&4) timers
M5411->X:$002315,0,24,S       ; &4 Isx11 timer (for synchronous assignment)
M5412->Y:$002315,0,24,S       ; &4 Isx12 timer (for synchronous assignment)
; Coordinate System 4 (&4) end-of-calculated-move positions
M5441->L:$002341              ; &4 A-axis target position (engineering units)
M5442->L:$002342              ; &4 B-axis target position (engineering units)
M5443->L:$002343              ; &4 C-axis target position (engineering units)
M5444->L:$002344              ; &4 U-axis target position (engineering units)
M5445->L:$002345              ; &4 V-axis target position (engineering units)
M5446->L:$002346              ; &4 W-axis target position (engineering units)
M5447->L:$002347              ; &4 X-axis target position (engineering units)
M5448->L:$002348              ; &4 Y-axis target position (engineering units)
M5449->L:$002349              ; &4 Z-axis target position (engineering units)
; Coordinate System 4 (&4) Status Bits
M5480->X:$002340,0,1          ; &4 Program-running bit
M5481->Y:$00233F,21,1         ; &4 Circle-radius-error bit
M5482->Y:$00233F,22,1         ; &4 Run-time-error bit
M5484->X:$002340,0,4          ; &4 Continuous motion request
M5487->Y:$00233F,17,1         ; &4 In-position bit (AND of motors)
M5488->Y:$00233F,18,1         ; &4 Warning-following-error bit (OR)
M5489->Y:$00233F,19,1         ; &4 Fatal-following-error bit (OR)
M5490->Y:$00233F,20,1         ; &4 Amp-fault-error bit (OR of motors)
```

```
; Coordinate System 4 (&4) Variables
M5497->X:$002300,0,24,S      ; &4 Host commanded time base (I10 units)
M5498->X:$002302,0,24,S      ; &4 Present time base (I10 units)
; Coordinate System 5 (&5) timers
M5511->X:$002415,0,24,S      ; &5 Isx11 timer (for synchronous assignment)
M5512->Y:$002415,0,24,S      ; &5 Isx12 timer (for synchronous assignment)
; Coordinate System 5 (&5) end-of-calculated-move positions
M5541->L:$002441             ; &5 A-axis target position (engineering units)
M5542->L:$002442             ; &5 B-axis target position (engineering units)
M5543->L:$002443             ; &5 C-axis target position (engineering units)
M5544->L:$002444             ; &5 U-axis target position (engineering units)
M5545->L:$002445             ; &5 V-axis target position (engineering units)
M5546->L:$002446             ; &5 W-axis target position (engineering units)
M5547->L:$002447             ; &5 X-axis target position (engineering units)
M5548->L:$002448             ; &5 Y-axis target position (engineering units)
M5549->L:$002449             ; &5 Z-axis target position (engineering units)
; Coordinate System 5 (&5) Status Bits
M5580->X:$002440,0,1         ; &5 Program-running bit
M5581->Y:$00243F,21,1        ; &5 Circle-radius-error bit
M5582->Y:$00243F,22,1        ; &5 Run-time-error bit
M5584->X:$002440,0,4         ; &5 Continuous motion request
M5587->Y:$00243F,17,1        ; &5 In-position bit (AND of motors)
M5588->Y:$00243F,18,1        ; &5 Warning-following-error bit (OR)
M5589->Y:$00243F,19,1        ; &5 Fatal-following-error bit (OR)
M5590->Y:$00243F,20,1        ; &5 Amp-fault-error bit (OR of motors)
; Coordinate System 5 (&5) Variables
M5597->X:$002400,0,24,S      ; &5 Host commanded time base (I10 units)
M5598->X:$002402,0,24,S      ; &5 Present time base (I10 units)
; Coordinate System 6 (&6) timers
M5611->X:$002515,0,24,S      ; &6 Isx11 timer (for synchronous assignment)
M5612->Y:$002515,0,24,S      ; &6 Isx12 timer (for synchronous assignment)
; Coordinate System 6 (&6) end-of-calculated-move positions
M5641->L:$002541             ; &6 A-axis target position (engineering units)
M5642->L:$002542             ; &6 B-axis target position (engineering units)
M5643->L:$002543             ; &6 C-axis target position (engineering units)
M5644->L:$002544             ; &6 U-axis target position (engineering units)
M5645->L:$002545             ; &6 V-axis target position (engineering units)
M5646->L:$002546             ; &6 W-axis target position (engineering units)
M5647->L:$002547             ; &6 X-axis target position (engineering units)
M5648->L:$002548             ; &6 Y-axis target position (engineering units)
M5649->L:$002549             ; &6 Z-axis target position (engineering units)
; Coordinate System 6 (&6) Status Bits
M5680->X:$002540,0,1         ; &6 Program-running bit
M5681->Y:$00253F,21,1        ; &6 Circle-radius-error bit
M5682->Y:$00253F,22,1        ; &6 Run-time-error bit
M5684->X:$002540,0,4         ; &6 Continuous motion request
```

```
M5687->Y:$00253F,17,1          ; &6 In-position bit (AND of motors)
M5688->Y:$00253F,18,1          ; &6 Warning-following-error bit (OR)
M5689->Y:$00253F,19,1          ; &6 Fatal-following-error bit (OR)
M5690->Y:$00253F,20,1          ; &6 Amp-fault-error bit (OR of motors)
; Coordinate System 6 (&6) Variables
M5697->X:$002500,0,24,S        ; &6 Host commanded time base (I10 units)
M5698->X:$002502,0,24,S        ; &6 Present time base (I10 units)
; Coordinate System 7 (&7) timers
M5711->X:$002615,0,24,S        ; &7 Isx11 timer (for synchronous assignment)
M5712->Y:$002615,0,24,S        ; &7 Isx12 timer (for synchronous assignment)
; Coordinate System 7 (&7) end-of-calculated-move positions
M5741->L:$002641               ; &7 A-axis target position (engineering units)
M5742->L:$002642               ; &7 B-axis target position (engineering units)
M5743->L:$002643               ; &7 C-axis target position (engineering units)
M5744->L:$002644               ; &7 U-axis target position (engineering units)
M5745->L:$002645               ; &7 V-axis target position (engineering units)
M5746->L:$002646               ; &7 W-axis target position (engineering units)
M5747->L:$002647               ; &7 X-axis target position (engineering units)
M5748->L:$002648               ; &7 Y-axis target position (engineering units)
M5749->L:$002649               ; &7 Z-axis target position (engineering units)
; Coordinate System 7 (&7) Status Bits
M5780->X:$002640,0,1           ; &7 Program-running bit
M5781->Y:$00263F,21,1          ; &7 Circle-radius-error bit
M5782->Y:$00263F,22,1          ; &7 Run-time-error bit
M5784->X:$002640,0,4           ; &7 Continuous motion request
M5787->Y:$00263F,17,1          ; &7 In-position bit (AND of motors)
M5788->Y:$00263F,18,1          ; &7 Warning-following-error bit (OR)
M5789->Y:$00263F,19,1          ; &7 Fatal-following-error bit (OR)
M5790->Y:$00263F,20,1          ; &7 Amp-fault-error bit (OR of motors)
; Coordinate System 7 (&7) Variables
M5797->X:$002600,0,24,S        ; &7 Host commanded time base (I10 units)
M5798->X:$002602,0,24,S        ; &7 Present time base (I10 units)
; Coordinate System 8 (&8) timers
M5811->X:$002715,0,24,S        ; &8 Isx11 timer (for synchronous assignment)
M5812->Y:$002715,0,24,S        ; &8 Isx12 timer (for synchronous assignment)
; Coordinate System 8 (&8) end-of-calculated-move positions
M5841->L:$002741               ; &8 A-axis target position (engineering units)
M5842->L:$002742               ; &8 B-axis target position (engineering units)
M5843->L:$002743               ; &8 C-axis target position (engineering units)
M5844->L:$002744               ; &8 U-axis target position (engineering units)
M5845->L:$002745               ; &8 V-axis target position (engineering units)
M5846->L:$002746               ; &8 W-axis target position (engineering units)
M5847->L:$002747               ; &8 X-axis target position (engineering units)
M5848->L:$002748               ; &8 Y-axis target position (engineering units)
M5849->L:$002749               ; &8 Z-axis target position (engineering units)
```

```
; Coordinate System 8 (&8) Status Bits
M5880->X:$002740,0,1          ; &8 Program-running bit
M5881->Y:$00273F,21,1         ; &8 Circle-radius-error bit
M5882->Y:$00273F,22,1         ; &8 Run-time-error bit
M5884->X:$002740,0,4          ; &8 Continuous motion request
M5887->Y:$00273F,17,1         ; &8 In-position bit (AND of motors)
M5888->Y:$00273F,18,1         ; &8 Warning-following-error bit (OR)
M5889->Y:$00273F,19,1         ; &8 Fatal-following-error bit (OR)
M5890->Y:$00273F,20,1         ; &8 Amp-fault-error bit (OR of motors)
; Coordinate System 8 (&8) Variables
M5897->X:$002700,0,24,S       ; &8 Host commanded time base (I10 units)
M5898->X:$002702,0,24,S       ; &8 Present time base (I10 units)
; Coordinate System 9 (&9) timers
M5911->X:$002815,0,24,S       ; &9 Isx11 timer (for synchronous assignment)
M5912->Y:$002815,0,24,S       ; &9 Isx12 timer (for synchronous assignment)
; Coordinate System 9 (&9) end-of-calculated-move positions
M5941->L:$002841              ; &9 A-axis target position (engineering units)
M5942->L:$002842              ; &9 B-axis target position (engineering units)
M5943->L:$002843              ; &9 C-axis target position (engineering units)
M5944->L:$002844              ; &9 U-axis target position (engineering units)
M5945->L:$002845              ; &9 V-axis target position (engineering units)
M5946->L:$002846              ; &9 W-axis target position (engineering units)
M5947->L:$002847              ; &9 X-axis target position (engineering units)
M5948->L:$002848              ; &9 Y-axis target position (engineering units)
M5949->L:$002849              ; &9 Z-axis target position (engineering units)
; Coordinate System 1 (&1) Status Bits
M5980->X:$002840,0,1          ; &9 Program-running bit
M5981->Y:$00283F,21,1         ; &9 Circle-radius-error bit
M5982->Y:$00283F,22,1         ; &9 Run-time-error bit
M5984->X:$002840,0,4          ; &9 Continuous motion request
M5987->Y:$00283F,17,1         ; &9 In-position bit (AND of motors)
M5988->Y:$00283F,18,1         ; &9 Warning-following-error bit (OR)
M5989->Y:$00283F,19,1         ; &9 Fatal-following-error bit (OR)
M5990->Y:$00283F,20,1         ; &9 Amp-fault-error bit (OR of motors)
; Coordinate System 1 (&1) Variables
M5997->X:$002800,0,24,S       ; &9 Host commanded time base (I10 units)
M5998->X:$002802,0,24,S       ; &9 Present time base (I10 units)
; Coordinate System 10 (&10) timers
M6011->X:$002915,0,24,S       ; &10 Isx11 timer (for synchronous assignment)
M6012->Y:$002915,0,24,S       ; &10 Isx12 timer (for synchronous assignment)
; Coordinate System 10 (&10) end-of-calculated-move positions
M6041->L:$002941              ; &10 A-axis target position (engineering units)
M6042->L:$002942              ; &10 B-axis target position (engineering units)
M6043->L:$002943              ; &10 C-axis target position (engineering units)
M6044->L:$002944              ; &10 U-axis target position (engineering units)
M6045->L:$002945              ; &10 V-axis target position (engineering units)
```

```
M6046->L:$002946              ; &10 W-axis target position (engineering units)
M6047->L:$002947              ; &10 X-axis target position (engineering units)
M6048->L:$002948              ; &10 Y-axis target position (engineering units)
M6049->L:$002949              ; &10 Z-axis target position (engineering units)
; Coordinate System 10 (&10) Status Bits
M6080->X:$002940,0,1          ; &10 Program-running bit
M6081->Y:$00293F,21,1         ; &10 Circle-radius-error bit
M6082->Y:$00293F,22,1         ; &10 Run-time-error bit
M6084->X:$002940,0,4          ; &10 Continuous motion request
M6087->Y:$00293F,17,1         ; &10 In-position bit (AND of motors)
M6088->Y:$00293F,18,1         ; &10 Warning-following-error bit (OR)
M6089->Y:$00293F,19,1         ; &10 Fatal-following-error bit (OR)
M6090->Y:$00293F,20,1         ; &10 Amp-fault-error bit (OR of motors)
; Coordinate System 10 (&10) Variables
M6097->X:$002900,0,24,S       ; &10 Host commanded time base (I10 units)
M6098->X:$002902,0,24,S       ; &10 Present time base (I10 units)
; Coordinate System 11 (&11) timers
M6111->X:$002A15,0,24,S       ; &11 Isx11 timer (for synchronous assignment)
M6112->Y:$002A15,0,24,S       ; &11 Isx12 timer (for synchronous assignment)
; Coordinate System 11 (&11) end-of-calculated-move positions
M6141->L:$002A41              ; &11 A-axis target position (engineering units)
M6142->L:$002A42              ; &11 B-axis target position (engineering units)
M6143->L:$002A43              ; &11 C-axis target position (engineering units)
M6144->L:$002A44              ; &11 U-axis target position (engineering units)
M6145->L:$002A45              ; &11 V-axis target position (engineering units)
M6146->L:$002A46              ; &11 W-axis target position (engineering units)
M6147->L:$002A47              ; &11 X-axis target position (engineering units)
M6148->L:$002A48              ; &11 Y-axis target position (engineering units)
M6149->L:$002A49              ; &11 Z-axis target position (engineering units)
; Coordinate System 11 (&11) Status Bits
M6180->X:$002A40,0,1          ; &11 Program-running bit
M6181->Y:$002A3F,21,1         ; &11 Circle-radius-error bit
M6182->Y:$002A3F,22,1         ; &11 Run-time-error bit
M6184->X:$002A40,0,4          ; &11 Continuous motion request
M6187->Y:$002A3F,17,1         ; &11 In-position bit (AND of motors)
M6188->Y:$002A3F,18,1         ; &11 Warning-following-error bit (OR)
M6189->Y:$002A3F,19,1         ; &11 Fatal-following-error bit (OR)
M6190->Y:$002A3F,20,1         ; &11 Amp-fault-error bit (OR of motors)
; Coordinate System 11 (&11) Variables
M6197->X:$002A00,0,24,S       ; &11 Host commanded time base (I10 units)
M6198->X:$002A02,0,24,S       ; &11 Present time base (I10 units)
; Coordinate System 12 (&12) timers
M6211->X:$002B15,0,24,S       ; &12 Isx11 timer (for synchronous assignment)
M6212->Y:$002B15,0,24,S       ; &12 Isx12 timer (for synchronous assignment)
```

```
; Coordinate System 12 (&12) end-of-calculated-move positions
M6241->L:$002B41            ; &12 A-axis target position (engineering units)
M6242->L:$002B42            ; &12 B-axis target position (engineering units)
M6243->L:$002B43            ; &12 C-axis target position (engineering units)
M6244->L:$002B44            ; &12 U-axis target position (engineering units)
M6245->L:$002B45            ; &12 V-axis target position (engineering units)
M6246->L:$002B46            ; &12 W-axis target position (engineering units)
M6247->L:$002B47            ; &12 X-axis target position (engineering units)
M6248->L:$002B48            ; &12 Y-axis target position (engineering units)
M6249->L:$002B49            ; &12 Z-axis target position (engineering units)
; Coordinate System 12 (&12) Status Bits
M6280->X:$002B40,0,1        ; &12 Program-running bit
M6281->Y:$002B3F,21,1       ; &12 Circle-radius-error bit
M6282->Y:$002B3F,22,1       ; &12 Run-time-error bit
M6284->X:$002B40,0,4        ; &12 Continuous motion request
M6287->Y:$002B3F,17,1       ; &12 In-position bit (AND of motors)
M6288->Y:$002B3F,18,1       ; &12 Warning-following-error bit (OR)
M6289->Y:$002B3F,19,1       ; &12 Fatal-following-error bit (OR)
M6290->Y:$002B3F,20,1       ; &12 Amp-fault-error bit (OR of motors)
; Coordinate System 12 (&12) Variables
M6297->X:$002B00,0,24,S     ; &12 Host commanded time base (I10 units)
M6298->X:$002B02,0,24,S     ; &12 Present time base (I10 units)
; Coordinate System 13 (&13) timers
M6311->X:$002C15,0,24,S     ; &13 Isx11 timer (for synchronous assignment)
M6312->Y:$002C15,0,24,S     ; &13 Isx12 timer (for synchronous assignment)
; Coordinate System 13 (&13) end-of-calculated-move positions
M6341->L:$002C41            ; &13 A-axis target position (engineering units)
M6342->L:$002C42            ; &13 B-axis target position (engineering units)
M6343->L:$002C43            ; &13 C-axis target position (engineering units)
M6344->L:$002C44            ; &13 U-axis target position (engineering units)
M6345->L:$002C45            ; &13 V-axis target position (engineering units)
M6346->L:$002C46            ; &13 W-axis target position (engineering units)
M6347->L:$002C47            ; &13 X-axis target position (engineering units)
M6348->L:$002C48            ; &13 Y-axis target position (engineering units)
M6349->L:$002C49            ; &13 Z-axis target position (engineering units)
; Coordinate System 13 (&13) Status Bits
M6380->X:$002C40,0,1        ; &13 Program-running bit
M6381->Y:$002C3F,21,1       ; &13 Circle-radius-error bit
M6382->Y:$002C3F,22,1       ; &13 Run-time-error bit
M6384->X:$002C40,0,4        ; &13 Continuous motion request
M6387->Y:$002C3F,17,1       ; &13 In-position bit (AND of motors)
M6388->Y:$002C3F,18,1       ; &13 Warning-following-error bit (OR)
M6389->Y:$002C3F,19,1       ; &13 Fatal-following-error bit (OR)
M6390->Y:$002C3F,20,1       ; &13 Amp-fault-error bit (OR of motors)
```

```
; Coordinate System 13 (&13) Variables
M6397->X:$002C00,0,24,S      ; &13 Host commanded time base (I10 units)
M6398->X:$002C02,0,24,S      ; &13 Present time base (I10 units)
; Coordinate System 14 (&14) timers
M6411->X:$002D15,0,24,S      ; &14 Isx11 timer (for synchronous assignment)
M6412->Y:$002D15,0,24,S      ; &14 Isx12 timer (for synchronous assignment)
; Coordinate System 14 (&14) end-of-calculated-move positions
M6441->L:$002D41             ; &14 A-axis target position (engineering units)
M6442->L:$002D42             ; &14 B-axis target position (engineering units)
M6443->L:$002D43             ; &14 C-axis target position (engineering units)
M6444->L:$002D44             ; &14 U-axis target position (engineering units)
M6445->L:$002D45             ; &14 V-axis target position (engineering units)
M6446->L:$002D46             ; &14 W-axis target position (engineering units)
M6447->L:$002D47             ; &14 X-axis target position (engineering units)
M6448->L:$002D48             ; &14 Y-axis target position (engineering units)
M6449->L:$002D49             ; &14 Z-axis target position (engineering units)
; Coordinate System 14 (&14) Status Bits
M6480->X:$002D40,0,1         ; &14 Program-running bit
M6481->Y:$002D3F,21,1        ; &14 Circle-radius-error bit
M6482->Y:$002D3F,22,1        ; &14 Run-time-error bit
M6484->X:$002D40,0,4         ; &14 Continuous motion request
M6487->Y:$002D3F,17,1        ; &14 In-position bit (AND of motors)
M6488->Y:$002D3F,18,1        ; &14 Warning-following-error bit (OR)
M6489->Y:$002D3F,19,1        ; &14 Fatal-following-error bit (OR)
M6490->Y:$002D3F,20,1        ; &14 Amp-fault-error bit (OR of motors)
; Coordinate System 14 (&14) Variables
M6497->X:$002D00,0,24,S      ; &14 Host commanded time base (I10 units)
M6498->X:$002D02,0,24,S      ; &14 Present time base (I10 units)
; Coordinate System 15 (&15) timers
M6511->X:$002E15,0,24,S      ; &15 Isx11 timer (for synchronous assignment)
M6512->Y:$002E15,0,24,S      ; &15 Isx12 timer (for synchronous assignment)
; Coordinate System 15 (&15) end-of-calculated-move positions
M6541->L:$002E41             ; &15 A-axis target position (engineering units)
M6542->L:$002E42             ; &15 B-axis target position (engineering units)
M6543->L:$002E43             ; &15 C-axis target position (engineering units)
M6544->L:$002E44             ; &15 U-axis target position (engineering units)
M6545->L:$002E45             ; &15 V-axis target position (engineering units)
M6546->L:$002E46             ; &15 W-axis target position (engineering units)
M6547->L:$002E47             ; &15 X-axis target position (engineering units)
M6548->L:$002E48             ; &15 Y-axis target position (engineering units)
M6549->L:$002E49             ; &15 Z-axis target position (engineering units)
; Coordinate System 15 (&15) Status Bits
M6580->X:$002E40,0,1         ; &15 Program-running bit
M6581->Y:$002E3F,21,1        ; &15 Circle-radius-error bit
M6582->Y:$002E3F,22,1        ; &15 Run-time-error bit
M6584->X:$002E40,0,4         ; &15 Continuous motion request
```

```
M6587->Y:$002E3F,17,1          ; &15 In-position bit (AND of motors)
M6588->Y:$002E3F,18,1          ; &15 Warning-following-error bit (OR)
M6589->Y:$002E3F,19,1          ; &15 Fatal-following-error bit (OR)
M6590->Y:$002E3F,20,1          ; &15 Amp-fault-error bit (OR of motors)
; Coordinate System 15 (&15) Variables
M6597->X:$002E00,0,24,S        ; &15 Host commanded time base (I10 units)
M6598->X:$002E02,0,24,S        ; &15 Present time base (I10 units)
; Coordinate System 16 (&16) timers
M6611->X:$002F15,0,24,S        ; &16 Isx11 timer (for synchronous assignment)
M6612->Y:$002F15,0,24,S        ; &16 Isx12 timer (for synchronous assignment)
; Coordinate System 16 (&16) end-of-calculated-move positions
M6641->L:$002F41               ; &16 A-axis target position (engineering units)
M6642->L:$002F42               ; &16 B-axis target position (engineering units)
M6643->L:$002F43               ; &16 C-axis target position (engineering units)
M6644->L:$002F44               ; &16 U-axis target position (engineering units)
M6645->L:$002F45               ; &16 V-axis target position (engineering units)
M6646->L:$002F46               ; &16 W-axis target position (engineering units)
M6647->L:$002F47               ; &16 X-axis target position (engineering units)
M6648->L:$002F48               ; &16 Y-axis target position (engineering units)
M6649->L:$002F49               ; &16 Z-axis target position (engineering units)
; Coordinate System 16 (&16) Status Bits
M6680->X:$002F40,0,1           ; &16 Program-running bit
M6681->Y:$002F3F,21,1          ; &16 Circle-radius-error bit
M6682->Y:$002F3F,22,1          ; &16 Run-time-error bit
M6684->X:$002F40,0,4           ; &16 Continuous motion request
M6687->Y:$002F3F,17,1          ; &16 In-position bit (AND of motors)
M6688->Y:$002F3F,18,1          ; &16 Warning-following-error bit (OR)
M6689->Y:$002F3F,19,1          ; &16 Fatal-following-error bit (OR)
M6690->Y:$002F3F,20,1          ; &16 Amp-fault-error bit (OR of motors)
; Coordinate System 16 (&16) Variables
M6697->X:$002F00,0,24,S        ; &16 Host commanded time base (I10 units)
M6698->X:$002F02,0,24,S        ; &16 Present time base (I10 units)
; Accessory 14 I/O M-Variables (1st ACC-14)
M7000->Y:$078A00,0,1           ; MI/O0
M7001->Y:$078A00,1,1           ; MI/O1
M7002->Y:$078A00,2,1           ; MI/O2
M7003->Y:$078A00,3,1           ; MI/O3
M7004->Y:$078A00,4,1           ; MI/O4
M7005->Y:$078A00,5,1           ; MI/O5
M7006->Y:$078A00,6,1           ; MI/O6
M7007->Y:$078A00,7,1           ; MI/O7
M7008->Y:$078A00,8,1           ; MI/O8
M7009->Y:$078A00,9,1           ; MI/O9
M7010->Y:$078A00,10,1          ; MI/O10
M7011->Y:$078A00,11,1          ; MI/O11
M7012->Y:$078A00,12,1          ; MI/O12
```

```
M7013->Y:$078A00,13,1        ; MI/O13
M7014->Y:$078A00,14,1        ; MI/O14
M7015->Y:$078A00,15,1        ; MI/O15
M7016->Y:$078A00,16,1        ; MI/O16
M7017->Y:$078A00,17,1        ; MI/O17
M7018->Y:$078A00,18,1        ; MI/O18
M7019->Y:$078A00,19,1        ; MI/O19
M7020->Y:$078A00,20,1        ; MI/O20
M7021->Y:$078A00,21,1        ; MI/O21
M7022->Y:$078A00,22,1        ; MI/O22
M7023->Y:$078A00,23,1        ; MI/O23
M7024->Y:$078A01,0,1         ; MI/O24
M7025->Y:$078A01,1,1         ; MI/O25
M7026->Y:$078A01,2,1         ; MI/O26
M7027->Y:$078A01,3,1         ; MI/O27
M7028->Y:$078A01,4,1         ; MI/O28
M7029->Y:$078A01,5,1         ; MI/O29
M7030->Y:$078A01,6,1         ; MI/O30
M7031->Y:$078A01,7,1         ; MI/O31
M7032->Y:$078A01,8,1         ; MI/O32
M7033->Y:$078A01,9,1         ; MI/O33
M7034->Y:$078A01,10,1        ; MI/O34
M7035->Y:$078A01,11,1        ; MI/O35
M7036->Y:$078A01,12,1        ; MI/O36
M7037->Y:$078A01,13,1        ; MI/O37
M7038->Y:$078A01,14,1        ; MI/O38
M7039->Y:$078A01,15,1        ; MI/O39
M7040->Y:$078A01,16,1        ; MI/O40
M7041->Y:$078A01,17,1        ; MI/O41
M7042->Y:$078A01,18,1        ; MI/O42
M7043->Y:$078A01,19,1        ; MI/O43
M7044->Y:$078A01,20,1        ; MI/O44
M7045->Y:$078A01,21,1        ; MI/O45
M7046->Y:$078A01,22,1        ; MI/O46
M7047->Y:$078A01,23,1        ; MI/O47
; Encoder Conversion Table Result Registers (M8xxx matches I8xxx)
M8000->X:$003501,0,24,S      ; Line 0 result from conversion table
M8001->X:$003502,0,24,S      ; Line 1 result from conversion table
M8002->X:$003503,0,24,S      ; Line 2 result from conversion table
M8003->X:$003504,0,24,S      ; Line 3 result from conversion table
M8004->X:$003505,0,24,S      ; Line 4 result from conversion table
M8005->X:$003506,0,24,S      ; Line 5 result from conversion table
M8006->X:$003507,0,24,S      ; Line 6 result from conversion table
M8007->X:$003508,0,24,S      ; Line 7 result from conversion table
M8008->X:$003509,0,24,S      ; Line 8 result from conversion table
M8009->X:$00350A,0,24,S      ; Line 9 result from conversion table
M8010->X:$00350B,0,24,S      ; Line 10 result from conversion table
```

```
M8011->X:$00350C,0,24,S      ; Line 11 result from conversion table
M8012->X:$00350D,0,24,S      ; Line 12 result from conversion table
M8013->X:$00350E,0,24,S      ; Line 13 result from conversion table
M8014->X:$00350F,0,24,S      ; Line 14 result from conversion table
M8015->X:$003510,0,24,S      ; Line 15 result from conversion table
M8016->X:$003511,0,24,S      ; Line 16 result from conversion table
M8017->X:$003512,0,24,S      ; Line 17 result from conversion table
M8018->X:$003513,0,24,S      ; Line 18 result from conversion table
M8019->X:$003514,0,24,S      ; Line 19 result from conversion table
M8020->X:$003515,0,24,S      ; Line 20 result from conversion table
M8021->X:$003516,0,24,S      ; Line 21 result from conversion table
M8022->X:$003517,0,24,S      ; Line 22 result from conversion table
M8023->X:$003518,0,24,S      ; Line 23 result from conversion table
M8024->X:$003519,0,24,S      ; Line 24 result from conversion table
M8025->X:$00351A,0,24,S      ; Line 25 result from conversion table
M8026->X:$00351B,0,24,S      ; Line 26 result from conversion table
M8027->X:$00351C,0,24,S      ; Line 27 result from conversion table
M8028->X:$00351D,0,24,S      ; Line 28 result from conversion table
M8029->X:$00351E,0,24,S      ; Line 29 result from conversion table
M8030->X:$00351F,0,24,S      ; Line 30 result from conversion table
M8031->X:$003520,0,24,S      ; Line 31 result from conversion table
M8032->X:$003521,0,24,S      ; Line 32 result from conversion table
M8033->X:$003522,0,24,S      ; Line 33 result from conversion table
M8034->X:$003523,0,24,S      ; Line 34 result from conversion table
M8035->X:$003524,0,24,S      ; Line 35 result from conversion table
M8036->X:$003525,0,24,S      ; Line 36 result from conversion table
M8037->X:$003526,0,24,S      ; Line 37 result from conversion table
M8038->X:$003527,0,24,S      ; Line 38 result from conversion table
M8039->X:$003528,0,24,S      ; Line 39 result from conversion table
M8040->X:$003529,0,24,S      ; Line 40 result from conversion table
M8041->X:$00352A,0,24,S      ; Line 41 result from conversion table
M8042->X:$00352B,0,24,S      ; Line 42 result from conversion table
M8043->X:$00352C,0,24,S      ; Line 43 result from conversion table
M8044->X:$00352D,0,24,S      ; Line 44 result from conversion table
M8045->X:$00352E,0,24,S      ; Line 45 result from conversion table
M8046->X:$00352F,0,24,S      ; Line 46 result from conversion table
M8047->X:$003530,0,24,S      ; Line 47 result from conversion table
M8048->X:$003531,0,24,S      ; Line 48 result from conversion table
M8049->X:$003532,0,24,S      ; Line 49 result from conversion table
M8050->X:$003533,0,24,S      ; Line 50 result from conversion table
M8051->X:$003534,0,24,S      ; Line 51 result from conversion table
M8052->X:$003535,0,24,S      ; Line 52 result from conversion table
M8053->X:$003536,0,24,S      ; Line 53 result from conversion table
M8054->X:$003537,0,24,S      ; Line 54 result from conversion table
M8055->X:$003538,0,24,S      ; Line 55 result from conversion table
M8056->X:$003539,0,24,S      ; Line 56 result from conversion table
M8057->X:$00353A,0,24,S      ; Line 57 result from conversion table
```

```
M8058->X:$00353B,0,24,S      ; Line 58 result from conversion table
M8059->X:$00353C,0,24,S      ; Line 59 result from conversion table
M8060->X:$00353D,0,24,S      ; Line 60 result from conversion table
M8061->X:$00353E,0,24,S      ; Line 61 result from conversion table
M8062->X:$00353F,0,24,S      ; Line 62 result from conversion table
M8063->X:$003540,0,24,S      ; Line 63 result from conversion table
```

UMAC TURBO SUGGESTED M-VARIABLE DEFINITIONS

; This file contains suggested definitions for M-variables on the UMAC Turbo. (For the 3U Turbo Stack, refer to the suggested M-variable definitions for Turbo PMAC2.) Note that these are only suggestions; the user is free to make whatever definitions are desired.

```
; Clear existing definitions
CLOSE                   ; Make sure no buffer is open
M0..8191->*             ; All M-variables are now self-referenced

; ACC-5E JI/O Port M-variables
M0->Y:$078400,0         ; I/O00 Data Line; J3 Pin 1
M1->Y:$078400,1         ; I/O01 Data Line; J3 Pin 2
M2->Y:$078400,2         ; I/O02 Data Line; J3 Pin 3
M3->Y:$078400,3         ; I/O03 Data Line; J3 Pin 4
M4->Y:$078400,4         ; I/O04 Data Line; J3 Pin 5
M5->Y:$078400,5         ; I/O05 Data Line; J3 Pin 6
M6->Y:$078400,6         ; I/O06 Data Line; J3 Pin 7
M7->Y:$078400,7         ; I/O07 Data Line; J3 Pin 8
M8->Y:$078400,8         ; I/O08 Data Line; J3 Pin 9
M9->Y:$078400,9         ; I/O09 Data Line; J3 Pin 10
M10->Y:$078400,10       ; I/O10 Data Line; J3 Pin 11
M11->Y:$078400,11       ; I/O11 Data Line; J3 Pin 12
M12->Y:$078400,12       ; I/O12 Data Line; J3 Pin 13
M13->Y:$078400,13       ; I/O13 Data Line; J3 Pin 14
M14->Y:$078400,14       ; I/O14 Data Line; J3 Pin 15
M15->Y:$078400,15       ; I/O15 Data Line; J3 Pin 16
M16->Y:$078400,16       ; I/O16 Data Line; J3 Pin 17
M17->Y:$078400,17       ; I/O17 Data Line; J3 Pin 18
M18->Y:$078400,18       ; I/O18 Data Line; J3 Pin 19
M19->Y:$078400,19       ; I/O19 Data Line; J3 Pin 20
M20->Y:$078400,20       ; I/O20 Data Line; J3 Pin 21
M21->Y:$078400,21       ; I/O21 Data Line; J3 Pin 22
M22->Y:$078400,22       ; I/O22 Data Line; J3 Pin 23
M23->Y:$078400,23       ; I/O23 Data Line; J3 Pin 24
M24->Y:$078401,0        ; I/O24 Data Line; J3 Pin 25
M25->Y:$078401,1        ; I/O25 Data Line; J3 Pin 26
M26->Y:$078401,2        ; I/O26 Data Line; J3 Pin 27
M27->Y:$078401,3        ; I/O27 Data Line; J3 Pin 28
M28->Y:$078401,4        ; I/O28 Data Line; J3 Pin 29
M29->Y:$078401,5        ; I/O29 Data Line; J3 Pin 30
M30->Y:$078401,6        ; I/O30 Data Line; J3 Pin 31
M31->Y:$078401,7        ; I/O31 Data Line; J3 Pin 32
M32->X:$078400,0,8      ; Direction control for I/O00 to I/O07
M33->Y:$070800,0        ; Buffer direction control for I/O00 to I/O07
M34->X:$078400,8,8      ; Direction control for I/O08 to I/O15
M35->Y:$070800,1        ; Buffer direction control for I/O08 to I/O15
M36->X:$078400,16,8     ; Direction control for I/O16 to I/O23
M37->Y:$070800,2        ; Buffer direction control for I/O16 to I/O23
M38->X:$078401,0,8      ; Direction control for I/O24 to I/O31
M39->Y:$070800,3        ; Buffer direction control for I/O24 to I/O31
```

```
; ACC-5E JTHW Thumbwheel Multiplexer Port M-variables
M40->Y:$078402,8          ; SEL0 Line; J2 Pin 4
M41->Y:$078402,9          ; SEL1 Line; J2 Pin 6
M42->Y:$078402,10         ; SEL2 Line; J2 Pin 8
M43->Y:$078402,11         ; SEL3 Line; J2 Pin 10
M44->Y:$078402,12         ; SEL4 Line; J2 Pin 12
M45->Y:$078402,13         ; SEL5 Line; J2 Pin 14
M46->Y:$078402,14         ; SEL6 Line; J2 Pin 16
M47->Y:$078402,15         ; SEL7 Line; J2 Pin 18
M48->Y:$078402,8,8,U      ; SEL0-7 Lines treated as a byte
M50->Y:$078402,0          ; DAT0 Line; J2 Pin 3
M51->Y:$078402,1          ; DAT1 Line; J2 Pin 5
M52->Y:$078402,2          ; DAT2 Line; J2 Pin 7
M53->Y:$078402,3          ; DAT3 Line; J2 Pin 9
M54->Y:$078402,4          ; DAT4 Line; J2 Pin 11
M55->Y:$078402,5          ; DAT5 Line; J2 Pin 13
M56->Y:$078402,6          ; DAT6 Line; J2 Pin 15
M57->Y:$078402,7          ; DAT7 Line; J2 Pin 17
M58->Y:$078402,0,8,U      ; DAT0-7 Lines treated as a byte
M60->X:$078402,0,8        ; Direction control for DAT0 to DAT7
M62->X:$078400,8,8        ; Direction control for SEL0 to SEL7
; Miscellaneous global registers
M70->X:$FFFF8C,0,24       ; Time between phase interrupts (CPU cycles/2)
M71->X:$000037,0,24       ; Time for phase tasks (CPU cycles/2)
M72->Y:$000037,0,24       ; Time for servo tasks (CPU cycles/2)
M73->X:$00000B,0,24       ; Time for RTI tasks (CPU cycles/2)
M80->X:$000025,0,24       ; Minimum watchdog timer count
M81->X:$000024,0,24       ; Pointer to display buffer
M82->Y:$001080,0,8        ; First character of display buffer
M83->X:$000006,12,1       ; Firmware checksum error bit
M84->X:$000006,13,1       ; Any memory checksum error bit
M85->X:$000006,5,1        ; MACRO auxiliary communications error bit
M86->X:$000006,6,1        ; ACC-34 serial parity error bit
; DPRAM active setup registers
M93->X:$070009,0,8        ; PC/104 Active DPRAM Base Address bits A23-A16 (Bits 0-7
                          ; from I93)
M94->X:$07000A,0,8        ; PC/104 Active DPRAM Base Address bits A15-A14, Enable &
                          ; Bank Select (Bits 0-7 from I94)
; Servo cycle counter (read only) -- counts up once per servo cycle
M100->X:$000000,0,24,S    ; 24-bit servo cycle counter
; Servo IC 2 Channel 2 Registers (usually for Motor #2)
M101->X:$078201,0,24,S    ; ENC1 24-bit counter position
M102->Y:$078202,8,16,S    ; OUT1A command value; DAC or PWM
M103->X:$078203,0,24,S    ; ENC1 captured position
M104->Y:$078203,8,16,S    ; OUT1B command value; DAC or PWM
M105->Y:$078205,8,16,S    ; ADC1A input value
M106->Y:$078206,8,16,S    ; ADC1B input value
```

```
M107->Y:$078204,8,16,S   ; OUT1C command value; PFM or PWM
M108->Y:$078207,0,24,S   ; ENC1 compare A position
M109->X:$078207,0,24,S   ; ENC1 compare B position
M110->X:$078206,0,24,S   ; ENC1 compare autoincrement value
M111->X:$078205,11       ; ENC1 compare initial state write enable
M112->X:$078205,12       ; ENC1 compare initial state
M114->X:$078205,14       ; AENA1 output status
M115->X:$078200,19       ; USER1 flag input status
M116->X:$078200,9        ; ENC1 compare output value
M117->X:$078200,11       ; ENC1 capture flag
M118->X:$078200,8        ; ENC1 count error flag
M119->X:$078200,14       ; CHC1 input status
M120->X:$078200,16       ; HMFL1 flag input status
M121->X:$078200,17       ; PLIM1 flag input status
M122->X:$078200,18       ; MLIM1 flag input status
M123->X:$078200,15       ; FAULT1 flag input status
M124->X:$078200,20       ; Channel 1 W flag input status
M125->X:$078200,21       ; Channel 1 V flag input status
M126->X:$078200,22       ; Channel 1 U flag input status
M127->X:$078200,23       ; Channel 1 T flag input status
M128->X:$078200,20,4     ; Channel 1 TUVW inputs as 4-bit value
; Motor #1 Status Bits
M130->Y:$0000C0,11,1     ; #1 Stopped-on-position-limit bit
M131->X:$0000B0,21,1     ; #1 Positive-end-limit-set bit
M132->X:$0000B0,22,1     ; #1 Negative-end-limit-set bit
M133->X:$0000B0,13,1     ; #1 Desired-velocity-zero bit
M135->X:$0000B0,15,1     ; #1 Dwell-in-progress bit
M137->X:$0000B0,17,1     ; #1 Running-program bit
M138->X:$0000B0,18,1     ; #1 Open-loop-mode bit
M139->X:$0000B0,19,1     ; #1 Amplifier-enabled status bit
M140->Y:$0000C0,0,1      ; #1 Background in-position bit
M141->Y:$0000C0,1,1      ; #1 Warning-following error bit
M142->Y:$0000C0,2,1      ; #1 Fatal-following-error bit
M143->Y:$0000C0,3,1      ; #1 Amplifier-fault-error bit
M144->Y:$0000C0,13,1     ; #1 Foreground in-position bit
M145->Y:$0000C0,10,1     ; #1 Home-complete bit
M146->Y:$0000C0,6,1      ; #1 Integrated following error fault bit
M147->Y:$0000C0,5,1      ; #1 I2T fault bit
M148->Y:$0000C0,8,1      ; #1 Phasing error fault bit
M149->Y:$0000C0,9,1      ; #1 Phasing search-in-progress bit
; MACRO IC 0 Node 0 Flag Registers (usually used for Motor #1)
M150->X:$003440,0,24     ; MACRO IC 0 Node 0 flag status register
M151->Y:$003440,0,24     ; MACRO IC 0 Node 0 flag command register
M153->X:$003440,20,4     ; MACRO IC 0 Node 0 TUVW flags
M154->Y:$003440,14,1     ; MACRO IC 0 Node 0 amplifier enable flag
M155->X:$003440,15,1     ; MACRO IC 0 Node 0 node/amplifier fault flag
M156->X:$003440,16,1     ; MACRO IC 0 Node 0 home flag
```

```
M157->X:$003440,17,1      ; MACRO IC 0 Node 0 positive limit flag
M158->X:$003440,18,1      ; MACRO IC 0 Node 0 negative limit flag
M159->X:$003440,19,1      ; MACRO IC 0 Node 0 user flag
; Motor #1 Move Registers
M161->D:$000088           ; #1 Commanded position (1/[Ixx08*32] cts)
M162->D:$00008B                ; #1 Actual position (1/[Ixx08*32] cts)
M163->D:$0000C7           ; #1 Target (end) position (1/[Ixx08*32] cts)
M164->D:$0000CC           ; #1 Position bias (1/[Ixx08*32] cts)
M166->X:$00009D,0,24,S    ; #1 Actual velocity (1/[Ixx09*32] cts/cyc)
M167->D:$00008D           ; #1 Present master pos (1/[Ixx07*32] cts)
M168->X:$0000BF,8,16,S    ; #1 Filter Output (16-bit DAC bits)
M169->D:$000090           ; #1 Compensation correction (1/[Ixx08*32] cts)
M170->D:$0000B4           ; #1 Present phase position (including fraction)
M171->X:$0000B4,24,S      ; #1 Present phase position (counts *Ixx70)
M172->L:$0000D7           ; #1 Variable jog position/distance (cts)
M173->Y:$0000CE,0,24,S    ; #1 Encoder home capture position (cts)
M174->D:$0000EF           ; #1 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M175->X:$0000B9,8,16,S    ; #1 Actual quadrature current
M176->Y:$0000B9,8,16,S    ; #1 Actual direct current
M177->X:$0000BC,8,16,S    ; #1 Quadrature current-loop integrator output
M178->Y:$0000BC,8,16,S    ; #1 Direct current-loop integrator output
M179->X:$0000AE,8,16,S    ; #1 PID internal filter result (16-bit DAC bits)
M188->Y:$078201,0,12,U    ; IC 2 Ch 1 Compare A fractional count
M189->Y:$078200,0,12,U    ; IC 2 Ch 1 Compare B fractional count
; Motor #1 Axis Definition Registers
M191->L:$0000CF           ; #1 X/U/A/B/C-Axis scale factor (cts/unit)
M192->L:$0000D0           ; #1 Y/V-Axis scale factor (cts/unit)
M193->L:$0000D1           ; #1 Z/W-Axis scale factor (cts/unit)
M194->L:$0000D2           ; #1 Axis offset (cts)
; Servo IC 2 Registers for PMAC2 Channel 2 (usually for Motor #2)
M201->X:$078209,0,24,S    ; ENC2 24-bit counter position
M202->Y:$07820A,8,16,S    ; OUT2A command value; DAC or PWM
M203->X:$07820B,0,24,S    ; ENC2 captured position
M204->Y:$07820B,8,16,S    ; OUT2B command value; DAC or PWM
M205->Y:$07820D,8,16,S    ; ADC2A input value
M206->Y:$07820E,8,16,S    ; ADC2B input value
M207->Y:$07820C,8,16,S    ; OUT2C command value; PFM or PWM
M208->Y:$07820F,0,24,S    ; ENC2 compare A position
M209->X:$07820F,0,24,S    ; ENC2 compare B position
M210->X:$07820E,0,24,S    ; ENC2 compare autoincrement value
M211->X:$07820D,11        ; ENC2 compare initial state write enable
M212->X:$07820D,12        ; ENC2 compare initial state
M214->X:$07820D,14        ; AENA2 output status
M215->X:$078208,19        ; USER2 flag input status
M216->X:$078208,9         ; ENC2 compare output value
M217->X:$078208,11        ; ENC2 capture flag
M218->X:$078208,8         ; ENC2 count error flag
```

```
M219->X:$078208,14        ; CHC2 input status
M220->X:$078208,16        ; HMFL2 flag input status
M221->X:$078208,17        ; PLIM2 flag input status
M222->X:$078208,18        ; MLIM2 flag input status
M223->X:$078208,15        ; FAULT2 flag input status
M224->X:$078208,20        ; Channel 2 W flag input status
M225->X:$078208,21        ; Channel 2 V flag input status
M226->X:$078208,22        ; Channel 2 U flag input status
M227->X:$078208,23        ; Channel 2 T flag input status
M228->X:$078208,20,4      ; Channel 2 TUVW inputs as 4-bit value
; Motor #2 Status Bits
M230->Y:$000140,11,1      ; #2 Stopped-on-position-limit bit
M231->X:$000130,21,1      ; #2 Positive-end-limit-set bit
M232->X:$000130,22,1      ; #2 Negative-end-limit-set bit
M233->X:$000130,13,1      ; #2 Desired-velocity-zero bit
M235->X:$000130,15,1      ; #2 Dwell-in-progress bit
M237->X:$000130,17,1      ; #2 Running-program bit
M238->X:$000130,18,1      ; #2 Open-loop-mode bit
M239->X:$000130,19,1      ; #2 Amplifier-enabled status bit
M240->Y:$000140,0,1       ; #2 Background in-position bit
M241->Y:$000140,1,1       ; #2 Warning-following error bit
M242->Y:$000140,2,1       ; #2 Fatal-following-error bit
M243->Y:$000140,3,1       ; #2 Amplifier-fault-error bit
M244->Y:$000140,13,1      ; #2 Foreground in-position bit
M245->Y:$000140,10,1      ; #2 Home-complete bit
M246->Y:$000140,6,1       ; #2 Integrated following error fault bit
M247->Y:$000140,5,1       ; #2 I2T fault bit
M248->Y:$000140,8,1       ; #2 Phasing error fault bit
M249->Y:$000140,9,1       ; #2 Phasing search-in-progress bit
; MACRO IC 0 Node 1 Flag Registers (usually used for Motor #2)
M250->X:$003441,0,24      ; MACRO IC 0 Node 1 flag status register
M251->Y:$003441,0,24      ; MACRO IC 0 Node 1 flag command register
M253->X:$003441,20,4      ; MACRO IC 0 Node 1 TUVW flags
M254->Y:$003441,14,1      ; MACRO IC 0 Node 1 amplifier enable flag
M255->X:$003441,15,1      ; MACRO IC 0 Node 1 node/amplifier fault flag
M256->X:$003441,16,1      ; MACRO IC 0 Node 1 home flag
M257->X:$003441,17,1      ; MACRO IC 0 Node 1 positive limit flag
M258->X:$003441,18,1      ; MACRO IC 0 Node 1 negative limit flag
M259->X:$003441,19,1      ; MACRO IC 0 Node 1 user flag
; Motor #2 Move Registers
M261->D:$000108           ; #2 Commanded position (1/[Ixx08*32] cts)
M262->D:$00010B           ; #2 Actual position (1/[Ixx08*32] cts)
M263->D:$000147           ; #2 Target (end) position (1/[Ixx08*32] cts)
M264->D:$00014C           ; #2 Position bias (1/[Ixx08*32] cts)
M266->X:$00011D,0,24,S    ; #2 Actual velocity (1/[Ixx09*32] cts/cyc)
M267->D:$00010D           ; #2 Present master pos (1/[Ixx07*32] cts)
M268->X:$00013F,8,16,S    ; #2 Filter Output (16-bit DAC bits)
```

```
M269->D:$000110          ; #2 Compensation correction (1/[Ixx08*32] cts)
M270->D:$000134          ; #2 Present phase position (including fraction)
M271->X:$000134,24,S     ; #2 Present phase position (counts *Ixx70)
M272->L:$000157          ; #2 Variable jog position/distance (cts)
M273->Y:$00014E,0,24,S   ; #2 Encoder home capture position (cts)
M274->D:$00016F          ; #2 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M275->X:$000139,8,16,S   ; #2 Actual quadrature current
M276->Y:$000139,8,16,S   ; #2 Actual direct current
M277->X:$00013C,8,16,S   ; #2 Quadrature current-loop integrator output
M278->Y:$00013C,8,16,S   ; #2 Direct current-loop integrator output
M279->X:$00012E,8,16,S   ; #2 PID internal filter result (16-bit DAC bits)
M288->Y:$078209,0,12,U   ; IC 2 Ch 2 Compare A fractional count
M289->Y:$078208,0,12,U   ; IC 2 Ch 2 Compare B fractional count
; Motor #2 Axis Definition Registers
M291->L:$00014F          ; #2 X/U/A/B/C-Axis scale factor (cts/unit)
M292->L:$000150          ; #2 Y/V-Axis scale factor (cts/unit)
M293->L:$000151          ; #2 Z/W-Axis scale factor (cts/unit)
M294->L:$000152          ; #2 Axis offset (cts)
; Servo IC 2 Channel 3 Registers (usually for Motor #3)
M301->X:$078211,0,24,S   ; ENC3 24-bit counter position
M302->Y:$078212,8,16,S   ; OUT3A command value; DAC or PWM
M303->X:$078213,0,24,S   ; ENC3 captured position
M304->Y:$078213,8,16,S   ; OUT3B command value; DAC or PWM
M305->Y:$078215,8,16,S   ; ADC3A input value
M306->Y:$078216,8,16,S   ; ADC3B input value
M307->Y:$078214,8,16,S   ; OUT3C command value; PFM or PWM
M308->Y:$078217,0,24,S   ; ENC3 compare A position
M309->X:$078217,0,24,S   ; ENC3 compare B position
M310->X:$078216,0,24,S   ; ENC3 compare autoincrement value
M311->X:$078215,11       ; ENC3 compare initial state write enable
M312->X:$078215,12       ; ENC3 compare initial state
M314->X:$078215,14       ; AENA3 output status
M315->X:$078210,19       ; USER3 flag input status
M316->X:$078210,9        ; ENC3 compare output value
M317->X:$078210,11       ; ENC3 capture flag
M318->X:$078210,8        ; ENC3 count error flag
M319->X:$078210,14       ; CHC3 input status
M320->X:$078210,16       ; HMFL3 flag input status
M321->X:$078210,17       ; PLIM3 flag input status
M322->X:$078210,18       ; MLIM3 flag input status
M323->X:$078210,15       ; FAULT3 flag input status
M324->X:$078210,20       ; Channel 3 W flag input status
M325->X:$078210,21       ; Channel 3 V flag input status
M326->X:$078210,22       ; Channel 3 U flag input status
M327->X:$078210,23       ; Channel 3 T flag input status
M328->X:$078210,20,4     ; Channel 3 TUVW inputs as 4-bit value
```

```
; Motor #3 Status Bits
M330->Y:$0001C0,11,1      ; #3 Stopped-on-position-limit bit
M331->X:$0001B0,21,1      ; #3 Positive-end-limit-set bit
M332->X:$0001B0,22,1      ; #3 Negative-end-limit-set bit
M333->X:$0001B0,13,1      ; #3 Desired-velocity-zero bit
M335->X:$0001B0,15,1      ; #3 Dwell-in-progress bit
M337->X:$0001B0,17,1      ; #3 Running-program bit
M338->X:$0001B0,18,1      ; #3 Open-loop-mode bit
M339->X:$0001B0,19,1      ; #3 Amplifier-enabled status bit
M340->Y:$0001C0,0,1       ; #3 Background in-position bit
M341->Y:$0001C0,1,1       ; #3 Warning-following error bit
M342->Y:$0001C0,2,1       ; #3 Fatal-following-error bit
M343->Y:$0001C0,3,1       ; #3 Amplifier-fault-error bit
M344->Y:$0001C0,13,1      ; #3 Foreground in-position bit
M345->Y:$0001C0,10,1      ; #3 Home-complete bit
M346->Y:$0001C0,6,1       ; #3 Integrated following error fault bit
M347->Y:$0001C0,5,1       ; #3 I2T fault bit
M348->Y:$0001C0,8,1       ; #3 Phasing error fault bit
M349->Y:$0001C0,9,1       ; #3 Phasing search-in-progress bit
; MACRO IC 0 Node 4 Flag Registers (usually used for Motor #3)
M350->X:$003444,0,24      ; MACRO IC 0 Node 4 flag status register
M351->Y:$003444,0,24      ; MACRO IC 0 Node 4 flag command register
M353->X:$003444,20,4      ; MACRO IC 0 Node 4 TUVW flags
M354->Y:$003444,14,1      ; MACRO IC 0 Node 4 amplifier enable flag
M355->X:$003444,15,1      ; MACRO IC 0 Node 4 node/amplifier fault flag
M356->X:$003444,16,1      ; MACRO IC 0 Node 4 home flag
M357->X:$003444,17,1      ; MACRO IC 0 Node 4 positive limit flag
M358->X:$003444,18,1      ; MACRO IC 0 Node 4 negative limit flag
M359->X:$003444,19,1      ; MACRO IC 0 Node 4 user flag
; Motor #3 Move Registers
M361->D:$000188           ; #3 Commanded position (1/[Ixx08*32] cts)
M362->D:$00018B           ; #3 Actual position (1/[Ixx08*32] cts)
M363->D:$0001C7           ; #3 Target (end) position (1/[Ixx08*32] cts)
M364->D:$0001CC           ; #3 Position bias (1/[Ixx08*32] cts)
M366->X:$00019D,0,24,S    ; #3 Actual velocity (1/[Ixx09*32] cts/cyc)
M367->D:$00018D           ; #3 Present master pos (1/[Ixx07*32] cts)
M368->X:$0001BF,8,16,S    ; #3 Filter Output (16-bit DAC bits)
M369->D:$000190           ; #3 Compensation correction (1/[Ixx08*32] cts)
M370->D:$0001B4           ; #3 Present phase position (including fraction)
M371->X:$0001B4,24,S      ; #3 Present phase position (counts *Ixx70)
M372->L:$0001D7           ; #3 Variable jog position/distance (cts)
M373->Y:$0001CE,0,24,S    ; #3 Encoder home capture position (cts)
M374->D:$0001EF           ; #3 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M375->X:$0001B9,8,16,S    ; #3 Actual quadrature current
M376->Y:$0001B9,8,16,S    ; #3 Actual direct current
M377->X:$0001BC,8,16,S    ; #3 Quadrature current-loop integrator output
M378->Y:$0001BC,8,16,S    ; #3 Direct current-loop integrator output
```

```
M379->X:$0001AE,8,16,S   ; #3 PID internal filter result (16-bit DAC bits)
M388->Y:$078211,0,12,U   ; IC 2 Ch 3 Compare A fractional count
M389->Y:$078210,0,12,U   ; IC 2 Ch 3 Compare B fractional count
; Motor #3 Axis Definition Registers
M391->L:$0001CF          ; #3 X/U/A/B/C-Axis scale factor (cts/unit)
M392->L:$0001D0          ; #3 Y/V-Axis scale factor (cts/unit)
M393->L:$0001D1          ; #3 Z/W-Axis scale factor (cts/unit)
M394->L:$0001D2          ; #3 Axis offset (cts)
; Servo IC 2 Channel 4 Registers (usually for Motor #4)
M401->X:$078219,0,24,S   ; ENC4 24-bit counter position
M402->Y:$07821A,8,16,S   ; OUT4A command value; DAC or PWM
M403->X:$07821B,0,24,S   ; ENC4 captured position
M404->Y:$07821B,8,16,S   ; OUT4B command value; DAC or PWM
M405->Y:$07821D,8,16,S   ; ADC4A input value
M406->Y:$07821E,8,16,S   ; ADC4B input value
M407->Y:$07821C,8,16,S   ; OUT4C command value; PFM or PWM
M408->Y:$07821F,0,24,S   ; ENC4 compare A position
M409->X:$07821F,0,24,S   ; ENC4 compare B position
M410->X:$07821E,0,24,S   ; ENC4 compare autoincrement value
M411->X:$07821D,11       ; ENC4 compare initial state write enable
M412->X:$07821D,12       ; ENC4 compare initial state
M414->X:$07821D,14       ; AENA4 output status
M415->X:$078218,19       ; USER4 flag input status
M416->X:$078218,9        ; ENC4 compare output value
M417->X:$078218,11       ; ENC4 capture flag
M418->X:$078218,8        ; ENC4 count error flag
M419->X:$078218,14       ; HMFL4 flag input status
M420->X:$078218,16       ; CHC4 input status
M421->X:$078218,17       ; PLIM4 flag input status
M422->X:$078218,18       ; MLIM4 flag input status
M423->X:$078218,15       ; FAULT4 flag input status
M424->X:$078218,20       ; Channel 4 W flag input status
M425->X:$078218,21       ; Channel 4 V flag input status
M426->X:$078218,22       ; Channel 4 U flag input status
M427->X:$078218,23       ; Channel 4 T flag input status
M428->X:$078218,20,4     ; Channel 4 TUVW inputs as 4-bit value
; Motor #4 Status Bits
M430->Y:$000240,11,1     ; #4 Stopped-on-position-limit bit
M431->X:$000230,21,1     ; #4 Positive-end-limit-set bit
M432->X:$000230,22,1     ; #4 Negative-end-limit-set bit
M433->X:$000230,13,1     ; #4 Desired-velocity-zero bit
M435->X:$000230,15,1     ; #4 Dwell-in-progress bit
M437->X:$000230,17,1     ; #4 Running-program bit
M438->X:$000230,18,1     ; #4 Open-loop-mode bit
M439->X:$000230,19,1     ; #4 Amplifier-enabled status bit
M440->Y:$000240,0,1      ; #4 Background in-position bit
M441->Y:$000240,1,1      ; #4 Warning-following error bit
```

```
M442->Y:$000240,2,1       ; #4 Fatal-following-error bit
M443->Y:$000240,3,1       ; #4 Amplifier-fault-error bit
M444->Y:$000240,13,1      ; #4 Foreground in-position bit
M445->Y:$000240,10,1      ; #4 Home-complete bit
M446->Y:$000240,6,1       ; #4 Integrated following error fault bit
M447->Y:$000240,5,1       ; #4 I2T fault bit
M448->Y:$000240,8,1       ; #4 Phasing error fault bit
M449->Y:$000240,9,1       ; #4 Phasing search-in-progress bit
; MACRO IC 0 Node 5 Flag Registers (usually used for Motor #4)
M450->X:$003445,0,24      ; MACRO IC 0 Node 5 flag status register
M451->Y:$003445,0,24      ; MACRO IC 0 Node 5 flag command register
M453->X:$003445,20,4      ; MACRO IC 0 Node 5 TUVW flags
M454->Y:$003445,14,1      ; MACRO IC 0 Node 5 amplifier enable flag
M455->X:$003445,15,1      ; MACRO IC 0 Node 5 node/amplifier fault flag
M456->X:$003445,16,1      ; MACRO IC 0 Node 5 home flag
M457->X:$003445,17,1      ; MACRO IC 0 Node 5 positive limit flag
M458->X:$003445,18,1      ; MACRO IC 0 Node 5 negative limit flag
M459->X:$003445,19,1      ; MACRO IC 0 Node 5 user flag
; Motor #4 Move Registers
M461->D:$000208           ; #4 Commanded position (1/[Ixx08*32] cts)
M462->D:$00020B           ; #4 Actual position (1/[Ixx08*32] cts)
M463->D:$000247           ; #4 Target (end) position (1/[Ixx08*32] cts)
M464->D:$00024C           ; #4 Position bias (1/[Ixx08*32] cts)
M466->X:$00021D,0,24,S    ; #4 Actual velocity (1/[Ixx09*32] cts/cyc)
M467->D:$00020D           ; #4 Present master pos (1/[Ixx07*32] cts)
M468->X:$00023F,8,16,S    ; #4 Filter Output (16-bit DAC bits)
M469->D:$000210           ; #4 Compensation correction (1/[Ixx08*32] cts)
M470->D:$000234           ; #4 Present phase position (including fraction)
M471->X:$000234,24,S      ; #4 Present phase position (counts *Ixx70)
M472->L:$000257           ; #4 Variable jog position/distance (cts)
M473->Y:$00024E,0,24,S    ; #4 Encoder home capture position (cts)
M474->D:$00026F           ; #4 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M475->X:$000239,8,16,S    ; #4 Actual quadrature current
M476->Y:$000239,8,16,S    ; #4 Actual direct current
M477->X:$00023C,8,16,S    ; #4 Quadrature current-loop integrator output
M478->Y:$00023C,8,16,S    ; #4 Direct current-loop integrator output
M479->X:$00022E,8,16,S    ; #4 PID internal filter result (16-bit DAC bits)
M488->Y:$078219,0,12,U    ; IC 2 Ch 4 Compare A fractional count
M489->Y:$078218,0,12,U    ; IC 2 Ch 4 Compare B fractional count
; Motor #4 Axis Definition Registers
M491->L:$00024F           ; #4 X/U/A/B/C-Axis scale factor (cts/unit)
M492->L:$000250           ; #4 Y/V-Axis scale factor (cts/unit)
M493->L:$000251           ; #4 Z/W-Axis scale factor (cts/unit)
M494->L:$000252           ; #4 Axis offset (cts)
; Servo IC 3 Channel 1 Registers (usually for Motor #5)
M501->X:$078301,0,24,S    ; ENC5 24-bit counter position
M502->Y:$078302,8,16,S    ; OUT5A command value; DAC or PWM
```

```
M503->X:$078303,0,24,S   ; ENC5 captured position
M504->Y:$078303,8,16,S   ; OUT5B command value; DAC or PWM
M505->Y:$078305,8,16,S   ; ADC5A input value
M506->Y:$078306,8,16,S   ; ADC5B input value
M507->Y:$078304,8,16,S   ; OUT5C command value; PFM or PWM
M508->Y:$078307,0,24,S   ; ENC5 compare A position
M509->X:$078307,0,24,S   ; ENC5 compare B position
M510->X:$078306,0,24,S   ; ENC5 compare autoincrement value
M511->X:$078305,11       ; ENC5 compare initial state write enable
M512->X:$078305,12       ; ENC5 compare initial state
M514->X:$078305,14       ; AENA5 output status
M515->X:$078300,19       ; USER5 flag input status
M516->X:$078300,9        ; ENC5 compare output value
M517->X:$078300,11       ; ENC5 capture flag
M518->X:$078300,8        ; ENC5 count error flag
M519->X:$078300,14       ; CHC5 input status
M520->X:$078300,16       ; HMFL5 flag input status
M521->X:$078300,17       ; PLIM5 flag input status
M522->X:$078300,18       ; MLIM5 flag input status
M523->X:$078300,15       ; FAULT5 flag input status
M524->X:$078300,20       ; Channel 5 W flag input status
M525->X:$078300,21       ; Channel 5 V flag input status
M526->X:$078300,22       ; Channel 5 U flag input status
M527->X:$078300,23       ; Channel 5 T flag input status
M528->X:$078300,20,4     ; Channel 5 TUVW inputs as 4-bit value
; Motor #5 Status Bits
M530->Y:$0002C0,11,1     ; #5 Stopped-on-position-limit bit
M531->X:$0002B0,21,1     ; #5 Positive-end-limit-set bit
M532->X:$0002B0,22,1     ; #5 Negative-end-limit-set bit
M533->X:$0002B0,13,1     ; #5 Desired-velocity-zero bit
M535->X:$0002B0,15,1     ; #5 Dwell-in-progress bit
M537->X:$0002B0,17,1     ; #5 Running-program bit
M538->X:$0002B0,18,1     ; #5 Open-loop-mode bit
M539->X:$0002B0,19,1     ; #5 Amplifier-enabled status bit
M540->Y:$0002C0,0,1      ; #5 Background in-position bit
M541->Y:$0002C0,1,1      ; #5 Warning-following error bit
M542->Y:$0002C0,2,1      ; #5 Fatal-following-error bit
M543->Y:$0002C0,3,1      ; #5 Amplifier-fault-error bit
M544->Y:$0002C0,13,1     ; #5 Foreground in-position bit
M545->Y:$0002C0,10,1     ; #5 Home-complete bit
M546->Y:$0002C0,6,1      ; #5 Integrated following error fault bit
M547->Y:$0002C0,5,1      ; #5 I2T fault bit
M548->Y:$0002C0,8,1      ; #5 Phasing error fault bit
M549->Y:$0002C0,9,1      ; #5 Phasing search-in-progress bit
```

```
; MACRO IC 0 Node 8 Flag Registers (usually used for Motor #5)
M550->X:$003448,0,24      ; MACRO IC 0 Node 8 flag status register
M551->Y:$003448,0,24      ; MACRO IC 0 Node 8 flag command register
M553->X:$003448,20,4      ; MACRO IC 0 Node 8 TUVW flags
M554->Y:$003448,14,1      ; MACRO IC 0 Node 8 amplifier enable flag
M555->X:$003448,15,1      ; MACRO IC 0 Node 8 node/amplifier fault flag
M556->X:$003448,16,1      ; MACRO IC 0 Node 8 home flag
M557->X:$003448,17,1      ; MACRO IC 0 Node 8 positive limit flag
M558->X:$003448,18,1      ; MACRO IC 0 Node 8 negative limit flag
M559->X:$003448,19,1      ; MACRO IC 0 Node 8 user flag
; Motor #5 Move Registers
M561->D:$000288       ; #5 Commanded position (1/[Ixx08*32] cts)
M562->D:$00028B           ; #5 Actual position (1/[Ixx08*32] cts)
M563->D:$0002C7           ; #5 Target (end) position (1/[Ixx08*32] cts)
M564->D:$0002CC           ; #5 Position bias (1/[Ixx08*32] cts)
M566->X:$00029D,0,24,S    ; #5 Actual velocity (1/[Ixx09*32] cts/cyc)
M567->D:$00028D           ; #5 Present master pos (1/[Ixx07*32] cts)
M568->X:$0002BF,8,16,S    ; #5 Filter Output (16-bit DAC bits)
M569->D:$000290           ; #5 Compensation correction (1/[Ixx08*32] cts)
M570->D:$0002B4           ; #5 Present phase position (including fraction)
M571->X:$0002B4,24,S      ; #5 Present phase position (counts *Ixx70)
M572->L:$0002D7           ; #5 Variable jog position/distance (cts)
M573->Y:$0002CE,0,24,S    ; #5 Encoder home capture position (cts)
M574->D:$0002EF           ; #5 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M575->X:$0002B9,8,16,S    ; #5 Actual quadrature current
M576->Y:$0002B9,8,16,S    ; #5 Actual direct current
M577->X:$0002BC,8,16,S    ; #5 Quadrature current-loop integrator output
M578->Y:$0002BC,8,16,S    ; #5 Direct current-loop integrator output
M579->X:$0002AE,8,16,S    ; #5 PID internal filter result (16-bit DAC bits)
M588->Y:$078301,0,12,U    ; IC 3 Ch 1 Compare A fractional count
M589->Y:$078300,0,12,U    ; IC 3 Ch 1 Compare B fractional count
; Motor #5 Axis Definition Registers
M591->L:$0002CF           ; #5 X/U/A/B/C-Axis scale factor (cts/unit)
M592->L:$0002D0           ; #5 Y/V-Axis scale factor (cts/unit)
M593->L:$0002D1           ; #5 Z/W-Axis scale factor (cts/unit)
M594->L:$0002D2           ; #5 Axis offset (cts)
; Servo IC 3 Channel 2 Registers (usually for Motor #6)
M601->X:$078309,0,24,S    ; ENC6 24-bit counter position
M602->Y:$07830A,8,16,S    ; OUT6A command value; DAC or PWM
M603->X:$07830B,0,24,S    ; ENC6 captured position
M604->Y:$07830B,8,16,S    ; OUT6B command value; DAC or PWM
M605->Y:$07830D,8,16,S    ; ADC6A input value
M606->Y:$07830E,8,16,S    ; ADC6B input value
M607->Y:$07830C,8,16,S    ; OUT6C command value; PFM or PWM
M608->Y:$07830F,0,24,S    ; ENC6 compare A position
M609->X:$07830F,0,24,S    ; ENC6 compare B position
M610->X:$07830E,0,24,S    ; ENC6 compare autoincrement value
```

```
M611->X:$07830D,11      ; ENC6 compare initial state write enable
M612->X:$07830D,12      ; ENC6 compare initial state
M614->X:$07830D,14      ; AENA6 output status
M615->X:$078308,19      ; USER6 flag input status
M616->X:$078308,9       ; ENC6 compare output value
M617->X:$078308,11      ; ENC6 capture flag
M618->X:$078308,8       ; ENC6 count error flag
M619->X:$078308,14      ; CHC6 input status
M620->X:$078308,16      ; HMFL6 flag input status
M621->X:$078308,17      ; PLIM6 flag input status
M622->X:$078308,18      ; MLIM6 flag input status
M623->X:$078308,15      ; FAULT6 flag input status
M624->X:$078308,20      ; Channel 6 W flag input status
M625->X:$078308,21      ; Channel 6 V flag input status
M626->X:$078308,22      ; Channel 6 U flag input status
M627->X:$078308,23      ; Channel 6 T flag input status
M628->X:$078308,20,4    ; Channel 6 TUVW inputs as 4-bit value
; Motor #6 Status Bits
M630->Y:$000340,11,1    ; #6 Stopped-on-position-limit bit
M631->X:$000330,21,1    ; #6 Positive-end-limit-set bit
M632->X:$000330,22,1    ; #6 Negative-end-limit-set bit
M633->X:$000330,13,1    ; #6 Desired-velocity-zero bit
M635->X:$000330,15,1    ; #6 Dwell-in-progress bit
M637->X:$000330,17,1    ; #6 Running-program bit
M638->X:$000330,18,1    ; #6 Open-loop-mode bit
M639->X:$000330,19,1    ; #6 Amplifier-enabled status bit
M640->Y:$000340,0,1     ; #6 Background in-position bit
M641->Y:$000340,1,1     ; #6 Warning-following error bit
M642->Y:$000340,2,1     ; #6 Fatal-following-error bit
M643->Y:$000340,3,1     ; #6 Amplifier-fault-error bit
M644->Y:$000340,13,1    ; #6 Foreground in-position bit
M645->Y:$000340,10,1    ; #6 Home-complete bit
M646->Y:$000340,6,1     ; #6 Integrated following error fault bit
M647->Y:$000340,5,1     ; #6 I2T fault bit
M648->Y:$000340,8,1     ; #6 Phasing error fault bit
M649->Y:$000340,9,1     ; #6 Phasing search-in-progress bit
; MACRO IC 0 Node 9 Flag Registers (usually used for Motor #6)
M650->X:$003449,0,24    ; MACRO IC 0 Node 9 flag status register
M651->Y:$003449,0,24    ; MACRO IC 0 Node 9 flag command register
M653->X:$003449,20,4    ; MACRO IC 0 Node 9 TUVW flags
M654->Y:$003449,14,1    ; MACRO IC 0 Node 9 amplifier enable flag
M655->X:$003449,15,1    ; MACRO IC 0 Node 9 node/amplifier fault flag
M656->X:$003449,16,1    ; MACRO IC 0 Node 9 home flag
M657->X:$003449,17,1    ; MACRO IC 0 Node 9 positive limit flag
M658->X:$003449,18,1    ; MACRO IC 0 Node 9 negative limit flag
M659->X:$003449,19,1    ; MACRO IC 0 Node 9 user flag
```

```
; Motor #6 Move Registers
M661->D:$000308           ; #6 Commanded position (1/[Ixx08*32] cts)
M662->D:$00030B           ; #6 Actual position (1/[Ixx08*32] cts)
M663->D:$000347           ; #6 Target (end) position (1/[Ixx08*32] cts)
M664->D:$00034C           ; #6 Position bias (1/[Ixx08*32] cts)
M666->X:$00031D,0,24,S    ; #6 Actual velocity (1/[Ixx09*32] cts/cyc)
M667->D:$00030D           ; #6 Present master pos (1/[Ixx07*32] cts)
M668->X:$00033F,8,16,S    ; #6 Filter Output (16-bit DAC bits)
M669->D:$000310           ; #6 Compensation correction (1/[Ixx08*32] cts)
M670->D:$000334           ; #6 Present phase position (including fraction)
M671->X:$000334,24,S      ; #6 Present phase position (counts *Ixx70)
M672->L:$000357           ; #6 Variable jog position/distance (cts)
M673->Y:$00034E,0,24,S    ; #6 Encoder home capture position (cts)
M674->D:$00036F           ; #6 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M675->X:$000339,8,16,S    ; #6 Actual quadrature current
M676->Y:$000339,8,16,S    ; #6 Actual direct current
M677->X:$00033C,8,16,S    ; #6 Quadrature current-loop integrator output
M678->Y:$00033C,8,16,S    ; #6 Direct current-loop integrator output
M679->X:$00032E,8,16,S    ; #6 PID internal filter result (16-bit DAC bits)
M688->Y:$078309,0,12,U    ; IC 3 Ch 2 Compare A fractional count
M689->Y:$078308,0,12,U    ; IC 3 Ch 2 Compare B fractional count
; Motor #6 Axis Definition Registers
M691->L:$00034F           ; #6 X/U/A/B/C-Axis scale factor (cts/unit)
M692->L:$000350           ; #6 Y/V-Axis scale factor (cts/unit)
M693->L:$000351           ; #6 Z/W-Axis scale factor (cts/unit)
M694->L:$000352           ; #6 Axis offset (cts)
; Servo IC 3 Channel 3 Registers (usually for Motor #7)
M701->X:$078311,0,24,S    ; ENC7 24-bit counter position
M702->Y:$078312,8,16,S    ; OUT7A command value; DAC or PWM
M703->X:$078313,0,24,S    ; ENC7 captured position
M704->Y:$078313,8,16,S    ; OUT7B command value; DAC or PWM
M705->Y:$078315,8,16,S    ; ADC7A input value
M706->Y:$078316,8,16,S    ; ADC7B input value
M707->Y:$078314,8,16,S    ; OUT7C command value; PFM or PWM
M708->Y:$078317,0,24,S    ; ENC7 compare A position
M709->X:$078317,0,24,S    ; ENC7 compare B position
M710->X:$078316,0,24,S    ; ENC7 compare autoincrement value
M711->X:$078315,11        ; ENC7 compare initial state write enable
M712->X:$078315,12        ; ENC7 compare initial state
M714->X:$078315,14        ; AENA7 output status
M715->X:$078310,19        ; CHC7 input status
M716->X:$078310,9         ; ENC7 compare output value
M717->X:$078310,11        ; ENC7 capture flag
M718->X:$078310,8         ; ENC7 count error flag
M719->X:$078310,14        ; CHC7 input status
M720->X:$078310,16        ; HMFL7 flag input status
M721->X:$078310,17        ; PLIM7 flag input status
```

```
M722->X:$078310,18        ; MLIM7 flag input status
M723->X:$078310,15        ; FAULT7 flag input status
M724->X:$078310,20        ; Channel 7 W flag input status
M725->X:$078310,21        ; Channel 7 V flag input status
M726->X:$078310,22        ; Channel 7 U flag input status
M727->X:$078310,23        ; Channel 7 T flag input status
M728->X:$078310,20,4      ; Channel 7 TUVW inputs as 4-bit value
; Motor #7 Status Bits
M730->Y:$0003C0,11,1      ; #7 Stopped-on-position-limit bit
M731->X:$0003B0,21,1      ; #7 Positive-end-limit-set bit
M732->X:$0003B0,22,1      ; #7 Negative-end-limit-set bit
M733->X:$0003B0,13,1      ; #7 Desired-velocity-zero bit
M735->X:$0003B0,15,1      ; #7 Dwell-in-progress bit
M737->X:$0003B0,17,1      ; #7 Running-program bit
M738->X:$0003B0,18,1      ; #7 Open-loop-mode bit
M739->X:$0003B0,19,1      ; #7 Amplifier-enabled status bit
M740->Y:$0003C0,0,1       ; #7 Background in-position bit
M741->Y:$0003C0,1,1       ; #7 Warning-following error bit
M742->Y:$0003C0,2,1       ; #7 Fatal-following-error bit
M743->Y:$0003C0,3,1       ; #7 Amplifier-fault-error bit
M744->Y:$0003C0,13,1      ; #7 Foreground in-position bit
M745->Y:$0003C0,10,1      ; #7 Home-complete bit
M746->Y:$0003C0,6,1       ; #7 Integrated following error fault bit
M747->Y:$0003C0,5,1       ; #7 I2T fault bit
M748->Y:$0003C0,8,1       ; #7 Phasing error fault bit
M749->Y:$0003C0,9,1       ; #7 Phasing search-in-progress bit
; MACRO IC 0 Node 12 Flag Registers (usually used for Motor #7)
M750->X:$00344C,0,24      ; MACRO IC 0 Node 12 flag status register
M751->Y:$00344C,0,24      ; MACRO IC 0 Node 12 flag command register
M753->X:$00344C,20,4      ; MACRO IC 0 Node 12 TUVW flags
M754->Y:$00344C,14,1      ; MACRO IC 0 Node 12 amplifier enable flag
M755->X:$00344C,15,1      ; MACRO IC 0 Node 12 node/amplifier fault flag
M756->X:$00344C,16,1      ; MACRO IC 0 Node 12 home flag
M757->X:$00344C,17,1      ; MACRO IC 0 Node 12 positive limit flag
M758->X:$00344C,18,1      ; MACRO IC 0 Node 12 negative limit flag
M759->X:$00344C,19,1      ; MACRO IC 0 Node 12 user flag
; Motor #7 Move Registers
M761->D:$000388           ; #7 Commanded position (1/[Ixx08*32] cts)
M762->D:$00038B           ; #7 Actual position (1/[Ixx08*32] cts)
M763->D:$0003C7           ; #7 Target (end) position (1/[Ixx08*32] cts)
M764->D:$0003CC           ; #7 Position bias (1/[Ixx08*32] cts)
M766->X:$00039D,0,24,S    ; #7 Actual velocity (1/[Ixx09*32] cts/cyc)
M767->D:$00038D           ; #7 Present master pos (1/[Ixx07*32] cts)
M768->X:$0003BF,8,16,S    ; #7 Filter Output (16-bit DAC bits)
M769->D:$000390           ; #7 Compensation correction (1/[Ixx08*32] cts)
M770->D:$0003B4           ; #7 Present phase position (including fraction)
M771->X:$0003B4,24,S      ; #7 Present phase position (counts *Ixx70)
```

```
M772->L:$0003D7          ; #7 Variable jog position/distance (cts)
M773->Y:$0003CE,0,24,S   ; #7 Encoder home capture position (cts)
M774->D:$0003EF          ; #7 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M775->X:$0003B9,8,16,S   ; #7 Actual quadrature current
M776->Y:$0003B9,8,16,S   ; #7 Actual direct current
M777->X:$0003BC,8,16,S   ; #7 Quadrature current-loop integrator output
M778->Y:$0003BC,8,16,S   ; #7 Direct current-loop integrator output
M779->X:$0003AE,8,16,S   ; #7 PID internal filter result (16-bit DAC bits)
M788->Y:$078311,0,12,U   ; IC 3 Ch 3 Compare A fractional count
M789->Y:$078310,0,12,U   ; IC 3 Ch 3 Compare B fractional count
; Motor #7 Axis Definition Registers
M791->L:$0003CF          ; #7 X/U/A/B/C-Axis scale factor (cts/unit)
M792->L:$0003D0          ; #7 Y/V-Axis scale factor (cts/unit)
M793->L:$0003D1          ; #7 Z/W-Axis scale factor (cts/unit)
M794->L:$0003D2          ; #7 Axis offset (cts)
; Servo IC 3 Channel 4 Registers (usually for Motor #8)
M801->X:$078319,0,24,S   ; ENC8 24-bit counter position
M802->Y:$07831A,8,16,S   ; OUT8A command value; DAC or PWM
M803->X:$07831B,0,24,S   ; ENC8 captured position
M804->Y:$07831B,8,16,S   ; OUT8B command value; DAC or PWM
M805->Y:$07831D,8,16,S   ; ADC8A input value
M806->Y:$07831E,8,16,S   ; ADC8B input value
M807->Y:$07831C,8,16,S   ; OUT8C command value; PFM or PWM
M808->Y:$07831F,0,24,S   ; ENC8 compare A position
M809->X:$07831F,0,24,S   ; ENC8 compare B position
M810->X:$07831E,0,24,S   ; ENC8 compare autoincrement value
M811->X:$07831D,11       ; ENC8 compare initial state write enable
M812->X:$07831D,12       ; ENC8 compare initial state
M814->X:$07831D,14       ; AENA8 output status
M815->X:$078318,19       ; USER8 flag input status
M816->X:$078318,9        ; ENC8 compare output value
M817->X:$078318,11       ; ENC8 capture flag
M818->X:$078318,8        ; ENC8 count error flag
M819->X:$078318,14       ; CHC8 input status
M820->X:$078318,16       ; HMFL8 flag input status
M821->X:$078318,17       ; PLIM8 flag input status
M822->X:$078318,18       ; MLIM8 flag input status
M823->X:$078318,15       ; FAULT8 flag input status
M824->X:$078318,20       ; Channel 8 W flag input status
M825->X:$078318,21       ; Channel 8 V flag input status
M826->X:$078318,22       ; Channel 8 U flag input status
M827->X:$078318,23       ; Channel 8 T flag input status
M828->X:$078318,20,4     ; Channel 8 TUVW inputs as 4-bit value
; Motor #8 Status Bits
M830->Y:$000440,11,1     ; #8 Stopped-on-position-limit bit
M831->X:$000430,21,1     ; #8 Positive-end-limit-set bit
M832->X:$000430,22,1     ; #8 Negative-end-limit-set bit
```

```
M833->X:$000430,13,1      ; #8 Desired-velocity-zero bit
M835->X:$000430,15,1      ; #8 Dwell-in-progress bit
M837->X:$000430,17,1      ; #8 Running-program bit
M838->X:$000430,18,1      ; #8 Open-loop-mode bit
M839->X:$000430,19,1      ; #8 Amplifier-enabled status bit
M840->Y:$000440,0,1       ; #8 Background in-position bit
M841->Y:$000440,1,1       ; #8 Warning-following error bit
M842->Y:$000440,2,1       ; #8 Fatal-following-error bit
M843->Y:$000440,3,1       ; #8 Amplifier-fault-error bit
M844->Y:$000440,13,1      ; #8 Foreground in-position bit
M845->Y:$000440,10,1      ; #8 Home-complete bit
M846->Y:$000440,6,1       ; #8 Integrated following error fault bit
M847->Y:$000440,5,1       ; #8 I2T fault bit
M848->Y:$000440,8,1       ; #8 Phasing error fault bit
M849->Y:$000440,9,1       ; #8 Phasing search-in-progress bit
; MACRO IC 0 Node 13 Flag Registers (usually used for Motor #8)
M850->X:$00344D,0,24      ; MACRO IC 0 Node 13 flag status register
M851->Y:$00344D,0,24      ; MACRO IC 0 Node 13 flag command register
M853->X:$00344D,20,4      ; MACRO IC 0 Node 13 TUVW flags
M854->Y:$00344D,14,1      ; MACRO IC 0 Node 13 amplifier enable flag
M855->X:$00344D,15,1      ; MACRO IC 0 Node 13 node/amplifier fault flag
M856->X:$00344D,16,1      ; MACRO IC 0 Node 13 home flag
M857->X:$00344D,17,1      ; MACRO IC 0 Node 13 positive limit flag
M858->X:$00344D,18,1      ; MACRO IC 0 Node 13 negative limit flag
M859->X:$00344D,19,1      ; MACRO IC 0 Node 13 user flag
; Motor #8 Move Registers
M861->D:$000408           ; #8 Commanded position (1/[Ixx08*32] cts)
M862->D:$00040B           ; #8 Actual position (1/[Ixx08*32] cts)
M863->D:$000447           ; #8 Target (end) position (1/[Ixx08*32] cts)
M864->D:$00044C           ; #8 Position bias (1/[Ixx08*32] cts)
M866->X:$00041D,0,24,S    ; #8 Actual velocity (1/[Ixx09*32] cts/cyc)
M867->D:$00040D           ; #8 Present master pos (1/[Ixx07*32] cts)
M868->X:$00043F,8,16,S    ; #8 Filter Output (16-bit DAC bits)
M869->D:$000410           ; #8 Compensation correction 1/[Ixx08*32] cts)
M870->D:$000434           ; #8 Present phase position (including fraction)
M871->X:$000434,24,S      ; #8 Present phase position (counts *Ixx70)
M872->L:$000457           ; #8 Variable jog position/distance (cts)
M873->Y:$00044E,0,24,S    ; #8 Encoder home capture position (cts)
M874->D:$00046F           ; #8 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M875->X:$000439,8,16,S    ; #8 Actual quadrature current
M876->Y:$000439,8,16,S    ; #8 Actual direct current
M877->X:$00043C,8,16,S    ; #8 Quadrature current-loop integrator output
M878->Y:$00043C,8,16,S    ; #8 Direct current-loop integrator output
M879->X:$00042E,8,16,S    ; #8 PID internal filter result (16-bit DAC bits)
M888->Y:$078319,0,12,U    ; IC 3 Ch 4 Compare A fractional count
M889->Y:$078318,0,12,U    ; IC 3 Ch 4 Compare B fractional count
```

```
; Motor #8 Axis Definition Registers
M891->L:$00044F          ; #8 X/U/A/B/C-Axis scale factor (cts/unit)
M892->L:$000450          ; #8 Y/V-Axis scale factor (cts/unit)
M893->L:$000451          ; #8 Z/W-Axis scale factor (cts/unit)
M894->L:$000452          ; #8 Axis offset (cts)
; Servo IC 4 Channel 1 Registers (usually for Motor #9)
M901->X:$079201,0,24,S   ; ENC1 24-bit counter position
M902->Y:$079202,8,16,S   ; OUT1A command value; DAC or PWM
M903->X:$079203,0,24,S   ; ENC1 captured position
M904->Y:$079203,8,16,S   ; OUT1B command value; DAC or PWM
M905->Y:$079205,8,16,S   ; ADC1A input value
M906->Y:$079206,8,16,S   ; ADC1B input value
M907->Y:$079204,8,16,S   ; OUT1C command value; PFM or PWM
M908->Y:$079207,0,24,S   ; ENC1 compare A position
M909->X:$079207,0,24,S   ; ENC1 compare B position
M910->X:$079206,0,24,S   ; ENC1 compare autoincrement value
M911->X:$079205,11       ; ENC1 compare initial state write enable
M912->X:$079205,12       ; ENC1 compare initial state
M914->X:$079205,14       ; AENA1 output status
M915->X:$079200,19       ; USER1 flag input status
M916->X:$079200,9        ; ENC1 compare output value
M917->X:$079200,11       ; ENC1 capture flag
M918->X:$079200,8        ; ENC1 count error flag
M919->X:$079200,14       ; CHC1 input status
M920->X:$079200,16       ; HMFL1 flag input status
M921->X:$079200,17       ; PLIM1 flag input status
M922->X:$079200,18       ; MLIM1 flag input status
M923->X:$079200,15       ; FAULT1 flag input status
M924->X:$079200,20       ; Channel 1 W flag input status
M925->X:$079200,21       ; Channel 1 V flag input status
M926->X:$079200,22       ; Channel 1 U flag input status
M927->X:$079200,23       ; Channel 1 T flag input status
M928->X:$079200,20,4     ; Channel 1 TUVW inputs as 4-bit value
; Motor #9 Status Bits
M930->Y:$0004C0,11,1     ; #9 Stopped-on-position-limit bit
M931->X:$0004B0,21,1     ; #9 Positive-end-limit-set bit
M932->X:$0004B0,22,1     ; #9 Negative-end-limit-set bit
M933->X:$0004B0,13,1     ; #9 Desired-velocity-zero bit
M935->X:$0004B0,15,1     ; #9 Dwell-in-progress bit
M937->X:$0004B0,17,1     ; #9 Running-program bit
M938->X:$0004B0,18,1     ; #9 Open-loop-mode bit
M939->X:$0004B0,19,1     ; #9 Amplifier-enabled status bit
M940->Y:$0004C0,0,1      ; #9 Background in-position bit
M941->Y:$0004C0,1,1      ; #9 Warning-following error bit
M942->Y:$0004C0,2,1      ; #9 Fatal-following-error bit
M943->Y:$0004C0,3,1      ; #9 Amplifier-fault-error bit
M944->Y:$0004C0,13,1     ; #9 Foreground in-position bit
```

```
M945->Y:$0004C0,10,1      ; #9 Home-complete bit
M946->Y:$0004C0,6,1       ; #9 Integrated following error fault bit
M947->Y:$0004C0,5,1       ; #9 I2T fault bit
M948->Y:$0004C0,8,1       ; #9 Phasing error fault bit
M949->Y:$0004C0,9,1       ; #9 Phasing search-in-progress bit
; MACRO IC 1 Node 0 Flag Registers (usually used for Motor #9)
M950->X:$003450,0,24      ; MACRO IC 1 Node 0 flag status register
M951->Y:$003450,0,24      ; MACRO IC 1 Node 0 flag command register
M953->X:$003450,20,4      ; MACRO IC 1 Node 0 TUVW flags
M954->Y:$003450,14,1      ; MACRO IC 1 Node 0 amplifier enable flag
M955->X:$003450,15,1      ; MACRO IC 1 Node 0 node/amplifier fault flag
M956->X:$003450,16,1      ; MACRO IC 1 Node 0 home flag
M957->X:$003450,17,1      ; MACRO IC 1 Node 0 positive limit flag
M958->X:$003450,18,1      ; MACRO IC 1 Node 0 negative limit flag
M959->X:$003450,19,1      ; MACRO IC 1 Node 0 user flag
; Motor #9 Move Registers
M961->D:$000488           ; #9 Commanded position (1/[Ixx08*32] cts)
M962->D:$00048B           ; #9 Actual position (1/[Ixx08*32] cts)
M963->D:$0004C7           ; #9 Target (end) position (1/[Ixx08*32] cts)
M964->D:$0004CC           ; #9 Position bias (1/[Ixx08*32] cts)
M966->X:$00049D,0,24,S    ; #9 Actual velocity (1/[Ixx09*32] cts/cyc)
M967->D:$00048D           ; #9 Present master pos (1/[Ixx07*32] cts)
M968->X:$0004BF,8,16,S    ; #9 Filter Output (16-bit DAC bits)
M969->D:$000490           ; #9 Compensation correction (1/[Ixx08*32] cts)
M970->D:$0004B4           ; #9 Present phase position (including fraction)
M971->X:$0004B4,24,S      ; #9 Present phase position (counts *Ixx70)
M972->L:$0004D7           ; #9 Variable jog position/distance (cts)
M973->Y:$0004CE,0,24,S    ; #9 Encoder home capture position (cts)
M974->D:$0004EF           ; #9 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M975->X:$0004B9,8,16,S    ; #9 Actual quadrature current
M976->Y:$0004B9,8,16,S    ; #9 Actual direct current
M977->X:$0004BC,8,16,S    ; #9 Quadrature current-loop integrator output
M978->Y:$0004BC,8,16,S    ; #9 Direct current-loop integrator output
M979->X:$0004AE,8,16,S    ; #9 PID internal filter result (16-bit DAC bits)
M988->Y:$079201,0,12,U    ; IC 4 Ch 1 Compare A fractional count
M989->Y:$079200,0,12,U    ; IC 4 Ch 1 Compare B fractional count
; Motor #9 Axis Definition Registers
M991->L:$0004CF           ; #9 X/U/A/B/C-Axis scale factor (cts/unit)
M992->L:$0004D0           ; #9 Y/V-Axis scale factor (cts/unit)
M993->L:$0004D1           ; #9 Z/W-Axis scale factor (cts/unit)
M994->L:$0004D2           ; #9 Axis offset (cts)
; Servo IC 4 Channel 2 Registers (usually for Motor #10)
M1001->X:$079209,0,24,S   ; ENC2 24-bit counter position
M1002->Y:$07920A,8,16,S   ; OUT2A command value; DAC or PWM
M1003->X:$07920B,0,24,S   ; ENC2 captured position
M1004->Y:$07920B,8,16,S   ; OUT2B command value; DAC or PWM
M1005->Y:$07920D,8,16,S   ; ADC2A input value
```

```
M1006->Y:$07920E,8,16,S  ; ADC2B input value
M1007->Y:$07920C,8,16,S  ; OUT2C command value; PFM or PWM
M1008->Y:$07920F,0,24,S  ; ENC2 compare A position
M1009->X:$07920F,0,24,S  ; ENC2 compare B position
M1010->X:$07920E,0,24,S  ; ENC2 compare autoincrement value
M1011->X:$07920D,11      ; ENC2 compare initial state write enable
M1012->X:$07920D,12      ; ENC2 compare initial state
M1014->X:$07920D,14      ; AENA2 output status
M1015->X:$079208,19      ; USER2 flag input status
M1016->X:$079208,9       ; ENC2 compare output value
M1017->X:$079208,11      ; ENC2 capture flag
M1018->X:$079208,8       ; ENC2 count error flag
M1019->X:$079208,14      ; CHC2 input status
M1020->X:$079208,16      ; HMFL2 flag input status
M1021->X:$079208,17      ; PLIM2 flag input status
M1022->X:$079208,18      ; MLIM2 flag input status
M1023->X:$079208,15      ; FAULT2 flag input status
M1024->X:$079208,20      ; Channel 2 W flag input status
M1025->X:$079208,21      ; Channel 2 V flag input status
M1026->X:$079208,22      ; Channel 2 U flag input status
M1027->X:$079208,23      ; Channel 2 T flag input status
M1028->X:$079208,20,4    ; Channel 2 TUVW inputs as 4-bit value
; Motor #10 Status Bits
M1030->Y:$000540,11,1    ; #10 Stopped-on-position-limit bit
M1031->X:$000530,21,1    ; #10 Positive-end-limit-set bit
M1032->X:$000530,22,1    ; #10 Negative-end-limit-set bit
M1033->X:$000530,13,1    ; #10 Desired-velocity-zero bit
M1035->X:$000530,15,1    ; #10 Dwell-in-progress bit
M1037->X:$000530,17,1    ; #10 Running-program bit
M1038->X:$000530,18,1    ; #10 Open-loop-mode bit
M1039->X:$000530,19,1    ; #10 Amplifier-enabled status bit
M1040->Y:$000540,0,1     ; #10 Background in-position bit
M1041->Y:$000540,1,1     ; #10 Warning-following error bit
M1042->Y:$000540,2,1     ; #10 Fatal-following-error bit
M1043->Y:$000540,3,1     ; #10 Amplifier-fault-error bit
M1044->Y:$000540,13,1    ; #10 Foreground in-position bit
M1045->Y:$000540,10,1    ; #10 Home-complete bit
M1046->Y:$000540,6,1     ; #10 Integrated following error fault bit
M1047->Y:$000540,5,1     ; #10 I2T fault bit
M1048->Y:$000540,8,1     ; #10 Phasing error fault bit
M1049->Y:$000540,9,1     ; #10 Phasing search-in-progress bit
; MACRO IC 1 Node 1 Flag Registers (usually used for Motor #10)
M1050->X:$003451,0,24    ; MACRO IC 1 Node 1 flag status register
M1051->Y:$003451,0,24    ; MACRO IC 1 Node 1 flag command register
M1053->X:$003451,20,4    ; MACRO IC 1 Node 1 TUVW flags
M1054->Y:$003451,14,1    ; MACRO IC 1 Node 1 amplifier enable flag
M1055->X:$003451,15,1    ; MACRO IC 1 Node 1 node/amplifier fault flag
```

```
M1056->X:$003451,16,1    ; MACRO IC 1 Node 1 home flag
M1057->X:$003451,17,1    ; MACRO IC 1 Node 1 positive limit flag
M1058->X:$003451,18,1    ; MACRO IC 1 Node 1 negative limit flag
M1059->X:$003451,19,1    ; MACRO IC 1 Node 1 user flag
; Motor #10 Move Registers
M1061->D:$000508          ; #10 Commanded position (1/[Ixx08*32] cts)
M1062->D:$00050B          ; #10 Actual position (1/[Ixx08*32] cts)
M1063->D:$000547          ; #10 Target (end) position (1/[Ixx08*32] cts)
M1064->D:$00054C          ; #10 Position bias (1/[Ixx08*32] cts)
M1066->X:$00051D,0,24,S   ; #10 Actual velocity (1/[Ixx09*32] cts/cyc)
M1067->D:$00050D          ; #10 Present master pos (1/[Ixx07*32] cts)
M1068->X:$00053F,8,16,S   ; #10 Filter Output (16-bit DAC bits)
M1069->D:$000510          ; #10 Compensation correction (1/[Ixx08*32] cts)
M1070->D:$000534          ; #10 Present phase position (including fraction)
M1071->X:$000534,24,S     ; #10 Present phase position (counts *Ixx70)
M1072->L:$000557          ; #10 Variable jog position/distance (cts)
M1073->Y:$00054E,0,24,S   ; #10 Encoder home capture position (cts)
M1074->D:$00056F          ; #10 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1075->X:$000539,8,16,S   ; #10 Actual quadrature current
M1076->Y:$000539,8,16,S   ; #10 Actual direct current
M1077->X:$00053C,8,16,S   ; #10 Quadrature current-loop integrator output
M1078->Y:$00053C,8,16,S   ; #10 Direct current-loop integrator output
M1079->X:$00052E,8,16,S   ; #10 PID internal filter result (16-bit DAC bits)
M1088->Y:$079209,0,12,U   ; IC 4 Ch 2 Compare A fractional count
M1089->Y:$079208,0,12,U   ; IC 4 Ch 2 Compare B fractional count
; Motor #10 Axis Definition Registers
M1091->L:$00054F          ; #10 X/U/A/B/C-Axis scale factor (cts/unit)
M1092->L:$000550          ; #10 Y/V-Axis scale factor (cts/unit)
M1093->L:$000551          ; #10 Z/W-Axis scale factor (cts/unit)
M1094->L:$000552          ; #10 Axis offset (cts)
; Servo IC 4 Channel 3 Registers (usually for Motor #11)
M1101->X:$079211,0,24,S    ; ENC3 24-bit counter position
M1102->Y:$079212,8,16,S    ; OUT3A command value; DAC or PWM
M1103->X:$079213,0,24,S    ; ENC3 captured position
M1104->Y:$079213,8,16,S    ; OUT3B command value; DAC or PWM
M1105->Y:$079215,8,16,S    ; ADC3A input value
M1106->Y:$079216,8,16,S    ; ADC3B input value
M1107->Y:$079214,8,16,S    ; OUT3C command value; PFM or PWM
M1108->Y:$079217,0,24,S    ; ENC3 compare A position
M1109->X:$079217,0,24,S    ; ENC3 compare B position
M1110->X:$079216,0,24,S    ; ENC3 compare autoincrement value
M1111->X:$079215,11        ; ENC3 compare initial state write enable
M1112->X:$079215,12        ; ENC3 compare initial state
M1114->X:$079215,14        ; AENA3 output status
M1115->X:$079210,19        ; USER3 flag input status
M1116->X:$079210,9         ; ENC3 compare output value
M1117->X:$079210,11        ; ENC3 capture flag
```

```
M1118->X:$079210,8          ; ENC3 count error flag
M1119->X:$079210,14         ; CHC3 input status
M1120->X:$079210,16         ; HMFL3 flag input status
M1121->X:$079210,17         ; PLIM3 flag input status
M1122->X:$079210,18         ; MLIM3 flag input status
M1123->X:$079210,15         ; FAULT3 flag input status
M1124->X:$079210,20         ; Channel 3 W flag input status
M1125->X:$079210,21         ; Channel 3 V flag input status
M1126->X:$079210,22         ; Channel 3 U flag input status
M1127->X:$079210,23         ; Channel 3 T flag input status
M1128->X:$079210,20,4       ; Channel 3 TUVW inputs as 4-bit value
; Motor #11 Status Bits
M1130->Y:$0005C0,11,1     ; #11 Stopped-on-position-limit bit
M1131->X:$0005B0,21,1     ; #11 Positive-end-limit-set bit
M1132->X:$0005B0,22,1     ; #11 Negative-end-limit-set bit
M1133->X:$0005B0,13,1     ; #11 Desired-velocity-zero bit
M1135->X:$0005B0,15,1     ; #11 Dwell-in-progress bit
M1137->X:$0005B0,17,1     ; #11 Running-program bit
M1138->X:$0005B0,18,1     ; #11 Open-loop-mode bit
M1139->X:$0005B0,19,1     ; #11 Amplifier-enabled status bit
M1140->Y:$0005C0,0,1      ; #11 Background in-position bit
M1141->Y:$0005C0,1,1      ; #11 Warning-following error bit
M1142->Y:$0005C0,2,1      ; #11 Fatal-following-error bit
M1143->Y:$0005C0,3,1      ; #11 Amplifier-fault-error bit
M1144->Y:$0005C0,13,1     ; #11 Foreground in-position bit
M1145->Y:$0005C0,10,1     ; #11 Home-complete bit
M1146->Y:$0005C0,6,1      ; #11 Integrated following error fault bit
M1147->Y:$0005C0,5,1      ; #11 I2T fault bit
M1148->Y:$0005C0,8,1      ; #11 Phasing error fault bit
M1149->Y:$0005C0,9,1      ; #11 Phasing search-in-progress bit
; MACRO IC 1 Node 4 Flag Registers (usually used for Motor #11)
M1150->X:$003454,0,24     ; MACRO IC 1 Node 4 flag status register
M1151->Y:$003454,0,24     ; MACRO IC 1 Node 4 flag command register
M1153->X:$003454,20,4     ; MACRO IC 1 Node 4 TUVW flags
M1154->Y:$003454,14,1     ; MACRO IC 1 Node 4 amplifier enable flag
M1155->X:$003454,15,1     ; MACRO IC 1 Node 4 node/amplifier fault flag
M1156->X:$003454,16,1     ; MACRO IC 1 Node 4 home flag
M1157->X:$003454,17,1     ; MACRO IC 1 Node 4 positive limit flag
M1158->X:$003454,18,1     ; MACRO IC 1 Node 4 negative limit flag
M1159->X:$003454,19,1     ; MACRO IC 1 Node 4 user flag
; Motor #11 Move Registers
M1161->D:$000588            ; #11 Commanded position (1/[Ixx08*32] cts)
M1162->D:$00058B            ; #11 Actual position (1/[Ixx08*32] cts)
M1163->D:$0005C7            ; #11 Target (end) position (1/[Ixx08*32] cts)
M1164->D:$0005CC            ; #11 Position bias (1/[Ixx08*32] cts)
M1166->X:$00059D,0,24,S   ; #11 Actual velocity (1/[Ixx09*32] cts/cyc)
M1167->D:$00058D            ; #11 Present master pos (1/[Ixx07*32] cts)
```

```
M1168->X:$0005BF,8,16,S   ; #11 Filter Output (16-bit DAC bits)
M1169->D:$000590          ; #11 Compensation correction (1/[Ixx08*32] cts)
M1170->D:$0005B4          ; #11 Present phase position (including fraction)
M1171->X:$0005B4,24,S     ; #11 Present phase position (counts *Ixx70)
M1172->L:$0005D7          ; #11 Variable jog position/distance (cts)
M1173->Y:$0005CE,0,24,S   ; #11 Encoder home capture position (cts)
M1174->D:$0005EF          ; #11 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1175->X:$0005B9,8,16,S   ; #11 Actual quadrature current
M1176->Y:$0005B9,8,16,S   ; #11 Actual direct current
M1177->X:$0005BC,8,16,S   ; #11 Quadrature current-loop integrator output
M1178->Y:$0005BC,8,16,S   ; #11 Direct current-loop integrator output
M1179->X:$0005AE,8,16,S   ; #11 PID internal filter result (16-bit DAC bits)
M1188->Y:$079211,0,12,U   ; IC 4 Ch 3 Compare A fractional count
M1189->Y:$079210,0,12,U   ; IC 4 Ch 3 Compare B fractional count
; Motor #11 Axis Definition Registers
M1191->L:$0005CF          ; #11 X/U/A/B/C-Axis scale factor (cts/unit)
M1192->L:$0005D0          ; #11 Y/V-Axis scale factor (cts/unit)
M1193->L:$0005D1          ; #11 Z/W-Axis scale factor (cts/unit)
M1194->L:$0005D2          ; #11 Axis offset (cts)
; Servo IC 4 Channel 4 Registers (usually for Motor #12)
M1201->X:$079219,0,24,S   ; ENC4 24-bit counter position
M1202->Y:$07921A,8,16,S   ; OUT4A command value; DAC or PWM
M1203->X:$07921B,0,24,S   ; ENC4 captured position
M1204->Y:$07921B,8,16,S   ; OUT4B command value; DAC or PWM
M1205->Y:$07921D,8,16,S   ; ADC4A input value
M1206->Y:$07921E,8,16,S   ; ADC4B input value
M1207->Y:$07921C,8,16,S   ; OUT4C command value; PFM or PWM
M1208->Y:$07921F,0,24,S   ; ENC4 compare A position
M1209->X:$07921F,0,24,S   ; ENC4 compare B position
M1210->X:$07921E,0,24,S   ; ENC4 compare autoincrement value
M1211->X:$07921D,11       ; ENC4 compare initial state write enable
M1212->X:$07921D,12       ; ENC4 compare initial state
M1214->X:$07921D,14       ; AENA4 output status
M1215->X:$079218,19       ; USER4 flag input status
M1216->X:$079218,9        ; ENC4 compare output value
M1217->X:$079218,11       ; ENC4 capture flag
M1218->X:$079218,8        ; ENC4 count error flag
M1219->X:$079218,14       ; HMFL4 flag input status
M1220->X:$079218,16       ; CHC4 input status
M1221->X:$079218,17       ; PLIM4 flag input status
M1222->X:$079218,18       ; MLIM4 flag input status
M1223->X:$079218,15       ; FAULT4 flag input status
M1224->X:$079218,20       ; Channel 4 W flag input status
M1225->X:$079218,21       ; Channel 4 V flag input status
M1226->X:$079218,22       ; Channel 4 U flag input status
M1227->X:$079218,23       ; Channel 4 T flag input status
M1228->X:$079218,20,4     ; Channel 4 TUVW inputs as 4-bit value
```

```
; Motor #12 Status Bits
M1230->Y:$000640,11,1      ; #12 Stopped-on-position-limit bit
M1231->X:$000630,21,1      ; #12 Positive-end-limit-set bit
M1232->X:$000630,22,1      ; #12 Negative-end-limit-set bit
M1233->X:$000630,13,1      ; #12 Desired-velocity-zero bit
M1235->X:$000630,15,1      ; #12 Dwell-in-progress bit
M1237->X:$000630,17,1      ; #12 Running-program bit
M1238->X:$000630,18,1      ; #12 Open-loop-mode bit
M1239->X:$000630,19,1      ; #12 Amplifier-enabled status bit
M1240->Y:$000640,0,1       ; #12 Background in-position bit
M1241->Y:$000640,1,1       ; #12 Warning-following error bit
M1242->Y:$000640,2,1       ; #12 Fatal-following-error bit
M1243->Y:$000640,3,1       ; #12 Amplifier-fault-error bit
M1244->Y:$000640,13,1      ; #12 Foreground in-position bit
M1245->Y:$000640,10,1      ; #12 Home-complete bit
M1246->Y:$000640,6,1       ; #12 Integrated following error fault bit
M1247->Y:$000640,5,1       ; #12 I2T fault bit
M1248->Y:$000640,8,1       ; #12 Phasing error fault bit
M1249->Y:$000640,9,1       ; #12 Phasing search-in-progress bit
; MACRO IC 1 Node 5 Flag Registers (usually used for Motor #12)
M1250->X:$003455,0,24      ; MACRO IC 1 Node 5 flag status register
M1251->Y:$003455,0,24      ; MACRO IC 1 Node 5 flag command register
M1253->X:$003455,20,4      ; MACRO IC 1 Node 5 TUVW flags
M1254->Y:$003455,14,1      ; MACRO IC 1 Node 5 amplifier enable flag
M1255->X:$003455,15,1      ; MACRO IC 1 Node 5 node/amplifier fault flag
M1256->X:$003455,16,1      ; MACRO IC 1 Node 5 home flag
M1257->X:$003455,17,1      ; MACRO IC 1 Node 5 positive limit flag
M1258->X:$003455,18,1      ; MACRO IC 1 Node 5 negative limit flag
M1259->X:$003455,19,1      ; MACRO IC 1 Node 5 user flag
; Motor #12 Move Registers
M1261->D:$000608           ; #12 Commanded position (1/[Ixx08*32] cts)
M1262->D:$00060B           ; #12 Actual position (1/[Ixx08*32] cts)
M1263->D:$000647           ; #12 Target (end) position (1/[Ixx08*32] cts)
M1264->D:$00064C           ; #12 Position bias (1/[Ixx08*32] cts)
M1266->X:$00061D,0,24,S    ; #12 Actual velocity (1/[Ixx09*32] cts/cyc)
M1267->D:$00060D           ; #12 Present master pos (1/[Ixx07*32] cts)
M1268->X:$00063F,8,16,S    ; #12 Filter Output (16-bit DAC bits)
M1269->D:$000610           ; #12 Compensation correction (1/[Ixx08*32] cts)
M1270->D:$000634           ; #12 Present phase position (including fraction)
M1271->X:$000634,24,S      ; #12 Present phase position (counts *Ixx70)
M1272->L:$000657           ; #12 Variable jog position/distance (cts)
M1273->Y:$00064E,0,24,S    ; #12 Encoder home capture position (cts)
M1274->D:$00066F           ; #12 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1275->X:$000639,8,16,S    ; #12 Actual quadrature current
M1276->Y:$000639,8,16,S    ; #12 Actual direct current
M1277->X:$00063C,8,16,S    ; #12 Quadrature current-loop integrator output
M1278->Y:$00063C,8,16,S    ; #12 Direct current-loop integrator output
```

```
M1279->X:$00062E,8,16,S   ; #12 PID internal filter result (16-bit DAC bits)
M1288->Y:$079219,0,12,U   ; IC 4 Ch 4 Compare A fractional count
M1289->Y:$079218,0,12,U   ; IC 4 Ch 4 Compare B fractional count
; Motor #12 Axis Definition Registers
M1291->L:$00064F          ; #12 X/U/A/B/C-Axis scale factor (cts/unit)
M1292->L:$000650          ; #12 Y/V-Axis scale factor (cts/unit)
M1293->L:$000651          ; #12 Z/W-Axis scale factor (cts/unit)
M1294->L:$000652          ; #12 Axis offset (cts)
; Servo IC 5 Channel 1 Registers (usually for Motor #13)
M1301->X:$079301,0,24,S   ; ENC5 24-bit counter position
M1302->Y:$079302,8,16,S   ; OUT5A command value; DAC or PWM
M1303->X:$079303,0,24,S   ; ENC5 captured position
M1304->Y:$079303,8,16,S   ; OUT5B command value; DAC or PWM
M1305->Y:$079305,8,16,S   ; ADC5A input value
M1306->Y:$079306,8,16,S   ; ADC5B input value
M1307->Y:$079304,8,16,S   ; OUT5C command value; PFM or PWM
M1308->Y:$079307,0,24,S   ; ENC5 compare A position
M1309->X:$079307,0,24,S   ; ENC5 compare B position
M1310->X:$079306,0,24,S   ; ENC5 compare autoincrement value
M1311->X:$079305,11       ; ENC5 compare initial state write enable
M1312->X:$079305,12       ; ENC5 compare initial state
M1314->X:$079305,14       ; AENA5 output status
M1315->X:$079300,19       ; USER5 flag input status
M1316->X:$079300,9        ; ENC5 compare output value
M1317->X:$079300,11       ; ENC5 capture flag
M1318->X:$079300,8        ; ENC5 count error flag
M1319->X:$079300,14       ; CHC5 input status
M1320->X:$079300,16       ; HMFL5 flag input status
M1321->X:$079300,17       ; PLIM5 flag input status
M1322->X:$079300,18       ; MLIM5 flag input status
M1323->X:$079300,15       ; FAULT5 flag input status
M1324->X:$079300,20       ; Channel 5 W flag input status
M1325->X:$079300,21       ; Channel 5 V flag input status
M1326->X:$079300,22       ; Channel 5 U flag input status
M1327->X:$079300,23       ; Channel 5 T flag input status
M1328->X:$079300,20,4     ; Channel 5 TUVW inputs as 4-bit value
; Motor #13 Status Bits
M1330->Y:$0006C0,11,1     ; #13 Stopped-on-position-limit bit
M1331->X:$0006B0,21,1     ; #13 Positive-end-limit-set bit
M1332->X:$0006B0,22,1     ; #13 Negative-end-limit-set bit
M1333->X:$0006B0,13,1     ; #13 Desired-velocity-zero bit
M1335->X:$0006B0,15,1     ; #13 Dwell-in-progress bit
M1337->X:$0006B0,17,1     ; #13 Running-program bit
M1338->X:$0006B0,18,1     ; #13 Open-loop-mode bit
M1339->X:$0006B0,19,1     ; #13 Amplifier-enabled status bit
M1340->Y:$0006C0,0,1      ; #13 Background in-position bit
M1341->Y:$0006C0,1,1      ; #13 Warning-following error bit
```

```
M1342->Y:$0006C0,2,1        ; #13 Fatal-following-error bit
M1343->Y:$0006C0,3,1        ; #13 Amplifier-fault-error bit
M1344->Y:$0006C0,13,1       ; #13 Foreground in-position bit
M1345->Y:$0006C0,10,1       ; #13 Home-complete bit
M1346->Y:$0006C0,6,1        ; #13 Integrated following error fault bit
M1347->Y:$0006C0,5,1        ; #13 I2T fault bit
M1348->Y:$0006C0,8,1        ; #13 Phasing error fault bit
M1349->Y:$0006C0,9,1        ; #13 Phasing search-in-progress bit
; MACRO IC 1 Node 8 Flag Registers (usually used for Motor #13)
M1350->X:$003458,0,24       ; MACRO IC 1 Node 8 flag status register
M1351->Y:$003458,0,24       ; MACRO IC 1 Node 8 flag command register
M1353->X:$003458,20,4       ; MACRO IC 1 Node 8 TUVW flags
M1354->Y:$003458,14,1       ; MACRO IC 1 Node 8 amplifier enable flag
M1355->X:$003458,15,1       ; MACRO IC 1 Node 8 node/amplifier fault flag
M1356->X:$003458,16,1       ; MACRO IC 1 Node 8 home flag
M1357->X:$003458,17,1       ; MACRO IC 1 Node 8 positive limit flag
M1358->X:$003458,18,1       ; MACRO IC 1 Node 8 negative limit flag
M1359->X:$003458,19,1       ; MACRO IC 1 Node 8 user flag
; Motor #13 Move Registers
M1361->D:$000688            ; #13 Commanded position (1/[Ixx08*32] cts)
M1362->D:$00068B            ; #13 Actual position (1/[Ixx08*32] cts)
M1363->D:$0006C7            ; #13 Target (end) position (1/[Ixx08*32] cts)
M1364->D:$0006CC            ; #13 Position bias (1/[Ixx08*32] cts)
M1366->X:$00069D,0,24,S     ; #13 Actual velocity (1/[Ixx09*32] cts/cyc)
M1367->D:$00068D            ; #13 Present master pos (1/[Ixx07*32] cts)
M1368->X:$0006BF,8,16,S     ; #13 Filter Output (16-bit DAC bits)
M1369->D:$000690            ; #13 Compensation correction (1/[Ixx08*32] cts)
M1370->D:$0006B4            ; #13 Present phase position (including fraction)
M1371->X:$0006B4,24,S       ; #13 Present phase position (counts *Ixx70)
M1372->L:$0006D7            ; #13 Variable jog position/distance (cts)
M1373->Y:$0006CE,0,24,S     ; #13 Encoder home capture position (cts)
M1374->D:$0006EF            ; #13 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1375->X:$0006B9,8,16,S     ; #13 Actual quadrature current
M1376->Y:$0006B9,8,16,S     ; #13 Actual direct current
M1377->X:$0006BC,8,16,S     ; #13 Quadrature current-loop integrator output
M1378->Y:$0006BC,8,16,S     ; #13 Direct current-loop integrator output
M1379->X:$0006AE,8,16,S     ; #13 PID internal filter result (16-bit DAC bits)
M1388->Y:$079301,0,12,U     ; IC 5 Ch 1 Compare A fractional count
M1389->Y:$079300,0,12,U     ; IC 5 Ch 1 Compare B fractional count
; Motor #13 Axis Definition Registers
M1391->L:$0006CF            ; #13 X/U/A/B/C-Axis scale factor (cts/unit)
M1392->L:$0006D0            ; #13 Y/V-Axis scale factor (cts/unit)
M1393->L:$0006D1            ; #13 Z/W-Axis scale factor (cts/unit)
M1394->L:$0006D2            ; #13 Axis offset (cts)
```

```
; Servo IC 5 Channel 2 Registers (usually for Motor #14)
M1401->X:$079309,0,24,S   ; ENC6 24-bit counter position
M1402->Y:$07930A,8,16,S   ; OUT6A command value; DAC or PWM
M1403->X:$07930B,0,24,S   ; ENC6 captured position
M1404->Y:$07930B,8,16,S   ; OUT6B command value; DAC or PWM
M1405->Y:$07930D,8,16,S   ; ADC6A input value
M1406->Y:$07930E,8,16,S   ; ADC6B input value
M1407->Y:$07930C,8,16,S   ; OUT6C command value; PFM or PWM
M1408->Y:$07930F,0,24,S   ; ENC6 compare A position
M1409->X:$07930F,0,24,S   ; ENC6 compare B position
M1410->X:$07930E,0,24,S   ; ENC6 compare autoincrement value
M1411->X:$07930D,11       ; ENC6 compare initial state write enable
M1412->X:$07930D,12       ; ENC6 compare initial state
M1414->X:$07930D,14       ; AENA6 output status
M1415->X:$079308,19       ; USER6 flag input status
M1416->X:$079308,9        ; ENC6 compare output value
M1417->X:$079308,11       ; ENC6 capture flag
M1418->X:$079308,8        ; ENC6 count error flag
M1419->X:$079308,14       ; CHC6 input status
M1420->X:$079308,16       ; HMFL6 flag input status
M1421->X:$079308,17       ; PLIM6 flag input status
M1422->X:$079308,18       ; MLIM6 flag input status
M1423->X:$079308,15       ; FAULT6 flag input status
M1424->X:$079308,20       ; Channel 6 W flag input status
M1425->X:$079308,21       ; Channel 6 V flag input status
M1426->X:$079308,22       ; Channel 6 U flag input status
M1427->X:$079308,23       ; Channel 6 T flag input status
M1428->X:$079308,20,4     ; Channel 6 TUVW inputs as 4-bit value
; Motor #14 Status Bits
M1430->Y:$000740,11,1     ; #14 Stopped-on-position-limit bit
M1431->X:$000730,21,1     ; #14 Positive-end-limit-set bit
M1432->X:$000730,22,1     ; #14 Negative-end-limit-set bit
M1433->X:$000730,13,1     ; #14 Desired-velocity-zero bit
M1435->X:$000730,15,1     ; #14 Dwell-in-progress bit
M1437->X:$000730,17,1     ; #14 Running-program bit
M1438->X:$000730,18,1     ; #14 Open-loop-mode bit
M1439->X:$000730,19,1     ; #14 Amplifier-enabled status bit
M1440->Y:$000740,0,1      ; #14 Background in-position bit
M1441->Y:$000740,1,1      ; #14 Warning-following error bit
M1442->Y:$000740,2,1      ; #14 Fatal-following-error bit
M1443->Y:$000740,3,1      ; #14 Amplifier-fault-error bit
M1444->Y:$000740,13,1     ; #14 Foreground in-position bit
M1445->Y:$000740,10,1     ; #14 Home-complete bit
M1446->Y:$000740,6,1      ; #14 Integrated following error fault bit
M1447->Y:$000740,5,1      ; #14 I2T fault bit
M1448->Y:$000740,8,1      ; #14 Phasing error fault bit
M1449->Y:$000740,9,1      ; #14 Phasing search-in-progress bit
```

```
; MACRO IC 1 Node 9 Flag Registers (usually used for Motor #14)
M1450->X:$003459,0,24      ; MACRO IC 1 Node 9 flag status register
M1451->Y:$003459,0,24      ; MACRO IC 1 Node 9 flag command register
M1453->X:$003459,20,4      ; MACRO IC 1 Node 9 TUVW flags
M1454->Y:$003459,14,1      ; MACRO IC 1 Node 9 amplifier enable flag
M1455->X:$003459,15,1      ; MACRO IC 1 Node 9 node/amplifier fault flag
M1456->X:$003459,16,1      ; MACRO IC 1 Node 9 home flag
M1457->X:$003459,17,1      ; MACRO IC 1 Node 9 positive limit flag
M1458->X:$003459,18,1      ; MACRO IC 1 Node 9 negative limit flag
M1459->X:$003459,19,1      ; MACRO IC 1 Node 9 user flag
; Motor #14 Move Registers
M1461->D:$000708           ; #14 Commanded position (1/[Ixx08*32] cts)
M1462->D:$00070B           ; #14 Actual position (1/[Ixx08*32] cts)
M1463->D:$000747           ; #14 Target (end) position (1/[Ixx08*32] cts)
M1464->D:$00074C           ; #14 Position bias (1/[Ixx08*32] cts)
M1466->X:$00071D,0,24,S    ; #14 Actual velocity (1/[Ixx09*32] cts/cyc)
M1467->D:$00070D           ; #14 Present master pos (1/[Ixx07*32] cts)
M1468->X:$00073F,8,16,S    ; #14 Filter Output (16-bit DAC bits)
M1469->D:$000710           ; #14 Compensation correction (1/[Ixx08*32] cts)
M1470->D:$000734           ; #14 Present phase position (including fraction)
M1471->X:$000734,24,S      ; #14 Present phase position (counts *Ixx70)
M1472->L:$000757           ; #14 Variable jog position/distance (cts)
M1473->Y:$00074E,0,24,S    ; #14 Encoder home capture position (cts)
M1474->D:$00076F           ; #14 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1475->X:$000739,8,16,S    ; #14 Actual quadrature current
M1476->Y:$000739,8,16,S    ; #14 Actual direct current
M1477->X:$00073C,8,16,S    ; #14 Quadrature current-loop integrator output
M1478->Y:$00073C,8,16,S    ; #14 Direct current-loop integrator output
M1479->X:$00072E,8,16,S    ; #14 PID internal filter result (16-bit DAC bits)
M1488->Y:$079309,0,12,U    ; IC 5 Ch 2 Compare A fractional count
M1489->Y:$079308,0,12,U    ; IC 5 Ch 2 Compare B fractional count
; Motor #14 Axis Definition Registers
M1491->L:$00074F           ; #14 X/U/A/B/C-Axis scale factor (cts/unit)
M1492->L:$000750           ; #14 Y/V-Axis scale factor (cts/unit)
M1493->L:$000751           ; #14 Z/W-Axis scale factor (cts/unit)
M1494->L:$000752           ; #14 Axis offset (cts)
; Servo IC 5 Channel 3 Registers (usually for Motor #15)
M1501->X:$079311,0,24,S    ; ENC7 24-bit counter position
M1502->Y:$079312,8,16,S    ; OUT7A command value; DAC or PWM
M1503->X:$079313,0,24,S    ; ENC7 captured position
M1504->Y:$079313,8,16,S    ; OUT7B command value; DAC or PWM
M1505->Y:$079315,8,16,S    ; ADC7A input value
M1506->Y:$079316,8,16,S    ; ADC7B input value
M1507->Y:$079314,8,16,S    ; OUT7C command value; PFM or PWM
M1508->Y:$079317,0,24,S    ; ENC7 compare A position
M1509->X:$079317,0,24,S    ; ENC7 compare B position
M1510->X:$079316,0,24,S    ; ENC7 compare autoincrement value
```

```
M1511->X:$079315,11        ; ENC7 compare initial state write enable
M1512->X:$079315,12        ; ENC7 compare initial state
M1514->X:$079315,14        ; AENA7 output status
M1515->X:$079310,19        ; CHC7 input status
M1516->X:$079310,9         ; ENC7 compare output value
M1517->X:$079310,11        ; ENC7 capture flag
M1518->X:$079310,8         ; ENC7 count error flag
M1519->X:$079310,14        ; CHC7 input status
M1520->X:$079310,16        ; HMFL7 flag input status
M1521->X:$079310,17        ; PLIM7 flag input status
M1522->X:$079310,18        ; MLIM7 flag input status
M1523->X:$079310,15        ; FAULT7 flag input status
M1524->X:$079310,20        ; Channel 7 W flag input status
M1525->X:$079310,21        ; Channel 7 V flag input status
M1526->X:$079310,22        ; Channel 7 U flag input status
M1527->X:$079310,23        ; Channel 7 T flag input status
M1528->X:$079310,20,4      ; Channel 7 TUVW inputs as 4-bit value
; Motor #15 Status Bits
M1530->Y:$0007C0,11,1      ; #15 Stopped-on-position-limit bit
M1531->X:$0007B0,21,1      ; #15 Positive-end-limit-set bit
M1532->X:$0007B0,22,1      ; #15 Negative-end-limit-set bit
M1533->X:$0007B0,13,1      ; #15 Desired-velocity-zero bit
M1535->X:$0007B0,15,1      ; #15 Dwell-in-progress bit
M1537->X:$0007B0,17,1      ; #15 Running-program bit
M1538->X:$0007B0,18,1      ; #15 Open-loop-mode bit
M1539->X:$0007B0,19,1      ; #15 Amplifier-enabled status bit
M1540->Y:$0007C0,0,1       ; #15 Background in-position bit
M1541->Y:$0007C0,1,1       ; #15 Warning-following error bit
M1542->Y:$0007C0,2,1       ; #15 Fatal-following-error bit
M1543->Y:$0007C0,3,1       ; #15 Amplifier-fault-error bit
M1544->Y:$000740,13,1      ; #15 Foreground in-position bit
M1545->Y:$0007C0,10,1      ; #15 Home-complete bit
M1546->Y:$0007C0,6,1       ; #15 Integrated following error fault bit
M1547->Y:$0007C0,5,1       ; #15 I2T fault bit
M1548->Y:$0007C0,8,1       ; #15 Phasing error fault bit
M1549->Y:$0007C0,9,1       ; #15 Phasing search-in-progress bit
; MACRO IC 1 Node 12 Flag Registers (usually used for Motor #15)
M1550->X:$00345C,0,24      ; MACRO IC 1 Node 12 flag status register
M1551->Y:$00345C,0,24      ; MACRO IC 1 Node 12 flag command register
M1553->X:$00345C,20,4      ; MACRO IC 1 Node 12 TUVW flags
M1554->Y:$00345C,14,1      ; MACRO IC 1 Node 12 amplifier enable flag
M1555->X:$00345C,15,1      ; MACRO IC 1 Node 12 node/amplifier fault flag
M1556->X:$00345C,16,1      ; MACRO IC 1 Node 12 home flag
M1557->X:$00345C,17,1      ; MACRO IC 1 Node 12 positive limit flag
M1558->X:$00345C,18,1      ; MACRO IC 1 Node 12 negative limit flag
M1559->X:$00345C,19,1      ; MACRO IC 1 Node 12 user flag
```

```
; Motor #15 Move Registers
M1561->D:$000788          ; #15 Commanded position (1/[Ixx08*32] cts)
M1562->D:$00078B          ; #15 Actual position (1/[Ixx08*32] cts)
M1563->D:$0007C7          ; #15 Target (end) position (1/[Ixx08*32] cts)
M1564->D:$0007CC          ; #15 Position bias (1/[Ixx08*32] cts)
M1566->X:$00079D,0,24,S   ; #15 Actual velocity (1/[Ixx09*32] cts/cyc)
M1567->D:$00078D          ; #15 Present master pos (1/[Ixx07*32] cts)
M1568->X:$0007BF,8,16,S   ; #15 Filter Output (16-bit DAC bits)
M1569->D:$000790          ; #15 Compensation correction (1/[Ixx08*32] cts)
M1570->D:$0007B4          ; #15 Present phase position (including fraction)
M1571->X:$0007B4,24,S     ; #15 Present phase position (counts *Ixx70)
M1572->L:$0007D7          ; #15 Variable jog position/distance (cts)
M1573->Y:$0007CE,0,24,S   ; #15 Encoder home capture position (cts)
M1574->D:$0007EF          ; #15 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1575->X:$0007B9,8,16,S   ; #15 Actual quadrature current
M1576->Y:$0007B9,8,16,S   ; #15 Actual direct current
M1577->X:$0007BC,8,16,S   ; #15 Quadrature current-loop integrator output
M1578->Y:$0007BC,8,16,S   ; #15 Direct current-loop integrator output
M1579->X:$0007AE,8,16,S   ; #15 PID internal filter result (16-bit DAC bits)
M1588->Y:$079311,0,12,U   ; IC 5 Ch 3 Compare A fractional count
M1589->Y:$079310,0,12,U   ; IC 5 Ch 3 Compare B fractional count
; Motor #15 Axis Definition Registers
M1591->L:$0007CF          ; #15 X/U/A/B/C-Axis scale factor (cts/unit)
M1592->L:$0007D0          ; #15 Y/V-Axis scale factor (cts/unit)
M1593->L:$0007D1          ; #15 Z/W-Axis scale factor (cts/unit)
M1594->L:$0007D2          ; #15 Axis offset (cts)
; Servo IC 5 Channel 4 Registers (usually for Motor #16)
M1601->X:$079319,0,24,S   ; ENC8 24-bit counter position
M1602->Y:$07931A,8,16,S   ; OUT8A command value; DAC or PWM
M1603->X:$07931B,0,24,S   ; ENC8 captured position
M1604->Y:$07931B,8,16,S   ; OUT8B command value; DAC or PWM
M1605->Y:$07931D,8,16,S   ; ADC8A input value
M1606->Y:$07931E,8,16,S   ; ADC8B input value
M1607->Y:$07931C,8,16,S   ; OUT8C command value; PFM or PWM
M1608->Y:$07931F,0,24,S   ; ENC8 compare A position
M1609->X:$07931F,0,24,S   ; ENC8 compare B position
M1610->X:$07931E,0,24,S   ; ENC8 compare autoincrement value
M1611->X:$07931D,11       ; ENC8 compare initial state write enable
M1612->X:$07931D,12       ; ENC8 compare initial state
M1614->X:$07931D,14       ; AENA8 output status
M1615->X:$079318,19       ; USER8 flag input status
M1616->X:$079318,9        ; ENC8 compare output value
M1617->X:$079318,11       ; ENC8 capture flag
M1618->X:$079318,8        ; ENC8 count error flag
M1619->X:$079318,14       ; CHC8 input status
M1620->X:$079318,16       ; HMFL8 flag input status
M1621->X:$079318,17       ; PLIM8 flag input status
```

```
M1622->X:$079318,18        ; MLIM8 flag input status
M1623->X:$079318,15        ; FAULT8 flag input status
M1624->X:$079318,20        ; Channel 8 W flag input status
M1625->X:$079318,21        ; Channel 8 V flag input status
M1626->X:$079318,22        ; Channel 8 U flag input status
M1627->X:$079318,23        ; Channel 8 T flag input status
M1628->X:$079318,20,4      ; Channel 8 TUVW inputs as 4-bit value
; Motor #16 Status Bits
M1630->Y:$000840,11,1      ; #16 Stopped-on-position-limit bit
M1631->X:$000830,21,1      ; #16 Positive-end-limit-set bit
M1632->X:$000830,22,1      ; #16 Negative-end-limit-set bit
M1633->X:$000830,13,1      ; #16 Desired-velocity-zero bit
M1635->X:$000830,15,1      ; #16 Dwell-in-progress bit
M1637->X:$000830,17,1      ; #16 Running-program bit
M1638->X:$000830,18,1      ; #16 Open-loop-mode bit
M1639->X:$000830,19,1      ; #16 Amplifier-enabled status bit
M1640->Y:$000840,0,1       ; #16 Background in-position bit
M1641->Y:$000840,1,1       ; #16 Warning-following error bit
M1642->Y:$000840,2,1       ; #16 Fatal-following-error bit
M1643->Y:$000840,3,1       ; #16 Amplifier-fault-error bit
M1644->Y:$000840,13,1      ; #16 Foreground in-position bit
M1645->Y:$000840,10,1      ; #16 Home-complete bit
M1646->Y:$000840,6,1       ; #16 Integrated following error fault bit
M1647->Y:$000840,5,1       ; #16 I2T fault bit
M1648->Y:$000840,8,1       ; #16 Phasing error fault bit
; MACRO IC 1 Node 13 Flag Registers (usually used for Motor #16)
M1650->X:$00345D,0,24      ; MACRO IC 1 Node 13 flag status register
M1651->Y:$00345D,0,24      ; MACRO IC 1 Node 13 flag command register
M1653->X:$00345D,20,4      ; MACRO IC 1 Node 13 TUVW flags
M1654->Y:$00345D,14,1      ; MACRO IC 1 Node 13 amplifier enable flag
M1655->X:$00345D,15,1      ; MACRO IC 1 Node 13 node/amplifier fault flag
M1656->X:$00345D,16,1      ; MACRO IC 1 Node 13 home flag
M1657->X:$00345D,17,1      ; MACRO IC 1 Node 13 positive limit flag
M1658->X:$00345D,18,1      ; MACRO IC 1 Node 13 negative limit flag
M1659->X:$00345D,19,1      ; MACRO IC 1 Node 13 user flag
; Motor #16 Move Registers
M1661->D:$000808           ; #16 Commanded position (1/[Ixx08*32] cts)
M1662->D:$00080B           ; #16 Actual position (1/[Ixx08*32] cts)
M1663->D:$000847           ; #16 Target (end) position (1/[Ixx08*32] cts)
M1664->D:$00084C           ; #16 Position bias (1/[Ixx08*32] cts)
M1666->X:$00081D,0,24,S    ; #16 Actual velocity (1/[Ixx09*32] cts/cyc)
M1667->D:$00080D           ; #16 Present master pos (1/[Ixx07*32] cts)
M1668->X:$00083F,8,16,S    ; #16 Filter Output (16-bit DAC bits)
M1669->D:$000810           ; #16 Compensation correction (1/[Ixx08*32] cts)
M1670->D:$000834           ; #16 Present phase position (including fraction)
M1671->X:$000834,24,S      ; #16 Present phase position (counts *Ixx70)
M1672->L:$000857           ; #16 Variable jog position/distance (cts)
```

```
M1673->Y:$00084E,0,24,S   ; #16 Encoder home capture position (cts)
M1674->D:$00086F          ; #16 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1675->X:$000839,8,16,S   ; #16 Actual quadrature current
M1676->Y:$000839,8,16,S   ; #16 Actual direct current
M1677->X:$00083C,8,16,S   ; #16 Quadrature current-loop integrator output
M1678->Y:$00083C,8,16,S   ; #16 Direct current-loop integrator output
M1679->X:$00082E,8,16,S   ; #16 PID internal filter result (16-bit DAC bits)
M1688->Y:$079319,0,12,U   ; IC 5 Ch 4 Compare A fractional count
M1689->Y:$079318,0,12,U   ; IC 5 Ch 4 Compare B fractional count
; Motor #16 Axis Definition Registers
M1691->L:$00084F          ; #16 X/U/A/B/C-Axis scale factor (cts/unit)
M1692->L:$000850          ; #16 Y/V-Axis scale factor (cts/unit)
M1693->L:$000851          ; #16 Z/W-Axis scale factor (cts/unit)
M1694->L:$000852          ; #16 Axis offset (cts)
; Servo IC 6 Channel 1 Registers (usually for Motor #17)
M1701->X:$07A201,0,24,S   ; ENC1 24-bit counter position
M1702->Y:$07A202,8,16,S   ; OUT1A command value; DAC or PWM
M1703->X:$07A203,0,24,S   ; ENC1 captured position
M1704->Y:$07A203,8,16,S   ; OUT1B command value; DAC or PWM
M1705->Y:$07A205,8,16,S   ; ADC1A input value
M1706->Y:$07A206,8,16,S   ; ADC1B input value
M1707->Y:$07A204,8,16,S   ; OUT1C command value; PFM or PWM
M1708->Y:$07A207,0,24,S   ; ENC1 compare A position
M1709->X:$07A207,0,24,S   ; ENC1 compare B position
M1710->X:$07A206,0,24,S   ; ENC1 compare autoincrement value
M1711->X:$07A205,11       ; ENC1 compare initial state write enable
M1712->X:$07A205,12       ; ENC1 compare initial state
M1714->X:$07A205,14       ; AENA1 output status
M1715->X:$07A200,19       ; USER1 flag input status
M1716->X:$07A200,9        ; ENC1 compare output value
M1717->X:$07A200,11       ; ENC1 capture flag
M1718->X:$07A200,8        ; ENC1 count error flag
M1719->X:$07A200,14       ; CHC1 input status
M1720->X:$07A200,16       ; HMFL1 flag input status
M1721->X:$07A200,17       ; PLIM1 flag input status
M1722->X:$07A200,18       ; MLIM1 flag input status
M1723->X:$07A200,15       ; FAULT1 flag input status
M1724->X:$07A200,20       ; Channel 1 W flag input status
M1725->X:$07A200,21       ; Channel 1 V flag input status
M1726->X:$07A200,22       ; Channel 1 U flag input status
M1727->X:$07A200,23       ; Channel 1 T flag input status
M1728->X:$07A200,20,4     ; Channel 1 TUVW inputs as 4-bit value
; Motor #17 Status Bits
M1730->Y:$0008C0,11,1     ; #17 Stopped-on-position-limit bit
M1731->X:$0008B0,21,1     ; #17 Positive-end-limit-set bit
M1732->X:$0008B0,22,1     ; #17 Negative-end-limit-set bit
M1733->X:$0008B0,13,1     ; #17 Desired-velocity-zero bit
```

```
M1735->X:$0008B0,15,1      ; #17 Dwell-in-progress bit
M1737->X:$0008B0,17,1      ; #17 Running-program bit
M1738->X:$0008B0,18,1      ; #17 Open-loop-mode bit
M1739->X:$0008B0,19,1      ; #17 Amplifier-enabled status bit
M1740->Y:$0008C0,0,1       ; #17 Background in-position bit
M1741->Y:$0008C0,1,1       ; #17 Warning-following error bit
M1742->Y:$0008C0,2,1       ; #17 Fatal-following-error bit
M1743->Y:$0008C0,3,1       ; #17 Amplifier-fault-error bit
M1744->Y:$0008C0,13,1      ; #17 Foreground in-position bit
M1745->Y:$0008C0,10,1      ; #17 Home-complete bit
M1746->Y:$0008C0,6,1       ; #17 Integrated following error fault bit
M1747->Y:$0008C0,5,1       ; #17 I2T fault bit
M1748->Y:$0008C0,8,1       ; #17 Phasing error fault bit
M1749->Y:$0008C0,9,1       ; #17 Phasing search-in-progress bit
; MACRO IC 2 Node 0 Flag Registers (usually used for Motor #17)
M1750->X:$003460,0,24      ; MACRO IC 2 Node 0 flag status register
M1751->Y:$003460,0,24      ; MACRO IC 2 Node 0 flag command register
M1753->X:$003460,20,4      ; MACRO IC 2 Node 0 TUVW flags
M1754->Y:$003460,14,1      ; MACRO IC 2 Node 0 amplifier enable flag
M1755->X:$003460,15,1      ; MACRO IC 2 Node 0 node/amplifier fault flag
M1756->X:$003460,16,1      ; MACRO IC 2 Node 0 home flag
M1757->X:$003460,17,1      ; MACRO IC 2 Node 0 positive limit flag
M1758->X:$003460,18,1      ; MACRO IC 2 Node 0 negative limit flag
M1759->X:$003460,19,1      ; MACRO IC 2 Node 0 user flag
; Motor #17 Move Registers
M1761->D:$000888           ; #17 Commanded position (1/[Ixx08*32] cts)
M1762->D:$00088B           ; #17 Actual position (1/[Ixx08*32] cts)
M1763->D:$0008C7           ; #17 Target (end) position (1/[Ixx08*32] cts)
M1764->D:$0008CC           ; #17 Position bias (1/[Ixx08*32] cts)
M1766->X:$00089D,0,24,S    ; #17 Actual velocity (1/[Ixx09*32] cts/cyc)
M1767->D:$00088D           ; #17 Present master pos (1/[Ixx07*32] cts)
M1768->X:$0008BF,8,16,S    ; #17 Filter Output (16-bit DAC bits)
M1769->D:$000890           ; #17 Compensation correction (1/[Ixx08*32] cts)
M1770->D:$0008B4           ; #17 Present phase position (including fraction)
M1771->X:$0008B4,24,S      ; #17 Present phase position (counts *Ixx70)
M1772->L:$0008D7           ; #17 Variable jog position/distance (cts)
M1773->Y:$0008CE,0,24,S    ; #17 Encoder home capture position (cts)
M1774->D:$0008EF           ; #17 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1775->X:$0008B9,8,16,S    ; #17 Actual quadrature current
M1776->Y:$0008B9,8,16,S    ; #17 Actual direct current
M1777->X:$0008BC,8,16,S    ; #17 Quadrature current-loop integrator output
M1778->Y:$0008BC,8,16,S    ; #17 Direct current-loop integrator output
M1779->X:$0008AE,8,16,S    ; #17 PID internal filter result (16-bit DAC bits)
M1788->Y:$07A201,0,12,U    ; IC 6 Ch 1 Compare A fractional count
M1789->Y:$07A200,0,12,U    ; IC 6 Ch 1 Compare B fractional count
```

```
; Motor #17 Axis Definition Registers
M1791->L:$0008CF            ; #17 X/U/A/B/C-Axis scale factor (cts/unit)
M1792->L:$0008D0            ; #17 Y/V-Axis scale factor (cts/unit)
M1793->L:$0008D1            ; #17 Z/W-Axis scale factor (cts/unit)
M1794->L:$0008D2            ; #17 Axis offset (cts)
; Servo IC 6 Channel 2 Registers (usually for Motor #18)
M1801->X:$07A209,0,24,S     ; ENC2 24-bit counter position
M1802->Y:$07A20A,8,16,S     ; OUT2A command value; DAC or PWM
M1803->X:$07A20B,0,24,S     ; ENC2 captured position
M1804->Y:$07A20B,8,16,S     ; OUT2B command value; DAC or PWM
M1805->Y:$07A20D,8,16,S     ; ADC2A input value
M1806->Y:$07A20E,8,16,S     ; ADC2B input value
M1807->Y:$07A20C,8,16,S     ; OUT2C command value; PFM or PWM
M1808->Y:$07A20F,0,24,S     ; ENC2 compare A position
M1809->X:$07A20F,0,24,S     ; ENC2 compare B position
M1810->X:$07A20E,0,24,S     ; ENC2 compare autoincrement value
M1811->X:$07A20D,11         ; ENC2 compare initial state write enable
M1812->X:$07A20D,12         ; ENC2 compare initial state
M1814->X:$07A20D,14         ; AENA2 output status
M1815->X:$07A208,19         ; USER2 flag input status
M1816->X:$07A208,9          ; ENC2 compare output value
M1817->X:$07A208,11         ; ENC2 capture flag
M1818->X:$07A208,8          ; ENC2 count error flag
M1819->X:$07A208,14         ; CHC2 input status
M1820->X:$07A208,16         ; HMFL2 flag input status
M1821->X:$07A208,17         ; PLIM2 flag input status
M1822->X:$07A208,18         ; MLIM2 flag input status
M1823->X:$07A208,15         ; FAULT2 flag input status
M1824->X:$07A208,20         ; Channel 2 W flag input status
M1825->X:$07A208,21         ; Channel 2 V flag input status
M1826->X:$07A208,22         ; Channel 2 U flag input status
M1827->X:$07A208,23         ; Channel 2 T flag input status
M1828->X:$07A208,20,4       ; Channel 2 TUVW inputs as 4-bit value
; Motor #18 Status Bits
M1830->Y:$000940,11,1       ; #18 Stopped-on-position-limit bit
M1831->X:$000930,21,1       ; #18 Positive-end-limit-set bit
M1832->X:$000930,22,1       ; #18 Negative-end-limit-set bit
M1833->X:$000930,13,1       ; #18 Desired-velocity-zero bit
M1835->X:$000930,15,1       ; #18 Dwell-in-progress bit
M1837->X:$000930,17,1       ; #18 Running-program bit
M1838->X:$000930,18,1       ; #18 Open-loop-mode bit
M1839->X:$000930,19,1       ; #18 Amplifier-enabled status bit
M1840->Y:$000940,0,1        ; #18 Background in-position bit
M1841->Y:$000940,1,1        ; #18 Warning-following error bit
M1842->Y:$000940,2,1        ; #18 Fatal-following-error bit
M1843->Y:$000940,3,1        ; #18 Amplifier-fault-error bit
M1844->Y:$0008C0,13,1       ; #18 Foreground in-position bit
```

```
M1845->Y:$000940,10,1      ; #18 Home-complete bit
M1846->Y:$000940,6,1       ; #18 Integrated following error fault bit
M1847->Y:$000940,5,1       ; #18 I2T fault bit
M1848->Y:$000940,8,1       ; #18 Phasing error fault bit
M1849->Y:$000940,9,1       ; #18 Phasing search-in-progress bit
; MACRO IC 2 Node 1 Flag Registers (usually used for Motor #18)
M1850->X:$003461,0,24      ; MACRO IC 2 Node 1 flag status register
M1851->Y:$003461,0,24      ; MACRO IC 2 Node 1 flag command register
M1853->X:$003461,20,4      ; MACRO IC 2 Node 1 TUVW flags
M1854->Y:$003461,14,1      ; MACRO IC 2 Node 1 amplifier enable flag
M1855->X:$003461,15,1      ; MACRO IC 2 Node 1 node/amplifier fault flag
M1856->X:$003461,16,1      ; MACRO IC 2 Node 1 home flag
M1857->X:$003461,17,1      ; MACRO IC 2 Node 1 positive limit flag
M1858->X:$003461,18,1      ; MACRO IC 2 Node 1 negative limit flag
M1859->X:$003461,19,1      ; MACRO IC 2 Node 1 user flag
; Motor #18 Move Registers
M1861->D:$000908           ; #18 Commanded position (1/[Ixx08*32] cts)
M1862->D:$00090B           ; #18 Actual position (1/[Ixx08*32] cts)
M1863->D:$000947           ; #18 Target (end) position (1/[Ixx08*32] cts)
M1864->D:$00094C           ; #18 Position bias (1/[Ixx08*32] cts)
M1866->X:$00091D,0,24,S    ; #18 Actual velocity (1/[Ixx09*32] cts/cyc)
M1867->D:$00090D           ; #18 Present master pos (1/[Ixx07*32] cts)
M1868->X:$00093F,8,16,S    ; #18 Filter Output (16-bit DAC bits)
M1869->D:$000910           ; #18 Compensation correction (1/[Ixx08*32] cts)
M1870->D:$000934           ; #18 Present phase position (including fraction)
M1871->X:$000934,24,S      ; #18 Present phase position (counts *Ixx70)
M1872->L:$000957           ; #18 Variable jog position/distance (cts)
M1873->Y:$00094E,0,24,S    ; #18 Encoder home capture position (cts)
M1874->D:$00096F           ; #18 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1875->X:$000939,8,16,S    ; #18 Actual quadrature current
M1876->Y:$000939,8,16,S    ; #18 Actual direct current
M1877->X:$00093C,8,16,S    ; #18 Quadrature current-loop integrator output
M1878->Y:$00093C,8,16,S    ; #18 Direct current-loop integrator output
M1879->X:$00092E,8,16,S    ; #18 PID internal filter result (16-bit DAC bits)
M1888->Y:$07A209,0,12,U    ; IC 6 Ch 2 Compare A fractional count
M1889->Y:$07A208,0,12,U    ; IC 6 Ch 2 Compare B fractional count
; Motor #18 Axis Definition Registers
M1891->L:$00094F           ; #18 X/U/A/B/C-Axis scale factor (cts/unit)
M1892->L:$000950           ; #18 Y/V-Axis scale factor (cts/unit)
M1893->L:$000951           ; #18 Z/W-Axis scale factor (cts/unit)
M1894->L:$000952           ; #18 Axis offset (cts)
; Servo IC 6 Channel 3 Registers (usually for Motor #19)
M1901->X:$07A211,0,24,S    ; ENC3 24-bit counter position
M1902->Y:$07A212,8,16,S    ; OUT3A command value; DAC or PWM
M1903->X:$07A213,0,24,S    ; ENC3 captured position
M1904->Y:$07A213,8,16,S    ; OUT3B command value; DAC or PWM
M1905->Y:$07A215,8,16,S    ; ADC3A input value
```

```
M1906->Y:$07A216,8,16,S   ; ADC3B input value
M1907->Y:$07A214,8,16,S   ; OUT3C command value; PFM or PWM
M1908->Y:$07A217,0,24,S   ; ENC3 compare A position
M1909->X:$07A217,0,24,S   ; ENC3 compare B position
M1910->X:$07A216,0,24,S   ; ENC3 compare autoincrement value
M1911->X:$07A215,11       ; ENC3 compare initial state write enable
M1912->X:$07A215,12       ; ENC3 compare initial state
M1914->X:$07A215,14       ; AENA3 output status
M1915->X:$07A210,19       ; USER3 flag input status
M1916->X:$07A210,9        ; ENC3 compare output value
M1917->X:$07A210,11       ; ENC3 capture flag
M1918->X:$07A210,8        ; ENC3 count error flag
M1919->X:$07A210,14       ; CHC3 input status
M1920->X:$07A210,16       ; HMFL3 flag input status
M1921->X:$07A210,17       ; PLIM3 flag input status
M1922->X:$07A210,18       ; MLIM3 flag input status
M1923->X:$07A210,15       ; FAULT3 flag input status
M1924->X:$07A210,20       ; Channel 3 W flag input status
M1925->X:$07A210,21       ; Channel 3 V flag input status
M1926->X:$07A210,22       ; Channel 3 U flag input status
M1927->X:$07A210,23       ; Channel 3 T flag input status
M1928->X:$07A210,20,4     ; Channel 3 TUVW inputs as 4-bit value
; Motor #19 Status Bits
M1930->Y:$0009C0,11,1     ; #19 Stopped-on-position-limit bit
M1931->X:$0009B0,21,1     ; #19 Positive-end-limit-set bit
M1932->X:$0009B0,22,1     ; #19 Negative-end-limit-set bit
M1933->X:$0009B0,13,1     ; #19 Desired-velocity-zero bit
M1935->X:$0009B0,15,1     ; #19 Dwell-in-progress bit
M1937->X:$0009B0,17,1     ; #19 Running-program bit
M1938->X:$0009B0,18,1     ; #19 Open-loop-mode bit
M1939->X:$0009B0,19,1     ; #19 Amplifier-enabled status bit
M1940->Y:$0009C0,0,1      ; #19 Background in-position bit
M1941->Y:$0009C0,1,1      ; #19 Warning-following error bit
M1942->Y:$0009C0,2,1      ; #19 Fatal-following-error bit
M1943->Y:$0009C0,3,1      ; #19 Amplifier-fault-error bit
M1944->Y:$0009C0,13,1     ; #19 Foreground in-position bit
M1945->Y:$0009C0,10,1     ; #19 Home-complete bit
M1946->Y:$0009C0,6,1      ; #19 Integrated following error fault bit
M1947->Y:$0009C0,5,1      ; #19 I2T fault bit
M1948->Y:$0009C0,8,1      ; #19 Phasing error fault bit
M1949->Y:$0009C0,9,1      ; #19 Phasing search-in-progress bit
; MACRO IC 2 Node 4 Flag Registers (usually used for Motor #19)
M1950->X:$003464,0,24     ; MACRO IC 2 Node 4 flag status register
M1951->Y:$003464,0,24     ; MACRO IC 2 Node 4 flag command register
M1953->X:$003464,20,4     ; MACRO IC 2 Node 4 TUVW flags
M1954->Y:$003464,14,1     ; MACRO IC 2 Node 4 amplifier enable flag
M1955->X:$003464,15,1     ; MACRO IC 2 Node 4 node/amplifier fault flag
```

```
M1956->X:$003464,16,1      ; MACRO IC 2 Node 4 home flag
M1957->X:$003464,17,1      ; MACRO IC 2 Node 4 positive limit flag
M1958->X:$003464,18,1      ; MACRO IC 2 Node 4 negative limit flag
M1959->X:$003464,19,1      ; MACRO IC 2 Node 4 user flag
; Motor #19 Move Registers
M1961->D:$000988           ; #19 Commanded position (1/[Ixx08*32] cts)
M1962->D:$00098B           ; #19 Actual position (1/[Ixx08*32] cts)
M1963->D:$0009C7           ; #19 Target (end) position (1/[Ixx08*32] cts)
M1964->D:$0009CC           ; #19 Position bias (1/[Ixx08*32] cts)
M1966->X:$00099D,0,24,S    ; #19 Actual velocity (1/[Ixx09*32] cts/cyc)
M1967->D:$00098D           ; #19 Present master pos (1/[Ixx07*32] cts)
M1968->X:$0009BF,8,16,S    ; #19 Filter Output (16-bit DAC bits)
M1969->D:$000990           ; #19 Compensation correction (1/[Ixx08*32] cts)
M1970->D:$0009B4           ; #19 Present phase position (including fraction)
M1971->X:$0009B4,24,S      ; #19 Present phase position (counts *Ixx70)
M1972->L:$0009D7           ; #19 Variable jog position/distance (cts)
M1973->Y:$0009CE,0,24,S    ; #19 Encoder home capture position (cts)
M1974->D:$0009EF           ; #19 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M1975->X:$0009B9,8,16,S    ; #19 Actual quadrature current
M1976->Y:$0009B9,8,16,S    ; #19 Actual direct current
M1977->X:$0009BC,8,16,S    ; #19 Quadrature current-loop integrator output
M1978->Y:$0009BC,8,16,S    ; #19 Direct current-loop integrator output
M1979->X:$0009AE,8,16,S    ; #19 PID internal filter result (16-bit DAC bits)
M1988->Y:$07A211,0,12,U    ; IC 6 Ch 3 Compare A fractional count
M1989->Y:$07A210,0,12,U    ; IC 6 Ch 3 Compare B fractional count
; Motor #19 Axis Definition Registers
M1991->L:$0009CF           ; #19 X/U/A/B/C-Axis scale factor (cts/unit)
M1992->L:$0009D0           ; #19 Y/V-Axis scale factor (cts/unit)
M1993->L:$0009D1           ; #19 Z/W-Axis scale factor (cts/unit)
M1994->L:$0009D2           ; #19 Axis offset (cts)
; Servo IC 6 Channel 4 Registers (usually for Motor #20)
M2001->X:$07A219,0,24,S    ; ENC4 24-bit counter position
M2002->Y:$07A21A,8,16,S    ; OUT4A command value; DAC or PWM
M2003->X:$07A21B,0,24,S    ; ENC4 captured position
M2004->Y:$07A21B,8,16,S    ; OUT4B command value; DAC or PWM
M2005->Y:$07A21D,8,16,S    ; ADC4A input value
M2006->Y:$07A21E,8,16,S    ; ADC4B input value
M2007->Y:$07A21C,8,16,S    ; OUT4C command value; PFM or PWM
M2008->Y:$07A21F,0,24,S    ; ENC4 compare A position
M2009->X:$07A21F,0,24,S    ; ENC4 compare B position
M2010->X:$07A21E,0,24,S    ; ENC4 compare autoincrement value
M2011->X:$07A21D,11        ; ENC4 compare initial state write enable
M2012->X:$07A21D,12        ; ENC4 compare initial state
M2014->X:$07A21D,14        ; AENA4 output status
M2015->X:$07A218,19        ; USER4 flag input status
M2016->X:$07A218,9         ; ENC4 compare output value
M2017->X:$07A218,11        ; ENC4 capture flag
```

```
M2018->X:$07A218,8          ; ENC4 count error flag
M2019->X:$07A218,14         ; HMFL4 flag input status
M2020->X:$07A218,16         ; CHC4 input status
M2021->X:$07A218,17         ; PLIM4 flag input status
M2022->X:$07A218,18         ; MLIM4 flag input status
M2023->X:$07A218,15         ; FAULT4 flag input status
M2024->X:$07A218,20         ; Channel 4 W flag input status
M2025->X:$07A218,21         ; Channel 4 V flag input status
M2026->X:$07A218,22         ; Channel 4 U flag input status
M2027->X:$07A218,23         ; Channel 4 T flag input status
M2028->X:$07A218,20,4       ; Channel 4 TUVW inputs as 4-bit value
; Motor #20 Status Bits
M2030->Y:$000A40,11,1       ; #20 Stopped-on-position-limit bit
M2031->X:$000A30,21,1       ; #20 Positive-end-limit-set bit
M2032->X:$000A30,22,1       ; #20 Negative-end-limit-set bit
M2033->X:$000A30,13,1       ; #20 Desired-velocity-zero bit
M2035->X:$000A30,15,1       ; #20 Dwell-in-progress bit
M2037->X:$000A30,17,1       ; #20 Running-program bit
M2038->X:$000A30,18,1       ; #20 Open-loop-mode bit
M2039->X:$000A30,19,1       ; #20 Amplifier-enabled status bit
M2040->Y:$000A40,0,1        ; #20 Background in-position bit
M2041->Y:$000A40,1,1        ; #20 Warning-following error bit
M2042->Y:$000A40,2,1        ; #20 Fatal-following-error bit
M2043->Y:$000A40,3,1        ; #20 Amplifier-fault-error bit
M2044->Y:$000A40,13,1       ; #20 Foreground in-position bit
M2045->Y:$000A40,10,1       ; #20 Home-complete bit
M2046->Y:$000A40,6,1        ; #20 Integrated following error fault bit
M2047->Y:$000A40,5,1        ; #20 I2T fault bit
M2048->Y:$000A40,8,1        ; #20 Phasing error fault bit
M2049->Y:$000A40,9,1        ; #20 Phasing search-in-progress bit
; MACRO IC 2 Node 5 Flag Registers (usually used for Motor #20)
M2050->X:$003465,0,24       ; MACRO IC 2 Node 5 flag status register
M2051->Y:$003465,0,24       ; MACRO IC 2 Node 5 flag command register
M2053->X:$003465,20,4       ; MACRO IC 2 Node 5 TUVW flags
M2054->Y:$003465,14,1       ; MACRO IC 2 Node 5 amplifier enable flag
M2055->X:$003465,15,1       ; MACRO IC 2 Node 5 node/amplifier fault flag
M2056->X:$003465,16,1       ; MACRO IC 2 Node 5 home flag
M2057->X:$003465,17,1       ; MACRO IC 2 Node 5 positive limit flag
M2058->X:$003465,18,1       ; MACRO IC 2 Node 5 negative limit flag
M2059->X:$003465,19,1       ; MACRO IC 2 Node 5 user flag
; Motor #20 Move Registers
M2061->D:$000A08            ; #20 Commanded position (1/[Ixx08*32] cts)
M2062->D:$000A0B            ; #20 Actual position (1/[Ixx08*32] cts)
M2063->D:$000A47            ; #20 Target (end) position (1/[Ixx08*32] cts)
M2064->D:$000A4C            ; #20 Position bias (1/[Ixx08*32] cts)
M2066->X:$000A1D,0,24,S     ; #20 Actual velocity (1/[Ixx09*32] cts/cyc)
M2067->D:$000A0D            ; #20 Present master pos (1/[Ixx07*32] cts)
```

```
M2068->X:$000A3F,8,16,S   ; #20 Filter Output (16-bit DAC bits)
M2069->D:$000A10          ; #20 Compensation correction (1/[Ixx08*32] cts)
M2070->D:$000A34          ; #20 Present phase position (including fraction)
M2071->X:$000A34,24,S     ; #20 Present phase position (counts *Ixx70)
M2072->L:$000A57          ; #20 Variable jog position/distance (cts)
M2073->Y:$000A4E,0,24,S   ; #20 Encoder home capture position (cts)
M2074->D:$000A6F          ; #20 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2075->X:$000A39,8,16,S   ; #20 Actual quadrature current
M2076->Y:$000A39,8,16,S   ; #20 Actual direct current
M2077->X:$000A3C,8,16,S   ; #20 Quadrature current-loop integrator output
M2078->Y:$000A3C,8,16,S   ; #20 Direct current-loop integrator output
M2079->X:$000A2E,8,16,S   ; #20 PID internal filter result (16-bit DAC bits)
M2088->Y:$07A219,0,12,U   ; IC 6 Ch 4 Compare A fractional count
M2089->Y:$07A218,0,12,U   ; IC 6 Ch 4 Compare B fractional count
; Motor #20 Axis Definition Registers
M2091->L:$000A4F          ; #20 X/U/A/B/C-Axis scale factor (cts/unit)
M2092->L:$000A50          ; #20 Y/V-Axis scale factor (cts/unit)
M2093->L:$000A51          ; #20 Z/W-Axis scale factor (cts/unit)
M2094->L:$000A52          ; #20 Axis offset (cts)
; Servo IC 7 Channel 1 Registers (usually for Motor #21)
M2101->X:$07A301,0,24,S   ; ENC5 24-bit counter position
M2102->Y:$07A302,8,16,S   ; OUT5A command value; DAC or PWM
M2103->X:$07A303,0,24,S   ; ENC5 captured position
M2104->Y:$07A303,8,16,S   ; OUT5B command value; DAC or PWM
M2105->Y:$07A305,8,16,S   ; ADC5A input value
M2106->Y:$07A306,8,16,S   ; ADC5B input value
M2107->Y:$07A304,8,16,S   ; OUT5C command value; PFM or PWM
M2108->Y:$07A307,0,24,S   ; ENC5 compare A position
M2109->X:$07A307,0,24,S   ; ENC5 compare B position
M2110->X:$07A306,0,24,S   ; ENC5 compare autoincrement value
M2111->X:$07A305,11       ; ENC5 compare initial state write enable
M2112->X:$07A305,12       ; ENC5 compare initial state
M2114->X:$07A305,14       ; AENA5 output status
M2115->X:$07A300,19       ; USER5 flag input status
M2116->X:$07A300,9        ; ENC5 compare output value
M2117->X:$07A300,11       ; ENC5 capture flag
M2118->X:$07A300,8        ; ENC5 count error flag
M2119->X:$07A300,14       ; CHC5 input status
M2120->X:$07A300,16       ; HMFL5 flag input status
M2121->X:$07A300,17       ; PLIM5 flag input status
M2122->X:$07A300,18       ; MLIM5 flag input status
M2123->X:$07A300,15       ; FAULT5 flag input status
M2124->X:$07A300,20       ; Channel 5 W flag input status
M2125->X:$07A300,21       ; Channel 5 V flag input status
M2126->X:$07A300,22       ; Channel 5 U flag input status
M2127->X:$07A300,23       ; Channel 5 T flag input status
M2128->X:$07A300,20,4     ; Channel 5 TUVW inputs as 4-bit value
```

```
; Motor #21 Status Bits
M2130->Y:$000AC0,11,1      ; #21 Stopped-on-position-limit bit
M2131->X:$000AB0,21,1      ; #21 Positive-end-limit-set bit
M2132->X:$000AB0,22,1      ; #21 Negative-end-limit-set bit
M2133->X:$000AB0,13,1      ; #21 Desired-velocity-zero bit
M2135->X:$000AB0,15,1      ; #21 Dwell-in-progress bit
M2137->X:$000AB0,17,1      ; #21 Running-program bit
M2138->X:$000AB0,18,1      ; #21 Open-loop-mode bit
M2139->X:$000AB0,19,1      ; #21 Amplifier-enabled status bit
M2140->Y:$000AC0,0,1       ; #21 Background in-position bit
M2141->Y:$000AC0,1,1       ; #21 Warning-following error bit
M2142->Y:$000AC0,2,1       ; #21 Fatal-following-error bit
M2143->Y:$000AC0,3,1       ; #21 Amplifier-fault-error bit
M2144->Y:$000AC0,13,1      ; #21 Foreground in-position bit
M2145->Y:$000AC0,10,1      ; #21 Home-complete bit
M2146->Y:$000AC0,6,1       ; #21 Integrated following error fault bit
M2147->Y:$000AC0,5,1       ; #21 I2T fault bit
M2148->Y:$000AC0,8,1       ; #21 Phasing error fault bit
M2149->Y:$000AC0,9,1       ; #21 Phasing search-in-progress bit
; MACRO IC 2 Node 8 Flag Registers (usually used for Motor #21)
M2150->X:$003468,0,24      ; MACRO IC 2 Node 8 flag status register
M2151->Y:$003468,0,24      ; MACRO IC 2 Node 8 flag command register
M2153->X:$003468,20,4      ; MACRO IC 2 Node 8 TUVW flags
M2154->Y:$003468,14,1      ; MACRO IC 2 Node 8 amplifier enable flag
M2155->X:$003468,15,1      ; MACRO IC 2 Node 8 node/amplifier fault flag
M2156->X:$003468,16,1      ; MACRO IC 2 Node 8 home flag
M2157->X:$003468,17,1      ; MACRO IC 2 Node 8 positive limit flag
M2158->X:$003468,18,1      ; MACRO IC 2 Node 8 negative limit flag
M2159->X:$003468,19,1      ; MACRO IC 2 Node 8 user flag
; Motor #21 Move Registers
M2161->D:$000A88           ; #21 Commanded position (1/[Ixx08*32] cts)
M2162->D:$000A8B           ; #21 Actual position (1/[Ixx08*32] cts)
M2163->D:$000AC7           ; #21 Target (end) position (1/[Ixx08*32] cts)
M2164->D:$000ACC           ; #21 Position bias (1/[Ixx08*32] cts)
M2166->X:$000A9D,0,24,S    ; #21 Actual velocity (1/[Ixx09*32] cts/cyc)
M2167->D:$000A8D           ; #21 Present master pos (1/[Ixx07*32] cts)
M2168->X:$000ABF,8,16,S    ; #21 Filter Output (16-bit DAC bits)
M2169->D:$000A90           ; #21 Compensation correction (1/[Ixx08*32] cts)
M2170->D:$000AB4           ; #21 Present phase position (including fraction)
M2171->X:$000AB4,24,S      ; #21 Present phase position (counts *Ixx70)
M2172->L:$000AD7           ; #21 Variable jog position/distance (cts)
M2173->Y:$000ACE,0,24,S    ; #21 Encoder home capture position (cts)
M2174->D:$000AEF           ; #21 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2175->X:$000AB9,8,16,S    ; #21 Actual quadrature current
M2176->Y:$000AB9,8,16,S    ; #21 Actual direct current
M2177->X:$000ABC,8,16,S    ; #21 Quadrature current-loop integrator output
M2178->Y:$000ABC,8,16,S    ; #21 Direct current-loop integrator output
```

```
M2179->X:$000AAE,8,16,S   ; #21 PID internal filter result (16-bit DAC bits)
M2188->Y:$07A301,0,12,U   ; IC 7 Ch 1 Compare A fractional count
M2189->Y:$07A300,0,12,U   ; IC 7 Ch 1 Compare B fractional count
; Motor #21 Axis Definition Registers
M2191->L:$000ACF          ; #21 X/U/A/B/C-Axis scale factor (cts/unit)
M2192->L:$000AD0          ; #21 Y/V-Axis scale factor (cts/unit)
M2193->L:$000AD1          ; #21 Z/W-Axis scale factor (cts/unit)
M2194->L:$000AD2          ; #21 Axis offset (cts)
; Servo IC 7 Channel 2 Registers (usually for Motor #22)
M2201->X:$07A309,0,24,S   ; ENC6 24-bit counter position
M2202->Y:$07A30A,8,16,S   ; OUT6A command value; DAC or PWM
M2203->X:$07A30B,0,24,S   ; ENC6 captured position
M2204->Y:$07A30B,8,16,S   ; OUT6B command value; DAC or PWM
M2205->Y:$07A30D,8,16,S   ; ADC6A input value
M2206->Y:$07A30E,8,16,S   ; ADC6B input value
M2207->Y:$07A30C,8,16,S   ; OUT6C command value; PFM or PWM
M2208->Y:$07A30F,0,24,S   ; ENC6 compare A position
M2209->X:$07A30F,0,24,S   ; ENC6 compare B position
M2210->X:$07A30E,0,24,S   ; ENC6 compare autoincrement value
M2211->X:$07A30D,11       ; ENC6 compare initial state write enable
M2212->X:$07A30D,12       ; ENC6 compare initial state
M2214->X:$07A30D,14       ; AENA6 output status
M2215->X:$07A308,19       ; USER6 flag input status
M2216->X:$07A308,9        ; ENC6 compare output value
M2217->X:$07A308,11       ; ENC6 capture flag
M2218->X:$07A308,8        ; ENC6 count error flag
M2219->X:$07A308,14       ; CHC6 input status
M2220->X:$07A308,16       ; HMFL6 flag input status
M2221->X:$07A308,17       ; PLIM6 flag input status
M2222->X:$07A308,18       ; MLIM6 flag input status
M2223->X:$07A308,15       ; FAULT6 flag input status
M2224->X:$07A308,20       ; Channel 6 W flag input status
M2225->X:$07A308,21       ; Channel 6 V flag input status
M2226->X:$07A308,22       ; Channel 6 U flag input status
M2227->X:$07A308,23       ; Channel 6 T flag input status
M2228->X:$07A308,20,4     ; Channel 6 TUVW inputs as 4-bit value
; Motor #22 Status Bits
M2230->Y:$000B40,11,1     ; #22 Stopped-on-position-limit bit
M2231->X:$000B30,21,1     ; #22 Positive-end-limit-set bit
M2232->X:$000B30,22,1     ; #22 Negative-end-limit-set bit
M2233->X:$000B30,13,1     ; #22 Desired-velocity-zero bit
M2235->X:$000B30,15,1     ; #22 Dwell-in-progress bit
M2237->X:$000B30,17,1     ; #22 Running-program bit
M2238->X:$000B30,18,1     ; #22 Open-loop-mode bit
M2239->X:$000B30,19,1     ; #22 Amplifier-enabled status bit
M2240->Y:$000B40,0,1      ; #22 Background in-position bit
M2241->Y:$000B40,1,1      ; #22 Warning-following error bit
```

```
M2242->Y:$000B40,2,1        ; #22 Fatal-following-error bit
M2243->Y:$000B40,3,1        ; #22 Amplifier-fault-error bit
M2244->Y:$000B40,13,1       ; #22 Foreground in-position bit
M2245->Y:$000B40,10,1       ; #22 Home-complete bit
M2246->Y:$000B40,6,1        ; #22 Integrated following error fault bit
M2247->Y:$000B40,5,1        ; #22 I2T fault bit
M2248->Y:$000B40,8,1        ; #22 Phasing error fault bit
M2249->Y:$000B40,9,1        ; #22 Phasing search-in-progress bit
; MACRO IC 2 Node 9 Flag Registers (usually used for Motor #22)
M2250->X:$003469,0,24       ; MACRO IC 2 Node 9 flag status register
M2251->Y:$003469,0,24       ; MACRO IC 2 Node 9 flag command register
M2253->X:$003469,20,4       ; MACRO IC 2 Node 9 TUVW flags
M2254->Y:$003469,14,1       ; MACRO IC 2 Node 9 amplifier enable flag
M2255->X:$003469,15,1       ; MACRO IC 2 Node 9 node/amplifier fault flag
M2256->X:$003469,16,1       ; MACRO IC 2 Node 9 home flag
M2257->X:$003469,17,1       ; MACRO IC 2 Node 9 positive limit flag
M2258->X:$003469,18,1       ; MACRO IC 2 Node 9 negative limit flag
M2259->X:$003469,19,1       ; MACRO IC 2 Node 9 user flag
; Motor #22 Move Registers
M2261->D:$000B08            ; #22 Commanded position (1/[Ixx08*32] cts)
M2262->D:$000B0B            ; #22 Actual position (1/[Ixx08*32] cts)
M2263->D:$000B47            ; #22 Target (end) position (1/[Ixx08*32] cts)
M2264->D:$000B4C            ; #22 Position bias (1/[Ixx08*32] cts)
M2266->X:$000B1D,0,24,S     ; #22 Actual velocity (1/[Ixx09*32] cts/cyc)
M2267->D:$000B0D            ; #22 Present master pos (1/[Ixx07*32] cts)
M2268->X:$000B3F,8,16,S     ; #22 Filter Output (16-bit DAC bits)
M2269->D:$000B10            ; #22 Compensation correction (1/[Ixx08*32] cts)
M2270->D:$000B34            ; #22 Present phase position (including fraction)
M2271->X:$000B34,24,S       ; #22 Present phase position (counts *Ixx70)
M2272->L:$000B57            ; #22 Variable jog position/distance (cts)
M2273->Y:$000B4E,0,24,S     ; #22 Encoder home capture position (cts)
M2274->D:$000B6F            ; #22 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2275->X:$000B39,8,16,S     ; #22 Actual quadrature current
M2276->Y:$000B39,8,16,S     ; #22 Actual direct current
M2277->X:$000B3C,8,16,S     ; #22 Quadrature current-loop integrator output
M2278->Y:$000B3C,8,16,S     ; #22 Direct current-loop integrator output
M2279->X:$000B2E,8,16,S     ; #22 PID internal filter result (16-bit DAC bits)
M2288->Y:$07A309,0,12,U     ; IC 7 Ch 1 Compare A fractional count
M2289->Y:$07A308,0,12,U     ; IC 7 Ch 1 Compare B fractional count
; Motor #22 Axis Definition Registers
M2291->L:$000B4F            ; #22 X/U/A/B/C-Axis scale factor (cts/unit)
M2292->L:$000B50            ; #22 Y/V-Axis scale factor (cts/unit)
M2293->L:$000B51            ; #22 Z/W-Axis scale factor (cts/unit)
M2294->L:$000B52            ; #22 Axis offset (cts)
; Servo IC 7 Channel 3 Registers (usually for Motor #23)
M2301->X:$07A311,0,24,S     ; ENC7 24-bit counter position
M2302->Y:$07A312,8,16,S     ; OUT7A command value; DAC or PWM
```

```
M2303->X:$07A313,0,24,S   ; ENC7 captured position
M2304->Y:$07A313,8,16,S   ; OUT7B command value; DAC or PWM
M2305->Y:$07A315,8,16,S   ; ADC7A input value
M2306->Y:$07A316,8,16,S   ; ADC7B input value
M2307->Y:$07A314,8,16,S   ; OUT7C command value; PFM or PWM
M2308->Y:$07A317,0,24,S   ; ENC7 compare A position
M2309->X:$07A317,0,24,S   ; ENC7 compare B position
M2310->X:$07A316,0,24,S   ; ENC7 compare autoincrement value
M2311->X:$07A315,11       ; ENC7 compare initial state write enable
M2312->X:$07A315,12       ; ENC7 compare initial state
M2314->X:$07A315,14       ; AENA7 output status
M2315->X:$07A310,19       ; CHC7 input status
M2316->X:$07A310,9        ; ENC7 compare output value
M2317->X:$07A310,11       ; ENC7 capture flag
M2318->X:$07A310,8        ; ENC7 count error flag
M2319->X:$07A310,14       ; CHC7 input status
M2320->X:$07A310,16       ; HMFL7 flag input status
M2321->X:$07A310,17       ; PLIM7 flag input status
M2322->X:$07A310,18       ; MLIM7 flag input status
M2323->X:$07A310,15       ; FAULT7 flag input status
M2324->X:$07A310,20       ; Channel 7 W flag input status
M2325->X:$07A310,21       ; Channel 7 V flag input status
M2326->X:$07A310,22       ; Channel 7 U flag input status
M2327->X:$07A310,23       ; Channel 7 T flag input status
M2328->X:$07A310,20,4     ; Channel 7 TUVW inputs as 4-bit value
; Motor #23 Status Bits
M2330->Y:$000BC0,11,1     ; #23 Stopped-on-position-limit bit
M2331->X:$000BB0,21,1     ; #23 Positive-end-limit-set bit
M2332->X:$000BB0,22,1     ; #23 Negative-end-limit-set bit
M2333->X:$000BB0,13,1     ; #23 Desired-velocity-zero bit
M2335->X:$000BB0,15,1     ; #23 Dwell-in-progress bit
M2337->X:$000BB0,17,1     ; #23 Running-program bit
M2338->X:$000BB0,18,1     ; #23 Open-loop-mode bit
M2339->X:$000BB0,19,1     ; #23 Amplifier-enabled status bit
M2340->Y:$000BC0,0,1      ; #23 Background in-position bit
M2341->Y:$000BC0,1,1      ; #23 Warning-following error bit
M2342->Y:$000BC0,2,1      ; #23 Fatal-following-error bit
M2343->Y:$000BC0,3,1      ; #23 Amplifier-fault-error bit
M2344->Y:$000BC0,13,1     ; #23 Foreground in-position bit
M2345->Y:$000BC0,10,1     ; #23 Home-complete bit
M2346->Y:$000BC0,6,1      ; #23 Integrated following error fault bit
M2347->Y:$000BC0,5,1      ; #23 I2T fault bit
M2348->Y:$000BC0,8,1      ; #23 Phasing error fault bit
M2349->Y:$000BC0,9,1      ; #23 Phasing search-in-progress bit
; MACRO IC 2 Node 12 Flag Registers (usually used for Motor #23)
M2350->X:$00346C,0,24     ; MACRO IC 2 Node 12 flag status register
M2351->Y:$00346C,0,24     ; MACRO IC 2 Node 12 flag command register
```

```
M2353->X:$00346C,20,4      ; MACRO IC 2 Node 12 TUVW flags
M2354->Y:$00346C,14,1      ; MACRO IC 2 Node 12 amplifier enable flag
M2355->X:$00346C,15,1      ; MACRO IC 2 Node 12 node/amplifier fault flag
M2356->X:$00346C,16,1      ; MACRO IC 2 Node 12 home flag
M2357->X:$00346C,17,1      ; MACRO IC 2 Node 12 positive limit flag
M2358->X:$00346C,18,1      ; MACRO IC 2 Node 12 negative limit flag
M2359->X:$00346C,19,1      ; MACRO IC 2 Node 12 user flag
; Motor #23 Move Registers
M2361->D:$000B88           ; #23 Commanded position (1/[Ixx08*32] cts)
M2362->D:$000B8B           ; #23 Actual position (1/[Ixx08*32] cts)
M2363->D:$000BC7           ; #23 Target (end) position (1/[Ixx08*32] cts)
M2364->D:$000BCC           ; #23 Position bias (1/[Ixx08*32] cts)
M2366->X:$000B9D,0,24,S    ; #23 Actual velocity (1/[Ixx09*32] cts/cyc)
M2367->D:$000B8D           ; #23 Present master pos (1/[Ixx07*32] cts)
M2368->X:$000BBF,8,16,S    ; #23 Filter Output (16-bit DAC bits)
M2369->D:$000B90           ; #23 Compensation correction (1/[Ixx08*32] cts)
M2370->D:$000BB4           ; #23 Present phase position (including fraction)
M2371->X:$000BB4,24,S      ; #23 Present phase position (counts *Ixx70)
M2372->L:$000BD7           ; #23 Variable jog position/distance (cts)
M2373->Y:$000BCE,0,24,S    ; #23 Encoder home capture position (cts)
M2374->D:$000BEF           ; #23 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2375->X:$000BB9,8,16,S    ; #23 Actual quadrature current
M2376->Y:$000BB9,8,16,S    ; #23 Actual direct current
M2377->X:$000BBC,8,16,S    ; #23 Quadrature current-loop integrator output
M2378->Y:$000BBC,8,16,S    ; #23 Direct current-loop integrator output
M2379->X:$000BAE,8,16,S    ; #23 PID internal filter result (16-bit DAC bits)
M2388->Y:$07A311,0,12,U    ; IC 7 Ch 3 Compare A fractional count
M2389->Y:$07A310,0,12,U    ; IC 7 Ch 3 Compare B fractional count
; Motor #23 Axis Definition Registers
M2391->L:$000BCF           ; #23 X/U/A/B/C-Axis scale factor (cts/unit)
M2392->L:$000BD0           ; #23 Y/V-Axis scale factor (cts/unit)
M2393->L:$000BD1           ; #23 Z/W-Axis scale factor (cts/unit)
M2394->L:$000BD2           ; #23 Axis offset (cts)
; Servo IC 7 Channel 4 Registers (usually for Motor #24)
M2401->X:$07A319,0,24,S    ; ENC8 24-bit counter position
M2402->Y:$07A31A,8,16,S    ; OUT8A command value; DAC or PWM
M2403->X:$07A31B,0,24,S    ; ENC8 captured position
M2404->Y:$07A31B,8,16,S    ; OUT8B command value; DAC or PWM
M2405->Y:$07A31D,8,16,S    ; ADC8A input value
M2406->Y:$07A31E,8,16,S    ; ADC8B input value
M2407->Y:$07A31C,8,16,S    ; OUT8C command value; PFM or PWM
M2408->Y:$07A31F,0,24,S    ; ENC8 compare A position
M2409->X:$07A31F,0,24,S    ; ENC8 compare B position
M2410->X:$07A31E,0,24,S    ; ENC8 compare autoincrement value
M2411->X:$07A31D,11        ; ENC8 compare initial state write enable
M2412->X:$07A31D,12        ; ENC8 compare initial state
M2414->X:$07A31D,14        ; AENA8 output status
```

```
M2415->X:$07A318,19        ; USER8 flag input status
M2416->X:$07A318,9         ; ENC8 compare output value
M2417->X:$07A318,11        ; ENC8 capture flag
M2418->X:$07A318,8         ; ENC8 count error flag
M2419->X:$07A318,14        ; CHC8 input status
M2420->X:$07A318,16        ; HMFL8 flag input status
M2421->X:$07A318,17        ; PLIM8 flag input status
M2422->X:$07A318,18        ; MLIM8 flag input status
M2423->X:$07A318,15        ; FAULT8 flag input status
M2424->X:$07A318,20        ; Channel 8 W flag input status
M2425->X:$07A318,21        ; Channel 8 V flag input status
M2426->X:$07A318,22        ; Channel 8 U flag input status
M2427->X:$07A318,23        ; Channel 8 T flag input status
M2428->X:$07A318,20,4      ; Channel 8 TUVW inputs as 4-bit value
; Motor #24 Status Bits
M2430->Y:$000C40,11,1      ; #24 Stopped-on-position-limit bit
M2431->X:$000C30,21,1      ; #24 Positive-end-limit-set bit
M2432->X:$000C30,22,1      ; #24 Negative-end-limit-set bit
M2433->X:$000C30,13,1      ; #24 Desired-velocity-zero bit
M2435->X:$000C30,15,1      ; #24 Dwell-in-progress bit
M2437->X:$000C30,17,1      ; #24 Running-program bit
M2438->X:$000C30,18,1      ; #24 Open-loop-mode bit
M2439->X:$000C30,19,1      ; #24 Amplifier-enabled status bit
M2440->Y:$000C40,0,1       ; #24 Background in-position bit
M2441->Y:$000C40,1,1       ; #24 Warning-following error bit
M2442->Y:$000C40,2,1       ; #24 Fatal-following-error bit
M2443->Y:$000C40,3,1       ; #24 Amplifier-fault-error bit
M2444->Y:$000C40,13,1      ; #24 Foreground in-position bit
M2445->Y:$000C40,10,1      ; #24 Home-complete bit
M2446->Y:$000C40,6,1       ; #24 Integrated following error fault bit
M2447->Y:$000C40,5,1       ; #24 I2T fault bit
M2448->Y:$000C40,8,1       ; #24 Phasing error fault bit
M2449->Y:$000C40,9,1       ; #24 Phasing search-in-progress bit
; MACRO IC 2 Node 13 Flag Registers (usually used for Motor #24)
M2450->X:$00346D,0,24      ; MACRO IC 2 Node 13 flag status register
M2451->Y:$00346D,0,24      ; MACRO IC 2 Node 13 flag command register
M2453->X:$00346D,20,4      ; MACRO IC 2 Node 13 TUVW flags
M2454->Y:$00346D,14,1      ; MACRO IC 2 Node 13 amplifier enable flag
M2455->X:$00346D,15,1      ; MACRO IC 2 Node 13 node/amplifier fault flag
M2456->X:$00346D,16,1      ; MACRO IC 2 Node 13 home flag
M2457->X:$00346D,17,1      ; MACRO IC 2 Node 13 positive limit flag
M2458->X:$00346D,18,1      ; MACRO IC 2 Node 13 negative limit flag
M2459->X:$00346D,19,1      ; MACRO IC 2 Node 13 user flag
; Motor #24 Move Registers
M2461->D:$000C08           ; #24 Commanded position (1/[Ixx08*32] cts)
M2462->D:$000C0B           ; #24 Actual position (1/[Ixx08*32] cts)
M2463->D:$000C47           ; #24 Target (end) position (1/[Ixx08*32] cts)
```

```
M2464->D:$000C4C          ; #24 Position bias (1/[Ixx08*32] cts)
M2466->X:$000C1D,0,24,S   ; #24 Actual velocity (1/[Ixx09*32] cts/cyc)
M2467->D:$000C0D          ; #24 Present master pos (1/[Ixx07*32] cts)
M2468->X:$000C3F,8,16,S   ; #24 Filter Output (16-bit DAC bits)
M2469->D:$000C10          ; #24 Compensation correction (1/[Ixx08*32] cts)
M2470->D:$000C34          ; #24 Present phase position (including fraction)
M2471->X:$000C34,24,S     ; #24 Present phase position (counts *Ixx70)
M2472->L:$000C57          ; #24 Variable jog position/distance (cts)
M2473->Y:$000C4E,0,24,S   ; #24 Encoder home capture position (cts)
M2474->D:$000C6F          ; #24 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2475->X:$000C39,8,16,S   ; #24 Actual quadrature current
M2476->Y:$000C39,8,16,S   ; #24 Actual direct current
M2477->X:$000C3C,8,16,S   ; #24 Quadrature current-loop integrator output
M2478->Y:$000C3C,8,16,S   ; #24 Direct current-loop integrator output
M2479->X:$000C2E,8,16,S   ; #24 PID internal filter result (16-bit DAC bits)
M2488->Y:$07A319,0,12,U   ; IC 7 Ch 4 Compare A fractional count
M2489->Y:$07A318,0,12,U   ; IC 7 Ch 4 Compare B fractional count
; Motor #24 Axis Definition Registers
M2491->L:$000C4F          ; #24 X/U/A/B/C-Axis scale factor (cts/unit)
M2492->L:$000C50          ; #24 Y/V-Axis scale factor (cts/unit)
M2493->L:$000C51          ; #24 Z/W-Axis scale factor (cts/unit)
M2494->L:$000C52          ; #24 Axis offset (cts)
; Servo IC 8 Channel 1 Registers (usually for Motor #25)
M2501->X:$07B201,0,24,S   ; ENC1 24-bit counter position
M2502->Y:$07B202,8,16,S   ; OUT1A command value; DAC or PWM
M2503->X:$07B203,0,24,S   ; ENC1 captured position
M2504->Y:$07B203,8,16,S   ; OUT1B command value; DAC or PWM
M2505->Y:$07B205,8,16,S   ; ADC1A input value
M2506->Y:$07B206,8,16,S   ; ADC1B input value
M2507->Y:$07B204,8,16,S   ; OUT1C command value; PFM or PWM
M2508->Y:$07B207,0,24,S   ; ENC1 compare A position
M2509->X:$07B207,0,24,S   ; ENC1 compare B position
M2510->X:$07B206,0,24,S   ; ENC1 compare autoincrement value
M2511->X:$07B205,11       ; ENC1 compare initial state write enable
M2512->X:$07B205,12       ; ENC1 compare initial state
M2514->X:$07B205,14       ; AENA1 output status
M2515->X:$07B200,19       ; USER1 flag input status
M2516->X:$07B200,9        ; ENC1 compare output value
M2517->X:$07B200,11       ; ENC1 capture flag
M2518->X:$07B200,8        ; ENC1 count error flag
M2519->X:$07B200,14       ; CHC1 input status
M2520->X:$07B200,16       ; HMFL1 flag input status
M2521->X:$07B200,17       ; PLIM1 flag input status
M2522->X:$07B200,18       ; MLIM1 flag input status
M2523->X:$07B200,15       ; FAULT1 flag input status
M2524->X:$07B200,20       ; Channel 1 W flag input status
M2525->X:$07B200,21       ; Channel 1 V flag input status
```

```
M2526->X:$07B200,22        ; Channel 1 U flag input status
M2527->X:$07B200,23        ; Channel 1 T flag input status
M2528->X:$07B200,20,4      ; Channel 1 TUVW inputs as 4-bit value
; Motor #25 Status Bits
M2530->Y:$000CC0,11,1      ; #25 Stopped-on-position-limit bit
M2531->X:$000CB0,21,1      ; #25 Positive-end-limit-set bit
M2532->X:$000CB0,22,1      ; #25 Negative-end-limit-set bit
M2533->X:$000CB0,13,1      ; #25 Desired-velocity-zero bit
M2535->X:$000CB0,15,1      ; #25 Dwell-in-progress bit
M2537->X:$000CB0,17,1      ; #25 Running-program bit
M2538->X:$000CB0,18,1      ; #25 Open-loop-mode bit
M2539->X:$000CB0,19,1      ; #25 Amplifier-enabled status bit
M2540->Y:$000CC0,0,1       ; #25 Background in-position bit
M2541->Y:$000CC0,1,1       ; #25 Warning-following error bit
M2542->Y:$000CC0,2,1       ; #25 Fatal-following-error bit
M2543->Y:$000CC0,3,1       ; #25 Amplifier-fault-error bit
M2544->Y:$000CC0,13,1      ; #25 Foreground in-position bit
M2545->Y:$000CC0,10,1      ; #25 Home-complete bit
M2546->Y:$000CC0,6,1       ; #25 Integrated following error fault bit
M2547->Y:$000CC0,5,1       ; #25 I2T fault bit
M2548->Y:$000CC0,8,1       ; #25 Phasing error fault bit
M2549->Y:$000CC0,9,1       ; #25 Phasing search-in-progress bit
; MACRO IC 3 Node 0 Flag Registers (usually used for Motor #25)
M2550->X:$003470,0,24      ; MACRO IC 3 Node 0 flag status register
M2551->Y:$003470,0,24      ; MACRO IC 3 Node 0 flag command register
M2553->X:$003470,20,4      ; MACRO IC 3 Node 0 TUVW flags
M2554->Y:$003470,14,1      ; MACRO IC 3 Node 0 amplifier enable flag
M2555->X:$003470,15,1      ; MACRO IC 3 Node 0 node/amplifier fault flag
M2556->X:$003470,16,1      ; MACRO IC 3 Node 0 home flag
M2557->X:$003470,17,1      ; MACRO IC 3 Node 0 positive limit flag
M2558->X:$003470,18,1      ; MACRO IC 3 Node 0 negative limit flag
M2559->X:$003470,19,1      ; MACRO IC 3 Node 0 user flag
; Motor #25 Move Registers
M2561->D:$000C88           ; #25 Commanded position (1/[Ixx08*32] cts)
M2562->D:$000C8B           ; #25 Actual position (1/[Ixx08*32] cts)
M2563->D:$000CC7           ; #25 Target (end) position (1/[Ixx08*32] cts)
M2564->D:$000CCC           ; #25 Position bias (1/[Ixx08*32] cts)
M2566->X:$000C9D,0,24,S    ; #25 Actual velocity (1/[Ixx09*32] cts/cyc)
M2567->D:$000C8D           ; #25 Present master pos (1/[Ixx07*32] cts)
M2568->X:$000CBF,8,16,S    ; #25 Filter Output (16-bit DAC bits)
M2569->D:$000C90           ; #25 Compensation correction (1/[Ixx08*32] cts)
M2570->D:$000CB4           ; #25 Present phase position (including fraction)
M2571->X:$000CB4,24,S      ; #25 Present phase position (counts *Ixx70)
M2572->L:$000CD7           ; #25 Variable jog position/distance (cts)
M2573->Y:$000CCE,0,24,S    ; #25 Encoder home capture position (cts)
M2574->D:$000CEF           ; #25 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2575->X:$000CB9,8,16,S    ; #25 Actual quadrature current
```

```
M2576->Y:$000CB9,8,16,S   ; #25 Actual direct current
M2577->X:$000CBC,8,16,S   ; #25 Quadrature current-loop integrator output
M2578->Y:$000CBC,8,16,S   ; #25 Direct current-loop integrator output
M2579->X:$000CAE,8,16,S   ; #25 PID internal filter result (16-bit DAC bits)
M2588->Y:$07B201,0,12,U   ; IC 8 Ch 1 Compare A fractional count
M2589->Y:$07B200,0,12,U   ; IC 8 Ch 1 Compare B fractional count
; Motor #25 Axis Definition Registers
M2591->L:$000CCF          ; #25 X/U/A/B/C-Axis scale factor (cts/unit)
M2592->L:$000CD0          ; #25 Y/V-Axis scale factor (cts/unit)
M2593->L:$000CD1          ; #25 Z/W-Axis scale factor (cts/unit)
M2594->L:$000CD2          ; #25 Axis offset (cts)
; Servo IC 8 Channel 2 Registers (usually for Motor #26)
M2601->X:$07B209,0,24,S   ; ENC2 24-bit counter position
M2602->Y:$07B20A,8,16,S   ; OUT2A command value; DAC or PWM
M2603->X:$07B20B,0,24,S   ; ENC2 captured position
M2604->Y:$07B20B,8,16,S   ; OUT2B command value; DAC or PWM
M2605->Y:$07B20D,8,16,S   ; ADC2A input value
M2606->Y:$07B20E,8,16,S   ; ADC2B input value
M2607->Y:$07B20C,8,16,S   ; OUT2C command value; PFM or PWM
M2608->Y:$07B20F,0,24,S   ; ENC2 compare A position
M2609->X:$07B20F,0,24,S   ; ENC2 compare B position
M2610->X:$07B20E,0,24,S   ; ENC2 compare autoincrement value
M2611->X:$07B20D,11       ; ENC2 compare initial state write enable
M2612->X:$07B20D,12       ; ENC2 compare initial state
M2614->X:$07B20D,14       ; AENA2 output status
M2615->X:$07B208,19       ; USER2 flag input status
M2616->X:$07B208,9        ; ENC2 compare output value
M2617->X:$07B208,11       ; ENC2 capture flag
M2618->X:$07B208,8        ; ENC2 count error flag
M2619->X:$07B208,14       ; CHC2 input status
M2620->X:$07B208,16       ; HMFL2 flag input status
M2621->X:$07B208,17       ; PLIM2 flag input status
M2622->X:$07B208,18       ; MLIM2 flag input status
M2623->X:$07B208,15       ; FAULT2 flag input status
M2624->X:$07B208,20       ; Channel 2 W flag input status
M2625->X:$07B208,21       ; Channel 2 V flag input status
M2626->X:$07B208,22       ; Channel 2 U flag input status
M2627->X:$07B208,23       ; Channel 2 T flag input status
M2628->X:$07B208,20,4     ; Channel 2 TUVW inputs as 4-bit value
; Motor #26 Status Bits
M2630->Y:$000D40,11,1     ; #26 Stopped-on-position-limit bit
M2631->X:$000D30,21,1     ; #26 Positive-end-limit-set bit
M2632->X:$000D30,22,1     ; #26 Negative-end-limit-set bit
M2633->X:$000D30,13,1     ; #26 Desired-velocity-zero bit
M2635->X:$000D30,15,1     ; #26 Dwell-in-progress bit
M2637->X:$000D30,17,1     ; #26 Running-program bit
M2638->X:$000D30,18,1     ; #26 Open-loop-mode bit
```

```
M2639->X:$000D30,19,1      ; #26 Amplifier-enabled status bit
M2640->Y:$000D40,0,1       ; #26 Background in-position bit
M2641->Y:$000D40,1,1       ; #26 Warning-following error bit
M2642->Y:$000D40,2,1       ; #26 Fatal-following-error bit
M2643->Y:$000D40,3,1       ; #26 Amplifier-fault-error bit
M2644->Y:$000D40,13,1      ; #26 Foreground in-position bit
M2645->Y:$000D40,10,1      ; #26 Home-complete bit
M2646->Y:$000D40,6,1       ; #26 Integrated following error fault bit
M2647->Y:$000D40,5,1       ; #26 I2T fault bit
M2648->Y:$000D40,8,1       ; #26 Phasing error fault bit
M2649->Y:$000D40,9,1       ; #26 Phasing search-in-progress bit
; MACRO IC 3 Node 1 Flag Registers (usually used for Motor #26)
M2650->X:$003471,0,24      ; MACRO IC 3 Node 1 flag status register
M2651->Y:$003471,0,24      ; MACRO IC 3 Node 1 flag command register
M2653->X:$003471,20,4      ; MACRO IC 3 Node 1 TUVW flags
M2654->Y:$003471,14,1      ; MACRO IC 3 Node 1 amplifier enable flag
M2655->X:$003471,15,1      ; MACRO IC 3 Node 1 node/amplifier fault flag
M2656->X:$003471,16,1      ; MACRO IC 3 Node 1 home flag
M2657->X:$003471,17,1      ; MACRO IC 3 Node 1 positive limit flag
M2658->X:$003471,18,1      ; MACRO IC 3 Node 1 negative limit flag
M2659->X:$003471,19,1      ; MACRO IC 3 Node 1 user flag
; Motor #26 Move Registers
M2661->D:$000D08           ; #26 Commanded position (1/[Ixx08*32] cts)
M2662->D:$000D0B           ; #26 Actual position (1/[Ixx08*32] cts)
M2663->D:$000D47           ; #26 Target (end) position (1/[Ixx08*32] cts)
M2664->D:$000D4C           ; #26 Position bias (1/[Ixx08*32] cts)
M2666->X:$000D1D,0,24,S    ; #26 Actual velocity (1/[Ixx09*32] cts/cyc)
M2667->D:$000D0D           ; #26 Present master pos (1/[Ixx07*32] cts)
M2668->X:$000D3F,8,16,S    ; #26 Filter Output (16-bit DAC bits)
M2669->D:$000D10           ; #26 Compensation correction (1/[Ixx08*32] cts)
M2670->D:$000D34           ; #26 Present phase position (including fraction)
M2671->X:$000D34,24,S      ; #26 Present phase position (counts *Ixx70)
M2672->L:$000D57           ; #26 Variable jog position/distance (cts)
M2673->Y:$000D4E,0,24,S    ; #26 Encoder home capture position (cts)
M2674->D:$000D6F           ; #26 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2675->X:$000D39,8,16,S    ; #26 Actual quadrature current
M2676->Y:$000D39,8,16,S    ; #26 Actual direct current
M2677->X:$000D3C,8,16,S    ; #26 Quadrature current-loop integrator output
M2678->Y:$000D3C,8,16,S    ; #26 Direct current-loop integrator output
M2679->X:$000D2E,8,16,S    ; #26 PID internal filter result (16-bit DAC bits)
M2688->Y:$07B209,0,12,U    ; IC 8 Ch 2 Compare A fractional count
M2689->Y:$07B208,0,12,U    ; IC 8 Ch 2 Compare B fractional count
; Motor #26 Axis Definition Registers
M2691->L:$000D4F           ; #26 X/U/A/B/C-Axis scale factor (cts/unit)
M2692->L:$000D50           ; #26 Y/V-Axis scale factor (cts/unit)
M2693->L:$000D51           ; #26 Z/W-Axis scale factor (cts/unit)
M2694->L:$000D52           ; #26 Axis offset (cts)
```

```
; Servo IC 8 Channel 3 Registers (usually for Motor #27)
M2701->X:$07B211,0,24,S   ; ENC3 24-bit counter position
M2702->Y:$07B212,8,16,S   ; OUT3A command value; DAC or PWM
M2703->X:$07B213,0,24,S   ; ENC3 captured position
M2704->Y:$07B213,8,16,S   ; OUT3B command value; DAC or PWM
M2705->Y:$07B215,8,16,S   ; ADC3A input value
M2706->Y:$07B216,8,16,S   ; ADC3B input value
M2707->Y:$07B214,8,16,S   ; OUT3C command value; PFM or PWM
M2708->Y:$07B217,0,24,S   ; ENC3 compare A position
M2709->X:$07B217,0,24,S   ; ENC3 compare B position
M2710->X:$07B216,0,24,S   ; ENC3 compare autoincrement value
M2711->X:$07B215,11       ; ENC3 compare initial state write enable
M2712->X:$07B215,12       ; ENC3 compare initial state
M2714->X:$07B215,14       ; AENA3 output status
M2715->X:$07B210,19       ; USER3 flag input status
M2716->X:$07B210,9        ; ENC3 compare output value
M2717->X:$07B210,11       ; ENC3 capture flag
M2718->X:$07B210,8        ; ENC3 count error flag
M2719->X:$07B210,14       ; CHC3 input status
M2720->X:$07B210,16       ; HMFL3 flag input status
M2721->X:$07B210,17       ; PLIM3 flag input status
M2722->X:$07B210,18       ; MLIM3 flag input status
M2723->X:$07B210,15       ; FAULT3 flag input status
M2724->X:$07B210,20       ; Channel 3 W flag input status
M2725->X:$07B210,21       ; Channel 3 V flag input status
M2726->X:$07B210,22       ; Channel 3 U flag input status
M2727->X:$07B210,23       ; Channel 3 T flag input status
M2728->X:$07B210,20,4     ; Channel 3 TUVW inputs as 4-bit value
; Motor #27 Status Bits
M2730->Y:$000DC0,11,1     ; #27 Stopped-on-position-limit bit
M2731->X:$000DB0,21,1     ; #27 Positive-end-limit-set bit
M2732->X:$000DB0,22,1     ; #27 Negative-end-limit-set bit
M2733->X:$000DB0,13,1     ; #27 Desired-velocity-zero bit
M2735->X:$000DB0,15,1     ; #27 Dwell-in-progress bit
M2737->X:$000DB0,17,1     ; #27 Running-program bit
M2738->X:$000DB0,18,1     ; #27 Open-loop-mode bit
M2739->X:$000DB0,19,1     ; #27 Amplifier-enabled status bit
M2740->Y:$000DC0,0,1      ; #27 Background in-position bit
M2741->Y:$000DC0,1,1      ; #27 Warning-following error bit
M2742->Y:$000DC0,2,1      ; #27 Fatal-following-error bit
M2743->Y:$000DC0,3,1      ; #27 Amplifier-fault-error bit
M2744->Y:$000DC0,13,1     ; #27 Foreground in-position bit
M2745->Y:$000DC0,10,1     ; #27 Home-complete bit
M2746->Y:$000DC0,6,1      ; #27 Integrated following error fault bit
M2747->Y:$000DC0,5,1      ; #27 I2T fault bit
M2748->Y:$000DC0,8,1      ; #27 Phasing error fault bit
M2749->Y:$000DC0,9,1      ; #27 Phasing search-in-progress bit
```

```
; MACRO IC 3 Node 4 Flag Registers (usually used for Motor #27)
M2750->X:$003474,0,24      ; MACRO IC 3 Node 4 flag status register
M2751->Y:$003474,0,24      ; MACRO IC 3 Node 4 flag command register
M2753->X:$003474,20,4      ; MACRO IC 3 Node 4 TUVW flags
M2754->Y:$003474,14,1      ; MACRO IC 3 Node 4 amplifier enable flag
M2755->X:$003474,15,1      ; MACRO IC 3 Node 4 node/amplifier fault flag
M2756->X:$003474,16,1      ; MACRO IC 3 Node 4 home flag
M2757->X:$003474,17,1      ; MACRO IC 3 Node 4 positive limit flag
M2758->X:$003474,18,1      ; MACRO IC 3 Node 4 negative limit flag
M2759->X:$003474,19,1      ; MACRO IC 3 Node 4 user flag
; Motor #27 Move Registers
M2761->D:$000D88           ; #27 Commanded position (1/[Ixx08*32] cts)
M2762->D:$000D8B           ; #27 Actual position (1/[Ixx08*32] cts)
M2763->D:$000DC7           ; #27 Target (end) position (1/[Ixx08*32] cts)
M2764->D:$000DCC           ; #27 Position bias (1/[Ixx08*32] cts)
M2766->X:$000D9D,0,24,S    ; #27 Actual velocity (1/[Ixx09*32] cts/cyc)
M2767->D:$000D8D           ; #27 Present master pos (1/[Ixx07*32] cts)
M2768->X:$000DBF,8,16,S    ; #27 Filter Output (16-bit DAC bits)
M2769->D:$000D90           ; #27 Compensation correction (1/[Ixx08*32] cts)
M2770->D:$000DB4           ; #27 Present phase position (including fraction)
M2771->X:$000DB4,24,S      ; #27 Present phase position (counts *Ixx70)
M2772->L:$000DD7           ; #27 Variable jog position/distance (cts)
M2773->Y:$000DCE,0,24,S    ; #27 Encoder home capture position (cts)
M2774->D:$000DEF           ; #27 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2775->X:$000DB9,8,16,S    ; #27 Actual quadrature current
M2776->Y:$000DB9,8,16,S    ; #27 Actual direct current
M2777->X:$000DBC,8,16,S    ; #27 Quadrature current-loop integrator output
M2778->Y:$000DBC,8,16,S    ; #27 Direct current-loop integrator output
M2779->X:$000DAE,8,16,S    ; #27 PID internal filter result (16-bit DAC bits)
M2788->Y:$07B211,0,12,U    ; IC 8 Ch 3 Compare A fractional count
M2789->Y:$07B210,0,12,U    ; IC 8 Ch 3 Compare B fractional count
; Motor #27 Axis Definition Registers
M2791->L:$000DCF           ; #27 X/U/A/B/C-Axis scale factor (cts/unit)
M2792->L:$000DD0           ; #27 Y/V-Axis scale factor (cts/unit)
M2793->L:$000DD1           ; #27 Z/W-Axis scale factor (cts/unit)
M2794->L:$000DD2           ; #27 Axis offset (cts)
; Servo IC 8 Channel 4 Registers (usually for Motor #28)
M2801->X:$07B219,0,24,S    ; ENC4 24-bit counter position
M2802->Y:$07B21A,8,16,S    ; OUT4A command value; DAC or PWM
M2803->X:$07B21B,0,24,S    ; ENC4 captured position
M2804->Y:$07B21B,8,16,S    ; OUT4B command value; DAC or PWM
M2805->Y:$07B21D,8,16,S    ; ADC4A input value
M2806->Y:$07B21E,8,16,S    ; ADC4B input value
M2807->Y:$07B21C,8,16,S    ; OUT4C command value; PFM or PWM
M2808->Y:$07B21F,0,24,S    ; ENC4 compare A position
M2809->X:$07B21F,0,24,S    ; ENC4 compare B position
M2810->X:$07B21E,0,24,S    ; ENC4 compare autoincrement value
```

```
M2811->X:$07B21D,11        ; ENC4 compare initial state write enable
M2812->X:$07B21D,12        ; ENC4 compare initial state
M2814->X:$07B21D,14        ; AENA4 output status
M2815->X:$07B218,19        ; USER4 flag input status
M2816->X:$07B218,9         ; ENC4 compare output value
M2817->X:$07B218,11        ; ENC4 capture flag
M2818->X:$07B218,8         ; ENC4 count error flag
M2819->X:$07B218,14        ; HMFL4 flag input status
M2820->X:$07B218,16        ; CHC4 input status
M2821->X:$07B218,17        ; PLIM4 flag input status
M2822->X:$07B218,18        ; MLIM4 flag input status
M2823->X:$07B218,15        ; FAULT4 flag input status
M2824->X:$07B218,20        ; Channel 4 W flag input status
M2825->X:$07B218,21        ; Channel 4 V flag input status
M2826->X:$07B218,22        ; Channel 4 U flag input status
M2827->X:$07B218,23        ; Channel 4 T flag input status
M2828->X:$07B218,20,4      ; Channel 4 TUVW inputs as 4-bit value
; Motor #28 Status Bits
M2830->Y:$000E40,11,1      ; #28 Stopped-on-position-limit bit
M2831->X:$000E30,21,1      ; #28 Positive-end-limit-set bit
M2832->X:$000E30,22,1      ; #28 Negative-end-limit-set bit
M2833->X:$000E30,13,1      ; #28 Desired-velocity-zero bit
M2835->X:$000E30,15,1      ; #28 Dwell-in-progress bit
M2837->X:$000E30,17,1      ; #28 Running-program bit
M2838->X:$000E30,18,1      ; #28 Open-loop-mode bit
M2839->X:$000E30,19,1      ; #28 Amplifier-enabled status bit
M2840->Y:$000E40,0,1       ; #28 Background in-position bit
M2841->Y:$000E40,1,1       ; #28 Warning-following error bit
M2842->Y:$000E40,2,1       ; #28 Fatal-following-error bit
M2843->Y:$000E40,3,1       ; #28 Amplifier-fault-error bit
M2844->Y:$000E40,13,1      ; #28 Foreground in-position bit
M2845->Y:$000E40,10,1      ; #28 Home-complete bit
M2846->Y:$000E40,6,1       ; #28 Integrated following error fault bit
M2847->Y:$000E40,5,1       ; #28 I2T fault bit
M2848->Y:$000E40,8,1       ; #28 Phasing error fault bit
M2849->Y:$000E40,9,1       ; #28 Phasing search-in-progress bit
; MACRO IC 3 Node 5 Flag Registers (usually used for Motor #28)
M2850->X:$003475,0,24      ; MACRO IC 3 Node 5 flag status register
M2851->Y:$003475,0,24      ; MACRO IC 3 Node 5 flag command register
M2853->X:$003475,20,4      ; MACRO IC 3 Node 5 TUVW flags
M2854->Y:$003475,14,1      ; MACRO IC 3 Node 5 amplifier enable flag
M2855->X:$003475,15,1      ; MACRO IC 3 Node 5 node/amplifier fault flag
M2856->X:$003475,16,1      ; MACRO IC 3 Node 5 home flag
M2857->X:$003475,17,1      ; MACRO IC 3 Node 5 positive limit flag
M2858->X:$003475,18,1      ; MACRO IC 3 Node 5 negative limit flag
M2859->X:$003475,19,1      ; MACRO IC 3 Node 5 user flag
```

```
; Motor #28 Move Registers
M2861->D:$000E08            ; #28 Commanded position (1/[Ixx08*32] cts)
M2862->D:$000E0B            ; #28 Actual position (1/[Ixx08*32] cts)
M2863->D:$000E47            ; #28 Target (end) position (1/[Ixx08*32] cts)
M2864->D:$000E4C            ; #28 Position bias (1/[Ixx08*32] cts)
M2866->X:$000E1D,0,24,S     ; #28 Actual velocity (1/[Ixx09*32] cts/cyc)
M2867->D:$000E0D            ; #28 Present master pos (1/[Ixx07*32] cts)
M2868->X:$000E3F,8,16,S     ; #28 Filter Output (16-bit DAC bits)
M2869->D:$000E10            ; #28 Compensation correction (1/[Ixx08*32] cts)
M2870->D:$000E34            ; #28 Present phase position (including fraction)
M2871->X:$000E34,24,S       ; #28 Present phase position (counts *Ixx70)
M2872->L:$000E57            ; #28 Variable jog position/distance (cts)
M2873->Y:$000E4E,0,24,S     ; #28 Encoder home capture position (cts)
M2874->D:$000E6F            ; #28 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2875->X:$000E39,8,16,S     ; #28 Actual quadrature current
M2876->Y:$000E39,8,16,S     ; #28 Actual direct current
M2877->X:$000E3C,8,16,S     ; #28 Quadrature current-loop integrator output
M2878->Y:$000E3C,8,16,S     ; #28 Direct current-loop integrator output
M2879->X:$000E2E,8,16,S     ; #28 PID internal filter result (16-bit DAC bits)
M2888->Y:$07B219,0,12,U     ; IC 8 Ch 4 Compare A fractional count
M2889->Y:$07B218,0,12,U     ; IC 8 Ch 4 Compare B fractional count
; Motor #28 Axis Definition Registers
M2891->L:$000E4F            ; #28 X/U/A/B/C-Axis scale factor (cts/unit)
M2892->L:$000E50            ; #28 Y/V-Axis scale factor (cts/unit)
M2893->L:$000E51            ; #28 Z/W-Axis scale factor (cts/unit)
M2894->L:$000E52            ; #28 Axis offset (cts)
; Servo IC 9 Channel 1 Registers (usually for Motor #29)
M2901->X:$07B301,0,24,S     ; ENC5 24-bit counter position
M2902->Y:$07B302,8,16,S     ; OUT5A command value; DAC or PWM
M2903->X:$07B303,0,24,S     ; ENC5 captured position
M2904->Y:$07B303,8,16,S     ; OUT5B command value; DAC or PWM
M2905->Y:$07B305,8,16,S     ; ADC5A input value
M2906->Y:$07B306,8,16,S     ; ADC5B input value
M2907->Y:$07B304,8,16,S     ; OUT5C command value; PFM or PWM
M2908->Y:$07B307,0,24,S     ; ENC5 compare A position
M2909->X:$07B307,0,24,S     ; ENC5 compare B position
M2910->X:$07B306,0,24,S     ; ENC5 compare autoincrement value
M2911->X:$07B305,11         ; ENC5 compare initial state write enable
M2912->X:$07B305,12         ; ENC5 compare initial state
M2914->X:$07B305,14         ; AENA5 output status
M2915->X:$07B300,19         ; USER5 flag input status
M2916->X:$07B300,9          ; ENC5 compare output value
M2917->X:$07B300,11         ; ENC5 capture flag
M2918->X:$07B300,8          ; ENC5 count error flag
M2919->X:$07B300,14         ; CHC5 input status
M2920->X:$07B300,16         ; HMFL5 flag input status
M2921->X:$07B300,17         ; PLIM5 flag input status
```

```
M2922->X:$07B300,18        ; MLIM5 flag input status
M2923->X:$07B300,15        ; FAULT5 flag input status
M2924->X:$07B300,20        ; Channel 5 W flag input status
M2925->X:$07B300,21        ; Channel 5 V flag input status
M2926->X:$07B300,22        ; Channel 5 U flag input status
M2927->X:$07B300,23        ; Channel 5 T flag input status
M2928->X:$07B300,20,4      ; Channel 5 TUVW inputs as 4-bit value
; Motor #29 Status Bits
M2930->Y:$000EC0,11,1      ; #29 Stopped-on-position-limit bit
M2931->X:$000EB0,21,1      ; #29 Positive-end-limit-set bit
M2932->X:$000EB0,22,1      ; #29 Negative-end-limit-set bit
M2933->X:$000EB0,13,1      ; #29 Desired-velocity-zero bit
M2935->X:$000EB0,15,1      ; #29 Dwell-in-progress bit
M2937->X:$000EB0,17,1      ; #29 Running-program bit
M2938->X:$000EB0,18,1      ; #29 Open-loop-mode bit
M2939->X:$000EB0,19,1      ; #29 Amplifier-enabled status bit
M2940->Y:$000EC0,0,1       ; #29 Background in-position bit
M2941->Y:$000EC0,1,1       ; #29 Warning-following error bit
M2942->Y:$000EC0,2,1       ; #29 Fatal-following-error bit
M2943->Y:$000EC0,3,1       ; #29 Amplifier-fault-error bit
M2944->Y:$000EC0,13,1      ; #29 Foreground in-position bit
M2945->Y:$000EC0,10,1      ; #29 Home-complete bit
M2946->Y:$000EC0,6,1       ; #29 Integrated following error fault bit
M2947->Y:$000EC0,5,1       ; #29 I2T fault bit
M2948->Y:$000EC0,8,1       ; #29 Phasing error fault bit
M2949->Y:$000EC0,9,1       ; #29 Phasing search-in-progress bit
; MACRO IC 3 Node 8 Flag Registers (usually used for Motor #29)
M2950->X:$003478,0,24      ; MACRO IC 3 Node 8 flag status register
M2951->Y:$003478,0,24      ; MACRO IC 3 Node 8 flag command register
M2953->X:$003478,20,4      ; MACRO IC 3 Node 8 TUVW flags
M2954->Y:$003478,14,1      ; MACRO IC 3 Node 8 amplifier enable flag
M2955->X:$003478,15,1      ; MACRO IC 3 Node 8 node/amplifier fault flag
M2956->X:$003478,16,1      ; MACRO IC 3 Node 8 home flag
M2957->X:$003478,17,1      ; MACRO IC 3 Node 8 positive limit flag
M2958->X:$003478,18,1      ; MACRO IC 3 Node 8 negative limit flag
M2959->X:$003478,19,1      ; MACRO IC 3 Node 8 user flag
; Motor #29 Move Registers
M2961->D:$000E88           ; #29 Commanded position (1/[Ixx08*32] cts)
M2962->D:$000E8B           ; #29 Actual position (1/[Ixx08*32] cts)
M2963->D:$000EC7           ; #29 Target (end) position (1/[Ixx08*32] cts)
M2964->D:$000ECC           ; #29 Position bias (1/[Ixx08*32] cts)
M2966->X:$000E9D,0,24,S    ; #29 Actual velocity (1/[Ixx09*32] cts/cyc)
M2967->D:$000E8D           ; #29 Present master pos (1/[Ixx07*32] cts)
M2968->X:$000EBF,8,16,S    ; #29 Filter Output (16-bit DAC bits)
M2969->D:$000E90           ; #29 Compensation correction (1/[Ixx08*32] cts)
M2970->D:$000EB4           ; #29 Present phase position (including fraction)
M2971->X:$000EB4,24,S      ; #29 Present phase position (counts *Ixx70)
```

```
M2972->L:$000ED7          ; #29 Variable jog position/distance (cts)
M2973->Y:$000ECE,0,24,S   ; #29 Encoder home capture position (cts)
M2974->D:$000EEF          ; #29 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M2975->X:$000EB9,8,16,S   ; #29 Actual quadrature current
M2976->Y:$000EB9,8,16,S   ; #29 Actual direct current
M2977->X:$000EBC,8,16,S   ; #29 Quadrature current-loop integrator output
M2978->Y:$000EBC,8,16,S   ; #29 Direct current-loop integrator output
M2979->X:$000EAE,8,16,S   ; #29 PID internal filter result (16-bit DAC bits)
M2988->Y:$07B301,0,12,U   ; IC 9 Ch 1 Compare A fractional count
M2989->Y:$07B300,0,12,U   ; IC 9 Ch 1 Compare B fractional count
; Motor #29 Axis Definition Registers
M2991->L:$000ECF          ; #29 X/U/A/B/C-Axis scale factor (cts/unit)
M2992->L:$000ED0          ; #29 Y/V-Axis scale factor (cts/unit)
M2993->L:$000ED1          ; #29 Z/W-Axis scale factor (cts/unit)
M2994->L:$000ED2          ; #29 Axis offset (cts)
; Servo IC 9 Channel 2 Registers (usually for Motor #30)
M3001->X:$07B309,0,24,S   ; ENC6 24-bit counter position
M3002->Y:$07B30A,8,16,S   ; OUT6A command value; DAC or PWM
M3003->X:$07B30B,0,24,S   ; ENC6 captured position
M3004->Y:$07B30B,8,16,S   ; OUT6B command value; DAC or PWM
M3005->Y:$07B30D,8,16,S   ; ADC6A input value
M3006->Y:$07B30E,8,16,S   ; ADC6B input value
M3007->Y:$07B30C,8,16,S   ; OUT6C command value; PFM or PWM
M3008->Y:$07B30F,0,24,S   ; ENC6 compare A position
M3009->X:$07B30F,0,24,S   ; ENC6 compare B position
M3010->X:$07B30E,0,24,S   ; ENC6 compare autoincrement value
M3011->X:$07B30D,11       ; ENC6 compare initial state write enable
M3012->X:$07B30D,12       ; ENC6 compare initial state
M3014->X:$07B30D,14       ; AENA6 output status
M3015->X:$07B308,19       ; USER6 flag input status
M3016->X:$07B308,9        ; ENC6 compare output value
M3017->X:$07B308,11       ; ENC6 capture flag
M3018->X:$07B308,8        ; ENC6 count error flag
M3019->X:$07B308,14       ; CHC6 input status
M3020->X:$07B308,16       ; HMFL6 flag input status
M3021->X:$07B308,17       ; PLIM6 flag input status
M3022->X:$07B308,18       ; MLIM6 flag input status
M3023->X:$07B308,15       ; FAULT6 flag input status
M3024->X:$07B308,20       ; Channel 6 W flag input status
M3025->X:$07B308,21       ; Channel 6 V flag input status
M3026->X:$07B308,22       ; Channel 6 U flag input status
M3027->X:$07B308,23       ; Channel 6 T flag input status
M3028->X:$07B308,20,4     ; Channel 6 TUVW inputs as 4-bit value
; Motor #30 Status Bits
M3030->Y:$000F40,11,1     ; #30 Stopped-on-position-limit bit
M3031->X:$000F30,21,1     ; #30 Positive-end-limit-set bit
M3032->X:$000F30,22,1     ; #30 Negative-end-limit-set bit
```

```
M3033->X:$000F30,13,1      ; #30 Desired-velocity-zero bit
M3035->X:$000F30,15,1      ; #30 Dwell-in-progress bit
M3037->X:$000F30,17,1      ; #30 Running-program bit
M3038->X:$000F30,18,1      ; #30 Open-loop-mode bit
M3039->X:$000F30,19,1      ; #30 Amplifier-enabled status bit
M3040->Y:$000F40,0,1       ; #30 Background in-position bit
M3041->Y:$000F40,1,1       ; #30 Warning-following error bit
M3042->Y:$000F40,2,1       ; #30 Fatal-following-error bit
M3043->Y:$000F40,3,1       ; #30 Amplifier-fault-error bit
M3044->Y:$000F40,13,1      ; #30 Foreground in-position bit
M3045->Y:$000F40,10,1      ; #30 Home-complete bit
M3046->Y:$000F40,6,1       ; #30 Integrated following error fault bit
M3047->Y:$000F40,5,1       ; #30 I2T fault bit
M3048->Y:$000F40,8,1       ; #30 Phasing error fault bit
M3049->Y:$000F40,9,1       ; #30 Phasing search-in-progress bit
; MACRO IC 3 Node 9 Flag Registers (usually used for Motor #30)
M3050->X:$003479,0,24      ; MACRO IC 3 Node 9 flag status register
M3051->Y:$003479,0,24      ; MACRO IC 3 Node 9 flag command register
M3053->X:$003479,20,4      ; MACRO IC 3 Node 9 TUVW flags
M3054->Y:$003479,14,1      ; MACRO IC 3 Node 9 amplifier enable flag
M3055->X:$003479,15,1      ; MACRO IC 3 Node 9 node/amplifier fault flag
M3056->X:$003479,16,1      ; MACRO IC 3 Node 9 home flag
M3057->X:$003479,17,1      ; MACRO IC 3 Node 9 positive limit flag
M3058->X:$003479,18,1      ; MACRO IC 3 Node 9 negative limit flag
M3059->X:$003479,19,1      ; MACRO IC 3 Node 9 user flag
; Motor #30 Move Registers
M3061->D:$000F08           ; #30 Commanded position (1/[Ixx08*32] cts)
M3062->D:$000F0B           ; #30 Actual position (1/[Ixx08*32] cts)
M3063->D:$000F47           ; #30 Target (end) position (1/[Ixx08*32] cts)
M3064->D:$000F4C           ; #30 Position bias (1/[Ixx08*32] cts)
M3066->X:$000F1D,0,24,S    ; #30 Actual velocity (1/[Ixx09*32] cts/cyc)
M3067->D:$000F0D           ; #30 Present master pos (1/[Ixx07*32] cts)
M3068->X:$000F3F,8,16,S    ; #30 Filter Output (16-bit DAC bits)
M3069->D:$000F10           ; #30 Compensation correction (1/[Ixx08*32] cts)
M3070->D:$000F34           ; #30 Present phase position (including fraction)
M3071->X:$000F34,24,S      ; #30 Present phase position (counts *Ixx70)
M3072->L:$000F57           ; #30 Variable jog position/distance (cts)
M3073->Y:$000F4E,0,24,S    ; #30 Encoder home capture position (cts)
M3074->D:$000F6F           ; #30 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3075->X:$000F39,8,16,S    ; #30 Actual quadrature current
M3076->Y:$000F39,8,16,S    ; #30 Actual direct current
M3077->X:$000F3C,8,16,S    ; #30 Quadrature current-loop integrator output
M3078->Y:$000F3C,8,16,S    ; #30 Direct current-loop integrator output
M3079->X:$000F2E,8,16,S    ; #30 PID internal filter result (16-bit DAC bits)
M3088->Y:$07B309,0,12,U    ; IC 9 Ch 2 Compare A fractional count
M3089->Y:$07B308,0,12,U    ; IC 9 Ch 2 Compare B fractional count
```

```
; Motor #30 Axis Definition Registers
M3091->L:$000F4F          ; #30 X/U/A/B/C-Axis scale factor (cts/unit)
M3092->L:$000F50          ; #30 Y/V-Axis scale factor (cts/unit)
M3093->L:$000F51          ; #30 Z/W-Axis scale factor (cts/unit)
M3094->L:$000F52          ; #30 Axis offset (cts)
; Servo IC 9 Channel 3 Registers (usually for Motor #31)
M3101->X:$07B311,0,24,S   ; ENC7 24-bit counter position
M3102->Y:$07B312,8,16,S   ; OUT7A command value; DAC or PWM
M3103->X:$07B313,0,24,S   ; ENC7 captured position
M3104->Y:$07B313,8,16,S   ; OUT7B command value; DAC or PWM
M3105->Y:$07B315,8,16,S   ; ADC7A input value
M3106->Y:$07B316,8,16,S   ; ADC7B input value
M3107->Y:$07B314,8,16,S   ; OUT7C command value; PFM or PWM
M3108->Y:$07B317,0,24,S   ; ENC7 compare A position
M3109->X:$07B317,0,24,S   ; ENC7 compare B position
M3110->X:$07B316,0,24,S   ; ENC7 compare autoincrement value
M3111->X:$07B315,11       ; ENC7 compare initial state write enable
M3112->X:$07B315,12       ; ENC7 compare initial state
M3114->X:$07B315,14       ; AENA7 output status
M3115->X:$07B310,19       ; CHC7 input status
M3116->X:$07B310,9        ; ENC7 compare output value
M3117->X:$07B310,11       ; ENC7 capture flag
M3118->X:$07B310,8        ; ENC7 count error flag
M3119->X:$07B310,14       ; CHC7 input status
M3120->X:$07B310,16       ; HMFL7 flag input status
M3121->X:$07B310,17       ; PLIM7 flag input status
M3122->X:$07B310,18       ; MLIM7 flag input status
M3123->X:$07B310,15       ; FAULT7 flag input status
M3124->X:$07B310,20       ; Channel 7 W flag input status
M3125->X:$07B310,21       ; Channel 7 V flag input status
M3126->X:$07B310,22       ; Channel 7 U flag input status
M3127->X:$07B310,23       ; Channel 7 T flag input status
M3128->X:$07B310,20,4     ; Channel 7 TUVW inputs as 4-bit value
; Motor #31 Status Bits
M3130->Y:$000FC0,11,1     ; #31 Stopped-on-position-limit bit
M3131->X:$000FB0,21,1     ; #31 Positive-end-limit-set bit
M3132->X:$000FB0,22,1     ; #31 Negative-end-limit-set bit
M3133->X:$000FB0,13,1     ; #31 Desired-velocity-zero bit
M3135->X:$000FB0,15,1     ; #31 Dwell-in-progress bit
M3137->X:$000FB0,17,1     ; #31 Running-program bit
M3138->X:$000FB0,18,1     ; #31 Open-loop-mode bit
M3139->X:$000FB0,19,1     ; #31 Amplifier-enabled status bit
M3140->Y:$000FC0,0,1      ; #31 Background in-position bit
M3141->Y:$000FC0,1,1      ; #31 Warning-following error bit
M3142->Y:$000FC0,2,1      ; #31 Fatal-following-error bit
M3143->Y:$000FC0,3,1      ; #31 Amplifier-fault-error bit
M3144->Y:$000FC0,13,1     ; #31 Foreground in-position bit
```

```
M3145->Y:$000FC0,10,1      ; #31 Home-complete bit
M3146->Y:$000FC0,6,1       ; #31 Integrated following error fault bit
M3147->Y:$000FC0,5,1       ; #31 I2T fault bit
M3148->Y:$000FC0,8,1       ; #31 Phasing error fault bit
M3149->Y:$000FC0,9,1       ; #31 Phasing search-in-progress bit
; MACRO IC 3 Node 12 Flag Registers (usually used for Motor #31)
M3150->X:$00347C,0,24      ; MACRO IC 3 Node 12 flag status register
M3151->Y:$00347C,0,24      ; MACRO IC 3 Node 12 flag command register
M3153->X:$00347C,20,4      ; MACRO IC 3 Node 12 TUVW flags
M3154->Y:$00347C,14,1      ; MACRO IC 3 Node 12 amplifier enable flag
M3155->X:$00347C,15,1      ; MACRO IC 3 Node 12 node/amplifier fault flag
M3156->X:$00347C,16,1      ; MACRO IC 3 Node 12 home flag
M3157->X:$00347C,17,1      ; MACRO IC 3 Node 12 positive limit flag
M3158->X:$00347C,18,1      ; MACRO IC 3 Node 12 negative limit flag
M3159->X:$00347C,19,1      ; MACRO IC 3 Node 12 user flag
; Motor #31 Move Registers
M3161->D:$000F88           ; #31 Commanded position (1/[Ixx08*32] cts)
M3162->D:$000F8B           ; #31 Actual position (1/[Ixx08*32] cts)
M3163->D:$000FC7           ; #31 Target (end) position (1/[Ixx08*32] cts)
M3164->D:$000FCC           ; #31 Position bias (1/[Ixx08*32] cts)
M3166->X:$000F9D,0,24,S    ; #31 Actual velocity (1/[Ixx09*32] cts/cyc)
M3167->D:$000F8D           ; #31 Present master pos (1/[Ixx07*32] cts)
M3168->X:$000FBF,8,16,S    ; #31 Filter Output (16-bit DAC bits)
M3169->D:$000F90           ; #31 Compensation correction (1/[Ixx08*32] cts)
M3170->D:$000FB4           ; #31 Present phase position (including fraction)
M3171->X:$000FB4,24,S      ; #31 Present phase position (counts *Ixx70)
M3172->L:$000FD7           ; #31 Variable jog position/distance (cts)
M3173->Y:$000FCE,0,24,S    ; #31 Encoder home capture position (cts)
M3174->D:$000FEF           ; #31 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3175->X:$000FB9,8,16,S    ; #31 Actual quadrature current
M3176->Y:$000FB9,8,16,S    ; #31 Actual direct current
M3177->X:$000FBC,8,16,S    ; #31 Quadrature current-loop integrator output
M3178->Y:$000FBC,8,16,S    ; #31 Direct current-loop integrator output
M3179->X:$000FAE,8,16,S    ; #31 PID internal filter result (16-bit DAC bits)
M3188->Y:$07B311,0,12,U    ; IC 9 Ch 3 Compare A fractional count
M3189->Y:$07B310,0,12,U    ; IC 9 Ch 3 Compare B fractional count
; Motor #31 Axis Definition Registers
M3191->L:$000FCF           ; #31 X/U/A/B/C-Axis scale factor (cts/unit)
M3192->L:$000FD0           ; #31 Y/V-Axis scale factor (cts/unit)
M3193->L:$000FD1           ; #31 Z/W-Axis scale factor (cts/unit)
M3194->L:$000FD2           ; #31 Axis offset (cts)
; Servo IC 9 Channel 4 Registers (usually for Motor #32)
M3201->X:$07B319,0,24,S    ; ENC8 24-bit counter position
M3202->Y:$07B31A,8,16,S    ; OUT8A command value; DAC or PWM
M3203->X:$07B31B,0,24,S    ; ENC8 captured position
M3204->Y:$07B31B,8,16,S    ; OUT8B command value; DAC or PWM
M3205->Y:$07B31D,8,16,S    ; ADC8A input value
```

```
M3206->Y:$07B31E,8,16,S   ; ADC8B input value
M3207->Y:$07B31C,8,16,S   ; OUT8C command value; PFM or PWM
M3208->Y:$07B31F,0,24,S   ; ENC8 compare A position
M3209->X:$07B31F,0,24,S   ; ENC8 compare B position
M3210->X:$07B31E,0,24,S   ; ENC8 compare autoincrement value
M3211->X:$07B31D,11       ; ENC8 compare initial state write enable
M3212->X:$07B31D,12       ; ENC8 compare initial state
M3214->X:$07B31D,14       ; AENA8 output status
M3215->X:$07B318,19       ; USER8 flag input status
M3216->X:$07B318,9        ; ENC8 compare output value
M3217->X:$07B318,11       ; ENC8 capture flag
M3218->X:$07B318,8        ; ENC8 count error flag
M3219->X:$07B318,14       ; CHC8 input status
M3220->X:$07B318,16       ; HMFL8 flag input status
M3221->X:$07B318,17       ; PLIM8 flag input status
M3222->X:$07B318,18       ; MLIM8 flag input status
M3223->X:$07B318,15       ; FAULT8 flag input status
M3224->X:$07B318,20       ; Channel 8 W flag input status
M3225->X:$07B318,21       ; Channel 8 V flag input status
M3226->X:$07B318,22       ; Channel 8 U flag input status
M3227->X:$07B318,23       ; Channel 8 T flag input status
M3228->X:$07B318,20,4     ; Channel 8 TUVW inputs as 4-bit value
; Motor #32 Status Bits
M3230->Y:$001040,11,1     ; #32 Stopped-on-position-limit bit
M3231->X:$001030,21,1     ; #32 Positive-end-limit-set bit
M3232->X:$001030,22,1     ; #32 Negative-end-limit-set bit
M3233->X:$001030,13,1     ; #32 Desired-velocity-zero bit
M3235->X:$001030,15,1     ; #32 Dwell-in-progress bit
M3237->X:$001030,17,1     ; #32 Running-program bit
M3238->X:$001030,18,1     ; #32 Open-loop-mode bit
M3239->X:$001030,19,1     ; #32 Amplifier-enabled status bit
M3240->Y:$001040,0,1      ; #32 Background in-position bit
M3241->Y:$001040,1,1      ; #32 Warning-following error bit
M3242->Y:$001040,2,1      ; #32 Fatal-following-error bit
M3243->Y:$001040,3,1      ; #32 Amplifier-fault-error bit
M3244->Y:$001040,13,1     ; #32 Foreground in-position bit
M3245->Y:$001040,10,1     ; #32 Home-complete bit
M3246->Y:$001040,6,1      ; #32 Integrated following error fault bit
M3247->Y:$001040,5,1      ; #32 I2T fault bit
M3248->Y:$001040,8,1      ; #32 Phasing error fault bit
M3249->Y:$001040,9,1      ; #32 Phasing search-in-progress bit
; MACRO IC 3 Node 13 Flag Registers (usually used for Motor #32)
M3250->X:$00347D,0,24     ; MACRO IC 3 Node 13 flag status register
M3251->Y:$00347D,0,24     ; MACRO IC 3 Node 13 flag command register
M3253->X:$00347D,20,4     ; MACRO IC 3 Node 13 TUVW flags
M3254->Y:$00347D,14,1     ; MACRO IC 3 Node 13 amplifier enable flag
M3255->X:$00347D,15,1     ; MACRO IC 3 Node 13 node/amplifier fault flag
```

```
M3256->X:$00347D,16,1      ; MACRO IC 3 Node 13 home flag
M3257->X:$00347D,17,1      ; MACRO IC 3 Node 13 positive limit flag
M3258->X:$00347D,18,1      ; MACRO IC 3 Node 13 negative limit flag
M3259->X:$00347D,19,1      ; MACRO IC 3 Node 13 user flag
; Motor #32 Move Registers
M3261->D:$001008           ; #32 Commanded position (1/[Ixx08*32] cts)
M3262->D:$00100B           ; #32 Actual position (1/[Ixx08*32] cts)
M3263->D:$001047           ; #32 Target (end) position (1/[Ixx08*32] cts)
M3264->D:$00104C           ; #32 Position bias (1/[Ixx08*32] cts)
M3266->X:$00101D,0,24,S    ; #32 Actual velocity (1/[Ixx09*32] cts/cyc)
M3267->D:$00100D           ; #32 Present master pos (1/[Ixx07*32] cts)
M3268->X:$00103F,8,16,S    ; #32 Filter Output (16-bit DAC bits)
M3269->D:$001010           ; #32 Compensation correction (1/[Ixx08*32] cts)
M3270->D:$001034           ; #32 Present phase position (including fraction)
M3271->X:$001034,24,S      ; #32 Present phase position (counts *Ixx70)
M3272->L:$001057           ; #32 Variable jog position/distance (cts)
M3273->Y:$00104E,0,24,S    ; #32 Encoder home capture position (cts)
M3274->D:$00106F           ; #32 Averaged actual velocity (1/[Ixx09*32] cts/cyc)
M3275->X:$001039,8,16,S    ; #32 Actual quadrature current
M3276->Y:$001039,8,16,S    ; #32 Actual direct current
M3277->X:$00103C,8,16,S    ; #32 Quadrature current-loop integrator output
M3278->Y:$00103C,8,16,S    ; #32 Direct current-loop integrator output
M3279->X:$00102E,8,16,S    ; #32 PID internal filter result (16-bit DAC bits)
M3288->Y:$07B319,0,12,U    ; IC 9 Ch 4 Compare A fractional count
M3289->Y:$07B318,0,12,U    ; IC 9 Ch 4 Compare B fractional count
; Motor #32 Axis Definition Registers
M3291->L:$00104F           ; #32 X/U/A/B/C-Axis scale factor (cts/unit)
M3292->L:$001050           ; #32 Y/V-Axis scale factor (cts/unit)
M3293->L:$001051           ; #32 Z/W-Axis scale factor (cts/unit)
M3294->L:$001052           ; #32 Axis offset (cts)
; De-multiplexed ADC values from Opt. 12, ACC-36
M5061->Y:$003400,12,12,U ; Demuxed low ADC register from I5061
M5062->Y:$003402,12,12,U ; Demuxed low ADC register from I5062
M5063->Y:$003404,12,12,U ; Demuxed low ADC register from I5063
M5064->Y:$003406,12,12,U ; Demuxed low ADC register from I5064
M5065->Y:$003408,12,12,U ; Demuxed low ADC register from I5065
M5066->Y:$00340A,12,12,U ; Demuxed low ADC register from I5066
M5067->Y:$00340C,12,12,U ; Demuxed low ADC register from I5067
M5068->Y:$00340E,12,12,U ; Demuxed low ADC register from I5068
M5069->Y:$003410,12,12,U ; Demuxed low ADC register from I5069
M5070->Y:$003412,12,12,U ; Demuxed low ADC register from I5070
M5071->Y:$003414,12,12,U ; Demuxed low ADC register from I5071
M5072->Y:$003416,12,12,U ; Demuxed low ADC register from I5072
M5073->Y:$003418,12,12,U ; Demuxed low ADC register from I5073
M5074->Y:$00341A,12,12,U ; Demuxed low ADC register from I5074
M5075->Y:$00341C,12,12,U ; Demuxed low ADC register from I5075
M5076->Y:$00341E,12,12,U ; Demuxed low ADC register from I5076
```

```
M5081->Y:$003401,12,12,U ; Demuxed high ADC register from I5061
M5082->Y:$003403,12,12,U ; Demuxed high ADC register from I5062
M5083->Y:$003405,12,12,U ; Demuxed high ADC register from I5063
M5084->Y:$003407,12,12,U ; Demuxed high ADC register from I5064
M5085->Y:$003409,12,12,U ; Demuxed high ADC register from I5065
M5086->Y:$00340B,12,12,U ; Demuxed high ADC register from I5066
M5087->Y:$00340D,12,12,U ; Demuxed high ADC register from I5067
M5088->Y:$00340F,12,12,U ; Demuxed high ADC register from I5068
M5089->Y:$003411,12,12,U ; Demuxed high ADC register from I5069
M5090->Y:$003413,12,12,U ; Demuxed high ADC register from I5070
M5091->Y:$003415,12,12,U ; Demuxed high ADC register from I5071
M5092->Y:$003417,12,12,U ; Demuxed high ADC register from I5072
M5093->Y:$003419,12,12,U ; Demuxed high ADC register from I5073
M5094->Y:$00341B,12,12,U ; Demuxed high ADC register from I5074
M5095->Y:$00341D,12,12,U ; Demuxed high ADC register from I5075
M5096->Y:$00341F,12,12,U ; Demuxed high ADC register from I5076
; Coordinate System 1 (&1) timers
M5111->X:$002015,0,24,S  ; &1 Isx11 timer (for synchronous assignment)
M5112->Y:$002015,0,24,S  ; &1 Isx12 timer (for synchronous assignment)
; Coordinate System 1 (&1) end-of-calculated-move positions
M5141->L:$002041         ; &1 A-axis target position (engineering units)
M5142->L:$002042         ; &1 B-axis target position (engineering units)
M5143->L:$002043         ; &1 C-axis target position (engineering units)
M5144->L:$002044         ; &1 U-axis target position (engineering units)
M5145->L:$002045         ; &1 V-axis target position (engineering units)
M5146->L:$002046         ; &1 W-axis target position (engineering units)
M5147->L:$002047         ; &1 X-axis target position (engineering units)
M5148->L:$002048         ; &1 Y-axis target position (engineering units)
M5149->L:$002049         ; &1 Z-axis target position (engineering units)
; Coordinate System 1 (&1) Status Bits
M5180->X:$002040,0,1     ; &1 Program-running bit
M5181->Y:$00203F,21,1    ; &1 Circle-radius-error bit
M5182->Y:$00203F,22,1    ; &1 Run-time-error bit
M5184->X:$002040,0,4     ; &1 Continuous motion request
M5187->Y:$00203F,17,1    ; &1 In-position bit (AND of motors)
M5188->Y:$00203F,18,1    ; &1 Warning-following-error bit (OR)
M5189->Y:$00203F,19,1    ; &1 Fatal-following-error bit (OR)
M5190->Y:$00203F,20,1    ; &1 Amp-fault-error bit (OR of motors)
; Coordinate System 1 (&1) Variables
M5197->X:$002000,0,24,S  ; &1 Host commanded time base (I10 units)
M5198->X:$002002,0,24,S  ; &1 Present time base (I10 units)
; Coordinate System 2 (&2) timers
M5211->X:$002115,0,24,S  ; &2 Isx11 timer (for synchronous assignment)
M5212->Y:$002115,0,24,S  ; &2 Isx12 timer (for synchronous assignment)
```

```
; Coordinate System 2 (&2) end-of-calculated-move positions
M5241->L:$002141            ; &2 A-axis target position (engineering units)
M5242->L:$002142            ; &2 B-axis target position (engineering units)
M5243->L:$002143            ; &2 C-axis target position (engineering units)
M5244->L:$002144            ; &2 U-axis target position (engineering units)
M5245->L:$002145            ; &2 V-axis target position (engineering units)
M5246->L:$002146            ; &2 W-axis target position (engineering units)
M5247->L:$002147            ; &2 X-axis target position (engineering units)
M5248->L:$002148            ; &2 Y-axis target position (engineering units)
M5249->L:$002149            ; &2 Z-axis target position (engineering units)
; Coordinate System 2 (&2) Status Bits
M5280->X:$002140,0,1      ; &2 Program-running bit
M5281->Y:$00213F,21,1       ; &2 Circle-radius-error bit
M5282->Y:$00213F,22,1       ; &2 Run-time-error bit
M5284->X:$002140,0,4        ; &2 Continuous motion request
M5287->Y:$00213F,17,1       ; &2 In-position bit (AND of motors)
M5288->Y:$00213F,18,1       ; &2 Warning-following-error bit (OR)
M5289->Y:$00213F,19,1       ; &2 Fatal-following-error bit (OR)
M5290->Y:$00213F,20,1       ; &2 Amp-fault-error bit (OR of motors)
; Coordinate System 2 (&2) Variables
M5297->X:$002100,0,24,S     ; &2 Host commanded time base (I10 units)
M5298->X:$002102,0,24,S     ; &2 Present time base (I10 units)
; Coordinate System 3 (&3) timers
M5311->X:$002215,0,24,S     ; &3 Isx11 timer (for synchronous assignment)
M5312->Y:$002215,0,24,S     ; &3 Isx12 timer (for synchronous assignment)
; Coordinate System 3 (&3) end-of-calculated-move positions
M5341->L:$002241            ; &3 A-axis target position (engineering units)
M5342->L:$002242            ; &3 B-axis target position (engineering units)
M5343->L:$002243            ; &3 C-axis target position (engineering units)
M5344->L:$002244            ; &3 U-axis target position (engineering units)
M5345->L:$002245            ; &3 V-axis target position (engineering units)
M5346->L:$002246            ; &3 W-axis target position (engineering units)
M5347->L:$002247            ; &3 X-axis target position (engineering units)
M5348->L:$002248            ; &3 Y-axis target position (engineering units)
M5349->L:$002249            ; &3 Z-axis target position (engineering units)
; Coordinate System 3 (&3) Status Bits
M5380->X:$002240,0,1        ; &3 Program-running bit
M5381->Y:$00223F,21,1       ; &3 Circle-radius-error bit
M5382->Y:$00223F,22,1       ; &3 Run-time-error bit
M5384->X:$002240,0,4        ; &3 Continuous motion request
M5387->Y:$00223F,17,1       ; &3 In-position bit (AND of motors)
M5388->Y:$00223F,18,1       ; &3 Warning-following-error bit (OR)
M5389->Y:$00223F,19,1       ; &3 Fatal-following-error bit (OR)
M5390->Y:$00223F,20,1       ; &3 Amp-fault-error bit (OR of motors)
```

```
; Coordinate System 3 (&3) Variables
M5397->X:$002200,0,24,S   ; &3 Host commanded time base (I10 units)
M5398->X:$002202,0,24,S   ; &3 Present time base (I10 units)
; Coordinate System 4 (&4) timers
M5411->X:$002315,0,24,S   ; &4 Isx11 timer (for synchronous assignment)
M5412->Y:$002315,0,24,S   ; &4 Isx12 timer (for synchronous assignment)
; Coordinate System 4 (&4) end-of-calculated-move positions
M5441->L:$002341          ; &4 A-axis target position (engineering units)
M5442->L:$002342          ; &4 B-axis target position (engineering units)
M5443->L:$002343          ; &4 C-axis target position (engineering units)
M5444->L:$002344          ; &4 U-axis target position (engineering units)
M5445->L:$002345          ; &4 V-axis target position (engineering units)
M5446->L:$002346          ; &4 W-axis target position (engineering units)
M5447->L:$002347          ; &4 X-axis target position (engineering units)
M5448->L:$002348          ; &4 Y-axis target position (engineering units)
M5449->L:$002349          ; &4 Z-axis target position (engineering units)
; Coordinate System 4 (&4) Status Bits
M5480->X:$002340,0,1      ; &4 Program-running bit
M5481->Y:$00233F,21,1     ; &4 Circle-radius-error bit
M5482->Y:$00233F,22,1     ; &4 Run-time-error bit
M5484->X:$002340,0,4      ; &4 Continuous motion request
M5487->Y:$00233F,17,1     ; &4 In-position bit (AND of motors)
M5488->Y:$00233F,18,1     ; &4 Warning-following-error bit (OR)
M5489->Y:$00233F,19,1     ; &4 Fatal-following-error bit (OR)
M5490->Y:$00233F,20,1     ; &4 Amp-fault-error bit (OR of motors)
; Coordinate System 4 (&4) Variables
M5497->X:$002300,0,24,S   ; &4 Host commanded time base (I10 units)
M5498->X:$002302,0,24,S   ; &4 Present time base (I10 units)
; Coordinate System 5 (&5) timers
M5511->X:$002415,0,24,S   ; &5 Isx11 timer (for synchronous assignment)
M5512->Y:$002415,0,24,S   ; &5 Isx12 timer (for synchronous assignment)
; Coordinate System 5 (&5) end-of-calculated-move positions
M5541->L:$002441          ; &5 A-axis target position (engineering units)
M5542->L:$002442          ; &5 B-axis target position (engineering units)
M5543->L:$002443          ; &5 C-axis target position (engineering units)
M5544->L:$002444          ; &5 U-axis target position (engineering units)
M5545->L:$002445          ; &5 V-axis target position (engineering units)
M5546->L:$002446          ; &5 W-axis target position (engineering units)
M5547->L:$002447          ; &5 X-axis target position (engineering units)
M5548->L:$002448          ; &5 Y-axis target position (engineering units)
M5549->L:$002449          ; &5 Z-axis target position (engineering units)
; Coordinate System 5 (&5) Status Bits
M5580->X:$002440,0,1      ; &5 Program-running bit
M5581->Y:$00243F,21,1     ; &5 Circle-radius-error bit
M5582->Y:$00243F,22,1     ; &5 Run-time-error bit
M5584->X:$002440,0,4      ; &5 Continuous motion request
```

```
M5587->Y:$00243F,17,1      ; &5 In-position bit (AND of motors)
M5588->Y:$00243F,18,1      ; &5 Warning-following-error bit (OR)
M5589->Y:$00243F,19,1      ; &5 Fatal-following-error bit (OR)
M5590->Y:$00243F,20,1      ; &5 Amp-fault-error bit (OR of motors)
; Coordinate System 5 (&5) Variables
M5597->X:$002400,0,24,S    ; &5 Host commanded time base (I10 units)
M5598->X:$002402,0,24,S    ; &5 Present time base (I10 units)
; Coordinate System 6 (&6) timers
M5611->X:$002515,0,24,S    ; &6 Isx11 timer (for synchronous assignment)
M5612->Y:$002515,0,24,S    ; &6 Isx12 timer (for synchronous assignment)
; Coordinate System 6 (&6) end-of-calculated-move positions
M5641->L:$002541           ; &6 A-axis target position (engineering units)
M5642->L:$002542           ; &6 B-axis target position (engineering units)
M5643->L:$002543           ; &6 C-axis target position (engineering units)
M5644->L:$002544           ; &6 U-axis target position (engineering units)
M5645->L:$002545           ; &6 V-axis target position (engineering units)
M5646->L:$002546           ; &6 W-axis target position (engineering units)
M5647->L:$002547           ; &6 X-axis target position (engineering units)
M5648->L:$002548           ; &6 Y-axis target position (engineering units)
M5649->L:$002549           ; &6 Z-axis target position (engineering units)
; Coordinate System 6 (&6) Status Bits
M5680->X:$002540,0,1       ; &6 Program-running bit
M5681->Y:$00253F,21,1      ; &6 Circle-radius-error bit
M5682->Y:$00253F,22,1      ; &6 Run-time-error bit
M5684->X:$002540,0,4       ; &6 Continuous motion request
M5687->Y:$00253F,17,1      ; &6 In-position bit (AND of motors)
M5688->Y:$00253F,18,1      ; &6 Warning-following-error bit (OR)
M5689->Y:$00253F,19,1      ; &6 Fatal-following-error bit (OR)
M5690->Y:$00253F,20,1      ; &6 Amp-fault-error bit (OR of motors)
; Coordinate System 6 (&6) Variables
M5697->X:$002500,0,24,S    ; &6 Host commanded time base (I10 units)
M5698->X:$002502,0,24,S    ; &6 Present time base (I10 units)
; Coordinate System 7 (&7) timers
M5711->X:$002615,0,24,S    ; &7 Isx11 timer (for synchronous assignment)
M5712->Y:$002615,0,24,S    ; &7 Isx12 timer (for synchronous assignment)
; Coordinate System 7 (&7) end-of-calculated-move positions
M5741->L:$002641           ; &7 A-axis target position (engineering units)
M5742->L:$002642           ; &7 B-axis target position (engineering units)
M5743->L:$002643           ; &7 C-axis target position (engineering units)
M5744->L:$002644           ; &7 U-axis target position (engineering units)
M5745->L:$002645           ; &7 V-axis target position (engineering units)
M5746->L:$002646           ; &7 W-axis target position (engineering units)
M5747->L:$002647           ; &7 X-axis target position (engineering units)
M5748->L:$002648           ; &7 Y-axis target position (engineering units)
M5749->L:$002649           ; &7 Z-axis target position (engineering units)
```

```
; Coordinate System 7 (&7) Status Bits
M5780->X:$002640,0,1        ; &7 Program-running bit
M5781->Y:$00263F,21,1       ; &7 Circle-radius-error bit
M5782->Y:$00263F,22,1       ; &7 Run-time-error bit
M5784->X:$002640,0,4        ; &7 Continuous motion request
M5787->Y:$00263F,17,1       ; &7 In-position bit (AND of motors)
M5788->Y:$00263F,18,1       ; &7 Warning-following-error bit (OR)
M5789->Y:$00263F,19,1       ; &7 Fatal-following-error bit (OR)
M5790->Y:$00263F,20,1       ; &7 Amp-fault-error bit (OR of motors)
; Coordinate System 7 (&7) Variables
M5797->X:$002600,0,24,S     ; &7 Host commanded time base (I10 units)
M5798->X:$002602,0,24,S     ; &7 Present time base (I10 units)
; Coordinate System 8 (&8) timers
M5811->X:$002715,0,24,S     ; &8 Isx11 timer (for synchronous assignment)
M5812->Y:$002715,0,24,S     ; &8 Isx12 timer (for synchronous assignment)
; Coordinate System 8 (&8) end-of-calculated-move positions
M5841->L:$002741            ; &8 A-axis target position (engineering units)
M5842->L:$002742            ; &8 B-axis target position (engineering units)
M5843->L:$002743            ; &8 C-axis target position (engineering units)
M5844->L:$002744            ; &8 U-axis target position (engineering units)
M5845->L:$002745            ; &8 V-axis target position (engineering units)
M5846->L:$002746            ; &8 W-axis target position (engineering units)
M5847->L:$002747            ; &8 X-axis target position (engineering units)
M5848->L:$002748            ; &8 Y-axis target position (engineering units)
M5849->L:$002749            ; &8 Z-axis target position (engineering units)
; Coordinate System 8 (&8) Status Bits
M5880->X:$002740,0,1        ; &8 Program-running bit
M5881->Y:$00273F,21,1       ; &8 Circle-radius-error bit
M5882->Y:$00273F,22,1       ; &8 Run-time-error bit
M5884->X:$002740,0,4        ; &8 Continuous motion request
M5887->Y:$00273F,17,1       ; &8 In-position bit (AND of motors)
M5888->Y:$00273F,18,1       ; &8 Warning-following-error bit (OR)
M5889->Y:$00273F,19,1       ; &8 Fatal-following-error bit (OR)
M5890->Y:$00273F,20,1       ; &8 Amp-fault-error bit (OR of motors)
; Coordinate System 8 (&8) Variables
M5897->X:$002700,0,24,S     ; &8 Host commanded time base (I10 units)
M5898->X:$002702,0,24,S     ; &8 Present time base (I10 units)
; Coordinate System 9 (&9) timers
M5911->X:$002815,0,24,S     ; &9 Isx11 timer (for synchronous assignment)
M5912->Y:$002815,0,24,S     ; &9 Isx12 timer (for synchronous assignment)
; Coordinate System 9 (&9) end-of-calculated-move positions
M5941->L:$002841            ; &9 A-axis target position (engineering units)
M5942->L:$002842            ; &9 B-axis target position (engineering units)
M5943->L:$002843            ; &9 C-axis target position (engineering units)
M5944->L:$002844            ; &9 U-axis target position (engineering units)
M5945->L:$002845            ; &9 V-axis target position (engineering units)
```

```
M5946->L:$002846            ; &9 W-axis target position (engineering units)
M5947->L:$002847            ; &9 X-axis target position (engineering units)
M5948->L:$002848            ; &9 Y-axis target position (engineering units)
M5949->L:$002849            ; &9 Z-axis target position (engineering units)
; Coordinate System 1 (&1) Status Bits
M5980->X:$002840,0,1        ; &9 Program-running bit
M5981->Y:$00283F,21,1       ; &9 Circle-radius-error bit
M5982->Y:$00283F,22,1       ; &9 Run-time-error bit
M5984->X:$002840,0,4        ; &9 Continuous motion request
M5987->Y:$00283F,17,1       ; &9 In-position bit (AND of motors)
M5988->Y:$00283F,18,1       ; &9 Warning-following-error bit (OR)
M5989->Y:$00283F,19,1       ; &9 Fatal-following-error bit (OR)
M5990->Y:$00283F,20,1       ; &9 Amp-fault-error bit (OR of motors)
; Coordinate System 1 (&1) Variables
M5997->X:$002800,0,24,S     ; &9 Host commanded time base (I10 units)
M5998->X:$002802,0,24,S     ; &9 Present time base (I10 units)
; Coordinate System 10 (&10) timers
M6011->X:$002915,0,24,S     ; &10 Isx11 timer (for synchronous assignment)
M6012->Y:$002915,0,24,S     ; &10 Isx12 timer (for synchronous assignment)
; Coordinate System 10 (&10) end-of-calculated-move positions
M6041->L:$002941            ; &10 A-axis target position (engineering units)
M6042->L:$002942            ; &10 B-axis target position (engineering units)
M6043->L:$002943            ; &10 C-axis target position (engineering units)
M6044->L:$002944            ; &10 U-axis target position (engineering units)
M6045->L:$002945            ; &10 V-axis target position (engineering units)
M6046->L:$002946            ; &10 W-axis target position (engineering units)
M6047->L:$002947            ; &10 X-axis target position (engineering units)
M6048->L:$002948            ; &10 Y-axis target position (engineering units)
M6049->L:$002949            ; &10 Z-axis target position (engineering units)
; Coordinate System 10 (&10) Status Bits
M6080->X:$002940,0,1        ; &10 Program-running bit
M6081->Y:$00293F,21,1       ; &10 Circle-radius-error bit
M6082->Y:$00293F,22,1       ; &10 Run-time-error bit
M6084->X:$002940,0,4        ; &10 Continuous motion request
M6087->Y:$00293F,17,1       ; &10 In-position bit (AND of motors)
M6088->Y:$00293F,18,1       ; &10 Warning-following-error bit (OR)
M6089->Y:$00293F,19,1       ; &10 Fatal-following-error bit (OR)
M6090->Y:$00293F,20,1       ; &10 Amp-fault-error bit (OR of motors)
; Coordinate System 10 (&10) Variables
M6097->X:$002900,0,24,S     ; &10 Host commanded time base (I10 units)
M6098->X:$002902,0,24,S     ; &10 Present time base (I10 units)
; Coordinate System 11 (&11) timers
M6111->X:$002A15,0,24,S     ; &11 Isx11 timer (for synchronous assignment)
M6112->Y:$002A15,0,24,S     ; &11 Isx12 timer (for synchronous assignment)
```

```
; Coordinate System 11 (&11) end-of-calculated-move positions
M6141->L:$002A41          ; &11 A-axis target position (engineering units)
M6142->L:$002A42          ; &11 B-axis target position (engineering units)
M6143->L:$002A43          ; &11 C-axis target position (engineering units)
M6144->L:$002A44          ; &11 U-axis target position (engineering units)
M6145->L:$002A45          ; &11 V-axis target position (engineering units)
M6146->L:$002A46          ; &11 W-axis target position (engineering units)
M6147->L:$002A47          ; &11 X-axis target position (engineering units)
M6148->L:$002A48          ; &11 Y-axis target position (engineering units)
M6149->L:$002A49          ; &11 Z-axis target position (engineering units)
; Coordinate System 11 (&11) Status Bits
M6180->X:$002A40,0,1      ; &11 Program-running bit
M6181->Y:$002A3F,21,1     ; &11 Circle-radius-error bit
M6182->Y:$002A3F,22,1     ; &11 Run-time-error bit
M6184->X:$002A40,0,4      ; &11 Continuous motion request
M6187->Y:$002A3F,17,1     ; &11 In-position bit (AND of motors)
M6188->Y:$002A3F,18,1     ; &11 Warning-following-error bit (OR)
M6189->Y:$002A3F,19,1     ; &11 Fatal-following-error bit (OR)
M6190->Y:$002A3F,20,1     ; &11 Amp-fault-error bit (OR of motors)
; Coordinate System 11 (&11) Variables
M6197->X:$002A00,0,24,S   ; &11 Host commanded time base (I10 units)
M6198->X:$002A02,0,24,S   ; &11 Present time base (I10 units)
; Coordinate System 12 (&12) timers
M6211->X:$002B15,0,24,S   ; &12 Isx11 timer (for synchronous assignment)
M6212->Y:$002B15,0,24,S   ; &12 Isx12 timer (for synchronous assignment)
; Coordinate System 12 (&12) end-of-calculated-move positions
M6241->L:$002B41          ; &12 A-axis target position (engineering units)
M6242->L:$002B42          ; &12 B-axis target position (engineering units)
M6243->L:$002B43          ; &12 C-axis target position (engineering units)
M6244->L:$002B44          ; &12 U-axis target position (engineering units)
M6245->L:$002B45          ; &12 V-axis target position (engineering units)
M6246->L:$002B46          ; &12 W-axis target position (engineering units)
M6247->L:$002B47          ; &12 X-axis target position (engineering units)
M6248->L:$002B48          ; &12 Y-axis target position (engineering units)
M6249->L:$002B49          ; &12 Z-axis target position (engineering units)
; Coordinate System 12 (&12) Status Bits
M6280->X:$002B40,0,1      ; &12 Program-running bit
M6281->Y:$002B3F,21,1     ; &12 Circle-radius-error bit
M6282->Y:$002B3F,22,1     ; &12 Run-time-error bit
M6284->X:$002B40,0,4      ; &12 Continuous motion request
M6287->Y:$002B3F,17,1     ; &12 In-position bit (AND of motors)
M6288->Y:$002B3F,18,1     ; &12 Warning-following-error bit (OR)
M6289->Y:$002B3F,19,1     ; &12 Fatal-following-error bit (OR)
M6290->Y:$002B3F,20,1     ; &12 Amp-fault-error bit (OR of motors)
```

```
; Coordinate System 12 (&12) Variables
M6297->X:$002B00,0,24,S   ; &12 Host commanded time base (I10 units)
M6298->X:$002B02,0,24,S   ; &12 Present time base (I10 units)
; Coordinate System 13 (&13) timers
M6311->X:$002C15,0,24,S   ; &13 Isx11 timer (for synchronous assignment)
M6312->Y:$002C15,0,24,S   ; &13 Isx12 timer (for synchronous assignment)
; Coordinate System 13 (&13) end-of-calculated-move positions
M6341->L:$002C41          ; &13 A-axis target position (engineering units)
M6342->L:$002C42          ; &13 B-axis target position (engineering units)
M6343->L:$002C43          ; &13 C-axis target position (engineering units)
M6344->L:$002C44          ; &13 U-axis target position (engineering units)
M6345->L:$002C45          ; &13 V-axis target position (engineering units)
M6346->L:$002C46          ; &13 W-axis target position (engineering units)
M6347->L:$002C47          ; &13 X-axis target position (engineering units)
M6348->L:$002C48          ; &13 Y-axis target position (engineering units)
M6349->L:$002C49          ; &13 Z-axis target position (engineering units)
; Coordinate System 13 (&13) Status Bits
M6380->X:$002C40,0,1      ; &13 Program-running bit
M6381->Y:$002C3F,21,1     ; &13 Circle-radius-error bit
M6382->Y:$002C3F,22,1     ; &13 Run-time-error bit
M6384->X:$002C40,0,4      ; &13 Continuous motion request
M6387->Y:$002C3F,17,1     ; &13 In-position bit (AND of motors)
M6388->Y:$002C3F,18,1     ; &13 Warning-following-error bit (OR)
M6389->Y:$002C3F,19,1     ; &13 Fatal-following-error bit (OR)
M6390->Y:$002C3F,20,1     ; &13 Amp-fault-error bit (OR of motors)
; Coordinate System 13 (&13) Variables
M6397->X:$002C00,0,24,S   ; &13 Host commanded time base (I10 units)
M6398->X:$002C02,0,24,S   ; &13 Present time base (I10 units)
; Coordinate System 14 (&14) timers
M6411->X:$002D15,0,24,S   ; &14 Isx11 timer (for synchronous assignment)
M6412->Y:$002D15,0,24,S   ; &14 Isx12 timer (for synchronous assignment)
; Coordinate System 14 (&14) end-of-calculated-move positions
M6441->L:$002D41          ; &14 A-axis target position (engineering units)
M6442->L:$002D42          ; &14 B-axis target position (engineering units)
M6443->L:$002D43          ; &14 C-axis target position (engineering units)
M6444->L:$002D44          ; &14 U-axis target position (engineering units)
M6445->L:$002D45          ; &14 V-axis target position (engineering units)
M6446->L:$002D46          ; &14 W-axis target position (engineering units)
M6447->L:$002D47          ; &14 X-axis target position (engineering units)
M6448->L:$002D48          ; &14 Y-axis target position (engineering units)
M6449->L:$002D49          ; &14 Z-axis target position (engineering units)
; Coordinate System 14 (&14) Status Bits
M6480->X:$002D40,0,1      ; &14 Program-running bit
M6481->Y:$002D3F,21,1     ; &14 Circle-radius-error bit
M6482->Y:$002D3F,22,1     ; &14 Run-time-error bit
M6484->X:$002D40,0,4      ; &14 Continuous motion request
```

```
M6487->Y:$002D3F,17,1      ; &14 In-position bit (AND of motors)
M6488->Y:$002D3F,18,1      ; &14 Warning-following-error bit (OR)
M6489->Y:$002D3F,19,1      ; &14 Fatal-following-error bit (OR)
M6490->Y:$002D3F,20,1      ; &14 Amp-fault-error bit (OR of motors)
; Coordinate System 14 (&14) Variables
M6497->X:$002D00,0,24,S    ; &14 Host commanded time base (I10 units)
M6498->X:$002D02,0,24,S    ; &14 Present time base (I10 units)
; Coordinate System 15 (&15) timers
M6511->X:$002E15,0,24,S    ; &15 Isx11 timer (for synchronous assignment)
M6512->Y:$002E15,0,24,S    ; &15 Isx12 timer (for synchronous assignment)
; Coordinate System 15 (&15) end-of-calculated-move positions
M6541->L:$002E41           ; &15 A-axis target position (engineering units)
M6542->L:$002E42           ; &15 B-axis target position (engineering units)
M6543->L:$002E43           ; &15 C-axis target position (engineering units)
M6544->L:$002E44           ; &15 U-axis target position (engineering units)
M6545->L:$002E45           ; &15 V-axis target position (engineering units)
M6546->L:$002E46           ; &15 W-axis target position (engineering units)
M6547->L:$002E47           ; &15 X-axis target position (engineering units)
M6548->L:$002E48           ; &15 Y-axis target position (engineering units)
M6549->L:$002E49           ; &15 Z-axis target position (engineering units)
; Coordinate System 15 (&15) Status Bits
M6580->X:$002E40,0,1       ; &15 Program-running bit
M6581->Y:$002E3F,21,1      ; &15 Circle-radius-error bit
M6582->Y:$002E3F,22,1      ; &15 Run-time-error bit
M6584->X:$002E40,0,4       ; &15 Continuous motion request
M6587->Y:$002E3F,17,1      ; &15 In-position bit (AND of motors)
M6588->Y:$002E3F,18,1      ; &15 Warning-following-error bit (OR)
M6589->Y:$002E3F,19,1      ; &15 Fatal-following-error bit (OR)
M6590->Y:$002E3F,20,1      ; &15 Amp-fault-error bit (OR of motors)
; Coordinate System 15 (&15) Variables
M6597->X:$002E00,0,24,S    ; &15 Host commanded time base (I10 units)
M6598->X:$002E02,0,24,S    ; &15 Present time base (I10 units)
; Coordinate System 16 (&16) timers
M6611->X:$002F15,0,24,S    ; &16 Isx11 timer (for synchronous assignment)
M6612->Y:$002F15,0,24,S    ; &16 Isx12 timer (for synchronous assignment)
; Coordinate System 16 (&16) end-of-calculated-move positions
M6641->L:$002F41           ; &16 A-axis target position (engineering units)
M6642->L:$002F42           ; &16 B-axis target position (engineering units)
M6643->L:$002F43           ; &16 C-axis target position (engineering units)
M6644->L:$002F44           ; &16 U-axis target position (engineering units)
M6645->L:$002F45           ; &16 V-axis target position (engineering units)
M6646->L:$002F46           ; &16 W-axis target position (engineering units)
M6647->L:$002F47           ; &16 X-axis target position (engineering units)
M6648->L:$002F48           ; &16 Y-axis target position (engineering units)
M6649->L:$002F49           ; &16 Z-axis target position (engineering units)
```

```
; Coordinate System 16 (&16) Status Bits
M6680->X:$002F40,0,1        ; &16 Program-running bit
M6681->Y:$002F3F,21,1       ; &16 Circle-radius-error bit
M6682->Y:$002F3F,22,1       ; &16 Run-time-error bit
M6684->X:$002F40,0,4        ; &16 Continuous motion request
M6687->Y:$002F3F,17,1       ; &16 In-position bit (AND of motors)
M6688->Y:$002F3F,18,1       ; &16 Warning-following-error bit (OR)
M6689->Y:$002F3F,19,1       ; &16 Fatal-following-error bit (OR)
M6690->Y:$002F3F,20,1       ; &16 Amp-fault-error bit (OR of motors)
; Coordinate System 16 (&16) Variables
M6697->X:$002F00,0,24,S     ; &16 Host commanded time base (I10 units)
M6698->X:$002F02,0,24,S     ; &16 Present time base (I10 units)
; UMAC UBUS Accesory I/O M-Variables (1st ACC-9E, 10E, 11E, 12E, 14E)
M7000->Y:$078C00,0,1        ; MI/O0
M7001->Y:$078C00,1,1        ; MI/O1
M7002->Y:$078C00,2,1        ; MI/O2
M7003->Y:$078C00,3,1        ; MI/O3
M7004->Y:$078C00,4,1        ; MI/O4
M7005->Y:$078C00,5,1        ; MI/O5
M7006->Y:$078C00,6,1        ; MI/O6
M7007->Y:$078C00,7,1        ; MI/O7
M7008->Y:$078C01,0,1        ; MI/O8
M7009->Y:$078C01,1,1        ; MI/O9
M7010->Y:$078C01,2,1        ; MI/O10
M7011->Y:$078C01,3,1        ; MI/O11
M7012->Y:$078C01,4,1        ; MI/O12
M7013->Y:$078C01,5,1        ; MI/O13
M7014->Y:$078C01,6,1        ; MI/O14
M7015->Y:$078C01,7,1        ; MI/O15
M7016->Y:$078C02,0,1        ; MI/O16
M7017->Y:$078C02,1,1        ; MI/O17
M7018->Y:$078C02,2,1        ; MI/O18
M7019->Y:$078C02,3,1        ; MI/O19
M7020->Y:$078C02,4,1        ; MI/O20
M7021->Y:$078C02,5,1        ; MI/O21
M7022->Y:$078C02,6,1        ; MI/O22
M7023->Y:$078C02,7,1        ; MI/O23
M7024->Y:$078C03,0,1        ; MI/O24
M7025->Y:$078C03,1,1        ; MI/O25
M7026->Y:$078C03,2,1        ; MI/O26
M7027->Y:$078C03,3,1        ; MI/O27
M7028->Y:$078C03,4,1        ; MI/O28
M7029->Y:$078C03,5,1        ; MI/O29
M7030->Y:$078C03,6,1        ; MI/O30
M7031->Y:$078C03,7,1        ; MI/O31
M7032->Y:$078C04,0,1        ; MI/O32
M7033->Y:$078C04,1,1        ; MI/O33
```

```
M7034->Y:$078C04,2,1      ; MI/O34
M7035->Y:$078C04,3,1      ; MI/O35
M7036->Y:$078C04,4,1      ; MI/O36
M7037->Y:$078C04,5,1      ; MI/O37
M7038->Y:$078C04,6,1      ; MI/O38
M7039->Y:$078C04,7,1      ; MI/O39
M7040->Y:$078C05,0,1      ; MI/O40
M7041->Y:$078C05,1,1      ; MI/O41
M7042->Y:$078C05,2,1      ; MI/O42
M7043->Y:$078C05,3,1      ; MI/O43
M7044->Y:$078C05,4,1      ; MI/O44
M7045->Y:$078C05,5,1      ; MI/O45
M7046->Y:$078C05,6,1      ; MI/O46
M7047->Y:$078C05,7,1      ; MI/O47
; Encoder Conversion Table Result Registers (M8xxx matches I8xxx)
M8000->X:$003501,0,24,S   ; Line 0 result from conversion table
M8001->X:$003502,0,24,S   ; Line 1 result from conversion table
M8002->X:$003503,0,24,S   ; Line 2 result from conversion table
M8003->X:$003504,0,24,S   ; Line 3 result from conversion table
M8004->X:$003505,0,24,S   ; Line 4 result from conversion table
M8005->X:$003506,0,24,S   ; Line 5 result from conversion table
M8006->X:$003507,0,24,S   ; Line 6 result from conversion table
M8007->X:$003508,0,24,S   ; Line 7 result from conversion table
M8008->X:$003509,0,24,S   ; Line 8 result from conversion table
M8009->X:$00350A,0,24,S   ; Line 9 result from conversion table
M8010->X:$00350B,0,24,S   ; Line 10 result from conversion table
M8011->X:$00350C,0,24,S   ; Line 11 result from conversion table
M8012->X:$00350D,0,24,S   ; Line 12 result from conversion table
M8013->X:$00350E,0,24,S   ; Line 13 result from conversion table
M8014->X:$00350F,0,24,S   ; Line 14 result from conversion table
M8015->X:$003510,0,24,S   ; Line 15 result from conversion table
M8016->X:$003511,0,24,S   ; Line 16 result from conversion table
M8017->X:$003512,0,24,S   ; Line 17 result from conversion table
M8018->X:$003513,0,24,S   ; Line 18 result from conversion table
M8019->X:$003514,0,24,S   ; Line 19 result from conversion table
M8020->X:$003515,0,24,S   ; Line 20 result from conversion table
M8021->X:$003516,0,24,S   ; Line 21 result from conversion table
M8022->X:$003517,0,24,S   ; Line 22 result from conversion table
M8023->X:$003518,0,24,S   ; Line 23 result from conversion table
M8024->X:$003519,0,24,S   ; Line 24 result from conversion table
M8025->X:$00351A,0,24,S   ; Line 25 result from conversion table
M8026->X:$00351B,0,24,S   ; Line 26 result from conversion table
M8027->X:$00351C,0,24,S   ; Line 27 result from conversion table
M8028->X:$00351D,0,24,S   ; Line 28 result from conversion table
M8029->X:$00351E,0,24,S   ; Line 29 result from conversion table
M8030->X:$00351F,0,24,S   ; Line 30 result from conversion table
M8031->X:$003520,0,24,S   ; Line 31 result from conversion table
```

```
M8032->X:$003521,0,24,S   ; Line 32 result from conversion table
M8033->X:$003522,0,24,S   ; Line 33 result from conversion table
M8034->X:$003523,0,24,S   ; Line 34 result from conversion table
M8035->X:$003524,0,24,S   ; Line 35 result from conversion table
M8036->X:$003525,0,24,S   ; Line 36 result from conversion table
M8037->X:$003526,0,24,S   ; Line 37 result from conversion table
M8038->X:$003527,0,24,S   ; Line 38 result from conversion table
M8039->X:$003528,0,24,S   ; Line 39 result from conversion table
M8040->X:$003529,0,24,S   ; Line 40 result from conversion table
M8041->X:$00352A,0,24,S   ; Line 41 result from conversion table
M8042->X:$00352B,0,24,S   ; Line 42 result from conversion table
M8043->X:$00352C,0,24,S   ; Line 43 result from conversion table
M8044->X:$00352D,0,24,S   ; Line 44 result from conversion table
M8045->X:$00352E,0,24,S   ; Line 45 result from conversion table
M8046->X:$00352F,0,24,S   ; Line 46 result from conversion table
M8047->X:$003530,0,24,S   ; Line 47 result from conversion table
M8048->X:$003531,0,24,S   ; Line 48 result from conversion table
M8049->X:$003532,0,24,S   ; Line 49 result from conversion table
M8050->X:$003533,0,24,S   ; Line 50 result from conversion table
M8051->X:$003534,0,24,S   ; Line 51 result from conversion table
M8052->X:$003535,0,24,S   ; Line 52 result from conversion table
M8053->X:$003536,0,24,S   ; Line 53 result from conversion table
M8054->X:$003537,0,24,S   ; Line 54 result from conversion table
M8055->X:$003538,0,24,S   ; Line 55 result from conversion table
M8056->X:$003539,0,24,S   ; Line 56 result from conversion table
M8057->X:$00353A,0,24,S   ; Line 57 result from conversion table
M8058->X:$00353B,0,24,S   ; Line 58 result from conversion table
M8059->X:$00353C,0,24,S   ; Line 59 result from conversion table
M8060->X:$00353D,0,24,S   ; Line 60 result from conversion table
M8061->X:$00353E,0,24,S   ; Line 61 result from conversion table
M8062->X:$00353F,0,24,S   ; Line 62 result from conversion table
M8063->X:$003540,0,24,S   ; Line 63 result from conversion table
```

FIRMWARE UPDATE LISTING

V1.933 Updates (July 1999)

1. Fixed operation of **PR** query command.
2. Fixed operation of **DWELL** when Motor 1 not in coordinate system.
3. Fixed blending to new jog command when in acceleration slope.
4. Fixed operation of move-until-trigger with software position capture.
5. Fixed operation of dual-ported RAM variable-write buffer.
6. Made sure rotary buffer pointer always at beginning of buffer after **DEFINE** or **CLEAR**.
7. Made data rounding in PLCC programs like that of PLC programs
8. Fixed operation of single-line single-step mode (Isx53=1)
9. Fixed calculations with intermediate values just slightly less than 1.0.
10. Extended range of constant values accepted to 19 decimal digits.
11. Extended range of values that can be displayed to 14 decimal digits.
12. Implemented MACRO master-to-master communications with **MACROASCII** command.
13. Changes of state of amplifier-enable flags on MACRO ring immediately put on ring instead of waiting for next background cycle.
14. Cutter compensation refinements:
    – Cutter compensation can now be maintained with multiple **DWELL**s between two compensated moves.
    – If cutter compensation is active and Turbo PMAC cannot find the next move in the compensation plane, Turbo PMAC no longer removes the compensation at the end of the move; instead it ends the move at the proper point to start an outside corner.
    – If cutter compensation direction is changed with compensation active, offset to new direction occurs at move boundary, not over the course of the next move (unless there is no intersection of compensated paths, in which case the change still occurs over the next move.
    – 180^{o} reversal with arc(s) can now be treated as inside corner, not always outside corner.
    – Introduction and removal of compensation on inside corners changed slightly; compensated intersection point now offset from uncompensated intersection point perpendicular to fully compensated move only.

V1.934 Updates (September 1999)

1. Fixed de-multiplexing of Opt 12/ACC-36 A/D converters with I5060 – I5096.
2. When control-panel-port selector switch on Turbo PMAC(1) selects '0', firmware no longer writes to indicator outputs, so they can be used as general-purpose outputs.
3. Fixed operation of synchronous M-variable assignments when used in the lookahead buffer.
4. Fixed operation of cutter-radius compensation when more than 8 DWELL commands execute between two compensated moves.
5. Fixed operation of **<CTRL-C>** global coordinate-system status request command.
6. Corrected time of **DWELL** commands when not in cutter-radius compensation.
7. Fixed operation of **HOMEZ** command when issued through dual-ported RAM control panel.
8. Fixed initial acceleration on resuming running after feed hold, jog away, and jog return.
9. Fixed operation of **DEFINE GATHER** command without argument when gathering set up for dual-ported RAM.
10. Corrected arctangent calculations for high-resolution encoder interpolation and "two-guess" phasing search.
11. Fixed the blocking of interrupts around writing to M-variables in background routines to ensure no corruption when foreground and background routines write to M-variables in the same word.
12. Added capability to stop on desired position limit if Bit 15 of Ixx24 is set to 1. Positive desired position limit is (Ixx13 – Ixx41); negative desired position limit is (Ixx13 + Ixx41). If Bit 14 of Ixx24 is also set to 1, program will not stop, but motor will saturate at the limit value.
13. Added "auto-detect" capability for Servo ICs and MACRO ICs. I65, I66, I67, and I69, which user formerly set to tell Turbo which ICs were present, are no longer used. I4900 and I4901 now report which ICs are present and which type.
14. I19 now specifies which Servo IC or MACRO IC is the source of the servo and phase clock signals for the system.
15. Added six more hex digits (24 status bits) to the response for the **??** coordinate-system status-query command, for a total of 18.
16. Improved operation of watchdog timer, so less likely to trip on heavy, but valid, CPU loading. Now permit user to set required background period for watchdog timer with I20.
17. **MSDATE{node#}** now returns 4-digit year value.
18. Implemented special forward-kinematic and inverse-kinematic motion-program subroutine buffers for each coordinate system, created with new **OPEN FORWARD** and **OPEN INVERSE** commands. New variable Isx50 set to 1 causes these buffers to be used. New axis-definition command **#{motor}->I** causes Turbo PMAC to use inverse kinematic routine to convert from axis to motor coordinates. Number of regular motion program buffers reduced from 256 to 224 to accommodate these new buffers.
19. Added new run-time error codes 7 (arc radius smaller than tool radius), 8 (forward-kinematic equation error), and 9 (inverse-kinematic equation error).
20. Added new on-line command **EAVERSION** to give more complete listing of firmware revision and type.

21. Added new command **IDC** to force synchronization of PMAC's RAM timer with Option 18B non-volatile timer.
22. Password protection extended to compiled PLCs, user-written servo algorithms, user-written phase algorithms, forward-kinematic programs, and inverse-kinematic programs.
23. Improved communications response time when multiple ports are used simultaneously.
24. If all motors in a coordinate system are undefined while coordinate system is running a motion program, program will now stop with a run-time error
25. New variable I21 now permits "lockout" of changes to classes of I-variables
26. New variable I14 permits retention of structure of "temporary" buffers through power-down or reset so they do not have to be redefined.

V1.935 Updates (February 2000)

1. Corrected problem with executing **P({expression})={expression}** statement in motion program.
2. Corrected problem in homing-search move with capture over MACRO ring when previous homing-search move was interrupted with a Kill command.
3. On-line addressing commands **#** and **&** are now port-specific; they only affect the addressing of commands on the port they were sent to, no longer the other ports.
4. The set of 8 motors whose data is reported on a **<CTRL-P>**, **<CTRL-V>**, **<CTRL-F>**, or **<CTRL-V>** command is now independent for each port, set by the new **##{constant}** command given on that port, not by the global variable I59.
5. I59, which now affects the Turbo PMAC(1) control-panel port selector only, can now be set from the control-panel port.
6. New commands **ADDRESS#P{constant}** and **ADDRESS&P{constant}** permit variable setting of the addressed motor and coordinate system, respectively, from within a PLC program for CMD statements within that PLC program.
7. Added registers that record instruction cycles spent in phase and servo interrupt tasks to help determine computational duty cycle status.
8. Added cutter compensation move buffer, created with **DEFINE CCBUFFER{constant}** command, erased with **DELETE CCBUFFER** command, to permit out-of-plane moves while in cutter compensation.
9. Added ERR019, which is reported when a command to change position value (**HOME**, **HOMEZ**, **PSET**) is attempted while moves are stored in the cutter compensation move buffer.
10. If the same axis letter repeats in a single motion-program line, each occurrence is a new move command. Formerly, the value(s) associated with the later occurrence overwrote the values associated with the earlier occurrence, and only a single move was executed. For example, the program line **TM1000 X10 X20 X30** now causes the execution of three consecutive X-axis moves, each one second long. Formerly it caused execution of only a single move (X30), one second long.
11. When entering **SPLINE** mode, the move time for the added introductory spline segment is now automatically set to the first move time declared with **TM** in **SPLINE** mode. Formerly, this segment time was left at the previous value (or the Isx89 default if no value had previously been declared).

12. Fixed problem with scaling of absolute phase position reads from MACRO Stations with Ixx91=$73nnnn or Ixx91=$74nnnn so they now match the scaling of ongoing feedback scaled in 1/32 count. Implemented new Ixx91 settings of $71nnnn and $72nnnn to match scaling of ongoing feedback scaled in counts.
13. Fixed internal stack error that occurred on abort and could cause overwriting of variables like Ixx68.

V1.936 Updates (April 2000)

Note:

After upgrading an older system to V1.936 and either getting the old configuration back from flash memory or reloading a configuration file, issue an **I20..24=*** command to set up any MACRO ICs and DPRAM ICs properly. If there were non-zero values in the old configuration for I20 and I21, enter these values into I40 and I41 respectively. After this, **SAVE** the configuration and reset the board normally.

1. Added support for UMAC Turbo systems.
2. Added new variables I20 – I23 to specify base addresses of MACRO ICs 0 – 3 respectively, providing flexibility in MACRO ring configurations on UMAC Turbo. These must be set properly to support automatic firmware functions using these ICs, including multiplexer port functions, display port, and I6800 – I6999.
3. Moved old variables I20 (watchdog timer reset value) and I21 (I-variable lockout control) to their proper locations of I40 and I41.
4. Default values of "address" I-variables made more system-specific to reflect what components such as Servo ICs and MACRO ICs are actually found by the processor.
5. Added 3D cutter-radius compensation with new program commands **CC3**, **NX{data}**, **NY{data}**, **NZ(data}**, **TR{data}**, **TX{data}**, **TY{data}**, and **TZ{data}**.
6. Added "altered destination" RAPID mode move on-line command **!{axis}{data}…** to be able to break into currently executing RAPID-mode move and change the move on the fly to a new destination, or execute a RAPID-mode move directly from an on-line command.
7. Extended I49nn controller configuration status I-variables
8. Made the communications ports independent with respect to opening of program buffers. Only the port over which the **OPEN** command was issued can accept buffered program commands, **LIST** the open buffer, **LEARN** points into the open buffer, and **CLOSE** the buffer. Other ports can be used simultaneously for on-line commands.
9. Refined error reporting when **CLOSE**ing a program buffer missing **ENDIF** and/or **ENDWHILE**. Now reports ERR009 (program structure error), and only reports the error on **CLOSE**ing this particular buffer.
10. Extended Ixx91, Ixx95, and I8000 – I8191 to support parallel position reads in byte-wide sections from ACC-3E1 and ACC-14E boards.
11. Corrected problem with hardware position capture over MACRO in V.1933,4,5.
12. Corrected problem in **DELETE GATHER** command that could cause buffer management problems.
13. Corrected problem in cutter compensation in sequencing with non-compensated moves (RAPID, DWELL, out-of-plane).

14. Corrected problem in cutter compensation with CIRCLE mode lead-in moves.
15. Corrected operation of Isx91 default program parameter after **$$$** software reset, and in repeated execution of program.
16. Corrected absolute phase position read of resolver-to-digital converter through MACRO station.
17. Corrected **PMATCH** problem when linear set of axes X, Y, and Z, or U, V, and W were defined "out of order" (e.g. #1->X, #2->Z, #3->Y).
18. Corrected operation of **PMATCH** when called from within a motion program (**CMD "&nPMATCH"**).

V1.937 Updates (November, 2000)

1. Changed I5061 to I5076 A/D de-multiplexing pointer variables to contain the full address of the A/D register, not just the offset from $078800. The old default value of 0 still selects $078800, but $078800 must be added to existing non-zero values to maintain compatibility.
2. Automatically sets I58 to 1, enabling DPRAM ASCII communications, at power-up/reset, if any DPRAM IC is detected.
3. Fixed operation of **J!** command so that commanded position is always rounded to nearest integer number of counts, regardless of the size of the following error.
4. Fixed glitch at the center 1/8-millionth section of long compensation tables (> ½-million counts long).
5. Fixed operation of background variable read buffer in multi-user mode.
6. Added foreground "in-position" check in servo interrupt, enabled by I13=1. Added foreground in-position motor status bit – bit 13 of Y:$0000C0, etc.
7. Permitted disabling of automatic command parsing on Option 9T auxiliary serial port by setting I43 to 1, permitting custom parsing algorithms to be written for serial input of data that is not in PMAC command format.
8. Permitted loading of binary rotary motion program commands from DPRAM to internal buffer as a foreground real-time interrupt task instead of a background task with I45=1.
9. Permitted disabling of A/D de-multiplexing with I5080=0.
10. Implemented alternate rotary-axis rollover mode in which the sign of the specified destination value also specifies the direction to turn to that point. Setting Ixx27 to a negative value enables this mode.
11. Implemented |I|T integrated current limiting function as alternate to existing I^2T integrated current limiting. In |I|T, the magnitude of the current itself, not the square of the current, is integrated and compared to the Ixx58 limit. Setting the Ixx57 continuous current magnitude value to a negative number enables this alternate mode. This more accurately tracks the thermal behavior of a constant voltage-drop device such as an IGBT, whereas I^2T is better for constant-resistance devices such as MOSFETs and motor windings.
12. Implemented automatic clearing of direct current-loop registers to improve direct-PWM control of permanent-magnet brush motors. This mode is enabled by setting Ixx96 to 1 when Ixx01 bit 0 = 1 (enabling commutation) and Ixx82 > 0 (enabling current loop).
13. Implemented **ABR** command, permitting fastest possible abort of currently executing program, and start or restart of a motion program.
14. Implemented "time remaining in move" register and **MOVETIME** query command, to support functions initiated at a fixed time before the end of a commanded move.

15. Implemented **SETPHASE** command (on-line and buffered) to copy Ixx75 phase value into phase position register. Useful for correcting the phase position at a known point (e.g. the index pulse) after an initial rough phasing (e.g. from Hall commutation sensors).
16. Implemented **LOCK** and **UNLOCK** commands (on-line and buffered) to control up to 8 process locking bits that can prevent possible conflict of foreground and background tasks attempting to manipulate the same register.
17. Implemented on-line **I{constant}=@I{constant}** command, permitting the value of one I-variable to be set to the address of another I-variable. The main purpose of this command is to be able to set an address I-variable (e.g. Ixx03, Ixx04, Ixx05, Isx93) to the address of a conversion-table entry without having to look up the address of that entry.
18. Added capability for UMAC Turbo CPUs to generate their own servo and phase clock signals when expected clock source is not found. Keeps watchdog timer from tripping so that new clock source can be established. Bit 3 of X:$000006 set if CPU is generating its own clocks.
19. Resolution of (previously undocumented) real-time clock register L:$000017 changed from 1/256 second to 1/1024 second.
20. Turbo PMACs with extended user data memory options (5x1 or 5x3) have default user buffer of 65,536 words.
21. I52 CPU clock speed multiplier parameter range extended to 14 to support CPU speeds of up to 150 MHz.
22. Resolution of (previously undocumented) real-time interrupt cycle time registers X:$00000B (latest time) and Y:$00000B (maximum time) changed to 2 CPU clock cycles.
23. Added background cycle time registers X:$000022 (latest time) and Y:$000022 (maximum time) with resolution of 2 CPU clock cycles.
24. Internal-use global status bits "Servo Active" (X:$000006 bit 21) and "RTI Active" (X:$000006 bit 23) removed.
25. Implemented anti-windup protection for current-loop integrators. If calculated output is more than 9/8 of saturated output, integrator value is reduced to that which would produce 9/8 of saturation.
26. Corrected operation of cutter compensation for compensated inside corner immediately following inside-corner introduction of compensation.
27. Extended I68n5/I69n5 and I7mn5 encoder variables to support de-multiplexing of hall commutation states from Yaskawa encoder third channel ("B" or newer revision of DSPGATE1/2 Servo/MACRO IC required).
28. Corrected operation of **LIST BLCOMP DEF** and **LIST TCOMP DEF** commands.
29. Corrected algorithm in compiled PLCs for taking INT of a quotient.
30. Corrected deadband gain algorithm for true deadband (Ixx64=-8) with small motor scale factor (Ixx08~1). With pulse-and-direction output, the previous small remaining residual could cause dithering.
31. Fixed action of BREQ (Buffer Request) interrupt and control-panel output when entering rotary motion program commands. This did not work properly in V1.935 and V1.936.

V1.938 Updates (June, 2001)

1. Corrected operation of the **UNDEFINE** command so it only clears axis definitions in the addressed coordinate system.
2. Corrected operation of DPRAM binary rotary buffer download through USB interface.
3. Increased commanded velocity saturation value for jog-to-position and RAPID-mode moves from 256M/Ix08 counts/second to 768M/Ix08 counts/second, consistent with other types of moves.
4. Improved noise immunity in conversion-table algorithms for ACC-51 high-resolution analog-encoder interpolators to decrease chance of "quadrant" errors due to high noise on analog lines.
5. Corrected VME mailbox communications so that responses of more than 15 characters can be read properly.
6. Implemented support for ACC-57E Yaskawa/Mitsubishi absolute encoder interface board in Ixx10 and Ixx95 variables. See ACC-57E manual for details.
7. Implemented support for "I-button" real-time clock/calendar chip on new "Flex" CPU design.
8. Limited number of commands that can possibly be pulled off individual port in one background cycle to 8 to prevent possibility of trapping the background cycle with very high-speed communications.
9. Corrected problem with **DEFINE UBUF** when background PLCs are enabled.
10. Added support for on-board IEC-1131 ladder/sequential-function-chart programs.
11. Added support for "Open Servo" compiled servo algorithms
12. Supported negative values of Ixx99 for Yaskawa Sigma I absolute encoders to permit reversal of the direction sense.
13. Corrected saving and restoring of the value of new variable I7mn9.

V1.939 Updates (March, 2002)

a. Added support for DSP56311 CPU (Option 5Ex).

b. Added new status bit at X:$000006 bit 21 that is set to 1 to indicate that CPU is DSP56311 type (X:$000006 bit 21 also set to 1 in this case).

c. Moved location of main serial-port communications buffer from $001Exx to $0036xx. Moved location of host-bus port communications buffer from $001Fxx to $0037xx. Moved location of synchronous M-variable buffer from $0036xx and $0037xx to $001Exx and $001Fxx. Changes necessary to support DSP56311 (Option 5Ex) properly.

d. Increased maximum value of Ixx71 Commutation Cycle Size variable from 8,388,607 to 16,777,215.

e. Changed range of Ixx75 Phase Position Offset variable from –8,388,608 – +8,388,607 to 0 – 16,777,215. If a negative value of Ixx75 is specified, it is stored as (Ixx71 + Ixx75), which provides the same effect (the proper value of Ixx71 must already be specified for the motor).

f. Added new variable I12 to better support on-the-fly changes in vector feedrate during lookahead.

g. Added new variable I30 to support automatic "wrapping" of compensation tables (the last entry in the table "wraps" to become the correction at zero position as well). Existing documentation incorrectly reported that this was done always in earlier firmware versions, but correction at zero position was always zero, regardless of last entry.

h. Added support for "hardware 1/T" using D-revision or newer PMAC2-style "DSPGATE1" Servo ICs. In new conversion table method ($C with mode bit set), the IC computes the timer-based fractional count value in hardware; the conversion table simply combines it with the whole-count value. This permits use of the alternate timer mode for sub-count capture and compare.

i. Computational efficiency of dual-ported RAM data reporting buffers was improved.

j. Computational efficiency of commutation calculations was improved about 20%.

k. Permitted "foreign" characters (ASCII value > 127) to be accepted in comments (after semi-colon) without causing an error to be reported.

l. Modified timing of multiplexer port interface signals to ACC-34 boards so they will work properly with Option 5Ex 160MHz CPUs.

m. Fixed problem with BREQ "buffer request" interrupt on ISA/PCI bus.

V1.940 Updates (June, 2003)

1. Added support for DSP56321 CPU (Option 5Fx). Range of I52 CPU Frequency Control variable extended to 31 (Option 5Fx can run at up to 240 MHz, with I52=23).
2. Implemented support for separate flag addresses for limits, amp flags, and capture flags with new variables Ixx42 (separate amplifier-flag address) and Ixx43 (separate limit-flag address).
3. Implemented support for sub-count position capture from Revision D PMAC2-style Servo ICs for move-until-trigger functions with bits 11 and 12 of Ixx24.
4. Does not permit enabling of any motors if global "phase clock error" bit (X:$000006 bit 3) is set. Enabling command is rejected in this case with ERR018.
5. Fixed lead-out move problem of 2D cutter compensation.
6. Fixed operation of |I|T protection.
7. Improved operation of Extended Servo Algorithm when saturated.
8. Fixed execution of phasing read so will work correctly even when in an overtravel limit.
9. Fixed listing of **`DISPLAY {variable}`** statement when 0 fractional digits specified.
10. Fixed operation of on-line coordinate-system **`Z`** command.
11. Implemented new variable I37 that can specify additional wait states above the default when accessing memory and/or I/O.