MPL MATH

Version 1.3c

19 Linden Park

Rochester, NY 14625

TABLE OF CONTENTS

MPL MATH brings a new level of power and capability to ORMEC's **Motion Programming Language, [MPL]**. It integrates powerful arithmetic and logical functions directly into MPL's intuitive motion commands. MPL MATH also provides commands that handle extended DIGITAL I/O, ANALOG I/O, and a variety of operator interface devices. In addition it provides internal algorithms for precise synchronization and ratioing, to make ORMEC's motion control products ideal for electronic lineshafting applications.

OVERVIEW OF MPL

ORMEC's Motion Programming Language is calculator like in simplicity, providing 20 intuitive commands for commanding or specifying motion. These commands include the **A** command for setting Acceleration, the **V** command for setting Velocity and the **I** command for commanding a position Index (move relative to the current position). Also included are commands like the **N** command for Normalizing the absolute position counter, the **G** command for Going to a specific absolute position, and the **D** command for Delaying program execution, either as a function of time or distance. MPL commands include the ability to display their current settings and/or real time information with regard to the motion of the servo.

Also included in MPL are commands for editing programs, controlling program flow, dealing with 16 discrete I/O points, creating complex motion profiles, and setting up a number of operational parameters. Math has always been a part of MPL, used internally to calculate the appropriate motion parameters. Now math is being provided to the user in the form of functional expressions, using non-volatile registers and several arithmetic and logical operators.

COMBINING MATH WITH MPL

Users of MPL with MATH can utilize 32-bit register arithmetic directly in MPL commands or store computed arithmetic values in any of one hundred non-volatile registers. Arithmetic and logical functions can also be programmed to utilize real-time motion data.

MPL MATH uses standard algebraic notation to compute values, similar to the expressions used with a programmable calculator. Constants, registers, and 20 mathematical & logical operators, give the programmer new tools to solve difficult control problems.

ELECTRONIC LINESHAFTING COMMANDS

MPL MATH includes two commands to enhance the capabilities of our motion controllers in electronic lineshafting applications.

The Gear Ratio Command provides a method to express a gear ratio as a fraction of two 16-bit integers. This capability allows the user to specify gear ratios as rational numbers (such as 1/3 or 23/67) exactly, resulting in "zero drift" operation of electronically lineshafted servos. The High Resolution Jog command extends the velocity resolution from twelve bits to 28 bits, allowing specification of speed ratios to a precision finer than 1 part in 250,000,000.

EXTENDED I/O CAPABILITIES

MPL MATH contains a number of commands which support ORMEC's **Extended I/O Module [EIO-900]**. The EIO-900 daughter board provides 24 additional digital I/O points, two 8-bit analog inputs, one 8-bit analog output, plus an auxilliary serial port. The combination of the EIO-900 and MPL MATH enables the user to interface ORMEC motion controllers to industrial keypads, thumbwheel switches, numeric readouts, alphanumeric displays and other operator interface devices.

The 24 digital I/O points can be used as discrete I/O, or alternatively configured to interface to thumbwheel switches and numeric displays. Five thumbwheel switches can be attached to the digital I/O (totalling up to 28 decades), or alternatively four six-digit numeric displays can be used. If using all the I/O points for this purpose, up to five thumbwheel switches (totaling 18 digits) and four numeric displays can be attached at the same time.

The EIO-900 also provides either one differential or two single-ended channels of 8-bit analog to digital (A/D) input. A single 8-bit digital to analog output (D/A) is also included.

The RS-422/485 Serial Port on the EIO-900 provides an interface to a wide variety of operator interface devices. MPL MATH provides software drivers for a variety of industrial control panels, data entry controllers, industrial keypads, and alphanumeric displays. Up to thirty-two individual 20 character by 1 line alphanumeric displays can be attached to this device.

HARDWARE COUNTING CAPABILITIES

In conjunction with the optional **Encoder Backup Compensator, EBC-900** daughter board, MPL MATH provides access to an AM-9513 LSI counter chip. It provides five powerful, software-controlled, gated 16-bit up/down counters which can be used for applications which demand high frequency counting capability.

VERSION ENHANCEMENTS TO MPL MATH

Major changes in MPL MATH from Version 1.0 to Version 1.1 are listed below:

1) The number of Registers was changed from 10 to 100.

2) The print command was enhanced to add character-mode output with a timeout and to automatically change the default output device each time a device is setup.

3) The RI command was enhanced to add character-mode input with a timeout and to automatically change the default input device each time a device is setup.

4) Additional error checking was added to the RI Command

5) The RE command was enhanced to add the ability to add a user defined error.

6) The R? IF/ELSE structured programming construct was added to the language.

7) The RP - PMC Talk Command was added to the language.
8) The RX - External Register Read/Write Command was added to the language.

9) RD and RX Extended I/O Functions were added.

10) A number of Hardware Counter Functions were added.

11) MPL MATH Error Codes were revised to be functionally grouped.

Major changes in MPL MATH from Version 1.1 to Version 1.2 are listed below:

1) The "u" and "y" system status poll commands were added to access all 100 registers instead of just the first 10.

2) The RX - External Register Read/Write Commands was enhanced to support all 100 registers via the "u" and "y" system status poll commands.

Major changes in MPL MATH from Version 1.2 to Version 1.3 are listed below:

1) The RS - Input Scan Command was added to the language.

SYNTAX EXPRESSION NOTATION

For MPL MATH commands shown in this document, both a command syntax and examples will be shown. Command syntax statements may be somewhat confusing at first, however they concisely and accurately describe the command, and once understood, provide the programmer the maximum understanding of the flexibility of the language. The following notation is used in syntax statements:

< > PARAMETER

Any term enclosed in brackets is an item which is a parameter in the syntax. e.g. An allowable syntax of the print command reads **?"<text>"**, with an example of this command being **?"Hello"**, which would print the text **Hello**. You can see that in this case, Hello is the parameter <text>.

 # # REPEATABLE

Any section of the syntax enclosed in "number signs" is repeatable. e.g. Another allowable syntax of the print command reads **?#<text>,#**, with an example of this command being **?"Enter","Position"**, which would print the text **Enter Position**.

[ ] OPTIONAL

Any section of the syntax enclosed in square brackets is optional. e.g. Another allowable syntax of the print command reads **?["<text>"]**, with an example of this command being **?**, which would print a blank line.

<< >> VALUE, REGISTER, or EXPRESSION

Any item enclosed in double angle braces is to be either a value, a register, or an expression enclosed by parentheses.

For any command which is highly flexible (and therefore complex in syntax), several syntax options will be shown, each accomplishing an example objective. For formal presentation, a "Full Syntax" will also be shown, illustrating in entirety the extensive power and flexibility of these commands.

REGISTERS

One hundred, 32-bit, general purpose, non-volatile registers are provided for use by MPL MATH. The **R** command is used to assign the result of an arithmetic or logical expression to a register. In addition, any register's contents can be used as a value in an arithmetic or logical expression or in an MPL command.

Each register stores 32-bit binary values, which can be interpreted as decimal integers between -2,147,483,648 and 2,147,483,647. They can alternatively be interpreted as hexadecimal integers or character strings (up to four characters).

Syntax: **R<<reg>>**[1]

where: <reg> is a one or two digit number from 0 to 99 representing the register number to be used; It can be either a constant or the result of an expression.

Examples: R3 is the designator for Register 3.
R(R80) designates the Register whose register # is contained in Register 80.

EXPRESSIONS

A arithmetic or logical expression is an algebraic formula which uses constants, registers, functions, and operators to calculate a value. The calculated value can then be stored in a register for future use or used directly as an argument of an MPL command. The syntax of MPL MATH expressions is standard algebraic notation, much the same as that used on a calculator. Operator precedence is supported (see section on Operators), and parentheses can be used to force a set of operations to be performed first.

Syntax: **<value> [# <binary operator> <value> #]**

where: <value> a register, constant, function, or another expression (surrounded by parentheses) which may be preceded by one or more unary operators.

<binary operator> a binary operator (see section Operators).

Examples:		
	1+1	value = 2
	123+12/6	value = 123+(12/6) = 123+2 = 125
	(123+12)/6	value = 135/6 = 22
	-123+12/6	value = (-123)+(12/6) = 2-123 = -121
	R3*12	value = 12 times the value in Register 3
	g?/6	value = commanded absolute position divided by 6

[1] If a Register is used to specify the register, it must be enclosed in parentheses. See the example.

CONSTANTS

Constants are values that can be used within expressions which are always equal to the same number. Each constant can be expressed as one of three "data types": the **decimal integer**, the **hexadecimal integer**, or the **character constant**.

Constants within expressions have their data type determined as described below. A constant's data type is totally independent of the PMC communications mode.

A **decimal integer** is any number that begins with a non-zero digit (1-9), and can range from 0 to 4294967295.

A **hexadecimal integer** is any number that begins with a zero (0), and can range from 0 to 0FFFFFFFF.

A **character constant** is usually a single character (but can be up to four characters) surrounded by single quotes ('). The value returned by a character constant is the ASCII[2] value of the character (or characters) within the quotes. If more than one character is in quotes, the first character represents the least significant byte of the 32-bit binary value.

Special characters can be represented using the caret (^) unary operator which modifies the ASCII code of the following character, to create a "control character". If this ASCII code is greater than 3fH (@ or above), the ASCII code is ANDed with 1fH otherwise the ASCII code in ANDed with 2fH. The caret operator can be used to print a control character by following it with a letter (A-Z) to represent the control character to be printed (i.e. ^G prints Ctrl-G which sounds a bell). Since spaces cannot be entered in the MPL program buffer, the tilde symbol (~) is used to represent a space. Examples of special characters are:

^0 = <space> ~ = <space> ^2 = " ^" = " ^3 = # ^4 = $ ^' = '

```
Examples:  R1=3141592  Register 1 assigned the value 3,141,592
           R4=0FFFF    Register 4 assigned the value 65,535
           R2='A'      Register 2 assigned the value 65
           R7=041      Register 7 assigned the value 65
           R5='321'    Register 5 assigned the value 3,224,115
           R6=0313233  Register 6 assigned the value 3,224,115
```

[2] American Standard Code for Information Interchange. See Appendix B for a table of ASCII codes.

OPERATORS

There are two types of operators available: unary and binary. Unary operators return a value by operating on another single value. Binary operators return a value resulting from an operation of two other values.

The following table lists and briefly describes the operators available. Operators in the same section have the same precedence, and are evaluated left to right. Sections are in order of decreasing precedence, which means that operators in sections at the top of the table are performed before operators in sections toward the end of the table.

Op	Description	Type
()	expressions in parentheses are evaluated first, precedence rules apply within (<expression>)	n/a
-	negate (two's complement)	unary
~	not (one's complement)	unary
[]	absolute value [<expression>]	unary
->	bit number <value> -> <bit number>	binary
*	multiply	binary
/	divide (16-bit divisor)	binary
%	remainder (16-bit divisor)	binary
+	add	binary
-	subtract	binary
>>	shift right (logical) <value> >> <shift bits>	binary
<<	shift left <value> << <shift bits>	binary
=	equal to	binary
<>	not equal to	binary
<	less than	binary
<=	less than or equal to	binary
>	greater than	binary
>=	greater than or equal to	binary
&	logical and	binary
|	logical or	binary
^	exclusive or	binary

The divide and remainder operations will generate a range error if the divisor is too large. The bit operator will return the value (0 or 1) of the specified bit (0 through 31) of the first argument. The comparison operators each return a value of -1 if the compare is TRUE, or 0 if the compare is FALSE.

Examples:		
	R2+R3/4	Register 3 divided by 4 plus Register 2
	V!*100	current velocity multiplied by 100
	R1->7	bit 7 of Register 1 (0 or 1)
	'C'-'A'+1	3
	V*100<R4	-1 (true) if the last entered velocity multiplied by 100 is less than the contents of Register 4; otherwise 0 (false)

FUNCTIONS

Many MPL commands support the use of a <display> terminator which causes a value associated with that command to be displayed. The MPL+MATH commands listed below can not only be used to display a value, but can also be used to insert that value in a numeric expression.

Standard MPL+MATH Functions

Syntax	Value Returned
A?	last entered acceleration rate for motion
A!	current acceleration rate for motion
AL?	last entered acceleration rate for limits
AQ?	last entered acceleration rate for contour motion stop
AS?	last entered acceleration rate for E-stop
G?	currently commanded absolute position of the system
G!	absolute position of the system
H?	last entered home velocity
H!	current system velocity (same as V!)
I?	last entered index distance
I!	distance remaining in the current or last index
J?	last entered jog velocity
J!	current system velocity (same as V!)
M..	reserved for ECS functions
O[!]	current state of the PMC; OUT1 through OUT8
RE?	current branch/trap on error label
RE!	last error code
RI?	number of arguments read in most recent RI command
RI!	device input status
RJ?	last entered high resolution jog rate
RJ!	current high resolution jog rate
RS[!]	input latch bits
SC[!]	current state of the PMC; IN1 through IN80
SLF	current forward travel software position limit
SLR	current reverse travel software position limit
SM?	last commanded system mode
SM!	current system mode
SS[!]	current motion profile register
SX	current value of Register X
SY	current value of Register Y
SZ	current value of Register Z
TCP	current position loop compensation
TCV	current velocity loop compensation
TE?	current normalization error
TE!	current position error
TF	current feedforward gain
TP	current position loop gain
TV	current velocity loop gain
TX	current external gain
V?	last entered velocity rate
V!	current system velocity

Note: The question mark (?) <display> terminator is optional in the above functions.

USING EXPRESSIONS IN STANDARD MPL COMMANDS

The integration of 100 Registers with arithmetic and logical expressions into a Programmable Motion Controller is a powerful feature[3]. It allows the user to set MPL command values based on a the result of a numeric or logical calculations, which can be based on 1) previous results from other commands using any of the above functions, 2) I/O data, or 3) the contents of one or more of the one hundred non-volatile registers.

The MPL arguments that can use expressions and registers are listed below:

<condition>	<position>	<speed>
<count>	<rate>	<time>
<distance>	<select>	<label> (Branch & Function only)

To use the contents of a register as an argument, use the register name (R<reg>) in place of the number that would normally be entered. To use an expression as an argument, the expression must be enclosed in parentheses, and can be substituted in place of the number just like a register name.

Examples:		
	VR1	Set the velocity to the contents of Register 1; note that parenthesis are not needed for a register reference only. The command could however also read V(R1).
	J(R3/100)+	Jog in the positive direction at the velocity set by the contents of Register 3 divided by 100.
	I(I!)+	Index positively the remaining index distance. Note that this command would complete an index command that had been stopped for some reason.

USING INTEGER ARITHMETIC

MPL MATH uses 32-bit integer arithmetic for speed of operation and compatibility with the parameters and ranges used in MPL. Integer arithmetic can readily be used to deal with applications which use parameters which include decimal points, as well as to perform math functions which appear to need decimal points, such as multiplying a number by 6.67 (actually $6\ ^{2}/_{3}$).

For example, you may want to enter and display position in inches with up to three decimal places. MPL MATH supports this with input (RI) and output (?) commands which allow specification of a "decimal format". However, it is up to the individual writing MPL application software to "maintain the understood decimal point" and the desired precision throughout the arithmetic expressions.

Examples:		
	R1*20/3	Multiply register R1 by $6\ ^{2}/_{3}$.
	R1*3/20	Divide register R1 by $6\ ^{2}/_{3}$.

Note that in both cases the multiplication was done first so as not to lose precision because the intermediate number became too small.

---

[3] NOTE: Expressions and registers cannot be used in MPL commands while the PMC communications mode is set to binary input (SZ10xxxxxx). Otherwise the use of expressions and registers is independent of communications mode of the PMC.

COMMANDS ADDED TO MPL

MPL MATH adds a number of commands to MPL. These commands are used for Operator I/O, assigning the results of a calculation to a Register, and providing the user the ability to perform error recovery programs. In addition, the Gear Ratio and High Resolution Jog commands provide advanced electronic lineshafting capabilities.

? - Print / Display Expression Command

Purpose: **Print to, setup, or select a "default output device"[4].**

Goal 1: Setup the output device driver for one of the display devices supported by the **?** command and/or select a device as the default output device.

Syntax: **?@<<device>>[:<dev spec>]<cr>**

where: <device> A number from 1 to 5 specifying the output device. The default output device after powerup is device 1. The following output devices are available:

1 Character-mode output to the serial communications interface; (PMC connector JM2).

2[5] Character-mode output to the (optional) auxiliary serial port; (EIO connector JM3).

3[5] Block-mode output to an ITM-270 or ITM-27 display panel attached to the serial communications interface (PMC connector JM2).

4[5] Block-mode output to ITM-270 or ITM-27 display panel attached to the (optional) auxiliary serial port (EIO connector JM3).

5[5] Block-mode output to the NRO-066 Numeric Readout attached through a parallel interface (either 8 or 10 parallel output lines) to the (optional) EIO daughter board (EIO connector JM2).

---

[4] Once a "default output device" is selected, it will be used by future ? commands unless a different output device is specified, or the default is changed.

[5] A device specifier parameter <dev spec> is required to define hardware characteristics for these output devices. Refer to Appendix A for a detailed description of the output devices supported by MPL MATH.

<dev spec> A string of letters and numbers which set the device specifications for a particular device. Each <device> has a unique <dev spec> syntax, as described below:

Device 1 Syntax: (none)
Device 2 Syntax: **[T<<timeout>>]**
Device 3 Syntax: **[<<address>>][<type>]**
Device 4 Syntax: **[<<address>>][<type>]**
Device 5 Syntax: **[<<select>>]**

<timeout> A decimal number representing the transmit time-out period in msec. If a character can't be sent within the allotted time-out period because handshaking is held up, an I/O time-out error will be generated. A <timeout> of zero (default) disables the time-out feature. The <timeout> is rounded up to the nearest 4 millisecond interval. **Note:** The input time-out (used by the RI command) is the same as the output time-out for output device 2.

<address> A decimal number from 1 to 63 represents the address sent to the terminal at the beginning of each line of printed output. Default = 1.

<type>
A Selects the block-mode ITM-270 polling protocol (default).
B Selects the block-mode ITM-27 polling protocol.

<select> A number from 0 to 4 representing the address of the NRO-066 device to be selected. Four units (addressed 1 to 4) can be attached to the EIO in parallel by setting an address in each unit, with the device driver selecting the appropriate unit with two of the parallel output lines. Alternatively one unit (address 0) may be attached without requiring the use of the address output lines. The default <address> is 0.

Example 1: ?@4:1A Specify an ITM-270 attached to the auxiliary serial port and configured for multi-drop address 01.

Example 2: ?@5:0 Specify a single NRO-066 Numeric Readout attached to the EIO parallel interface (requires 8 outputs).

Example 3: ?@3:21B Specify an ITM-27 attached to the primary serial port and configured for multi-drop address 21.

Example 4: ?@5:1 Specify an NRO-066 Numeric Readout attached to the EIO parallel interface with address 1. Up to three additional devices can be attached in parallel as devices 2-4 (requires 10 outputs).

Example 5: ?@5:4 Specify an NRO-066 Numeric Readout attached to the EIO parallel interface with address 4. The unit must have its address yset internally to correspond with address 4.

Goal 2: Print text and/or other data to the current default output device.

Syntax: **? #["<text>"][,][<expr>[:<format>]][,]# <cr>**

where: <text> A string of text to be printed; See the discussion of character constants in the Constants Section.

<expr> A numeric expression to be printed; See the Expressions Section.

<format>[6] A string of characters specifying the format in which to display the result of <expr>. The syntax for <format> are described below:

[*n*][**D**] Print a decimal integer, right justified in a field with *n* characters. Leading zeros are replaced by spaces. If the number is longer then *n* characters, the full number is printed with no leading space. The *n* is optional and defaults to **1**.

[*n*].*m*[**D**] Print a fixed-point decimal integer with *m* digits to the right of the decimal point, right justified in a field with *n* characters. Leading zeros are replaced by spaces. If the number is longer then *n* characters, the full number is printed with no leading space. The *n* is optional and defaults to **1**.

[*n*]**X** Print the least significant *n* digits of a hexadecimal integer, including leading zeros. The *n* is optional and defaults to **8**, which is its maximum.

[6] If no <format> is specified, all output to Device 1 will use the numeric format specified by the current PMC Communications Output Mode, decimal or hex (SZ Command, Bits 7-6). Output defaults to decimal for other devices.

[*n*]**C** Print the first *n* characters of a stored character string, which may be four characters maximum. The *n* is optional and defaults to **4**, which is its maximum.

[*n*]**B** Print a binary number of exactly *n* bytes. The high order byte is sent first, followed by the low order bytes. The *n* is optional and defaults to **4**, which is its maximum.

, (comma) Separate multiple strings of text and expressions in the same print statement. Normally, the **?** command prints a complete line of text and then advances by printing a carriage return and linefeed. To continue printing on the same line with the next **?** command, use a comma as the last character of the **?** command.

Example 1: ?12+R1/2 — Print a value equal to half of the contents of Register 1 plus 12.

Example 2: ?"Hello." — Print "Hello.", followed by a carriage return and linefeed.[7]

Example 3: ?"Position",R1,"." — If Register 1 contains the number 123, this statement will print "Position 123.", followed by a carriage return and linefeed.

Example 4: ?"Total~=~$",R3:1.2, — If Register 3 contains the number 36595, this statement will print "Total = $365.95", with no following carriage return and linefeed (because of the comma).

Example 5: ?"The~system~is",R2:c — If Register 2 contains the character string '~off', this statement will print "The system is off".

Example 6: ?"^GError~^3",R4:2x — If Register 4 contains 160 ($00A0_H$), this statement will sound the bell of the serial device (if it has one), and print "Error #A0".

---

[7] Using the ? command interactively with "Device 1" selected produces confusing results when printing strings. This is because Device 1 is also the programming console, and the command is executed interpretively on a character by character basis.

It is convenient however, to use the ? command interactively to print the contents of registers or expressions. To try some examples of printing strings to the console, see the MPL program listed under the RI command.

Goal 3: Print to a device other than the current default device.

Syntax: **?@<<device>>[:<devspec>],["<text>"][,][<expr>[:<format>]][,]<cr>**

Examples: ?@4,"Enter~Position:~"

Full Syntax for the ? Command:

Syntax 1: **?[@<<device>>[:<devspec>],]#["<text>"][,]**
[<expr>[:<format>]][,]#<cr>

Syntax 2: **?@<<device>>[:<dev spec>]<cr>**

Syntax 3: **?@?** Display the current default output device.

R - Register Assignment Command

Purpose: **Store the results of an arithmetic or logical expression in one of one hundred non-volatile registers.**

Syntax: **R<<reg>> = <expr><cr>**

where:

<reg>	The number (from 0 to 99) of the register to be assigned the result of <expr>; This number can also be the result of an expression enclosed in parentheses. See Registers Section.
<expr>	A numeric expression; See Expressions Section.

Example 1: R1=R2*45/100 Register 1 = Register 2 * 0.45

Example 2: R3=SS! Register 3 = current motion profile register

Example 3: R(R80)=g! The register whose register number is contained in Register 80 = the current absolute position.

R? - IF/ELSE Construct

Purpose: **Provide an IF/ELSE construct for the MPL programmer.** Execute a set of commands based on a condition defined by an expression or set of expressions.

There are actually four parts to the IF command: IF, ELSEIF, ELSE, and ENDIF. The IF and ENDIF parts are required, and the ELSEIF and ELSE parts are optional. The ELSEIF part can be repeated. The IF and ELSEIF statements are followed by commands which will be executed if the expression <expr> in that statement is evaluated as TRUE. The commands following the ELSE statement will be executed only if all the IF and ELSEIF statements present are false.

		IF Construct	CASE Construct

Syntax:	**R?<expr><cr>**
	#<command>#
	[#R:<expr><cr>#]
	#<command>#
	[R:<cr>]
	#<command>#
	R?<cr>
		IF <expr>
		
		ELSEIF <expr>
		
		ELSE
		
		ENDIF
			CASE <expr>:
			
			CASE <expr>:
			
			DEFAULT:
			
			ENDCASE

where: <expr> A boolean expression; Program execution will continue starting with the next command if the result of this expression is non-zero, otherwise MPL program execution will continue at the next **R:** or **R?** command. If this expression is not included, program execution will continue with the next command.

<command> An MPL+MATH command to be executed if the above condition is true.

Example 1:

```
R?_R3>5                          IF R3 > 5
?"Too~big.~~Setting~to~5."       Print a warning message.
R3=5                             Set R3 equal to 5.
R?                               ENDIF
```

Example 2:

```
R?_RE!=0A1                       IF last error was #A1
?"Invalid~command~error"         Print the error message.
R:_RE!=0A2                       ELSE IF last error was #A2
?"Terminator~error"              Print the error message.
R:_RE!=0A4                       ELSE IF last error was #A4
?"Value~out~of~range~error"      Print the error message.
R:                               ELSE
?"Error~",RE!:2x                 Print the error code.
R?                               ENDIF
```

RE - Branch or Trap on Error / Generate Error

Purpose: **Cause MPL program execution to branch (transfer operation) to the specified label in the program buffer when a non-programming error occurs.**

Syntax 1:	**REB <label>**	Branch on error (many)
Syntax 2:	**RET <label>**	Trap on error (once)
Syntax 3:	**RE <cr>**	Disable branch/trap on error
Syntax 4:	**RE ***	Clear the last error code
Syntax 5:	**REE [<<error>>] <cr>**	Generate an MPL error
Syntax 6:	**REF <<error>> <cr>**	Force an error
Syntax 7:	**RE <display>**	Display last error or error label
Usage:	R<<reg>> = **RE[<display>]**	Get last error or error label

where:

<label> The program marker label of the error handler where MPL command execution will be transferred when an error occurs.

<cr> Clear the error <label>. All subsequent errors will be handled normally and no branch will occur on error.

* Clear the last error code (returned by **RE!**) to zero. Clearing the last error is not required for normal error processing.

<error> A two-digit hexadecimal number representing the error code to be generated. In the **REE** command, if <error> is not specified it defaults to the last error (**RE!**).

<display> **?** Display the current label to branch/trap to on error.
! Display the last error code.

The **REB** (branch on error) and **RET** (trap on error) commands both install an error handler routine to which execution will be transferred when an error occurs. If the **REB** command is specified, the jump to error <label> occurs each time a non-programming error occurs. For the **RET** command, only the first non-programming error causes a jump. The Trap feature is particularly useful during system startup since the **REB** command can lead to an infinite loop if MPL errors exist.

Programming errors (#C0 through #C9) and User Abort errors (#B5, #D0, #D2, and #22) never call the error handler, even if one is installed using **REB** or **RET**. These errors will always generate a "#<error>" message and abort the program just like MPL would without an error handler installed. All other errors call the error handler.

The **REE** and **REF** commands are functionally similar, but they are used for different purposes. The **REE** command is designed for use inside an error handler. It generates a "#<error>" message and aborts the program just like MPL would without an error handler installed. It should be used for all error codes that are not handled in a special way by the error handler.

The **REF** command is used to generate a specific error that is to be handled like all other errors. If an error handler is installed (using **REB** or **RET**), the **REF** command will call it, and **RE!** will reflect the error code generated by the **REF** command. If no error handler is installed, the **REF** command will behave like the **REE** command described above. This command is useful for testing MPL error handler code, or for forcing a user-defined error to occur during execution of an MPL program.

```
Example 1: REBe                  Branch on error to program label "e".

Example 2: @error_handler        Error handler
           Bv:RE!=0D9            Branch to label "v" if PSJ overflow error
           Bs:RE!=0D1|RE!=0B3    Branch to label "s" if STOP asserted error
           REE                   Handle all other errors normally (abort)
```

RG - Gear Ratio Command

Purpose: **Set a precise speedy ratio in electronic lineshafting applications (external mode) as the ratio of two integers.**

Syntax 1: **RG [<<ingear>>[/<<outgear>>]]#<term>#<cr>** Gear ratio
Syntax 2: **RG <cr>** Disable gear ratioing
Syntax 3: **RG <display>** Display gear ratio

where: <ingear> The number of teeth on the imaginary input gear, with a range of 1 to 65535. This value is entered into the motion parameter buffer.

<outgear> The number of teeth on the imaginary output gear, with a range 2 to 65536; If not specified, the <outgear> default is 10,000, allowing the user to specify the speed ratio to a precision of hundredths of a percent. This value is entered into the motion parameter buffer.

Note: This command has a constraint that <ingear> must be less than or equal to <outgear>*4095/4096. To run the slave servomotor at a ratio of 1:1, use j10000, which causes the servomotor to run at 100.00% of the electronic lineshaft speed.

<term>
+ Start motion in the positive direction.
- Start motion in the negative direction.
* Stop motion.

<cr> If <cr> follows <term>, the system first waits for constant velocity to be reached, and then enables high resolution gear ratioing. If <cr> immediately follows the **RG** command (with no other parameters), high resolution gear ratioing is disabled, but motion continues at the current low resolution jog speed.

<display>
? Display the last specified gear ratio.
! Display the current gear ratio.
% Repeatedly display the actual ratio.

Example: RG2/3+ Accelerate to, and run at a constant speed which is precisely two thirds of the rate of the electronic lineshaft (motion reference bus).

RG - Gear Ratio Command (continued)

The two integers <ingear> and <outgear> can be thought of as specifying the number of gear teeth on imaginary "input" and "output" gears, where the input gear is be mounted on the electronic lineshaft, and the output gear is mounted on the servomotor under control. Since the output gear, specified by <outgear>, must have more teeth than the input gear, you can see that the speed of the servomotor (in terms of encoder counts/sec) must always be slower than the speed of the electronic lineshaft.[8]

Once top speed is reached, (which can be all the time for slave servomotors) the speed of the slave servomotor will be ratioed to the electronic lineshaft by the specified ratio. e.g. 1/3 specifies that the servomotor will run at 1/3 (in encoder counts per lineshaft count) of the speed of the electronic lineshaft.

Notes:

1) The **RG** command functions the same as the Jog command during the acceleration and deceleration segments. Motion is accelerated at the **A** rate to the closest specifiable coarse speed less than that specified by the **RG** command. The <cr> command terminator enables the final acceleration step to the precision specified speed. This process is reversed on deceleration. The **RG** command therefore has an implied semicolon (;) wait for top speed synchronization character built in. MPL execution is suspended until top speed is reached.

2) The **RG** Command changes the jog speed value stored in the Motion Buffer.

[8] For slave servomotor speeds greater than the electronic lineshaft, the slave servomotor must have an encoder resolution less than the resolution of the electronic lineshaft. e.g. If the motor driving the electronic lineshaft has 24,000 counts per revolution, the slave servomotor has a resolution of 6,000 counts per revolution, and the slave servomotor is commanded to run at 2/3 of the electronic lineshaft, its actual speed will be (24,000/6,000)*2/3 = 8/3. For this case, when the master motor runs 300 RPM, the slave servomotor will run 800 RPM.

RI - Read Input / Get Input Status

Purpose: **Read operator input into a Register from an operator input device.** This command allows straight numerical input or formatted input, including either decimal point or character format. It also supports different types of input devices.

Goal 1: Setup an input device driver to read from one of the operator interface devices supported by the **RI** command and/or select a device as the "default input device"[9].

Syntax: **RI@<<device>>:<dev spec><cr>**

where: <device> is a number from 1 to 5 specifying the device to be selected as the default input device;

- **1** Read from the serial communications interface (PMC connector JM2) using line-mode or character-mode.
- **2** Read from the auxiliary serial port (EIO connector JM3) using line-mode or character-mode.
- **3**[10] Read from an ITM-270 or ITM-27 Operator Keypad attached to the serial communications interface (PMC connector JM2) using block-mode.
- **4**[10] Read from an ITM-270 or ITM-27 Operator Keypad attached to the auxiliary serial port (EIO, connector JM3) using block-mode.
- **5**[10] Read from a bank of thumbwheel switches through a parallel interface (PMC or EIO). The **RT** command is normally used for this purpose.

---

[9] Once a "default input device" is selected, it will be used by future RI commands unless a different device is specified or the default is changed.

[10] A device specifier parameter <dev spec> is required to define hardware characteristics for these input devices. Refer to Appendix A for a detailed description of the input devices supported by MPL MATH.

<dev spec> A string of letters and numbers which set the device specifications for a particular device. Each <device> has a unique <dev spec> syntax, as described below:

Device 1 Syntax: **[<mode>][T<timeout>]**
Device 2 Syntax: **[<mode>][T<timeout>]**
Device 3 Syntax: **[<<address>>][<type>]**
Device 4 Syntax: **[<<address>>][<type>]**
Device 5 Syntax: **[<<id>><spec>]**

<type>
- **L** Selects line input mode (default). All input will be done a line at a time, allowing editing of the line as it is typed.
- **C** Selects character input mode. All input will be done on a character-by-character basis.

<timeout> A decimal number representing the receive character timeout period in msec. If a character is not received within the allotted time-out period, an I/O time-out error will be generated. A <timeout> of zero (default) disables the time-out feature. The <timeout> is rounded up to the nearest 4 msec interval.
Note: The input time-out is the same as the output time-out for device 2.

<address> A decimal number from 1 to 63 represents the address sent to the terminal at the beginning of each line. Default = 1.

<type>
- **A** Selects the block-mode ITM-270 polling protocol (default).
- **B** Selects the block-mode ITM-27 polling protocol.

<id> See the **RT** command for a description of <id>.

<spec> See the **RT** command for a description of <spec>.

Example 1: RI@4:1A Specifies an ITM-270 attached to the auxiliary serial port and configured for multi-drop address 01.

Example 2: RI@3:21B Specifies an ITM-27 attached to the primary serial port and configured for multi-drop address 21.

Example 3: RI@1:cT0 Specifies a dumb terminal (or IBM-PC with MAX or MAX-II) attached to the primary serial port with single character input and no timeout.

Goal 2: Read a decimal number from the default operator interface device.

Syntax: **RI R<reg> <cr>**

where: <reg> A number from 0 to 99 specifies the register to receive the input.

Example: RIR3 Read operator input into Register 3; for entry 123, followed by a "carriage return", Register 3 = 123.

Goal 3: Read one or more formatted input parameters from an operator input device.

Syntax: **RI #R<reg>[:<format>]<sep># <cr>**

where: <reg> A number from 0 to 99 specifies the register to receive the input.

<format>[11] A string of characters specifying the format in which to input the number. The syntax for <format> are described below:

[*n*][**D**] Read a decimal integer of up to *n* characters. The *n* is optional and defaults to the longest possible entry.

[*n*].*m*[**D**] Read a fixed-point decimal integer of up to *n* characters, with *m* digits to the right of the decimal point. The *n* is optional and defaults to the longest possible entry. The integer value returned by this function is equal to the number entered at the terminal * 10^m (see Examples below).

[*n*]**X** Read a hexadecimal integer of up to *n* digits. The *n* is optional and defaults to **8**.

[*n*]**C** Read a character string of up to **4** characters. Only the first *n* characters will be read. The *n* is optional and defaults to **4**.

Note: The *n* specifies the maximum number of characters to read, including the sign and decimal point for D format; Characters in excess of the number n will be discarded at the completion of the read. The size of the input line buffer is 32 characters.

<sep> A separator is used to separate register inputs in the same read statement. The separators are as follows:

, When a comma is used, the user must type a space or a comma to delimit values entered on the input line. The final comma is not needed.

; When a semicolon is used, no delimiter is expected between input fields, and the next input field is identified only by an invalid character in the previous field. A semicolon must be used at the end of a line that may not end in a carriage return.

[11] The default <format> is decimal.

Example 1: RIR3:6.2D	Read a formatted number of up to six characters long, and with up to 2 decimal places, into Register 3; for entry 123.45, Register 3 = 12345
Example 2: RIR3:6.2D,R9:4.1D	Read a formatted number of up to six total characters long into Register 3 and a formatted number of up to 4 characters long into Register 4.

Note: The RI command can't be used interactively to input a number from the primary serial port (Device 1). To try the examples below, write an MPL program as illustrated:

```
@1_RI_Example                establish program label 1
?                            print a blank line
?"Enter~number:~",           prompt the operator for input
?                            print a blank line
RIR3                         read input into Register 3
?"Reg~3~=~",R3               print the value of Register 3
?                            print a blank line
@2                           establish program label 2
?"Enter~formatted~num.:~",   prompt the operator
RIR3:6.2D                    read a formatted decimal number into Register 3
?"Reg~3=",R3,"~~",R3:6.2D    print Register 3, unformatted and formatted
B2:(R3<>-1234)               branch to label 2 if R3 is not equal to -1234
R1='z'                       initialize Register 1 to z
?                            print a blank line
@3                           establish program label 3
?"Enter~Character:~",        prompt the operator
RIR1:C                       read a single character into Register 1
B3:(R1<>'c')                 branch back to label 3 is R1 is not equal to c
?"RI~Example~Complete"       print "RI Example Complete"
E                            exit to command level
```

Once the program above is entered, run it by typing B1<cr> and try the examples below:

Example 1: RIR3	Read terminal input into Register 3; Enter 123<cr> and Register 3 = 123
Example 2: RIR3:6.2D	Enter 123.45, Register 3 = 12345 Enter 123.4, Register 3 = 12340 Enter 12.345, Register 3 = 1234 Enter .1, Register 3 = 10 Enter -12.34, Register 3 = -1234
Example 3: RIR1:C	Read a single character into Register 1 Enter a, Register 3 = a Enter b, Register 3 = b Enter c, Register 3 = c

Goal 4: Read input from a device other than the current default device.

Syntax: **RI@<<device>>[:<dev spec>], #R<reg>[:<format>]<sep># <cr>**

Example: RI@4,R9 Read input from Device 4 into Register 9.

Goal 5: Determine the status of the last Read Input operation.

Syntax: **RI?** Display status of last read operation

Usage: R<<reg>> = **RI[?]** Get status of last read operation

This command displays the number of registers modified by the last read input operation (Goals 2, 3, and 4) or the input error code. This value is incremented for each successful read of input. If input is not available (null input) the value is zero. If an input error occurs, a negative value is returned indicating the error code. The following error codes may be returned:

-1 **Input device time-out error.** A character was not received from the input device for the number of milliseconds specified in <dev spec> for that device. This code can only be returned by devices which support a <timeout> parameter in the <dev spec>.

-2 **Input value too large.** A number entered by the user was too large to fit in a 32-bit register.

-3 **Invalid character in input field.** A character that is not valid for the input format specified was entered by the user. For example, the letter 'B' was typed when a decimal number was expected.

```
Example:  RIR1,R2,R3     Read three numbers
          B3:RI?=3       Branch if all three numbers were entered correctly
          Bt:RI?=-1      Branch if a time-out error occurred
          Bv:RI?=-2      Branch if one of the values was too large
          Bc:RI?=-3      Branch if an invalid character was entered
          ?"Lack data"   Otherwise, there must have been too few entries
          E
```

Goal 6: Determine if there is a character available from a device.

Syntax:	**RI[@<<device>>]!**	Display input status of port
Usage:	R<<reg>> = **RI[@<<device>>]!**	Get input status of port

This command displays **01** if a character is available at the input port, and **00** if no character is available. **This command only works with devices 1 and 2.** All other devices display **00**.

When used with device 1, **RI!** may display **00** even when a character has been received by the PMC. This is because MPL removes characters from the input buffer while checking for a received **Esc** character between commands. Device 2 will always return **1** if a character is available.

Example:	RI@2:C	Character input from auxiliary serial port
	@wait	Wait loop
	Fa	Call the function at label 'a'
	Bw:RI!=0	Loop if no character has been received
	RIR1:1C;	Read the character into R1
	?"Received~";R1:C	Print the character received
	E	Stop

Full Syntax for the RI Command:

Syntax 1: **RI[@<<device>>[:<dev spec>],] #R<reg>[:<format>]<sep># <cr>**

Syntax 2: **RI@<<device>>[:<dev spec>]<cr>**

Syntax 3:	**RI@?**	Display current default input device
Syntax 4:	**RI?**	Display status of last read operation
Syntax 5:	**RI[@<<device>>]!**	Display input status of port

Usage 1:	R<<reg>> = **RI[?]**	Get status of last read operation
Usage 2:	R<<reg>> = **RI[@<<device>>]!**	Get input status of port

Note: You can read inputs into multiple registers with a single **RI** command. If multiple registers are specified in the **RI** command, items typed on the input line may be delimited by one or more spaces or commas. If less items are entered than expected, the registers with no input will be left unchanged. If a blank line is entered, all registers are left unchanged and the **RI?** command or function will return zero.

RJ - High Resolution Jog Command

Purpose: **Set a very high resolution speed ratio in electronic lineshafting applications (external mode).** Set a very high resolution speed (internal mode).

Syntax 1:	**RJ <<ingear>> #<term># <cr>**	High resolution jog
Syntax 2:	**RJ <cr>**	Disable high resolution
Syntax 3:	**RJ <display>**	Display high resolution speed

Usage 1:	R<<reg>> = **RJ[?]**	Get the last entered hi-res speed
Usage 2:	R<<reg>> = **RJ!**	Get the current hi-res speed

where: <ingear> The number of whole teeth on the imaginary input gear using an output gear of 268,435,456 teeth. Range: 1 to 268,435,455. This value is entered into the motion parameter buffer.

<term>
- **+** Start motion in the positive direction.
- **-** Start motion in the negative direction.
- ***** Stop motion and terminate the command.

<cr> If <cr> follows <term>, the system first waits for constant velocity to be reached, and then enables high resolution jog mode. If <cr> immediately follows the **RJ** command (with no other parameters), high resolution jogging is disabled, but motion continues at the current low resolution jog speed.

<display>
- **?** Display the last entered speed (ratio) value.
- **!** Display current actual speed (ratio) value (0 = inactive).
- **%** Repeatedly display the actual speed (ratio) value.

Example: RJ67108864+ Accelerate to, and run at a speed ratio of precisely one quarter of the rate of the electronic lineshaft (motion reference bus), if the system is in the External Mode. If the system is in the 384 kHz internal Mode, it will accelerate to and run at precisely one quarter of the internal, crystal controlled 384 kHz clock (96 kHz).

RJ - High Resolution Jog Command (continued)

The standard PMC velocity resolution of twelve bits (or 4096 parts) is extended to 28 bits (or 268,435,456 parts) by this command, allowing over eight decimal places of precision. The actual speed ratio set by this command is: <ingear>/268,435,456

The two integers can be thought of as specifying the number of gear teeth on imaginary "input" and "output" gears, where the input gear is be mounted on the electronic lineshaft, and the output gear is mounted on the servomotor under control. The output gear is fixed, and specified to have 268,435,456 teeth. Since this is more than the input gear, you can see that the speed of the servomotor (in terms of encoder counts/sec) must always be slower than the speed of the electronic lineshaft. Further, the speed of the servomotor will always be ratioed to the electronic lineshaft (once top speed is reached) by the specified ratio. e.g. 1/3 specifies that the servomotor will run at 1/3 (in encoder counts per lineshaft count) of the speed of the electronic lineshaft.

Notes:

1) This command has a constraint that <ingear> must be less than or equal to 268,435,455. For a speed ratio of 1:1, use j10000, which causes the servomotor to run at 100.00% of the electronic lineshaft speed.

2) The **RJ** command functions the same as the Jog command during the acceleration and deceleration segments. Motion is accelerated at the **A** rate to the closest specifiable coarse speed less than that specified by the **RJ** command. The <cr> command terminator enables the final acceleration step to the precision specified speed. This process is reversed on deceleration. The **RJ** command therefore has an implied semicolon (;) wait for top speed synchronization character built in. MPL execution is suspended until top speed is reached.

3) The **RJ** Command changes the jog speed value stored in the Motion Buffer.

4) See the footnote for the **RG** command.

RS - Input Scan Command

Purpose: **Set up one or more of the Machine Inputs to be automatically scanned every four milliseconds.** If one of the eight inputs are at the specified input state during the scan, an *input latch bit* is set. This command can also be programmed to generate an MPL error (#01) when an input latch bit is set, and an MPL-MATH error handler can be used to perform the appropriate action.

Syntax 1:	**RS <<select>>[/<<level>>][+]<cr>**	Enable the input scanner
Syntax 2:	**RS <cr>**	Disable the input scanner
Syntax 3:	**RS [<<clear>>]***	Clear input latch bits
Syntax 4:	**RS <display>**	Display the results
Usage:	R<<reg>> = **RS[!]**	Read the input latch bits

where:

<select> A hexadecimal number that defines which inputs are to generate an error #01 when they become latched (1=error, 0=no error). Also indicates which input latch bits are to be cleared before enabling the input scanner (1=clear, 0=don't clear). When error #01 occurs, the <select> parameter is set to zero to prevent further errors from occurring. The state at which each input is latched is defined by the <level> parameter.

<level> A hexadecimal number that defines the state at which each input is to be latched (1=low/active, 0=high/inactive). If this parameter is not specified, the last specified value is used (FF at power-up).

\+ Prevent clearing of the input latch bits specified by the <select> parameter. When this option is specified, the input latch bits must be manually cleared by using the **RS[<clear>]*** command.

<cr> When used with parameters (syntax 1), this terminator enables the 4 millisecond input scanner. When used without parameters (syntax 2), this terminator disables the input scanner.

<clear> A hexadecimal number that specifies which input latch bits to clear (1=clear, 0=don't clear). The input latch bits can be displayed using the **RS!** command.

* Clear the input latch bits specified by the <clear> parameter. If no <clear> parameter is specified, then all input latch bits are cleared.

<display> ? Display the current <select> and <level> parameters.

! Display the input latch bits. These bits indicate which inputs have reached their specified level at least once (1=latched, 0=not latched).

% Repeatedly display the input latch bits.

Note: When an error #01 occurs, the <select> parameter is automatically set to zero to prevent multiple errors from occurring. The input scanner remains enabled to detect other inputs which may become latched during the error processing routine.

Example 1: **RS81<cr>** Generate an MPL error when input IN80' or IN1' becomes latched. Once the error occurs, the <select> value is automatically set to zero to prevent continuous branching from occurring. To "re-arm" the input for error generation, the command must be re-executed.

Example 2: **RS?** 00/00 The "00" returned by the PMC indicates that the RS command will not generate any errors.

Example 3: **RS?** 81/FF The "81/FF" returned by the PMC indicates that the RS command is active, and looking for either IN80' or IN1' to become active (low).

Example 4: **RS!** 05 The "05" returned by the PMC indicates that inputs IN4' and IN1' have been latched.

Use: One immediate use for this capability is to deal with the "MCS-S E-Stop Interlocks" as described below:

In the MCS-S Series motion control systems, "Main Power" is three phase 220 VAC for powering the servomotor; "Control Power" is single phase power for powering the Programmable Motion Controller (PMC) and the servodrive microprocessor, which performs servodrive diagnostic functions. In the case of a servodrive fault, or if the Emergency Stop pushbutton is pressed, it is important to turn off the Main Power, but maintain the Control Power so that the diagnostic functions will continue to operate. Under these conditions, the PMC should disable the servo loop circuitry using a SM0 command, and print an error message to the user.

When the fault is cleared and/or the Emergency Stop circuitry is reset enabling the servodrive, the PMC will find the servodrive is disabled, wait for it to stabilize and then enable the servo loop circuitry.

The solution used in the MCS-S Series is to interface the D-ENABLE signal to discrete input (IN2) of the PMC using an IDC-5 Opto-22 compatible module. In the automatic powerup program, an **rebE** command is used to initialize error handling by the "**E**nable motor" routine. The RS command is then used to set it up to be scanned every four milliseconds, and generate error #01 if it is found asserted.

When D-ENABLE is asserted, the PMC is interrupted and branches to routine **E** which ascertains that the servodrive is disabled, waits for it to stabilize and then enables the servo loop circuitry. When this is complete, routine **E** sets up the "**D**isable motor" routine and initiates scanning of D-ENABLE input every four milliseconds. These routines also allow any PSJ Overflow error to be cleared by pressing the E-Stop and then resetting it. No other MPL errors are trapped by these routines.

RT - Read Thumbwheel Switch Input

Purpose: **Read input from thumbwheel switches or specify a thumbwheel switch configuration.**

Goal 1: Specify a thumbwheel switch device configuration.

Syntax: **RT [<<id>>]<spec><cr>**

where: <id> A digit from 1 to 5 representing a specific thumbwheel switch; If <id> is omitted, it will default to the last <id> used. On powerup <id> defaults to 1.

<spec> A string of characters containing the following configuration commands (mutually exclusive commands may not both appear in the same <spec> string):

P<decades> specify the number of decades to read, use the PMC inputs for BCD data. Range 1 to 15.

Note: P and E are mutually exclusive.

E<decades> specify the number of decades to read, use the EIO inputs for BCD data. Range 1 to 15.

H Use inputs IN10' through IN80' for BCD data input if PMC inputs are selected (P), or use inputs EIO-4 through EIO-7 if EIO inputs are selected (E).

Note: H and L are mutually exclusive.

L Use inputs IN1' through IN8' for BCD data input if PMC inputs are selected (P), or use inputs EIO-0 through EIO-3 if EIO inputs are selected (E).

N Read input data with the Normal polarity.

Note: N and C are mutually exclusive.

C Complement input data when reading it.

S<bit#> Specify the select output for the most significant decade of the thumbwheel switches for this <id>. Digit selection will proceed toward lower output bits (OUT8' to OUT1', or EIO-23 to EIO-0) as less significant digits are selected. See the table Digital I/O Signal Assignments for RD and RO Commands for <bit#> mapping.

T<time> Set the delay (in msec) before reading each BCD decade after the select output is asserted (4 ms resolution). Range 0 to 15.

Example: RT2_P3_H_C_S26_T0<cr> Define Thumbwheel Switch 2 as three decades, using PMC inputs IN10' through IN80' as complemented data inputs, and using PMC outputs OUT4' through OUT1' as select lines, with no delay between setting the select lines and reading the input data.

The configuration parameters in the <spec> are saved in internal registers. Five thumbwheel switch definitions can be saved in addition to the currently active definition. The factory default thumbwheel switch definitions are defined below:

RT1_P2_H_C_S25_T4 Thumbwheel Switch 1: 2 decades, inputs IN10' through IN80', complementary inputs, using select outputs 24 and 25 (OUT1' and OUT2'), and using a 4 msec time delay

RT2_P3_H_C_S26_T4 Thumbwheel Switch 2: 3 decades, inputs IN10' through IN80', complementary inputs, using select outputs 24, 25 and 26 (OUT1' - OUT4'), and using a 4 msec time delay

RT3_P4_H_C_S27_T4 Thumbwheel Switch 3: 4 decades, inputs IN10' through IN80', complementary inputs, using select outputs 24 to 27 (OUT1' - OUT8'), and using a 4 msec time delay

RT4_E5_H_C_S12_T4 Thumbwheel Switch 4: 5 decades, inputs EIO-4 through EIO-7, complementary inputs, using select outputs 8 to 12 (EIO-8 to EIO-12), and using a 4 msec time delay

RT5_E6_H_C_S13_T4 Thumbwheel Switch 4: 6 decades, inputs EIO-4 through EIO-7, complementary inputs, using select outputs 8 to 13 (EIO-8 to EIO-13), and using a 4 msec time delay

RT - Read Thumbwheel Switch Input (continued)

Goal 2: Read input from a thumbwheel switch.

Syntax: **RT [<id>],R<reg>[:<format>]<cr>**

where: <id> A digit from 1 to 5 representing a specific thumbwheel switch; If <id> is omitted, it will default to the last <id> used. On powerup <id> defaults to 1.

<format> A string of characters specifying the format in which to input the number. The syntax for <format> are described below:

n[**D**] Read a fixed-point decimal integer of *n* decades.

*n***X** Read a fixed-point hexadecimal integer of *n* digits.

Example 1: RT2,R6<cr> Read Thumbwheel Switch 2 into Register 6.

Example 2: RT1,R9:5D Read the first five decades of Thumbwheel Switch 1 into Register 9, as a decimal integer.

Example 3: RT1,R9:5X Read the first five decades of Thumbwheel Switch 1 into Register 9, as a hexadecimal integer.

Full Syntax of the RT Command:

Syntax 1: **RT [<id>][#<spec>#]#,R<reg>[:<format>]#<cr>** Read thumbwheel <id>
Syntax 2: **RT [<id>]<spec><cr>** Set up a new thumbwheel <spec> for <id>
Syntax 3: **RT <id><cr>** Make <id> the current default thumbwheel
Syntax 4: **RT [<id>]?** Display the thumbwheel <spec> for <id>

Example 1: RT2_L,R7<cr> Read value from the previously specified Thumbwheel Switch 3 into Register 7 using PMC inputs IN1' through IN8'.

Example 2: RT3<cr> Select Thumbwheel Switch 3 as the current default thumbwheel switch.

Example 3: RT_E5_L_N_S12_T8<cr> Define the current default thumbwheel switch setup as five decades, using EIO inputs EIO-0 through EIO-3 as data inputs, and EIO outputs EIO-12 through EIO-8 as select lines, with an eight millisecond delay before reading.

Example 4: RT2? Display the specification for Thumbwheel Switch 2.

Example 5: RT? Display the current default Thumbwheel Switch.

MPL COMMANDS ENHANCED BY MPL MATH

MPL MATH provides enhanced syntax for the Branch and Function commands to allow them to use MPL MATH boolean expressions.

B - Branch Command (expanded syntax)

Purpose: **Transfer MPL program execution (branch) to a program label, based on a condition defined by an expression.** This command supports conditional or unconditional, direct or indirect branches.

Syntax: **B <<label>>[12][:<expr>]<cr>**

where:

<label> Label where MPL program execution will transfer, or an expression in parenthesis which represents that label; Since the left parenthesis '(' indicates the start of an expression, it cannot be used as a label unless it is the result of an expression.

<expr> A boolean expression; Program execution will be transferred by the Branch command if the result of this expression is non-zero, otherwise MPL program execution will continue at the next statement. If this expression is not included, the branch will always be taken.

Example 1: R1='B' Assign ASCII code for **B** to Register 1.

Example 2: B(R1):G!>1000 Branch to the label in Register 1 if the current system position is greater than 1000.

Example 3: BA:RD1! Branch to label "A" if digital I/O point EIO-1 is asserted.

[12] If a Register is used to specify a label, it must be enclosed in parentheses, and contain the ASCII code for that label.

F - Function Command (expanded syntax)

Purpose: **Call a subroutine (function) based on a condition defined by an expression.** This command supports conditional or unconditional, direct or indirect branches.

Syntax: **F <<label>>[:<expr>]<cr>**

where: <label> Label where MPL program execution will transfer, or an expression in parentheses represents the label to branch to; Since the left parenthesis '('indicates the start of an expression, it cannot be used as a label unless it is the result of an expression.

<expr> A boolean expression; The function call will be executed if the result of this expression is non-zero, otherwise MPL execution will continue at the next statement without execution of the function call first. If this expression is not included, the branch will always be taken.

Example 1: F(R7):V!<450 Call the function at the label in Register 7 if the current velocity is less than 450.

Example 2: FX:RD6! Call function "X" if digital I/O point EIO-6 is asserted.

MPL MATH SYSTEM STATUS POLLING FUNCTIONS

Syntax: **<attn> <sys poll>**

where: <attn> Ctrl] (ASCII $1D_H$)

<sys poll> system status poll command from the table below:

<sys poll>	Description	Equivalent MPL command
a	Read analog input 1	RA1!
b	Read analog input 2	RA2!
d	Read EIO inputs/outputs	RD!
f	Show MPL-MATH status register	RF?
q <reg1>[13]	Query register (R0-R9)	?R<reg1>
w <reg1>[13]<value><cr>	Write register (R0-R9)	R<reg1>=<value>
w f <status>	Write flags register (low nybble)	RF0F/<status>
y <reg2>	Query any register (R0-R99)	?R<reg2>
u <reg2><value><cr>	Write any register (R0-R99)	R<reg2>=<value>

The **q**, **w**, **y**, and **u** polling commands have a special action in that after the <sys poll> command is received, a prompt character is returned acknowledging the request. At this time, MPL program execution is temporarily suspended until the register number is entered. The **q** and **w** polling commands accept only a single-digit register number from 0 to 9, and the **y** and **u** commands take a two-digit register number from 00 to 99. If it is a **q** or **y** command, the register value is immediately displayed and MPL program execution resumes. If it is a **w** or **u** command, the <value> is then entered (in the current input mode of the PMC) with a <cr> as the terminator, and MPL program execution then resumes. If an invalid character is entered (non-digit) at any time, the system status poll command is quietly aborted (no error is generated).

[13] Note that only Registers 0-9 can be accessed through system status polling with these commands.

EXTENDED I/O CAPABILITIES

The following functions and commands are provided to enhance MPL's interface with the PMC's standard machine I/O, and also to provide a consistent interface with the additional I/O added to the PMC by the optional EIO-900 daughter board. The EIO-900 adds one serial port, 24 discrete I/O points, two 8-bit analog inputs and one 8-bit analog output.

RA - Analog Input/Output Command

Purpose: **Set the value of the analog output or read one of the analog inputs.**

Syntax 1:	**RA<<value>>[<sign>][<cr>]**	Set analog output level
Syntax 2:	**RA[<<channel>>]<display>[<<time>><cr>]**	Display analog input/ output
Usage 1:	R<<reg>> = **RA<channel>[!]**	Read analog input channel
Usage 2:	R<<reg>> = **RA[?]**	Read analog output

where:

<value> A decimal value that defines the desired voltage level of the general purpose analog output; This value can range from 0 to 1000 representing an output voltage from 0 to 10 VDC. The voltage polarity is specified by <sign>. See RF command for raw analog output.

<sign> The sign of the analog output voltage:
- **+** Specify positive output voltage
- **-** Specify negative output voltage

<channel> The analog input channel or channels to be read:
- **1** Single-ended AN1 input (JM1 pin 1 = +voltage, pin 2 = ground)
- **2** Single-ended AN2 input (JM1 pin 3 = +voltage, pin 4 = ground)
- **3** Differential +/- (JM1 pin 1 = +voltage, pin 3 = -voltage)

<display>
- **?** Display the current setting of the general purpose analog output.
- **!** Read the specified analog input channel and display the result as a number from -1000 to 1000, representing an approximate input voltage from -10 to 10 VDC.
- **%** Repeatedly display the specified analog input voltage until an SCI character is received.

<time> The rate in msec that the % output is repeated (default=100)

Example 1: RA100- Set the general purpose analog output to -1 volt

Example 2: RA? Display the voltage at the analog output

Example 3: RAR4 Set the general purpose analog output to the voltage in R4

RB - Set Baud Rate

Purpose: **Set the baud rate for the auxiliary serial port on the EIO-900.**

Syntax 1: **RB <<baud>><cr>** Set baud rate
Syntax 2: **RB <display>** Display current baud rate

Usage: R<<reg>> = **RB[<display>]** Get current baud rate

where: <baud> Value specifying the baud rate; from 300 to 38400

<display> **?** Display currently selected baud rate.
! Same as ?

RD - Digital Input/Output Command

Purpose: **Set or read individual general purpose digital inputs/outputs.**

Syntax: **RD #<<I/O #>><operation>#<cr>** Set/display digital inputs/outputs

Usage 1: R<<reg>> = **RD<<I/O #>>[?]** Read state of one digital output
Usage 2: R<<reg>> = **RD<<I/O #>>!** Read state of one digital input

where: <I/O #> A number from 0 to 31 specifying the I/O point to be displayed or changed. See the table on the following page.

Note: Attempting to change an EIO point which is configured as an input will have no effect.

<operation>
- **+** Turn "on" the appropriate output (TTL low level).
- **-** Turn "off" the appropriate output (TTL high level).
- ***** Toggle the appropriate output to the opposite state.
- **?** Display the last entered state for a general purpose machine output; A 0 or 1 will be returned.
- **!** Display the status of <I/O #> point; A 0 or 1 will be returned.
- **%** Repeatedly display the status of <I/O #> point.

Example 1: RD20+ Turns on output EIO-20

Example 2: RD9+8- Turns on output EIO-9, turns off output EIO-8

Example 3: RD5? Display status of EIO-5, 00 if off, 01 if on

DIGITAL I O SIGNAL ASSIGNMENTS FOR RD & RO COMMANDS

Hex Address FFFFFFFF Bit#	Opto#	Signal Name	Connector	Function
0	0	EIO-0	JM2-47	in/out*
1	1	EIO-1	JM2-45	in/out*
2	2	EIO-2	JM2-43	in/out*
3	3	EIO-3	JM2-41	in/out*
4	4	EIO-4	JM2-39	in/out*
5	5	EIO-5	JM2-37	in/out*
6	6	EIO-6	JM2-35	in/out*
7	7	EIO-7	JM2-33	in/out*

* RF command, bit 6 configures I/O points EIO-0 through EIO-7 as outputs or inputs

8	8	EIO-8	JM2-31	out/in**
9	9	EIO-9	JM2-29	out/in**
10	10	EIO-10	JM2-27	out/in**
11	11	EIO-11	JM2-25	out/in**
12	12	EIO-12	JM2-23	out/in**
13	13	EIO-13	JM2-21	out/in**
14	14	EIO-14	JM2-19	out/in**
15	15	EIO-15	JM2-17	out/in**

** RF command, bit 5 configures I/O points EIO-8 through EIO-15 as outputs or inputs. The EIO-900 manual documents the necessary hardware change.

16	16	EIO-16	JM2-15	out/in***
17	17	EIO-17	JM2-13	out/in***
18	18	EIO-18	JM2-11	out/in***
19	19	EIO-19	JM2-09	out/in***
20	20	EIO-20	JM2-07	out/in***
21	21	EIO-21	JM2-05	out/in***
22	22	EIO-22	JM2-03	out/in***
23	23	EIO-23	JM2-01	out/in***

*** RF command, bit 4 configures I/O points EIO-16 through EIO-23 as outputs or inputs. The EIO-900 manual documents the necessary hardware change.

Bit#	Input Signal[14]	PMC Connector	Output Signal[15]	PMC Connector
24	IN1′	JM1-23	OUT1′	JM1-31
25	IN2′	JM1-21	OUT2′	JM1-29
26	IN4′	JM1-19	OUT4′	JM1-27
27	IN8′	JM1-17	OUT8′	JM1-25
28	IN10′	JM1-15		
29	IN20′	JM1-13		
30	IN40′	JM1-11		
31	IN80′	JM1-9		

[14] These input signals are read with the RO! or RD! commands.

[15] These output signals are read with the RO? or RD? commands.

RF - Flag Status Register

Purpose: **Set or show status and general purpose bit flags.** These bit flags are stored in non-volatile memory, and do not change when the power is cycled. If non-volatile memory is lost, the flag bits are set to 40_H.

Syntax 1: **RF <<select>>[/<<status>>]<cr>** Set flags register
Syntax 2: **RF <display>** Display status of flags register

Usage: R<<reg>> = **RF[<display>]** Get status of flags register

where: <select> A hex value which indicates which of the general purpose flag bits are to be changed. A bit value of 1 indicates that the flag bit is to be changed to the corresponding bit value in the optional <status> parameter. A bit value of 0 indicates that the bit is to remain unchanged. The flag bits are assigned as follows:

Bit 7: RAW ANALOG I/O ENABLE causes the RA command to skip the normalization process before reading and writing to the A/D and D/A converters. The input and output values will be 8-bit unsigned numbers (0 to 255) equal to the values read and written to the analog converter chips.

Bit 6: EIO-0 THROUGH EIO-7 CONFIGURE selects the direction of digital I/O bits EIO-0 through EIO-7 as either inputs (1) or outputs (0). This port is reconfigured immediately upon execution of the RF command.

Bit 5: EIO-8 THROUGH EIO-15 CONFIGURE selects the direction of digital I/O bits EIO-8 through EIO-15 as either inputs (1) or outputs (0). A hardware configuration jumper must also be switched to affect this change. **A change of this bit will only take effect at the next PMC reset.**

Bit 4: EIO-16 THROUGH EIO-23 CONFIGURE selects the direction of digital I/O bits EIO-16 through EIO-23 as either inputs (1) or outputs (0). A hardware configuration jumper must also be switched to affect this change. **A change of this bit will only take effect at the next PMC reset.**

Bit 3: General purpose bit to be set or read by an MPL program.

Bit 2: General purpose bit to be set or read by an MPL program.

Bit 1: General purpose bit to be set or read by an MPL program.

Bit 0: General purpose bit to be set or read by an MPL program.

<status> A hex value which sets the state of the selected bits; the default value is FF.

<display> **?** Display the last entered flags register value
! Display the current flags register value

Example 1: RF30/0<cr> configures EIO-8 through EIO-23 as outputs after the next hardware reset or power-up. This is the default configuration shipped by ORMEC.

Example 2: RF80<cr> enables RAW analog I/O mode.

RO - Extended Digital Input/Output Command

Purpose: **Set selected general purpose digital machine outputs or read general purpose digital machine inputs.**

Syntax 1: **RO <<select>>[/<<status>>]<cr>** Set digital outputs
Syntax 2: **RO <display>[<<time>><cr>]** Display digital inputs/outputs

Usage 1: R<<reg>> = **RO[?]** Read state of digital outputs
Usage 2: R<<reg>> = **RO!** Read state of digital inputs

where: <select> A hexadecimal number that specifies which outputs (of 32 total) will be driven to a status. A specific bank of 4 I/O points is handled by each hexadecimal digit. See the table on the page following the **RD** command. Eight hexadecimal digits could effectively mask or unmask each of the 32 I/O points to which the <status> (on/off) pattern would be applied.

<status> A hexadecimal number, providing an output pattern for up to 32 outputs. Individual outputs masked out by the <select> argument will be unaffected by this pattern. Any of the 32 I/O points configured as inputs will also be unaffected by this output pattern.

<display> **?** Display the last entered state of the general purpose machine outputs; An eight digit hexadecimal value will be returned.
! Display the current state of the general purpose machine inputs.
% Repeatedly display the current state of the general purpose machine inputs until an SCI character is received.

<time> The rate in msec at which the % output is repeated (default = 100)

This command is designed to be used with Opto-22 compatible modules and a 24 slot I/O rack, and consequently turning an output "on" defines an active (low) TTL signal level, lighting the status LED and causing the Opto Module output to conduct. Conversely, turning an output "off" defines an inactive (high) TTL signal level, which turns off the Opto-22 module. Also, the Bit #'s in the <select> and <status> parameters, and the EIO Signal Names, correspond with the Opto-22 I/O numbers printed on the I/O Rack.

Example 1: RO400<cr> Turn "on" output EIO-10

Example 2: RO900<cr> Turn "on" outputs EIO-11 and EIO-8

Example 3: RO4200/0<cr> Turn "off" outputs EIO-14 and EIO-9

Example 4: RO! Display current state of the Digital I/O

Example 5: BZ:RO!->4<cr> Branch to label "Z" if EIO-4 is asserted

Example 6: BW:RO!->4=0<cr> Branch to label "W" if EIO-4 is not asserted

RP - PMC Talk Command

Purpose: **To allow the user to communicate through the serial communications interface with auxiliary PMCs connected to the auxiliary serial port.**

Syntax: **RP [<<axis>>]<cr>**

where: <axis> The axis ID to select before entering talk mode;

Once talk mode is entered, all characters that go into the PMC programming port, except for special characters, will be sent directly out the EIO auxiliary serial port. All characters that go into the EIO auxiliary serial port will be sent directly out the PMC programming port. The following characters perform a special function when they are received by the programming serial port, and are not sent directly to the auxiliary serial port:

Key Sequence	Hex Value	Function Performed
Ctrl/[	1D	Select axis or system status poll (see PMC manual). This command is interpreted by the master PMC and not by the slaves connected to the auxiliary serial port.
Ctrl/A	01	Sends a $1D_{hex}$ out the auxiliary serial port. This is used to select a slave axis or perform system status polling on a slave.
Ctrl/X	18	Exits talk mode, returns to MPL command mode.

RX - External Register Read/Write Command

Purpose: **Set or display the contents of a register on an auxiliary PMC who's programming serial port is connected to the auxiliary serial port of the main PMC.**

Note: The auxiliary PMC must be in hexadecimal communications mode (input and output) with no echo for this command to function properly (**SZ41**). This command uses I/O device 2 for communications with the auxiliary PMC, and the communications time-out should be set using the **RI** or **?** command.

Syntax 1:	**RX <<reg>>[<axis>] = <expr><cr>**	Write register
Syntax 2:	**RX=<axis><cr>**	Select axis
Syntax 3:	**RX <poll>[<axis>],R<<reg>>[:<format>][;]<cr>**	Status poll input
Usage:	R<<reg>> = **RX<<reg>>[<axis>]**	Read register

where:

<reg> The number (from 0 to 99) of the register to write to on the auxiliary PMC. This number can also be the result of an expression enclosed in parentheses. Only register numbers 0 through 9 are supported by system status polling for MPL-MATH versions prior to 1.1a. See Registers Section.

<axis> The axis ID to select before sending the command (A-Z)

<expr> The value to write to the auxiliary PMC (for register writes only);

<poll> A letter representing the system status poll command to send to the auxiliary PMC before accepting input;

Note: There is also an RX function with the same syntax 1 as the RX command.

Example 1: ?RX2	Read and display the contents of Register 2 on the current auxiliary axis (sends **<attn>q2**).
Example 2: ?RX4B	Read and display the contents of register 4 on axis B; (sends **<attn>B<attn>q4**).
Example 3: RX1=123	Write the value 123 to Register 1 on the current auxiliary axis (sends **<attn>w10000007B<cr>**).
Example 4: RX1B=012345678	Write the hex value 12345678 to register 1 on axis B (sends **<attn>B<attn>w112345678<cr>**).
Example 5: RX=A	Selects auxiliary axis A (sends **<attn>A**).
Example 6: RXcA,R45:2x;	Poll the inputs on axis A, and store the result in R45 (sends **<attn>A<attn>c** and reads 2 hex digits).
Example 7: ?RX2A+RX4B+R7	Calculate and display the sum of the three numbers in R2 on axis A, R4 on axis B, and R7 on the local axis.
Example 8: RX5=RX5-1	Decrement the value in register 5 on the currently selected auxiliary axis.
Example 9: R15=RX(R10)C	Store the contents of the axis C register pointed to by local R10 into local R15.

HARDWARE COUNTING CAPABILITIES

The optional Encoder Backup Compensator,EBC-900 daughter board, provides access to five AM-9513 programmable Up/Down hardware counters[16]. The functions and commands used to access this capability from MPL are described below.

It should be noted that this documentation does not try to describe the features of the AM-9513 counters, which are extensive. Due to the complexity (and capability) of the AM-9513 counters, it is recommended that the user consult the manual listed in the footnote before attempting to use these commands.

[16] Refer to the Advanced Micro Devices "Am9513A/Am9513 System Timing Controller Technical Manual" for detailed information on the operation of the counters.

RC - Counter Command

Purpose: **Allow the specification, loading, arming, and disarming of the five general purpose hardware counters on the optional EBC-900 daughter board.**

Syntax 1: **RC [#<counter>#] [#[<cmd>[<<value>>][.]][<sync>]#] <cr>**
Syntax 2: **RC [#<counter>#] <display>**

Usage 1:	R<<reg>> =**RCE[?]**	Contents of EBC mode register
Usage 2:	R<<reg>> =**RC<counter>H[?]**	Contents of <counter> Hold register
Usage 3:	R<<reg>> =**RC<counter>[L][?]**	Contents of <counter> Load register
Usage 4:	R<<reg>> =**RC<counter>M[?]**	Contents of <counter> Mode register
Usage 5:	R<<reg>> =**RC<counter>O[?]**	State of <counter> Output
Usage 6:	R<<reg>> =**RC<counter>S[?]**	Contents of <counter> via Hold reg.
Usage 7:	R<<reg>> =**RC<counter>!**	Contents of <counter> via Hold reg.

where: <counter> The counter number, 1 through 5, to be used; This also selects the output and the gate to be used as described in Table 1. Some counter commands (A, C, D, S) allow more than one counter to be selected while others (H, L, M, O) use the first counter number specified.

Table 1 - Output and Gate Assignments

Counter	Output	Gate
1	(not used)	"EBCR" input on EBC
2	"SNSROUT" to PMC	"SENSIN" input on PMC
3	"ECROUT" to PMC	"ENCR" input on PMC
4*	To counter source S3	Net EBC forward counts
5	EBC "TIMEOUT1"output	EBC flip flop

* If the Encoder Backup Compensator (EBC) function is being used on the EBC board, **counter 4 must not be used** by the **RC** command. This would cause unpredictable results.

<cmd> **A** **A**rm counters. Arms all of the selected counters simultaneously. If <value> is specified, it is stored in the first specified counter's Load register, and then all counters are loaded before arming.

C Load **C**ounters. Loads all of the selected counters simultaneously with the contents of their corresponding Load registers. If <value> is specified, it is stored in the first specified counter's Load register before the counters are loaded.

D **D**isarm counters. Disarms all of the selected counters simultaneously. No <value> is specified for this command.

E Select **E**BC mode. Stores <value> in the EBC mode register. <value> must be a hexadecimal number as described in Table 2. If <value> is not specified, it defaults to zero.

E? display the contents of the EBC mode register

Table 2 - EBC Mode Register Bit Assignments

Bit 7: EBC Reset
0 = release reset (allow normal EBC function)
1 = hold reset on (don't allow EBC to count)

Bit 6: Sensor Source Select
0 = select PMC "SENSIN" input
1 = select EBC "SNSROUT" (counter 2 output)

Bit 5: Encoder Reference Source Select
0 = select PMC "ENCR" input
1 = select EBC "ECROUT" (counter 3 output)

Bit 4: EBC Flip Flop Clock

Bit 3: reserved

Bit 2: reserved

Bit 1: EBC General-Purpose "OUT2" Output

Bit 0: EBC General-Purpose "OUT1" Output

H Store to counter **H**old Register. Stores <value> in the first specified counter's Hold Register. If <value> is not specified, it defaults to zero.

H? display the contents of the Hold Register

L Store to counter **L**oad Register. Stores <value> in the first specified counter's Load Register. If <value> is not specified, it defaults to zero.

L? display the contents of the Load Register

M Store to Counter **M**ode Register. Stores <value> in the first specified counter's Mode Register. <value> must be a hexadecimal number as described in Table 3. If <value> is not specified, it defaults to zero.

M? display the contents of the Counter Mode Register

Table 3 - Counter Mode Register Bit Assignments

Bits 15-13: Gating Control (n = first specified <counter>)
(see **Table 1**, Gate Assignments)

- 000 = no gating
- 001 = active high terminal count from counter n-1
- 010 = active high level gate n+1
- 011 = active high level gate n-1
- 100 = active high level gate n
- 101 = active low level gate n
- 110 = active high edge gate n
- 111 = active low edge gate n

Bit 12: Source Edge

- 0 = count on rising edge
- 1 = count on falling edge

Bits 11-8: Count Source Selection

- 0000 = 0 = terminal count from previous counter
- 0001 = 1 = PMC command pulses
- 0010 = 2 = EBC "SNSR2" sensor input
- 0011 = 3 = output from counter 4
- 0100 = 4 = EBC counter overflow
- 0101 = 5 = PMC velocity range frequency x 4
- 0110 = 6 = EBC "EBCR" encoder reference input
- 0111 = 7 = PMC "SENSIN" sensor input
- 1000 = 8 = PMC "ENCR" encoder reference input
- 1001 = 9 = EBC net forward encoder counts
- 1010 = A = EBC flip flop
- 1011 = B = 3.072 MHz clock
- 1100 = C = 192.0 kHz clock
- 1101 = D = 12.00 kHz clock
- 1110 = E = 750.0 Hz clock
- 1111 = F = 46.875 Hz clock

Bit 7: Special Gate

- 0 = disable special gate
- 1 = enable special gate

Bit 6: Reload Source

- 0 = reload from Load Register
- 1 = reload from Load or Hold Register

Bit 5: Count Control

- 0 = count once
- 1 = count repetitively

Bit 4: Count Base

- 0 = binary count
- 1 = BCD count

Bit 3: Count Direction

- 0 = count down
- 1 = count up

Bits 2-0: Output Control (see **Table 1**, Output Assignments)

- 000 = inactive, output low
- 001 = active high terminal count pulse
- 010 = toggle on terminal count pulse
- 011 = (illegal)
- 100 = inactive, output high impedance
- 101 = active low terminal count pulse
- 110 = (illegal)
- 111 = (illegal)

<cmd> cont'd

O Set or clear **O**utput. If <value> is non-zero, the first counter's output is set (turned on), otherwise it is cleared (turned off).

O? display the state of the selected outputs

S **S**ave counters. Saves all of the counters simultaneously in their corresponding Hold Registers. No <value> is specified for this command.

S? display the new contents of the first counter's Hold Register

<value> An optional argument to <cmd>. This argument is in units of the current input mode of the PMC unless otherwise specified. If the value is a hexadecimal number and the next command could also be interpreted as a hexadecimal digit (A - F), then a decimal point (.) should be used to signify the end of the hexadecimal number.

. Separate a hexadecimal argument from the following command if the following command could also be interpreted as a hexadecimal digit (A - F).

<display> **?** Display the contents of the counter Load Register.

! Display the current count. This terminator will cause the current count to be saved in the Hold Register, and previous contents will be destroyed.

% Repeatedly display !

Example 1: RC2_M0805.E40.A20<cr> Configure counter 2 to count PMC encoder reference pulses, counting down from 20, and assert the PMC sensor input when the count reaches zero. The '_' character is optional, but the '.' is required to separate the **E** and the **A** commands from the previous command's hexadecimal argument.

Example 2: RCA<cr> Re-arm the counter to repeat the last specified sequence.

MPL MATH ERROR CODES

Special Non-Error Codes:

01 **Input Latch Detected:** An input latch was detected by the **RS** command.

Math Error Codes:

10 **Division by Zero Error:** An attempt was made to divide a number by zero in an arithmetic expression.

11 **Overflow Error:** Overflow occurred during an arithmetic operation.

12 **Illegal Expression Syntax:** A binary operator or closing parenthesis was expected.

13 **Illegal Register Number:** The R register designator character was followed by a character other than register number 0 through 9.

14 **Value Expected:** A value such as an integer, register, unary operator, function name, or an expression in parenthesis was expected.

15 **Expression Too Complex:** Too many levels of parentheses and/or operations with different precedence have been attempted. Break up the expression into several simpler ones using registers.

I/O Error Codes:

20 **Unassigned I/O Device Number:** The specified I/O device number is not currently assigned to any device.

21 **Device Specifier Syntax Error:** Syntax error in <dev spec> portion of a **?**, **RI**, or **RT** command.

22 **Input/Output Operation Aborted by User:** The user pressed the Escape key while the system was waiting to receive or send a character through an I/O device.

23 **Invalid character in input field.** A character that is not valid for the input format specified was entered by the user.

24 **Input Value Too Large.** A number entered by the user was too large to fit in a 32-bit register.

25 **I/O Device Timeout:** A character was not sent to or received from the I/O device for the number of milliseconds specified in <dev spec> for that device.

26 **System Status Polling Disabled on Slave:** An **RX** command failed because system status polling was disabled on the slave PMC.

27 **Invalid Slave Response:** An **RX** command was unsuccessful because the slave responded with an invalid character.

Miscellaneous Errors:

30 **IF without matching ENDIF:** A **R?<expr>** command was encountered without a matching **R?** command.

31 **Extended Velocity Out of Range:** The specified extended velocity is out of the allowable range.

32 **Invalid Counter Number:** Counter number specified is outside range of 1 to 5 (hardware counters available on the optional EBC-900).

33 **Invalid Counter Command:** The counter command letter given in the **RC** command is invalid. It must be one of the command letters described under the <command> argument of the **RC** command description. (for hardware counters available on the optional EBC-900)

APPENDIX A: I/O DEVICE DRIVERS

DEVICE 1 - MAIN SERIAL PORT TERMINAL DEVICE

This device driver assumes that there is an IBM-PC compatible running our motion control development software, or a dumb terminal serial device, connected to the main serial port on the PMC. All output from the **?** command is sent directly to the terminal a character at a time. Each line of output is terminated by a carriage return and a line feed. When the **RI** command is executed, it waits for the user to type a line of up to 32 characters on the terminal. All characters typed by the user are entered into the input buffer and echoed back to the terminal, except for the following special characters:

Return [CR] (0Dh) - Enters the line currently being typed, and resumes execution of the RI command.
Back Space [BS] (08h) & **Delete [DEL] (7Fh)** - Deletes the previous character and moves the cursor back one space.
Ctrl/X [CAN] (18h) - Erases the entire line being edited, and starts over.
Escape [ESC] (1Bh) - Aborts the program currently being executed and generates an "I/O Operation Aborted by User" error. (Error code E7.)

With this device driver, MPL execution is suspended until the **RI** command is completed. The **RI Command** will be completed only when character entry is done and the **Return** key is pressed to enter the line.

The baud rate of the main serial port is configured by the **SB** command. The data format is always 8 data bits, 1 stop bit, no parity.

DEVICE 2 - AUXILIARY SERIAL PORT TERMINAL

This device driver is identical to Device 1 with the following exceptions:

1. It uses the auxiliary serial port on the EIO daughter board instead of the main serial port.

2. **Escape [ESC] (1Bh)** performs the same function as **Ctrl/X [CAN] (18h)** while entering a line, and does not abort the MPL program.

3. The baud rate of the auxiliary serial port is configured by the **RB** command instead of the **SB** command. The data format is always 8 data bits, 1 stop bit, no parity.

With this device driver, MPL execution is suspended until the **RI** command is completed. The **RI Command** will be completed only when character entry is done and the **Return** key is pressed to enter the line.

DEVICE 3 - MAIN SERIAL PORT ITM-270 / ITM-27 TERMINAL

This device driver supports both the ITM-270 and the ITM-27 Industrial Numeric Keyboard & Display units in a multi-drop polled mode. At the beginning of each output line that is sent to the terminal, the device address is sent. The address is a two-digit ASCII number from 01 to 63 which is configurable via the device specifier. Following the address, all output from the **?** command is sent directly to the terminal a character at a time. Each line of output is terminated by a single carriage return.

When the **RI** command is executed, it sends a **poll** command to the terminal to ask for the contents of the input buffer, and then immediately receives the contents of the buffer. If for some reason the terminal does not send the contents of the input buffer, a timeout occurs and an empty input buffer is returned -- the driver will send a **retransmit** command to the terminal on the next call to the **RI** command. The **poll** and **retransmit** commands are different between the ITM-27 and ITM-270 terminals, and the device specifier is used to select the appropriate protocol.

Device Specifier Syntax: **[<<address>>] [<type>]**

where: <address> a decimal number from 1 to 63 representing the address to be sent to the terminal at the beginning of each line. Defaults to 1.

<type> A selects the ITM-270 polling protocol (default).
B selects the ITM-27 polling protocol.

The baud rate of the main serial port is configured by the **SB** command. The data format is always 8 data bits, 1 stop bit, no parity.

DEVICE 4 - AUXILIARY SERIAL PORT ITM-270 / ITM-27 TERMINAL

This device driver is identical to Device 3 with the following exceptions:

1. Uses auxiliary serial port on the EIO daughter board instead of the main serial port.

2. The baud rate of the auxiliary serial port is configured by the **RB** command instead of the **SB** command. The data format is always 8 data bits, 1 stop bit, no parity.

OUTPUT DEVICE 5 - NRO-066 NUMERIC READOUT

This device driver is designed to interface to the NRO-066 Six Digit Numeric Readout. This readout is primarily a numeric display, but can display all of the following characters: "-.0123456789EHILOPS" and space (all letters are upper-case). The output is buffered internally and the display is not updated until the end of the line. The display time is 4 milliseconds per character -- 24 milliseconds to update the entire display. Up to four displays can be connected at one time, using the optional display select outputs (EIO-14 & EIO-15).

Device Specifier Syntax: **<select>** A number from 1 to 4 representing the value to be placed on the display select outputs (EIO-14 & EIO-15) when the display is updated. If this number is zero (0), the display select outputs are not controlled, and can be used for other purposes.

EIO to NRO-066 Display Connections

EIO JM2 Pin #	& Name	Display Pin #	& Name
1	EIO-23	8	DECIMAL POINT
3	EIO-22	7	DIGIT SELECT 4
5	EIO-21	6	DIGIT SELECT 2
7	EIO-20	5	DIGIT SELECT 1
9	EIO-19	4	BCD 8 DATA
11	EIO-18	3	BCD 4 DATA
13	EIO-17	2	BCD 2 DATA
15	EIO-16	1	BCD 1 DATA
17	EIO-15 (optional)	10	DISPLAY SELECT 2
19	EIO-14 (optional)	9	DISPLAY SELECT 1
49	+5V	11	DC POWER +
EVEN	DGND	12	DC GROUND

Note: Before using the BCD display driver the digital I/O ports must be configured properly. Outputs EIO-16 through EIO-23 must be configured as outputs by clearing bit 4 of the flags register (RF10/0), and then cycling power. If the display select outputs are used, outputs EIO-8 through EIO-15 must also be configured as outputs by clearing bit 5 of the flags register (RF20/0), and then cycling power.

INPUT DEVICE 5 - THUMBWHEEL SWITCHES

The thumbwheel switch driver reads the current setting of a bank of thumbwheel switches, converts each decade to an ASCII character (0-9, A-F) and returns the result in the input buffer. The **RT** command can then interpret the characters in the input buffer as either a decimal number or a hexadecimal number, depending on the application.

The thumbwheel switches are configured such that each thumbwheel switch input (each digit) is connected to a unique select line from the PMC (or EIO). The thumbwheel switch BCD outputs are connected to a common bus which is connected to four PMC (or EIO) inputs. The select lines are each strobed low, in sequence, starting with the most significant digit of the number, and the data is stored in the input buffer. The device specifier determines which outputs are used as select lines to strobe the individual digits, and which are used as BCD data inputs.

The configuration parameters in the <spec> are saved in internal registers. There is one set of internal registers for each <id> number (a total of five).

Before using the thumbwheel switch driver the digital I/O ports must be configured properly. If the EIO inputs are used for the BCD data, they must be configured as inputs by setting bit 6 of the flags register (RF40). If the EIO outputs are used as select lines, they must be configured as outputs by clearing bits 5 and/or 4 of the flags register (RF30/0), and then cycling power.

APPENDIX B: ASCII TABLE

Each character that the PMC can print has a unique number associated with it. The number that represents each character is defined by the **American Standard Code for Information Interchange** (abbreviated **ASCII**). The table below lists all of the printable and non-printable control characters along with their corresponding number (in both decimal and hexadecimal).

Dec	Hex	Char	Key	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	Ctrl-@	32	20	SPACE	64	40	@	96	60	`
1	01	SOH	Ctrl-A	33	21	!	65	41	A	97	61	a
2	02	STX	Ctrl-B	34	22	"	66	42	B	98	62	b
3	03	ETX	Ctrl-C	35	23	#	67	43	C	99	63	c
4	04	EOT	Ctrl-D	36	24	$	68	44	D	100	64	d
5	05	ENQ	Ctrl-E	37	25	%	69	45	E	101	65	e
6	06	ACK	Ctrl-F	38	26	&	70	46	F	102	66	f
7	07	BEL	Ctrl-G	39	27	'	71	47	G	103	67	g
8	08	BS	Ctrl-H	40	28	(	72	48	H	104	68	h
9	09	HT	Ctrl-I	41	29	)	73	49	I	105	69	i
10	0A	LF	Ctrl-J	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	Ctrl-K	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	Ctrl-L	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	Ctrl-M	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	Ctrl-N	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	Ctrl-O	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	Ctrl-P	48	30	0	80	50	P	112	70	p
17	11	DC1	Ctrl-Q	49	31	1	81	51	Q	113	71	q
18	12	DC2	Ctrl-R	50	32	2	82	52	R	114	72	r
19	13	DC3	Ctrl-S	51	33	3	83	53	S	115	73	s
20	14	DC4	Ctrl-T	52	34	4	84	54	T	116	74	t
21	15	NAK	Ctrl-U	53	35	5	85	55	U	117	75	u
22	16	SYN	Ctrl-V	54	36	6	86	56	V	118	76	v
23	17	ETB	Ctrl-W	55	37	7	87	57	W	119	77	w
24	18	CAN	Ctrl-X	56	38	8	88	58	X	120	78	x
25	19	EM	Ctrl-Y	57	39	9	89	59	Y	121	79	y
26	1A	SUB	Ctrl-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	Ctrl-[	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	Ctrl-\	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	Ctrl-]	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	Ctrl-^	62	3E	>	94	5E	^	126	7E	~
31	1F	US	Ctrl-_	63	3F	?	95	5F	_	127	7F	DEL