A100/200 Series
SMALL INDUSTRIAL ROBOT SYSTEM

RAPL-II PROGRAMMING MANUAL

Manual Order Number: UME-09-015

830 Harrington Court, Burlington, Ontario, L7N 3N4
CANADA

REV.	REVISION HISTORY	DATE
-001	Original Issue	12/85
-002	RAPL V1.2	2/86
-003	Version 3.30	9/86
-004	Version 3.4.5	2/87
-005	Version 3.5.0	6/87
-006	Version 4.10	2/88
-007	RAPL-II	8/89
-008	M1B/M2B brake support	2/90
-009	Support for A100/200 series	10/90

Additional copies of this manual, or other CRS Plus literature, may be obtained from:

CRS Plus Inc.
P.O. Box 163 Stn 'A'
Burlington, Ontario,
CANADA.
L7N 3N4

Telephone: (416) 639-0086
Facsimile: (416) 639-4248

This manual provides general programming information for the CRS Plus RAPL-II programming language. Additional information is available in the following documents:

* A100/200 Series Small Industrial Robot System - Tutorial Manual

* A100/200 Series Small Industrial Robot System - Technical Manual

* A100/200 Series Small Industrial Robot System - IBM-PC Host Interface (Robcomm-II)

TABLE OF CONTENTS:

FIGURES

TABLES

TABLE OF CONTENTS (Continued)

CHAPTER ONE - OVERVIEW

1-1 INTRODUCTION

This chapter is designed to give the programmer a quick overview of RAPL-II and the programming environment. It should give the programmer an idea of: how RAPL-II command lines are structured, the RAPL-II operating system, data types and their naming conventions and the different operators (logical, mathematical and relational).

1-2 RAPL-II DESCRIPTION

The CRS Plus family of industrial robots use the CRS Plus proprietary language known as RAPL-II. RAPL-II (Robotic Automation Programming Language - II) is an automation-oriented, line-structured language, designed to facilitate the design of robot system applications.

RAPL-II uses "english-like" commands to provide a user friendly interface for the operator. Features of the RAPL-II language include efficient coding structures to optimize memory usage, alternate command identifiers, and advanced mathematical expressions.

Experienced programmers may find that RAPL-II is similar to BASIC in many ways. Users of C will notice that certain conventions from C are also used.

1-3 NOTATIONAL CONVENTIONS USED IN MANUAL

NOTATION	DESCRIPTION
BOLD PRINT	Indicates a Command in RAPL-II. Also used in command line examples to indicate the characters which must be typed by the operator to implement the command.
VAR_NAME	Refers to a variable name (sometimes as Var_Name).
PRG_NAME	Refers to a program name (sometimes as Prg_Name).
LOC_NAME	Refers to a Cartesian Point location name (sometimes as Loc_Name).
#LOC_NME	Refers to a Precision Point name (sometimes as #Loc_Nme).
STR_NUM	Refers to the string number (1 - 4) (sometimes as &n).
CHAR_INDEX	Refers to the character index within a string.
< >	Indicates REQUIRED arguments or parameters.
[ ]	Indicates OPTIONAL arguments or parameters.
[,...] <,...>	Several arguments may be entered, but each argument must be separated by a comma.
|	Separates options within [] or <> brackets.
	Comma separates arguments.
+	High level true (input/output commands) - used in conjunction with digital I/O port commands (refer to **WAIT, IFSIG, OUTPUT**).
	Low level true (input/output commands) - used in conjunction with digital I/O port commands (refer to **WAIT, IFSIG, OUTPUT**).
<cr>	Indicates a carriage return.
&	Indicates that a string number follows (numbers 1-4).
^	Indicates an Array-Type location name.
<ABORT>	Indicates pressing <Ctrl-C> or <Ctrl-X> at terminal device or the ABORT button on the Teach Pendant.
[,S]	Straight Line argument.

1-4 RAPL-II COMMAND FORMAT

A command line is an instruction to the robot controller to execute a function. A command line starts after a prompt and is terminated by a <cr>. In RAPL-II there are two ways of giving the controller an instruction: (1) from a running program and (2) an immediately entered and executed command from the keyboard.

A command line may or may not start with a line number. If, when entered at the terminal device, a command line starts with a line number, the command is not executed immediately, but is stored in a program for later execution. Command lines within programs generally have line numbers, but in RAPL-II it is not necessary. However line numbers must be used when in the Program Line Entry mode or with the Edit mode (refer to chapter 1-6 for more information about these modes). If a line number is placed at the beginning of a command line, a space must separate the line number and the command.

A legal RAPL-II command is a necessary part of any command line. The command must be the first item in the command line, unless a line number and space is used to insert the line into a program. A list of all RAPL-II the commands with a full description of each can be found in chapter 4.

Following the command, a command line may contain several variable names, arguments and/or parameters depending on the actual command (refer to chapter 4). Variable names, arguments or parameters, when required, must have a space between them and the command. Multiple arguments must be separated by commas or spaces. Commas are recommended for improved program readability.

Refer to the following page for an example of a typical RAPL-II command. Also carefully read the following important notes:

** *If a line number is entered, there must be a space left between the line number and command. There should also be a space left between the command and any additional parameters, arguments, or location names.*

** *Only UPPERCASE characters are permitted (except within string definition or comments).*

** *Maximum line length is 128 characters.*

** *Valid line numbers are from 1 to 65536. Entering a line number larger than 65536 will result in an entry equivalent to:([Line#] mod 65536) even though it may be displayed as entered. Entering a line number of 0 will result in an error.*

1-4 RAPL-II COMMAND FORMAT (Continued)

Example of a typical RAPL-II command:

APPRO command

[Line#] **APPRO <LOC_NAME>,<DISTANCE>**[,S]**<cr>**

ITEM	DESCRIPTION
[Line#]	optional (required for Program Line Entry and Edit modes)
APPRO	required parameter (command name)
<LOC_NAME>	required location name
,	comma must separate all arguments
<DISTANCE>	required argument
[,S]	optional argument
<cr>	required carriage return

1-5 RAPL-II OPERATING ENVIRONMENT

Three levels of control are available with the RAPL-II operating system (refer to FIGURE 1-1). The levels are Immediate {I}, Program {P}, and Manual {M}:

{I}	Immediate	Used for the execution of single commands
{M}	Manual	Used for manual manipulation of robot arm
{P}	Program	Used for the execution of user programs

Upon powering up, unless an Autostart is being executed (refer to appendix D), RAPL-II automatically defaults to the Immediate mode.

Immediate {I}:

The Immediate mode allows the user to enter commands that will be executed upon completion of the command line. Additional lines may also be added to a program from Immediate mode (refer to Program Line Entry mode, section 1-6). From within the Immediate mode, use of the RAPL-II syntax builder (known as the Help feature) is available (refer to section 1-6). The Immediate mode also allows the user to enter the Line Editor (refer to section 2-6 for more information).

Manual {M}:

The Manual mode allows the user to move the robot with the teach pendant or with commands from the keyboard. There are actually two ways to manipulate the arm when in Manual mode: Joint Manual mode and Cylindrical Manual mode (refer to section 2-8). As with Immediate mode, Manual mode allows the user to enter the Edit mode, the Program Line Entry mode and use the Help feature. Note that while in Manual mode, if any motion command is executed from the keyboard, control will be returned to Immediate mode.

Program {P}:

The Program mode allows users to execute programs stored in the user memory of the controller. RAPL-II uses an interpreter, so errors in programs will only be detected during the execution of a program. The user may tokenize (refer to section 1-7) his or her programs to "speed up" execution. The tokenized format allows the interpreter to read instructions faster.

System Prompts:

RAPL-II uses a prompt to inform the user that the system is waiting for input. The type of prompt identifies the mode in which the system is operating. WHEN OPERATING IN PROGRAM MODE, NO PROMPT IS SEEN.

>*	Edit mode	Enter from Immediate or Manual mode
>>	Help or PLE[1] mode	Immediate mode
J>	Help or PLE mode	Joint Manual mode
C>	Help or PLE mode	Cylindrical Manual mode

[1] PLE - Program Line Entry mode (see chapter 1-6)

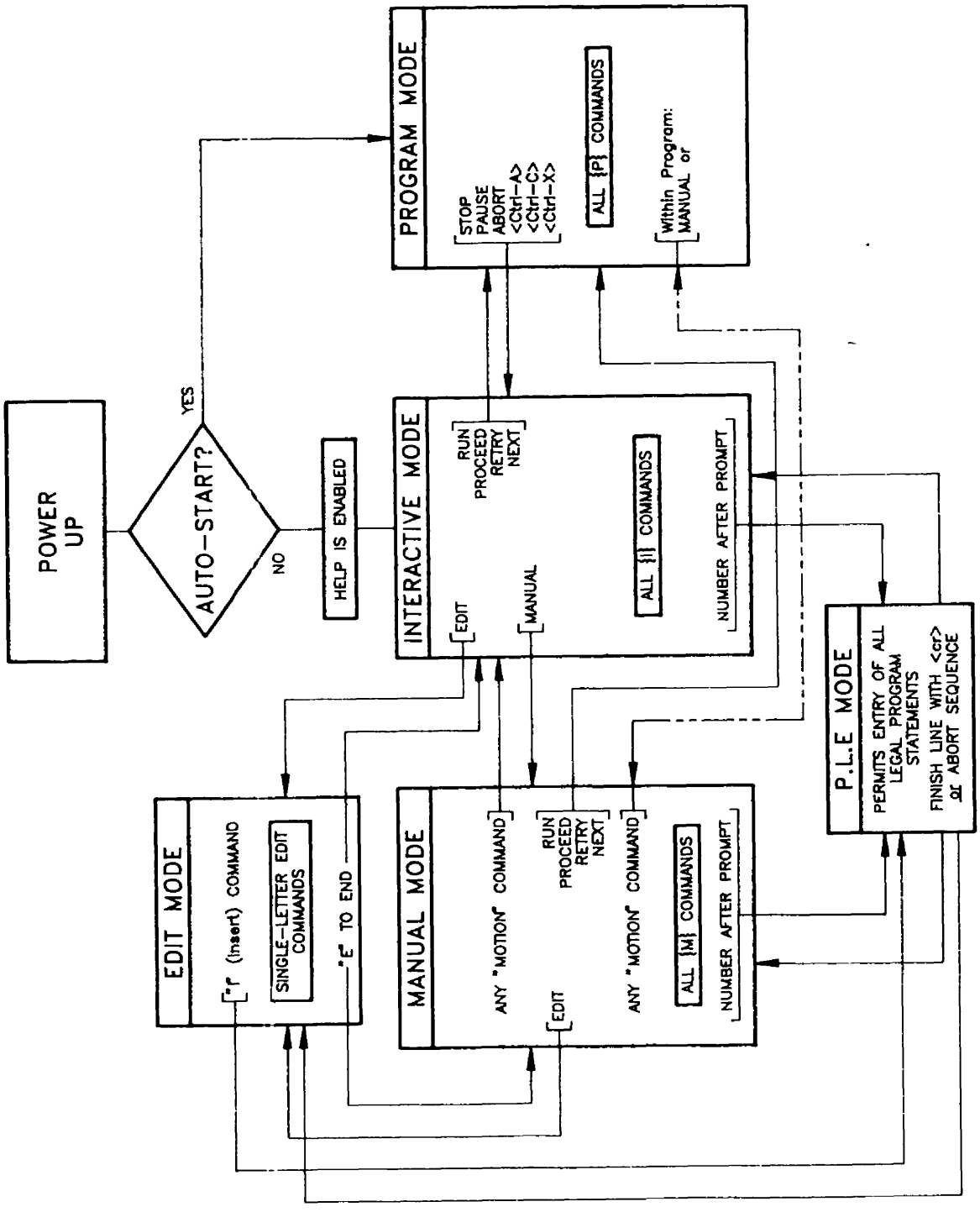


FIGURE 1-1 Operating System Structure

1-6 RAPL-II ENVIRONMENT PART II

Program Line Entry (PLE) mode:

A program line is a command line stored in a RAPL-II program for execution in the program mode. Program lines can be entered in one of three ways: inserted in the Edit mode, loaded from a host computer or entered from PLE mode.

A programmer enters PLE mode whenever a command line begins with a numeric character while in Immediate or Manual mode. While in the Edit or PLE mode a line number is required to permit RAPL-II to enter the command into the program currently being edited in the correct order. The following apply to PLE mode only:

** *Program line entry terminates with the <cr>, but the command is executed only at program run-time.*

** *DELETING THE NUMERIC CHARACTER WILL NOT TERMINATE THE PLE MODE. While starting a command line with a numeric character starts PLE mode, deleting the numeric characters will not terminate PLE mode. To exit PLE mode without entering the line into the program an <ABORT> sequence must be used.*

** *When entering a program line, any part of the command line can be edited before the <cr>, which enters the line into program memory.*

** *NO SYNTAX CHECK IS PERFORMED ON A LINE BEFORE IT IS INSERTED. However, when PLE mode is exited with a <cr>, RAPL-II checks to see that a valid line number was entered.*

Help feature (syntax builder):

The Help feature is intended to aid inexperienced users of the RAPL-II system. This assistance takes the form of a "syntax builder". The Help feature is automatically enabled upon system power up. In addition the Help feature can be turned on using **HELP** or **ENABLE HELP** commands. To turn-off the Help feature use the **DISABLE HELP** or **NOHELP** commands.

As the command is formulated at the terminal, the operator need only enter enough characters to make the command unique (when Help feature is enabled). The controller then finishes the command for the operator. The number of characters needed depends on whether other commands use the same first letters. Take for example the two commands **DELAY** and **DELETE**. The operator would have to type four letters of the command before the syntax builder would be able to identify which command it was, and complete it.

The RAPL-II Help feature also prompts the user for further information needed to complete the command line (once the command itself is complete). Three typical command line entries are seen below. In these examples the user input is **bold** while the RAPL-II response is in the normal type:

```
>>SHIFT   <LOCATION>  :  PICK, <DX,DY,DZ> :1,0,-.5<cr>
>>APPRO   <LOCATION>, PICK, <BY A DISTANCE OF>: 2.5<cr>
>>MOVE    <TO DESTINATION> : PICK, <S> FOR STRAIGHT: S<cr>
```

1-6 RAPL-II ENVIRONMENT PART II (continued)

Help feature (continued):

Notice that RAPL-II prompts for the correct syntax of the command, including spaces and commas if needed. The following apply when using the Help feature:

** *RAPL-II executes the command line as soon as it receives the <cr>.*

** *Changing or editing of a command line is permitted only within the current item (the command and each argument are separate items). Once an item is terminated (by a space, comma or <cr>), it can no longer be changed. The only editing allowed in a command line, is to back-space over information not yet terminated.*

** *A character is deleted when the 7F hex (rubout) character is received from the terminal device.*

** *At any time before receiving the carriage return, a command line can be aborted with an <ABORT> sequence.*

Edit mode:

RAPL-II provides the programmer with a line editing feature that can be used to create programs or modify existing ones. The Edit mode can be entered from Manual or Immediate modes. To enter the Edit mode use the **EDIT** command. Once all program editing and developing is complete, the **DISABLE EDIT** command prevents further changes to the program from the terminal device. This prevents the user from using Edit commands or the PLE mode to alter programs. This feature is useful in preventing unauthorized program editing. It also prevents other computer devices from altering user programs when hooked up to the terminal device. The **ENABLE EDIT** command re-enables the Edit mode and allow the user to once again edit programs. For more information refer to section 2-6.

1-7 TOKENIZED FORMAT

In this format, a number (Token) preceded by a slash is substituted for the english language RAPL-II command. The tokenized format of RAPL-II commands permits program line decoding at a rate of roughly ten times that of the source format. Due to the speed possible in interpreting the number, RAPL-II can determine a tokenized command quicker.

Tokenized format is recommended for all programs once they are thoroughly debugged. The available Robcomm-II software package, for use with IBM PC or compatible computers, contains a function to automatically tokenize a program.

Mixing of tokenized and non-tokenized commands is possible. Programs may be written directly in tokenized form. Unfortunately, this makes program debugging more difficult. The RAPL-II Trace mode allows the user to trace tokenized programs line-by-line during program execution. The tokenized commands are displayed in their full mnemonics at this time. Chapter 4 contains a description of the RAPL-II commands and the command line syntax. Also shown is the equivalent tokenized command format. APPENDIX A contains a short form list of standard RAPL-II commands, their functions and tokens.

1-8 RAPL-II DATA TYPES

RAPL-II supports three kinds of data: variables, locations and strings.

Variables:

In RAPL-II all variables are stored as REAL NUMBERS. RAPL-II allows for up to 24 significant bits of accuracy. Thus numbers with absolute values larger than 2^{24} can result in rounding or chopping errors. A variable uses 14 bytes of memory in RAPL-II 8 bytes for the variable name, 4 bytes for the actual number stored in real format and 2 bytes for the checksum.

Locations:

RAPL-II allows the user to store robot locations in terms of points in space (Cartesian points), or in terms of motor pulses (Precision points). Since there can be up to eight different axes of motion, each location must set enough space aside to account for this possibility. A location takes up 42 bytes of memory: 8 bytes for the name, 4 bytes for each of the eight axes and 2 bytes for the checksum* . Cartesian points are stored as real numbers (4 bytes per real number). Precision points are stored as double-size integer values (4 bytes per axis, for all 8 axes).

NOTE: Refer to chapter 2-3 to see how the robot defines its cartesian coordinate system.

Tables:

It is important to note that RAPL-II stores variables and locations in different sections of memory. Each of these two data types is stored in its own "table" in memory. The size of each table is determined by the programmer using the **ALLOC** command. Each time a variable is stored in memory, its name and value is stored in the variable table. Similarly each time a location is stored its name and coordinates (or motor pulses) are stored in the location table.

Strings:

RAPL-II allows the user to manipulate up to four text strings. The strings can be up to 32 characters in length. Strings may include any ASCII character. A string uses up 32 bytes of memory. For more information on Strings and their functions, refer to chapter 3-8.

* RAPL-II has a series of checks which protects the system against being used when the memory has been damaged. Each area of user memory has its own checksum. A checksum is the sum of all of the byte values in that area of memory. Any time a section of memory is altered, the checksum for that area of memory is recalculated. The value of the checksum is confirmed each time that particular element is used. If the sums do not match then a checksum error will result and activity ceases to avoid damage.

1-9 NAMING CONVENTIONS

RAPL-II uses the same naming convention for variables (Var_Name), locations (Loc_Name), and program names (Prg_Name).

Names can consist of up to eight alphabetical and numeric characters, however, the first character of the name can not be a number. For example:

'A1234567' is legal
'A123' is legal
'1A' is illegal

For improved readability underline characters (_) may be used. For example:

'PICK_1' rather than 'PICK1'

Variable Names:

In addition to standard variable names (i.e.- any name with eight characters starting with a letter), RAPL-II allows the programmer to enter array-type variables. These variables consist of a five character template and a three digit argument. Arrays have the form of the template name followed by the argument surrounded by square brackets.

e.g.- A[1] stored as A____001
COUNT[100] stored as COUNT100

Notice how RAPL-II pads the name with underscore characters if all five characters in the template are not used. Also note that 1 is stored as 001. This is because the argument can have up to three digits in it. This also means that the maximum number of variables per array name is 999. The strength of the array type variable is that the argument can be replaced with variables, expressions or even other arrays (as well as constants).

Location Names:

RAPL-II allows for both Precision point and Cartesian point variables. Precision point locations store the robot position in motor pulse coordinates. Precision point names are identified by the special '#' character located in the first character of the name. Only seven additional characters are then allowed. For example:

'#PICK_1' is a precision point name
'PICK_1' is a cartesian point name

Both location types can co-exist with the same key name (such as PICK_1 and #PICK_1) in the location directory.

Locations (Continued):

Another special form of a location name is the Array-type location. These stored locations have a hybrid name composed of a template name and an index value. Thus if the template is LOCN, then the first stored location is named LOCN_001. The fourth stored location is named LOCN_004. The template can have a maximum of five characters. If the template entered is less than five characters, the remaining character spaces are filled with "under-score" characters:

Template = LOCN, Index = 1	LOCN_001
Template = PT, Index = 7	PT___007
Template = #TRAY, Index = 73	#TRAY073

The last example would be a precision point.

The template and the counter are both defined within the **TEACH** command itself. Refer to section 2-7 for more information on the Teach mode

RAPL-II allows a special notation for Array-type locations. This is accomplished by the **WITH** command. **WITH** uses the template from a group of Array-type locations. Once the **WITH** command has been issued, a programmer can access locations using the ^ character in place of the template.

For example, assume a programmer has taught 10 points using the template ARB. Now the programmer types in:

```
>>WITH ARB
```

Assume that the points have the locations ARB__001, ARB__002, ... ARB__010. To move to location ARB__005 the programmer can type >>**MOVE ARB__005**<cr> or just >>**MOVE ^5**<cr>.

The number of Array-type locations corresponds to the number of Teach type points taught under the corresponding template. Note that no new locations are created with the **WITH** command. This is just an alternative way of addressing previously defined points without having to type out the entire name of the location. The Array-type location also allows the programmer to access the locations through the use of variables. Thus the line **MOVE** ^X is also a legal statement. The robot then moves to the location defined by X. If X were equal to 7, then the robot would move to location ARB__007.

Strings:

Strings are identified by using the '&' character followed by a number (1-4).

e.g.- ! &4 = 'this is the name of string #4'

1-10 MATHEMATICAL OPERATORS, FUNCTIONS AND EXPRESSIONS

Arithmetic Operators

RAPL-II provides the simple arithmetic operators:

+	- addition
-	- subtraction
*	- multiplication
/	- division

Mathematical Functions

In addition, RAPL-II also provides the user with 16 mathematical functions. This includes trigonometric, logarithmic and exponential type functions. It also includes other useful mathematical functions such as absolute values and integer portions of numbers. The following is a list of all of the mathematical functions available in RAPL-II:

SIN(X)	- SIN of angle X in radians
COS(X)	- COS of angle X in radians
TAN(X)	- TAN of angle X in radians
ASIN(X)	- ARCSIN of argument X, radians returned
ACOS(X)	- ARCCOS of argument X, radians returned
ATAN2(A,B)	- ARCTAN(A/B), radians returned
POW(A,B)	- A to the power B
SQRT(X)	- square root of argument
ABS(X)	- absolute value of argument
INT(X)	- returns integer portion of argument
LN(X)	- natural logarithm of argument
LOG(X)	- log base 10 of argument
RAD(X)	- degrees into radians (X is in degrees)
DEG(X)	- radians into degrees (X is in radians)

Logical Functions

There are also four Logical Functions supported by RAPL-II logical OR function, exclusive OR function, logical AND function and the modulus function. All four of these operations are performed on two arguments. The arguments are treated as double word size, 32 bit unsigned values. However because RAPL-II saves all variables as real numbers, only 24 bits are stored. To avoid a loss of significant digits, all variables used in logical statements must be limited to absolute values of less than 2^{24} .

The logical functions are:

AND(x,y)	- logical AND
OR(x,y)	- logical OR
XOR(x,y)	- exclusive OR
MOD(x,y)	- modulus

1-10 MATHEMATICAL OPERATORS, FUNCTIONS AND EXPRESSIONS (Continued)

Location Component Extraction Functions

In addition to the mathematical functions, there are a set of additional RAPL-II functions which are designed to extract component values from a stored location. For example, to shift a location relative to the difference between two reference locations:

```
1100 SHIFT NEWLOC BY (XCOMP(REFLOC1) - XCOMP(REFLOC2)),(YCOMP(REFLOC1) - YCOMP(REFLOC2)),0
```

COMP(NAME,#) - extracts the component from a specified location name (numbers 1-8)
XCOMP(NAME) - x component from specified cartesian location
YCOMP(NAME) - y component from specified cartesian location
ZCOMP(NAME) - z component from specified cartesian location
OCOMP(NAME) - yaw component from specified cartesian location
ACOMP(NAME) - pitch component from specified cartesian location
TCOMP(NAME) - roll component from specified cartesian location

Additional Functions

ANALOG(INPUT#) - This function returns the digital value of the measured voltage from one of the optional analog channels. Analog channel number 9 is used to read the speed dial potentiometer on the Teach pendant. There are 16 optional analog channels (numbered 10 to 24) available with this system when the COMBO/32 card is installed.

For all channels of analog input, accuracy of the converters is one part in 256 (8 bit resolution) for an analog input voltage from 0 to +5 volts. The result will be an integer from 0 to 255 so that each increment represents about a 20 mV increase in voltage.

WGRIP() - This function reads the current position of the fingers of the servo gripper. The value returned is the distance between the fingers expressed in inches or millimetres depending on the setting of the english/metric DIP switch.

Hierarchy of Operators

When evaluating expressions there is a hierarchy of operations that is followed. As seen in the chart below, brackets are always evaluated first, while the logical OR operator is always evaluated last.

Brackets () . . . . . most significant

Intrinsic and Logical Functions*

* /
\+ - . . . . . least significant

* Intrinsic functions include all mathematical and additional functions in RAPL-II

1-10 MATHEMATICAL OPERATORS, FUNCTIONS AND EXPRESSIONS (Continued)

The Parser

The Parser is the part of RAPL-II that evaluates mathematical expressions based on the hierarchy listed above. A mathematical expression can be anything from a simple "x + y", up to a complicated combination of mathematical (and logical) operators and functions. Whenever RAPL-II encounters a mathematical or logical expression the Parser is activated.

The Parser disassembles the expression character by character and builds a reverse Polish expression out of it. When the Parser is finished with the expression, it then evaluates the reverse Polish expression and returns a value to the assigned variable.

Errors in an expression are handled by the parser which has its own error codes. The parser error code is displayed on the screen before the RAPL-II error code which will normally be a bad syntax error if the expression is faulty.

```
e.g.-   ! X = ( A + ( B - C )<cr>
        029 ----------------------------------> PARSER ERROR CODE
        BAD SYNTAX - 014

        (Error code 029 means a missing termination parenthesis)
```

(NOTE: ! is the assignment command)

A complete list of these error codes can be found in Appendix B, section B-2 of this manual.

The Parser allows the user to program up to 8 nesting levels in an expression. [e.g.- this expression has three nesting levels: ! Y = A + (B * (C + D)))]. The number of nesting levels is determined by expression complexity. For example, if the expression is kept simple (only mathematical operators *,/,+,-) then the expression could be nested up to 15 or 16 levels. For a complicated expression the Parser might only be able to handle 8 nesting levels. Thus 8 nesting levels is the minimum number of levels that is guaranteed to work with all functions and operators.

In RAPL-II it is also legal to use to use expressions within the arguments of mathematical functions and logical operators. The following are entirely legal:

```
A = LOG( X + Y / ( 3 + C ))          or,

B = XOR( C + V * SQRT( V ), F * Y )
```

Just consider the expressions within the arguments to be another nested level. Also note that within logical functions, all arguments are converted to integer values before they are evaluated. Fractional values are chopped off the integer portion before the logical operation is performed. This is important since all values in RAPL-II are stored as REAL numbers.

Arrays can also have expressions within their arguments. Thus: A[X+Y*SQRT(A[2])] is perfectly valid. Consider the argument in the array to be another nested level for purposes of expressions.

1-11 RELATIONAL OPERATORS

There are six Relational Operations which can be performed on two arguments. These are used in the IF command.

Relational Operators:

==, EQ	- equal to
> , GT	- greater than
!=, NE	- not equal to
<=, LE	- less than or equal to
< , LT	- less than
>=, GE	- greater than or equal to

2-1 INTRODUCTION

This chapter explains some of the parameters with which RAPL-II is programmed. It defines the robot coordinate system and how the robot uses locations. It explains Manual mode and the Line Editor. This chapter also defines some advanced features of RAPL-II such aS AUTO-START and TOOL coordinate programming.

2-2 PROGRAMMING AIDS

RAPL-II includes a simple line oriented editor for on-line program creation and editing. The user enters the editor with the EDIT command. Refer to section 2-9 for a full description of the editor.

The Program Line Entry mode (see section 1-?) allows faster program development by inserting lines typed directly at the prompt into the currently editing program.

The Help feature (the RAPL-II syntax builder) helps reduce command line entry errors by prompting the user for information and structuring the appropriate format. HELP also reduces typing time by requiring the programmer to type only enough letters of a command to ensure that the command can be uniquely determined. RAPL-II then prompts the user for any further information required. Advanced programmers may find this feature a hinderance and can disable it with the DISABLE HELP command.

In addition to the editor in RAPL-II, CRS has designed a software package called Robcomm-II which runs on personal computers using the MS-DOS® operating system. This package permits programs, locations and variables as defined by RAPL-II to be saved to and loaded from the PC disk. It also includes useful programming features such as a program Tokenizer and Untokenizer and a program renumbering system. It also supports text editors which run in the DOS environment to permit advanced screen editing of RAPL-II programs from disk. It is strongly recommended that RAPL-II system programmers use the Robcomm-II package.

2-3 DEFINING THE WORKSPACE

Joint Coordinate System

The robot joint space for the A100/200 series consists of 5 revolute joints. These are seen in Figure 2-1. The arrow shown indicates the positive (+) direction. Zero for joint 1 is pointing straight forward at the mid-point of travel. Zero for joints 2, 3, and 4 is with the link in question pointing at the horizon. For joint 5, zero is also at the mid-point of travel.

World Coordinate System

Refer to figure 2-2 for illustration of the A100/200 Series world co-ordinate system. Note that 0,0,0 is at the centre of the base on the ground plane. The arrow indicates the positive (+) direction. The positive directions of the cartesian axes are defined by the "Right-Hand Rule".

USING THE FINGERS OF THE RIGHT HAND, POINT THE INDEX FINGER TO INDICATE THE POSITIVE X-AXIS, POINT THE MIDDLE FINGER AT RIGHT ANGLES TO THE PALM TO INDICATE THE POSITIVE Y-AXIS AND POINT THE THUMB UP TO INDICATE THE POSITIVE Z AXIS.

The orientation of the wrist is defined by three angles: yaw (orientation about the z axis), pitch (orientation about the y axis) and roll (orientation about the x axis). The right-hand rule applies in determining the direction of these angles as well.

USING THE RIGHT HAND, POINT THE THUMB ALONG THE POSITIVE AXIS; THE FINGERS WILL CURL IN THE DIRECTION OF POSITIVE ROTATION.

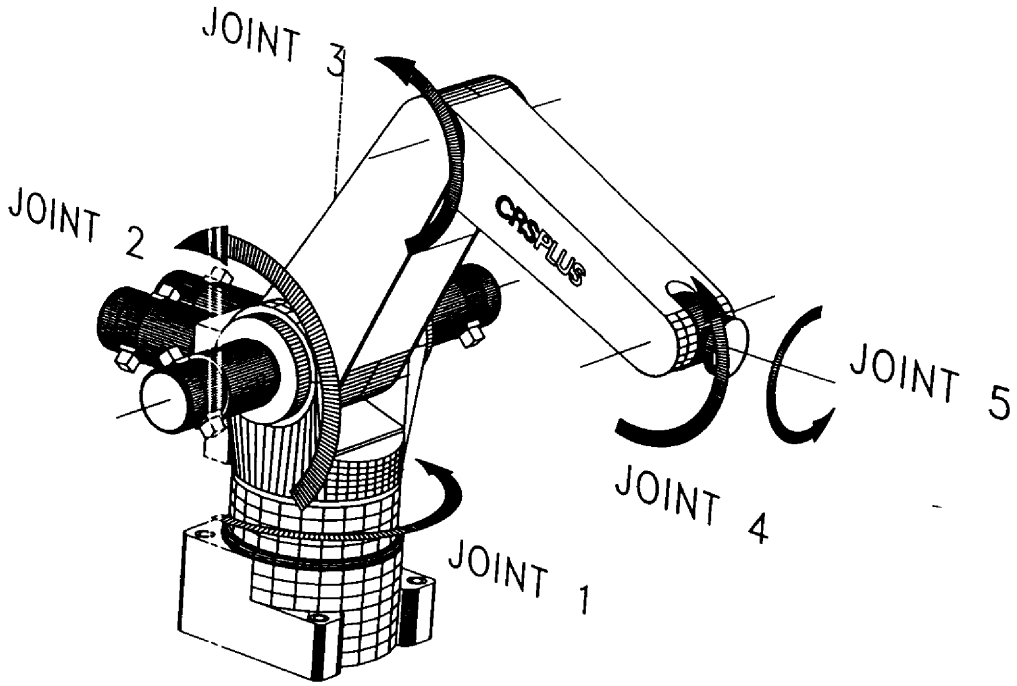


FIGURE 2-1 Defining the workspace - Joint Co-ordinates

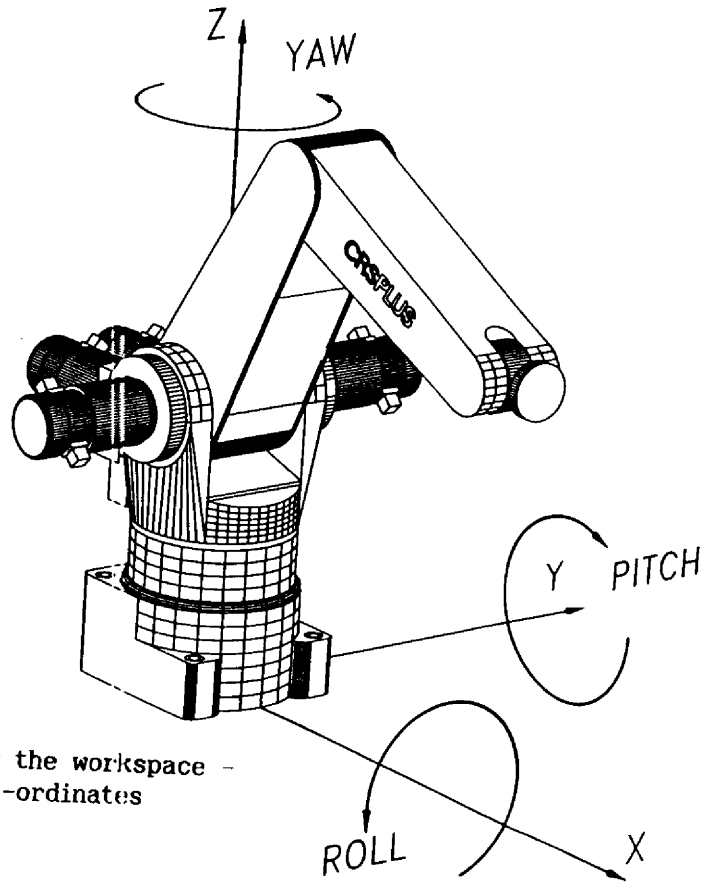


FIGURE 2-2 Defining the workspace - World Co-ordinates

2-4 TOOL COORDINATE PROGRAMMING

Introduction

Because the geometry of the tool or gripper which the robot carries can vary, RAPL-II includes a command to permit the programmer to define the location of the tool tip (the Tool Centre Point or TCP) relative to the end of the robot arm. This is the **TOOL** command (refer to chapter 4). This command calls a transform which is stored in the location table which defines the offset of the end of the tool being used in relation to the origin of the tool coordinate system. In the default case, the tool transform has zero values for all coordinates and thus the TCP is at the origin of the tool coordinate system (the centre of the tool flange with the tool axis normal to the flange - see Figure 2-3). The current transform can be displayed at any time using the **STATUS** or the **TOOL** command.

Function of Tool Coordinate Programming

When any cartesian robot motion is performed, the final pose of the arm will be such that the Tool Centre Point is located at the desired cartesian location and the tool axis is aligned with the location axis (as defined by the yaw, pitch and roll coordinates of the location). In the default condition, the centre of the tool flange will be positioned at the location with the flange perpendicular to the location axis. If a non-null tool has been defined, the new TCP will be positioned at the location with the new tool axis lined up with the location axis.

The **APPRO** and **DEPART** commands move the arm so that the final pose positions the TCP a defined distance away from the reference cartesian location along the tool axis. The ALIGN command positions the arm so that the TCP remains at its current location but the tool axis is aligned with the nearest major axis of the world coordinate system.

In all cases of controlled path motion, straight line, **CPATH**, **GOPATH**, and **VIA**, the TCP will move along the defined path. If the **CARTVEL** flag is **ENABLED**, the speed of the arm will be such that the TCP will move at a constant CARTesian VELocity.

The TOOL command allows the RAPL-II user to specify the position of the centre of interest (Tool Centre Point or TCP) on whatever tool the robot is carrying. The TCP is then the position to which all location measurements are made. Straight line moves for instance, will keep the TCP moving linearly at a constant speed, the ALIGN command will keep the TCP fixed in space while re-aligning the wrist pitch to a major axis, and any DEPART or APPRO commands use the tool axis as the direction of the motion relative to the end point.

The Tool Coordinate System

The tool coordinate system is an orthogonal system with its origin at the centre of the outer surface of the tool flange, flush with its outer surface (refer to FIGURE 2-3). The tool coordinate system moves with the tool flange.

The tool X-axis is normal to the tool flange at the origin. The Y- and Z- axes are orthogonal to the X- and are aligned with the world Y- and Z- axes when the robot is at the READY position (see FIGURE 2-3). The tool coordinate angles are defined by three angles: YAW (orientation about the tool Z-axis), PITCH (orientation about the tool Y-axis) and ROLL (orientation about the tool X-axis). As with the world coordinate system, the RIGHT-HAND-RULE applies to determine the direction of the positive axes and angles.

2-4 TOOL COORDINATE PROGRAMMING (Continued)

As an example, a transform with coordinates 2,0,0,0,0,0 would indicate that the TCP is located on the X-axis, 2 units (inches or millimetres) away from the face of the tool flange.

The angular components (YAW, PITCH and ROLL) of the tool transform determine the direction of the tool axis. The origin of the tool axis is always the TCP. The direction of the axis is determined by adding the angles in order of YAW PITCH then ROLL according to the directions shown in FIGURE 2-3.

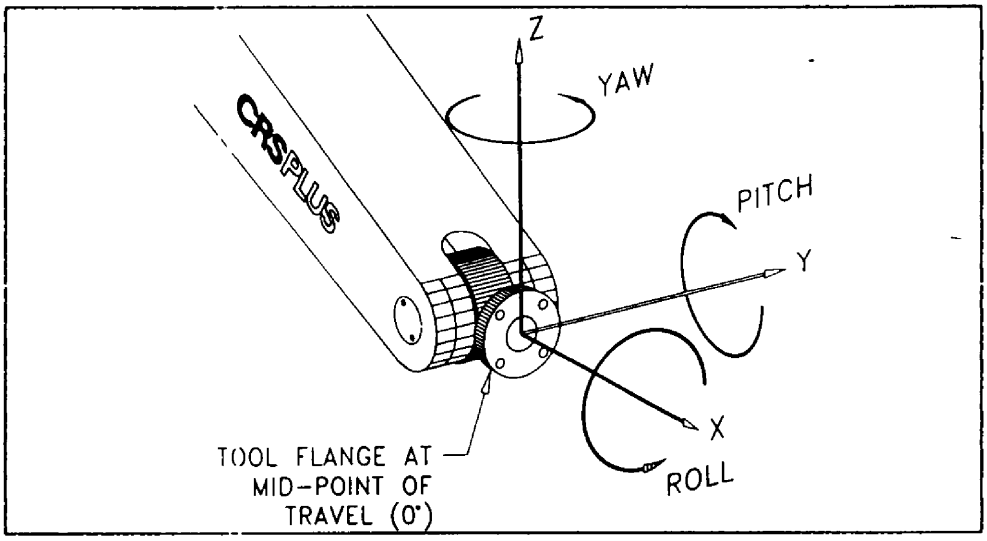


FIGURE 2-3 - The Tool Coordinate System

Example 1

For palletizing applications, pallet locations should be defined so that the TCP is located at the centre of the part. In this way, the shift command is changing the locations correctly.

If an air gripper is being used, the TCP must be defined as the point on the fingers where the part is grasped. Figure 2-4 shows the gripper in this arrangement. In this example it will be an X-axis offset from the default value. The tool transform would be as follows:

```
NAME          X        Y        Z       Yaw     Pitch    Roll
GRIPPER    +004.100 +000.000 +000.000 +000.000 +000.000 +000.000
```

This would be entered with the **POINT** command:

```
>>POINT ENTER LOCATION NAME: GRIPPER<cr>

NAME          X        Y        Z       Yaw     Pitch    Roll
GRIPPER    +000.000 +000.000 +000.000 +000.000 +000.000 +000.000

CHANGE? Y
X,Y,Z,YAW,PITCH,ROLL:
4.1,0,0,0,0,0<cr>
```

Example 1 (continued)

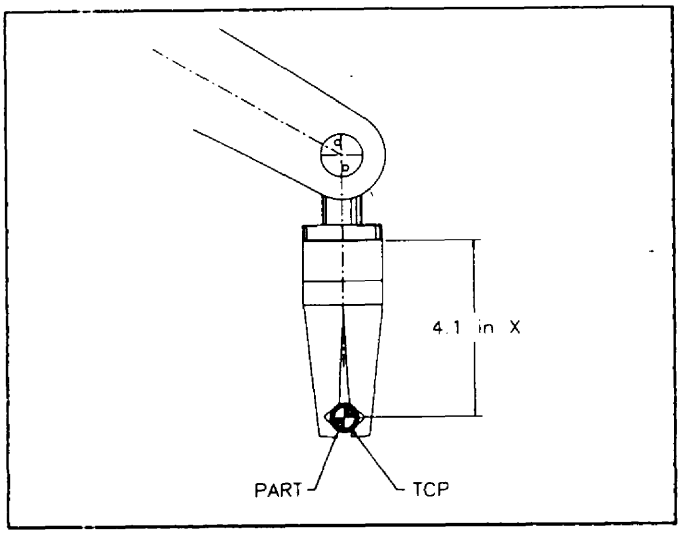


FIGURE 2-4 Gripper Arrangement, Example 1

Example 2

In cases where a symmetrical tool is carried by a 5-axis robot, it is an advantage to align the axis of symmetry with the world YAW-axis. In this way, the robot can use the pitch and roll axes of the arm (joints 4 and 5) to align the tool in any orientation within the limits of joint travel.

One type of tool which benefits from this approach is a glue dispense head. Since the nozzle is symmetrical about its axis, aligning the nozzle axis must be aligned across the flange, pointing down at the table when the robot is at READY (see FIGURE 2-5).

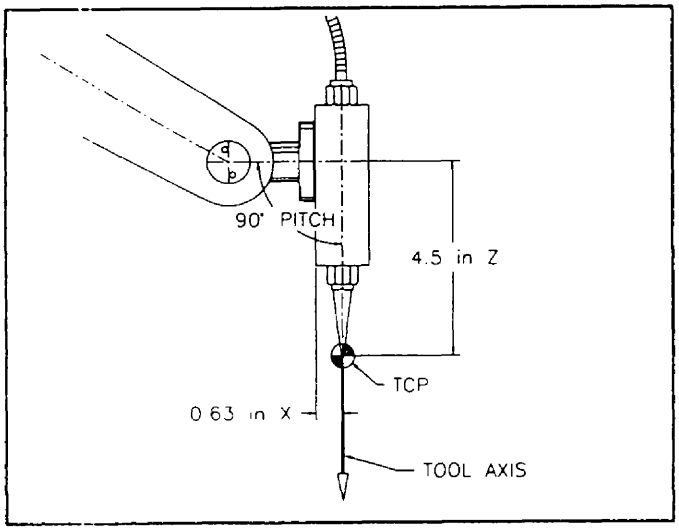


FIGURE 2-5 Dispense Head Arrangement - Example 2

2-4 TOOL COORDINATE PROGRAMMING (Continued)

Example 2 (continued)

It is important in dispensing to control the speed of the nozzle tip. Secondly, an **APPRO** or **DEPART** command should move the robot along the dispense axis. Both of these goals can be accomplished using the correct tool transform. We must set the TCP to the tip of the nozzle and use a +90° Pitch to align the tool axis with the nozzle axis.

The result will be a transform as follows:

```
NAME          X         Y         Z        Yaw     Pitch    Roll
DISPENSR   +000.630  +000.000  +004.500  +000.000  +090.000  +000.000
```

2-5 AUTOSTART

The RAPL-II system provides the user an Autostart feature allowing the robot to be homed and to run a predetermined program in a production environment without using a terminal.

The autostart sequence is activated when the Auto Start switch is depressed as the system power is turned on. When this happens, the controller searches its memory for a program called '**AUTO_ST**'. If the program is found, it is executed; if not, an error is issued. Refer to appendix 'D' for more information and a sample Auto Start procedure.

NOTE: An internal jumper on the front power logic board (refer to the technical manual, Section 7-6) can be set to short rout the AUTO-START switch. When this jumper is set, the controller will attempt an AUTO-START sequence every time it is turned on. This saves the operator a step when starting, however, since the switch is shorted, the **ONSTART** and **IFSTART** commands will no longer function correctly. **IFSTART** will always return true and **ONSTART** will stop operation completely.

2-6 TEACH MODE

This is a special mode which activates the TEACH button on the Teach pendant to store, when pressed and released, the current position of the robot. It can be activated from Immediate, Program or Manual mode. To set up the TEACH mode, the **TEACH** command is used:

[Line#] **TEACH** <TEMPLATE> [,<INDEX>,<NUMBER>,<INSERT|PACK>]

The template represents the first 5 characters of the location name and the index fills the remaining 3. The number entered as the <INDEX> in the command line is the starting index. Each time the Teach button is pressed after that, the index is increased by one. The maximum index value is 255. After 255, it returns to 0.

The Teach mode is enabled using the command >>**ENABLE TEACH**<cr>. The **DISABLE TEACH** or the **NOTEACH** command can be used to turn it off. After setting up the Teach Mode, locations can be stored by moving the robot (either by hand while LIMPed or using the Teach Pendant) to the desired position and pressing the Teach Button. The terminal displays the result by displaying the name of the location stored and issuing a beep for audible feedback. The robot may then be moved to the next location in the sequence and the procedure repeated.

The <NUMBER> in the command line represent the maximum number permitted before the Teach Mode is DISABLED. The <PACK> command tells RAPL-II to remove any "gaps" in the array of locations. The <INSERT> "opens" up a spot, or several spots in the array for teaching more locations. For more detail on the use of the TEACH command, refer to Chapter 4.

2-8 EDITOR DESCRIPTION

RAPL-II provides the programmer with a line editing feature which can be used to create or modify programs. The **EDIT** command is used to access the editor. If a program name is given in the command line, then that program is the object of all future editing or Program Line Entry (PLE) commands. If no program with that name exists, RAPL-II creates one (if there is space in the program table). The editor and PLE modes can be locked out using the DISABLE EDIT command.

Line numbers are optional in RAPL-II at run time. However in order for the editor to be able to construct the program properly, line numbers are required in all RAPL-II line entries.

NOTES:

** *NO SYNTAX CHECK IS MADE ON THE LINE CONTENTS BEFORE IT IS INSERTED. ERRORS WILL SHOW UP ONLY AT RUN TIME.*

** *A LINE CANNOT BE ENTERED IF A LINE WITH THE SAME NUMBER ALREADY EXISTS. THUS CHANGING A LINE REQUIRES DELETING THE OLD LINE FIRST.*

2-8 EDITOR DESCRIPTION (Continued)

Once in the EDIT mode, the prompt changes from ">>" to ">*". Several editing commands are available. Each is entered with a single keystroke; RAPL-II then echoes the complete command. Each command letter must be directly followed by a line number upon which the command is to act. A non-existent line number will result in an error. The commands are as follows:

Copy: It may be necessary to use identical command lines in the same program. In order to save keystrokes, the copy command can be used to copy one line to another.

Format: COPY<LINE#>,<NEWLINE#>

Example: In the following program:

```
10 READY
20 SPEED 50
30 APPRO A,2
40 MOVE A,S
45 FINISH
50 GRIP 1
70 DEPART 2
90 STOP
```

A FINISH command should be between the GRIP 1 (line 50) and the DEPART 2 (line 70). When in the editor the Copy command can be used to duplicate line 45 to line 60

```
>*COPY45,60<cr>
```

Now the program will be:

```
10 READY
20 SPEED 50
30 APPRO A,2
40 MOVE A,S
45 FINISH
50 GRIP 1
60 FINISH
70 DEPART 2
90 STOP
```

Delete: This command deletes a program line. Once deleted, a line cannot be restored.

Format: DELETE<Line#>

Example: In the above program, the speed at which the robot runs is coded into the program line #20 To make the speed more controllable, it makes sense to use a variable in the program which can be changed globally without editing the program line. To change line 20, we must first delete the old one.

```
>*DELETE20<cr>
```

Now the program will be:

```
10 READY
30 APPRO A,2
40 MOVE A,S
45 FINISH
50 GRIP 1
60 FINISH
70 DEPART 2
90 STOP
```

Insert: This command enters the system into PLE mode. It is used to insert a new command line into the program. A line cannot be inserted over another with the same number.

Format: INSERT<Line#> <Command> <arguments...><cr>

Example: To complete the above example, the new speed command with a variable name for an argument must be inserted:

```
>*INSERT20 SPEED NORMAL<cr>

10 READY
20 SPEED NORMAL
30 APPRO A,2
40 MOVE A,S
45 FINISH
50 GRIP 1
60 FINISH
70 DEPART 2
90 STOP
```

Move: It may be necessary to move a command line from one place to another within the same program. When a line number is re-assigned, it is removed from the original spot and re-installed in the correct numerical order.

Format: MOVE<Line#>,<New Line#>

Example: In the above example, it has been determined that the arm should move to the READY pose after the sequence and not before. In this case, the READY command line can be moved.

```
>*MOVE10,80<cr>

20 SPEED 50
30 APPRO A,2
40 MOVE A,S
45 FINISH
50 GRIP 1
60 FINISH
70 DEPART 2
80 READY
90 STOP
```

List: This command is used to display the contents of a command line.

Format: LIST<Line#>

Example: In our example, the programmer wants to delete line 40 but wants to be sure it is the correct line before the non-recoverable deletion.

```
>*LIST40<cr>
```

RAPL-II responds with:

```
40 MOVE A,S
```

End: Exits the line editor

FORMAT: END

Example: After all editing is complete, the End command is issued:

```
>*END<cr>
>>
```

Control is passed back to the Immediate mode command interpreter.

2-9 MANUAL MODE

Manual mode allows the operator to move the robot using the Teach Pendant.

RAPL-II provides two modes of manual control. The first is the JOINT MANUAL mode. In this mode, the 8 toggle switches on the teach pendant, the control each robot joint independently. This is the only possible MANUAL mode until the robot has been homed. The second mode, which is available after the robot is homed, is CYLINDRICAL MANUAL mode. In this mode, joints 2 and 3 are synchronized so the tool flange will move in a purely radial or vertical (Z axis) direction. This simplifies arm positioning and location teaching. When in this mode, toggle switch 2 moves the flange radially while switch 3 moves it vertically.

If fewer than 7 axes are selected, two additional functions are available on the teach pendant during MANUAL mode:

ALIGN: Switch #7 is equivalent to entering the **ALIGN** at the keyboard. RAPL-II will re-position the robot so that the tool centre point is at the same position as before but the tool X axis is aligned with the nearest major axis, vertical or horizontal (see **ALIGN** in CHAPTER 4). *This feature will not work when MANUAL Mode is entered through a program command. Nor will it work if a command has been partially entered at the terminal.*

LIMP: Switch #8 is equivalent to entering **LIMP** or **NOLIMP** at the keyboard, depending on which direction the switch is pushed. This opens the servo-loop so a user can move the robot by hand while the controller keeps track of the arm's position. This simplifies rough positioning.

The function of the switches under various situations is seen in TABLE 2-1. For more information, read the **MANUAL, ALIGN, LIMP** and **NOLIMP** commands in CHAPTER 4.

SWITCH #	JOINT mode 5 - 6 axes	CYL mode 5 - 6 axes	JOINT mode 7 - 8 axes	CYL mode 7 - 8 axes
1	JOINT 1	JOINT 1	JOINT 1	JOINT 1
2	JOINT 2	RADIUS	JOINT 2	RADIUS
3	JOINT 3	VERTICAL	JOINT 3	VERTICAL
4	JOINT 4	JOINT 4	JOINT 4	JOINT 4
5	JOINT 5	JOINT 5	JOINT 5	JOINT 5
6	-	-	JOINT 6	JOINT 6
7	ALIGN	ALIGN	JOINT 7	JOINT 7
8	LIMP/NOL.	LIMP/NOL.	JOINT 8	JOINT 8

TABLE 2-1 Switch functions in MANUAL Mode.

3-1 INTRODUCTION

The commands in RAPL-II can be grouped roughly into 9 catagories. This chapter provides a quick overview of these commands separated by type. Prior to using any commands from this list, the user should refer to chapter 4 for the correct syntax and function of each command.

3-2 MOTION COMMANDS

RAPL-II commands and controls movement of the arm with one of the Motion Commands. The controller can command up to eight axes of motion. The speed of the motion is controlled with the **SPEED** command, expressed as a percentage of maximum.

RAPL-II includes a **LOCK** command which excludes specified joints from further Motion Commands. This command becomes especially useful when using the controller to control two separate machines which must perform non-synchronized tasks. A machine may be defined as a separate piece of equipment or a defined group of axes. For example, suppose the robot is mounted on a track and the programmer does not want the robot moving along the track while trying to pick up a certain test tube. The programmer can **LOCK** joint 6 and prevent any further motion along the track. **UNLOCK** will reactivate a locked joint.

There are three different types of programmable motion in RAPL-II: Joint Interpolated, Straight Line and Path motion:

Joint Interpolated:

This type of motion coordinates the motion of all joints so that they start and stop at the same time. By default, the velocity of each joint varies according to a spline velocity profile. This gives the arm a smooth motion. For faster, but less smooth motion, a software switch sets up a trapezoidal velocity profile for each joint (Slew mode). This switch is toggled with the **ENABLE SLEW** and **DISABLE SLEW** commands. The Slew mode should not be used for normal motions, but may be used to decrease cycle time if, for instance, long sweeps of joint 1 were required. Most Motion Commands use Joint Interpolated motion (e.g.- **MOVE, DEPART**).

Straight-Line:

This type of motion keeps the Tool Centre Point moving in a straight line. RAPL-II does this by using five axis straight line interpolation. **JOG** is the only command that is always Straight-Line motion. Other commands such as **APPRO, DEPART** and **MOVE** may use Straight-Line motion depending on whether an [,S] argument is used with them. For example, if the user wanted the **MOVE** command to be in Straight-Line motion, then the command is:

[Line#] **MOVE** <LOC_NAME>,S

Path:

Path motion in RAPL-II uses a cubic-spline interpolation technique to smoothly move the arm through a sequence of pre-programmed locations (each of these locations are referred to as knots). The Tool Centre Point on the robot will then follow the path set by this function. By default, Path motions move the arm to minimize loading on all controlled joints. When the CARTVEL function is turned on (with the **ENABLE CARTVEL** command), a different approach is used: the controller uses cartesian distances to determine the path knot timing to keep the Tool Centre Point moving at a constant speed.

3-2 MOTION COMMANDS (continued)

Notes on Motion Commands in General:

** *ALL MOTION COMMANDS EXECUTED IN THE MANUAL CONTROL LEVEL FROM THE KEYBOARD WILL RETURN THE USER TO THE INTERACTIVE CONTROL LEVEL.*

** *Before the robot will accept any Motion Command which references the cartesian work space (such as JOG), the robot must be Homed. The Motion Commands **JOINT** and **MOTOR** may be used before Homing the machine. **USE EXTREME CARE WITH THESE COMMANDS AND MANUAL MOVES BEFORE HOMING, SINCE THE JOINT LIMIT CHECKING DOES NOT FUNCTION UNTIL THE HOME PROCEDURE HAS BEEN COMPLETED.***

** *Once the robot has been Homed (see **HOME** command in chapter 4), all Motion Commands (except the MOTOR command) are tested for movement outside of software limits prior to the robot starting the motion. These limits are defined by the kinematics of the robot structure.*

ALIGN	Align the tool with the nearest major axis.
APPRO	Move the robot to a position a specified distance away from a location.
CIRCLE	Move robot in a user defined circular path.
CPATH	Execute a continuous path.
CTPATH	Program a continuous path comprised of teach points.
DEPART	Move the robot away from its current location.
FINISH	Flag to instruct a motion command to complete motion prior to decoding the next program line.
GAIN	Change the positional gain of the servos.
GOPATH	Execute a continuous path programmed with **CTPATH**.
HALT	Stops all motion or sets up "feed-hold" on input.
HOME	Initialize robot position registers.
JOG	Move robot by a cartesian increment in a straight line.
JOINT	Move a single joint by an angular displacement.
LIMP	Disengage positional servos.
LOCK	Prevent selected joints from motion.
MA	Move all 5 joints to an absolute radian value.
MI	Move all 5 joints by an incremental radian value.
MOTOR	Drive selected motor by a selected number of pulses.
MOVE	Move the robot to a specified location.
NOLIMP	Re-engage positional servos.
READY	Move arm to READY position.
SPEED	Set speed of robot motion.
UNLOCK	Allow selected joints to move.
VIA	Move through selected points in JOINT or WORLD mode.

TABLE 3-1 Motion Command Summary

3-3 INPUT/OUTPUT COMMANDS

RAPL-II input/output commands permit the robot to "talk" to a number of different external devices. RAPL-II can access digital input and output ports, dual RS232 communication channels, and optional analog inputs and outputs.

The digital inputs can be used to sense switch closure signals in the work-place, providing logical decisions for use in program flow control. The digital outputs can be used to control external equipment.

The Dual serial communication channels provide input and output to a video display terminal, host computers, serial printers, or any other serial device. In standard configuration, they conform to the RS-232 standard. Optionally, they can be converted to RS-422.

The optional analog inputs can provide feedback from sensor elements in many robot applications. Similarly the analog outputs can send signals to other machines and devices.

AOUT	Send analog voltage to selected channel.
ARM	Enable or disable arm power.
CONFIG	Set Configuration of the RS232 ports.
DEVICE	Select the RS232 port for use.
FLASH	Set Flash Interval of light on Teach pendant panel.
IFAUX	Conditional statement based on status of the auxiliary input.
IFPOWER	Check status of arm power.
IFSIG	Conditional statement based on status of selected input(s).
IGNORE	Turn off auto-interrupt feature of robot.
INPUT	Input data into user program during run time.
NOFLASH	Stop flashing of light on Teach pendant panel.
ONAUX	Wait for a condition of the auxiliary input.
ONPOWER	Wait for arm power to come on.
ONSIG	Turn on auto-interrupt feature of robot.
OUTPUT	Turn on selected output(s).
PRINT	Output user information to printer port.
PRINTI	Output variables in an integer format to printer port.
PRINTV	Output variables to printer port.
SERIAL	Display the RS232 port status.
TRIGGER	Change the state of a digital output in path move.
TYPE	Display user information on video display.
TYPEI	Display variables in integer format on video display.
TYPEV	Display variables on video display.
WAIT	Wait for a selected condition of a digital input port.

TABLE 3-2 Input/Output Command Summary

3-4 GRIPPER COMMANDS

RAPL-II is designed to control pneumatic, servo (electric) and electromagnetic grippers.

The pneumatic gripper is controlled by the **OPEN** and **CLOSE** commands. The air pressure must be manually regulated to provide the desired gripping force.

The servo gripper is controlled in the position mode by the **GRIP** command. It can be opened or closed in the force mode by using the **OPEN** and **CLOSE** commands.

The magnetic gripper is controlled by the **MAGGRIP** command. Gripping force of the electromagnetic gripper is set as a percentage from 1 to 100. The actual gripping force depends on the gain of the magnetic amplifier and the coil type.

CLOSE	Close gripper (at specified force if SERVO).
GRIP	Change position of Servo-Gripper fingers.
MAGGRIP	Set gripping force level for Magnetic gripper.
OPEN	Open gripper (at specified force if SERVO).
TOOL	Specify a Tool Centre Point.
WGRIP()	A function within the ! Command permitting assignment of the Servo-Gripper finger position to a variable.

TABLE 3-3 Gripper Command Summary

3-5 SYSTEM COMMANDS

The RAPL-II system commands provide general program manipulation, memory allocation, system timer clock, controller status, and human interface.

Elementary program line editing commands allow the programmer to alter the contents of a program in the memory.

RAPL-II supports a variety of memory oriented commands intended to provide the user with a working memory space which can be tailored to specific needs.

RAPL-II also includes timing commands. These commands give the user the ability to build programs that are totally dependent on timing, synchronization and coordination. The **DELAY** command allows the user to halt all robot activity for a certain specific amount of time (accuracy within milliseconds). With the real time clock installed, the user can program the robot around an accurate time schedule. This is accomplished with the use of the **RTCLOCK** command. The **TIME** command reads the value of the system "ticker" clock and stores it in a variable. This command can be used for measuring time increments down to about 40 milliseconds.

RAPL-II also contains commands that allow the user to communicate directly with physical addresses within memory and the I/O chip. The commands **IORD** and **IOWR** read and write respectively to output port (I/O chip) of the 8086. The **MEMRD** and **MEMWR** commands read and write respectively to the 8086 memory. These are advanced commands intended for experienced users, but provide tremendous programming flexibility.

?	List Commands.
;	Documentation statement (comment)
ALLOC	Partition and clear robot memory.
AXSTATUS	Displays status of each axis card.
COPY	Duplicate program.
DELAY	Provide a time delay.
DELETE	Delete a program.
DIR	List program names that are in the program table.
DISABLE	Turn off a desired software "switch".
EDIT	Enter line editor or create a new program.
ELBOW	Determine the orientation of the elbow (UP/DOWN) for subsequent moves.
ENABLE	Turn on a desired software "switch".
EXECUTE	Causes execution of an 8086 machine language program.
FREE	Display user memory status.
HELP	Turn syntax-building feature on (Help mode).
HIMEM	Reserve part of the program buffer for other uses.
INVERT	Invert the Z-axis to permit correct inverted robot operation.
IORD	Read value of byte or word at 8086 output port.
IOWR	Write value of a constant or variable to the output port of the 8086.
LISTP	List program to a selected device.
MANUAL	Enable the Teach pendant for manual movement of robot.
MEMRD	Read contents of a memory address.
MEMWR	Write a constant or variable to a memory address.
NEW	Clear user memory.
NEXT	Single step through program.
NOHELP	Turn syntax building feature off.
NOMANUAL	Disable the Manual mode.
NOTEACH	Disable the Teach button on the Teach pendant.
NOTRACE	Disable Trace mode.
PASSWORD	Permit access to Monitor Level Commands.
REACH	Determine the reach type (FORWARD/BACK) for subsequent moves.
REMOTE	Permit a remote host to issue motion commands to guide the arm.
RENAME	Change program name.
RTCLOCK	Read current state of MK48T12 real time clock.
RUN	Execute a program from memory.
STATUS	Display operating status of robot.
TEACH	Enable the Teach button on Teach pendant.
TIME	Read the system timer clock.
TRACE	Program line numbers are displayed on terminal as instructions are executed.
WRIST	Determine the orientation of the wrist (FLIP/NOFLIP) for subsequent moves.

TABLE 3-4 System Command Summary

3-6 ROBOT LOCATION ASSIGNMENT

One of the most useful capabilities of RAPL-II is its ability to deal with named robot locations. There are commands which are used to inform the operator of the robot's current location and others to input and modify locations.

RAPL-II deals with locations in either inches or millimetres. A switch must be set on the mother board of the RAPL-II controller to change from one to the other. Changing without re-defining locations is dangerous and can lead to collisions.

There are eight elements used to describe each robot location. The cartesian (X,Y,Z) location of the Tool Point is defined by its location in space referenced to the centre of the robot base (unless an **OFFSET** command has been issued). Three angles describe the orientation of the location to the Tool Coordinate System: Yaw (orientation about the Z axis), Pitch (orientation about the Y axis) and Roll (orientation about the X axis).

The ability to define a location in terms of a remote coordinate system may also be useful. This is performed by the **OFFSET** function which will move the base origin to a new position. On the other hand, if the physical tool changes, the **TOOL** function permits changing the Tool Centre Point.

ACTUAL	Define a location as the actual robot position.
DLOCN	Delete a stored location.
HERE	Define a location as the current commanded robot position.
LISTL	List any or all locations stored in memory.
OFFSET	Redefine the base coordinate.
POINT	Define a location.
SET	Equate a new location with an existing location.
SHIFT	Shift a location by an incremental amount in X,Y,Z.
SHIFTA	Shift a location by an incremental amount in any or all coordinates.
TOOL	Specifies the location to be used as a Tool Centre Point
W0	Display robot's current commanded position (Joint and World coordinates).
W1	Continually display robot's actual position (Motor coordinates).
WE1	Continually display actual position of the extra axes (Motor coordinates).
W2	Display the robot's actual position.
W3	Continually display the robot's commanded position (Motor coordinates).
WE3	Continually display the commanded position of the extra axes (Motor coordinates).
W4	Display the robot's next end point.
W5	Continually display the robot's position error (Motor coordinates).
WE5	Continually display the position error of the extra axes. (Motor coordinates).
WITH	Permit numbered locations to be accessed using a Teach mode template. Used for Implied location access.

TABLE 3-5 Location Assignment Command Summary

3-7 PROGRAM FLOW COMMANDS

RAPL-II program flow commands permit conditional and unconditional branches within programs. Line number destinations may be defined by stored variables which permit "case" type programming.

ABORT	Terminate program execution and stop motion.
GOSUB	Pass control to the specified sub-program.
GOTO	Unconditional branch to a line number.
IF	Branch if variable expression is true.
IFAUX	Branch on status of auxiliary input.
IFPOWER	Branch on status of arm power.
IFSIG	Branch on status of selected input(s).
IFSTART	Branch on status of Auto Start switch.
IFSTRING	Branch on status of a string comparison.
ONAUX	Wait for a condition of an auxiliary input.
ONERR	Trap an error condition.
ONPOWER	Wait for arm power to be turned on.
ONSIG	Turn on auto-interrupt feature of robot.
ONSTART	Wait for Auto Start switch to be pressed.
PAUSE	Halt program flow until the **PROCEED** command.
PROCEED	Continues program flow after a **PAUSE** command.
RETRY	Following an error correction, this command will retry the line.
RETURN	Return control to a calling program from a sub-program.
RUN	Start a program or repeat it a number of times.
STOP	Program completion command.

TABLE 3-6 Program Flow Command Summary

3-8 STRING COMMANDS

The RAPL-II language can manipulate up to 4 text strings. The strings are identified by the special character '&', followed by a number 1 to 4. These strings can be up to 32 characters in length. Strings are bounded by the single quote marks when they are being assigned. A string can consist of any printable character or a decimal byte value. For instance the following command assigns the string Hello into string 1:

```
>>! &1 = 'Hello'<cr>
```

To encode the sound of a bell into the string, the ASCII bell character (ASCII value 7) can be added to the string. Since this is a "non-printing" character, it can be included as follows:

```
>>! &1 = 'Hello\007'<cr>
```

The back-slash character tells the string interpreter that a three digit decimal code is to be placed in the string. Another example of this technique is to include carriage returns and line feeds into strings. A carriage return has a decimal value 013, while a line feed has a value of 010. (A complete table of ASCII values may be found in Appendix E.):

```
>>! &1 = 'Type this\013\010then a new line'<cr>
>>TYPE &1<cr>Type this
then a new line
>>
```

To get back to the RAPL-II prompt, a carriage return from the keyboard is required.

Strings can be concatenated, using the standard assignment command (!). As an example, type in the following sequence at the terminal:

```
>>! &1 = 'Hello'<cr>
>>! &2 = ' World'<cr>
>>! &3 = &1 + &2<cr>
>>TYPE &3<cr>Hello World
>>
```

** *NOTE: The string number appearing on the left side of the assignment statement can not appear on the right side in the same statement.*

As shown in the above example, the **TYPE** command can be used to display a string. The operator can also enter a string, when in Program mode, using the **INPUT** command as shown here:

```
>>1 INPUT &1
```

The string entry will be terminated when the operator enters a carriage return. The carriage return will not be part of the stored string value.

Special commands exist that permit encoding of variable-type data into a string, or decoding data from a string. This is useful when the robot is "talking" to devices that transmit data in serial form, or which require special serial commands for operation.

CUT	Remove a portion of the string.
DECODE	Decode a data variable value from a string.
ENCODE	Encode a data variable into a string.
IFSTRING	Compare two strings and branch on result.
PASTE	Paste a new portion of text into a string.
STRPOS	Return the character location of a sub-string.

TABLE 3-7 String Command Summary

3-9 VARIABLE COMMANDS

RAPL-II contains a set of commands that are used to modify and display variables.

!	Assign a numeric value to a variable name.
++	Increments a variable by 1.
--	Decrements a variable by 1.
DVAR	Delete a variable name.
LISTV	List variables stored in memory.

TABLE 3-8 Variable Definition Command Summary

3-10 EXTRA AXIS COMMANDS

The RAPL-II language provide control of up to eight axes: five to six are normally used for the robot arm, and the others may be used to control up to three different servo axes. Two common applications of these extra axes are mounting the robot on a track, and mounting it on a gantry, suspended from above.

To configure RAPL-II for use with extra axes, several monitor level commands are used. These are included here for completeness, but are fully documented only in the robot system Technical Manual. Operation of the axes is controlled using "standard" RAPL-II commands.

MONITOR FUNCTIONS:

@XMAXVEL	Enter the maximum possible encoder speed for axis.
@XLIMITS	Enter joint limits for extra axes.
@XPULSES	Enter number of pulses per turn of extra axis encoder.
@XRATIO	Enter transmission ratio for extra axis.
@GANTRY	Set up axes six and seven as X and Y gantry axes.
@TRACK	Set up axis six as a track - X or Y axis.

STANDARD RAPL-II COMMANDS:

JOINT	Command motion for an axis - extra or robot.
LOCK	Lock an axis against motion commands.
UNLOCK	Permit motion of a formerly locked axis.
XCAL	Calibrate an extra axis.
XHOME	Home an extra axis.
XREADY	Move an extra axis to its Zero position.
XZERO	Set the axis current position to zero.

TABLE 3-9 Extra Axis Command Summary

3-11 PATH COMMANDS

RAPL-II contains several path type motion commands. These are listed under motion commands in section 3-2 but since they have fundamental differences from other commands, they deserve special mention here.

CIRCLE	Move tool-centre-point in a circle.
CPATH	Execute a continuous path through up to 16 points.
CTPATH	Program a continuous path comprised of TEACH points.
GOPATH	Execute a continuous path programmed with **CTPATH**.
VIA	Move through selected points in JOINT or WORLD mode.

TABLE 3-10 PATH Command Summary

RAPL-II path commands can be grouped into two forms: CPATH types and the VIA command. The CPATH commands (CPATH, CTPATH and GOPATH) all cause the robot to move through a sequence of locations. The VIA command moves the arm close to the but not actually through a series of locations. It allows the arm to deviate slightly depending on speed and acceleration rates during the motion. This concept is illustrated in Figure 3-1.

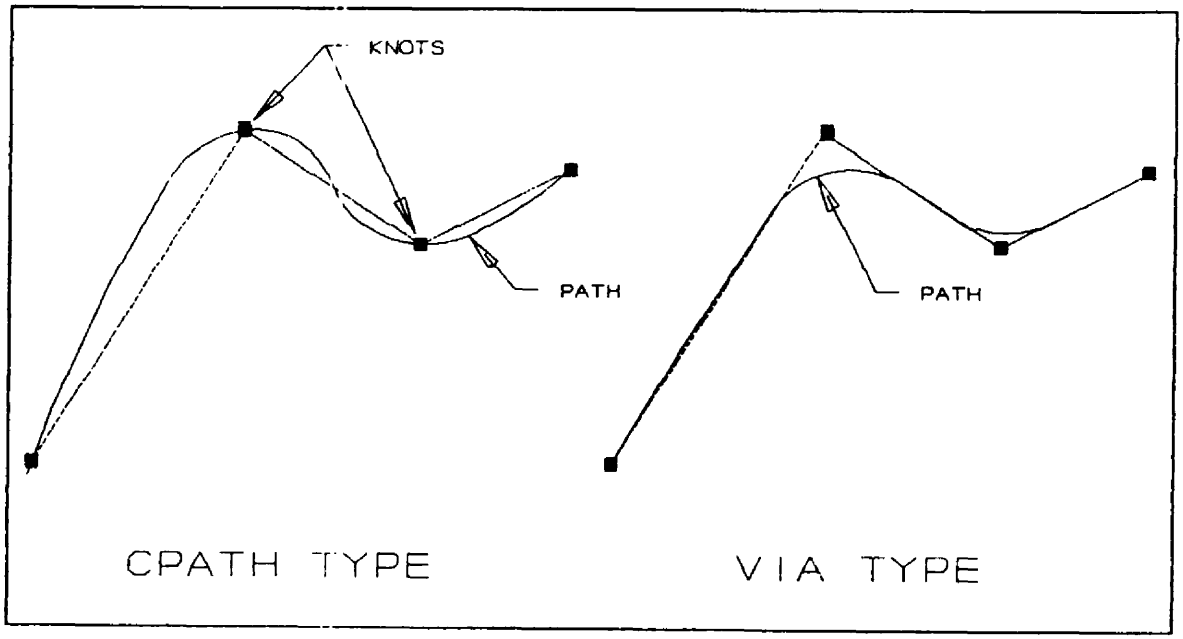


FIGURE 3-1 CPATH Type (THROUGH) Path vs. VIA Path

CPATH commands use a cubic spline to compute a path joining the locations. A term used to help visualize this is calling the locations in the path knots. It is as if the path to be followed is a string. The string's shape is determined by the fact that it is tied down at certain knots along its length. The more knots, the more accurately it is tied to the desired shape. Similarly, the more locations in a given length of path, the more "determined:" the shape of the path will be. This is seen in figure 3-2.

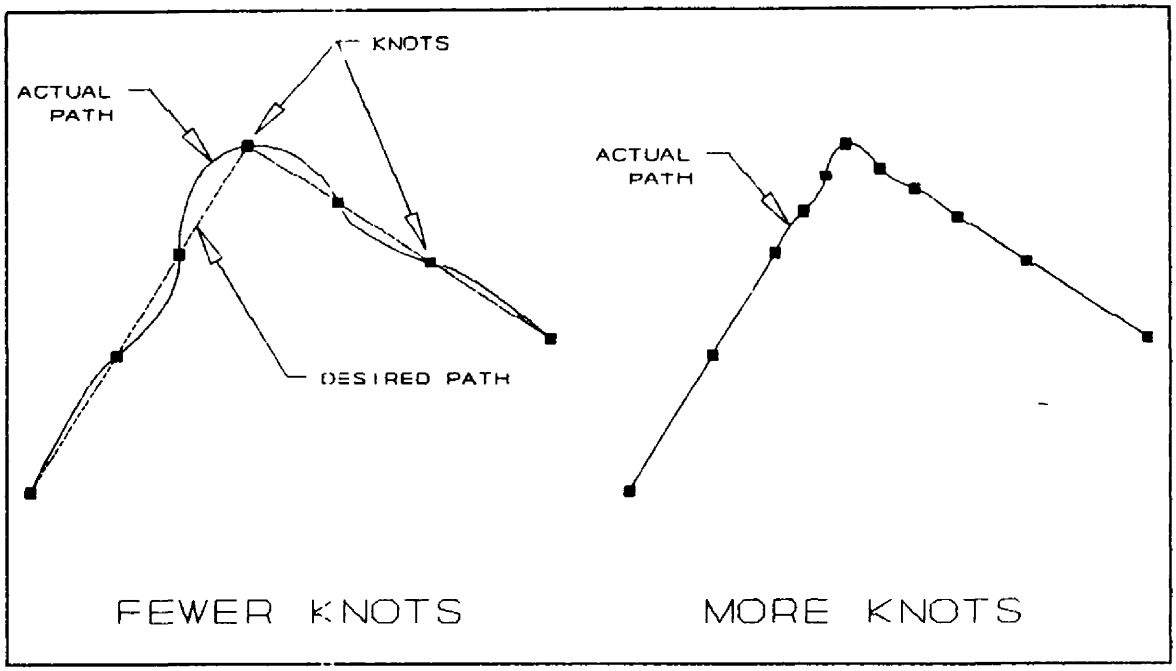


FIGURE 3-2 More knots constrain a curve more precisely

The CARTVEL flag has an affect on the performance of both types of path motion commands. CARTVEL stands for CARTesian VELocity. In the case of CPATH type commands, if this flag is enabled during the path generation stage, the path stored or executed will move the tool centre point through space at a roughly constant speed. This speed is determined as a percentage (the global speed value) of the Maximum linear speed factor (see STATUS command). For example, if the robot was commanded to be moving at 40% speed and the linear speed factor was set to 12 inches/second, the path speed would be 4.8 inches/second. If CARTVEL is not set, the speed will be determined by optimizing joint speed through the path. Not only does the CARTVEL flag influence the speed of the path motion, but due to the mathematics involved in creating the spline, the path taken can be influenced slightly as well.

In the case of the VIA command, the CARTVEL flag decides if the motion will be optimized for joint motion or world motion. In joint optimized VIA, the arm moves in PTP form along the path. That is, it moves toward each intermediate location in joint interpolated form but does not stop at them. It just moves on toward the next one when it gets sufficiently close to the location. If CARTVEL is enabled, all motions are in straight line form towards each location.

There are limitations as to what commands are available depending on the controller hardware and memory available. Table 3-11 lists these:

COMMAND	ROBOT SYSTEM	MEMORY REQ'D
CIRCLE	A400	STANDARD
CPATH	ALL	STANDARD
CTPATH	ALL	EXPANSION
VIA	A200/400	STANDARD

TABLE 3-11 Limitation on PATH Commands

4-1 INTRODUCTION TO COMMAND LIST

This Chapter contains an alphabetical list of RAPL-II commands describing their syntax, function, modes of operation, any associated warnings, and examples of command usage.

In examples within this chapter, all information which the user must type in, is shown in **BOLD**. The remainder of the command will be shown in normal text. For example:

>>STATUS<cr>

Indicates that the user must type the "STA" (which is enough letters to generate a unique command). As soon as that has been typed, RAPL-II completes the command entry by typing the "TUS". To enter the command, the user types the <cr>. In this way, users who prefer to use the HELP mode can determine just how many letters to type to make a command unique. Those who prefer to turn HELP off can see the complete command and its syntax.

In many instances RAPL-II prompts you for more information with a message. In those cases, the message will be shown. In subsequent examples of the command, the prompting message may be omitted, but if it is, the message:

(prompt messages not shown)

will appear above that usage.

?

TOKEN: /000

FORMAT:

? <COMMAND_NAME | TOKEN_NUMBER>

DESCRIPTION:

Displays RAPL-II commands and their numeric tokens. Monitor level commands will be included in the display only if the correct PASSWORD has been entered.

Entering a command name after the command displays the token for that command. Entering part of a command name displays all commands which match that pattern and their token values. Entering a number after the ? command displays the command for which that number is a token. These features permit easier entry or translation of tokenized commands and programs.

APPLICABLE MODES:

{I},{M}

EXAMPLE:

1. Display RAPL-II commands.

```
>>??????? ENTER COMMAND NAME, A TOKEN NUMBER, OR <CR>: <cr>
000 ??????  001 ALIGN   002 APPRO  004 DEPART  005 JOG  006 JOINT
etc.
```

2. Determine the command associated with a token:

```
>>??????? ENTER COMMAND NAME, A TOKEN NUMBER, OR <CR>: 12<cr>
TOKEN NUMBER #012 IS COMMAND MOVE
>>_
```

3. Determine tokens for possible commands:

```
>>??????? ENTER COMMAND NAME, A TOKEN NUMBER, OR <CR>: MO<cr>
TOKEN NUMBER #011 IS COMMAND MOTOR
TOKEN NUMBER #012 IS COMMAND MOVE
>>_
```

CROSS-REFERENCE:

RAPL-II COMMANDS: **PASSWORD**

TOKEN: /192

FORMAT:
[Line#] ++ <VARIABLE NAME>

DESCRIPTION:

<VARIABLE NAME>
Variable names only.

This command increments a variable. It is an abbreviation for the mathematical assignment:

```
C = C + 1
```

as shown here:

```
>>++ VARIABLE: C<cr>
```

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

!, --

--

TOKEN: /193

FORMAT:
[Line#] -- <VARIABLE NAME>

DESCRIPTION:

<VARIABLE NAME>
Variable names only.

This command decrements a variable. It is an abbreviation for the mathematical assignment:

```
C = C - 1
```

as shown here:

```
>>-- VARIABLE: C<cr>
```

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:
!, ++

TOKEN: /070

FORMAT:

[Line#] ! <&N | VARIABLE>=<&N | 'String' | ARG1[+ | - | * | /]ARG2>

DESCRIPTION:

Permits entry and modification of variable or string data. For variables, the four standard arithmetic operations are supported: '+,-,*,/'. The right hand side arguments can be variable references or explicit constant values. Variable references can be preceded by a plus or minus sign as a unary operator. The assigned value will be treated accordingly.

Using parentheses, the mathematical expression may be expanded to provide up to 9 levels of nesting. A function is considered as a nesting level. For instance:

```
>>! A = 1 + 2 *(2+A)/4*(3+(ABS(8)*SIN(X))<cr>
```

RAPL-II provides the following standard built-in functions:

Trigonometric Functions

Y = SIN(X)	Sine of X (X in radians)
Y = COS(X)	Cosine of X (X in radians)
Y = TAN(X)	Tangent of X (X in radians)
X = ASIN(Y)	Arc sine of Y returned in radians
X = ACOS(Y)	Arc cosine of Y returned in radians
X = ATAN2(A,B)	Arc Tangent of (A/B) returned in radians

Other Mathematical Functions

X = POW(A,B)	A raised to the power B is returned
X = SQRT(Y)	Square root of Y returned
X = ABS(Y)	Absolute value of Y returned
X = INT(Y)	The integer portion of the real argument returned
X = LN(Y)	The natural logarithm of Y returned
X = LOG(Y)	The log base 10 of the argument is returned
X = MOD(A,B)	The modulus of A and B is returned

Conversion Functions

X = DEG(Y)	Converts the argument Y to degrees from radians
X = RAD(Y)	Converts the argument Y to radians from degrees

! (continued)

Location Component Extraction Functions

C = COMP(NAME,n)	This function extracts the component value from the specified location name. Legal component numbers (n) are from 1 to 8. This can be used on Cartesian or Precision Locations.

The following can be used on Cartesian Locations only:

X = XCOMP(NAME)	Returns the X component of the location NAME
Y = YCOMP(NAME)	Returns the Y component of the location NAME
Z = ZCOMP(NAME)	Returns the Z component of the location NAME
O = OCOMP(NAME)	Returns the Orientation (yaw) component of the location NAME
A = ACOMP(NAME)	Returns the Azimuth (pitch) component of the location NAME
T = TCOMP(NAME)	Returns the Tool (roll) component of the location NAME

Logical Functions:

X = OR(A,B)	X is the value of A ORed with B.
X = AND(A,B)	X is the value of A ANDed with B.
X = XOR(A,B)	X is the value of A EXCLUSIVE ORed with B.
X = MOD(A,B)	X is the result of A MODulus B

Miscellaneous Functions:

X = WGRIP()	Returns the finger position of the Servo Gripper in current units.
X = ANALOG(n)	Returns a value proportional to the input voltage present on the specified analog input chanel. Legal analog channels are from 9 to 26. Channel 9 is the teach pendant speed dial. Channels 10 to 26 require an analog expansion option, such as the COMBO/32 card. The value returned will be: (Vdc/5)*255.

If the variable on the left hand side of the assignment is previously undefined, a new variable will be created with the value of the operation. If a variable is used on the right side that is previously undefined, a value of 0 is assigned to that variable. Spaces between arguments in the expression are optional.

A String-type variable can also be defined using the ! command. This command permits setting one string equal to another string or to a literal string of text defined within single quote marks. For example:

```
>>! &1 = &2<cr>

>>! &3 = 'INSERT PART IN NEST #1'<cr>

>>! &2 = &1 + &3<cr>
```

** *NOTE: One important note regarding string assignment: A string appearing on the left side of the expression cannot appear on the right side as well.*

APPLICABLE MODES:
{I},{M},{P}

EXAMPLES:

1. Set String #1 to 'Transferred WIDGETS':

```
>>! &1 = 'TRANSFERRED WIDGETS'<cr>
```

2. Set string #2 to String #1:

```
>>! &2 = &1<cr>
```

3. An expression to define a point on a circle:

```
>>! X = RADIUS*COS(ANGLE)<cr>
>>! Y = RADIUS*SIN(ANGLE)<cr>
```

4. An example of a Unary minus sign to invert the sense of a variable. STACK_S is now equal to -A (-3) multiplied by 4:

```
>>! STACK_S = -A * 4<cr>
```

5. To store the current position of the teach pendant speed dial:

```
>>! SPEED1 = ANALOG(9)<cr>
```

6. To store the voltage present on analog input number 10 (first analog input of the COMBO/32 card):

```
>>! DCVOLTS = 5 * (ANALOG(10)/255)<cr>
```

CROSS-REFERENCE:

RAPL-II COMMANDS: LISTV, DVAR, IF, ++, --

;

TOKEN: /098

FORMAT:
[Line#] ; [Message]

DESCRIPTION:

This command indicates a comment statement. These can be used throughout programs to provide documentation. A space must always follow the ; command.

Although the ; statement is not an active command, it does use memory space, and slows down the operation of the robot program slightly as the comment line must be scanned just like any other. Comments should be used in a concise fashion in order to reduce both effects.

APPLICABLE MODES:
{P}

EXAMPLE:
```
100 ; THIS IS A COMMENT.
```

TOKEN: /071

FORMAT:
[Line#] **ABORT**

DESCRIPTION:

The ABORT statement is an abrupt method of ending a program. Any motion in progress will be stopped immediately and the program terminated. This command is the software equivalent of the ABORT button on the Teach pendant.

APPLICABLE MODES:
{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **HALT**, **PAUSE** and **STOP** are other means of stopping the robot and/or a program

ACTUAL

TOKEN: /152

FORMAT:
[Line#] **ACTUAL** <LOC_NAME>

DESCRIPTION:

<LOC_NAME>
Can use a location name, a string or an ARRAY-Type location.

Read the actual position of the robot arm, and store the information as an entry in the location table in memory. The location can be stored in cartesian (X, Y, Z, YAW, PITCH, and ROLL) or precision (motor encoder pulses) form. If ENTER is typed without specifying a location name, the current actual position is displayed.

This command differs from the HERE command in that it reads the actual position including any positional error in the arm. Thus, a move to a location taught with ACTUAL will not move to the same location as the one taught: it will have its own error associated with it. ACTUAL is most useful for determining current positional error so that compensation may be made. Refer to the example below.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. A section of a program to determine the vertical position (Z-axis) error at a programmed location (POS1) is as follows:

```
1000 MOVE POS1
1100 FINISH
1200 ACTUAL ERRPOS1
1300 ! Z ERR = ZCOMP(ERRPOS1) - ZCOMP(POS1)
1400 ...
```

CROSS-REFERENCES:

RAPL-II COMMANDS: HERE, SET, W3

TOKEN: /001

FORMAT:
[Line#] ALIGN

DESCRIPTION:

This command provides the user with a simple method of aligning the tool flange with either the horizontal or vertical plane. It maintains the Tool Centre Point at the same cartesian location while moving the arm in Joint Interpolated motion, until the tool axis is aligned with the nearest (horizontal or vertical) plane. (If the tool pitch is exactly 45 degrees it will align to the vertical plane.)

The Manual mode has a hardware version of this command built in. If 6 or less axes are installed in the system, switch 7 on the Teach pendant performs the equivalent of:

```
NOMANUAL
ALIGN
MANUAL
```

This feature can only be activated in the MANUAL mode (ie. not while in a program), and requires that there not be a command partially entered through the terminal when the switch is depressed.

WARNINGS:

1. Using this command before the robot is HOMEd will cuase a RAPL-II error.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: MANUAL

ALLOC

TOKEN: /018

FORMAT:

ALLOC <#_PROGRAMS>,<#_VARIABLES>,<#_LOCATIONS>,<#_PATHKNOTS)

DESCRIPTION:

ALLOC is an acronym for ALLOCate memory. This command clears and re-partitions the robot memory. This can be done with [A] Automatic or [N] Non-automatic partitioning (see examples).

If the [A] option is selected, the memory is automatically divided according to a formula leaving 16 programs in the program table and dividing the remaining memory into 12% variable space, 38% location space and 50% program space. If the [N] option is used, RAPL-II prompts for the space to be allocated to the program, variable and location tables, and Path memory area. "Reserved" memory can also be set aside for locating user defined executable routines. This area must be partitioned by using the HIMEM command after allocating user memory.

WARNINGS:

1. The user memory is always cleared after the ALLOC command is issued.

2. A program table item takes 12 bytes, a location 42 bytes, and a variable 14 bytes of storage. An error will result if not enough memory is available for the user's desired allocation.

3. Changing RAM partitioning may affect useage of the EEPROM RAM backup option.

APPLICABLE MODES:
{I},{M}

EXAMPLE:

1. To manually allocate memory (20 programs, 30 variables, 50 locations, leaving 5832 bytes program), Enter:

```
>>ALLOC<cr>
ARE YOU SURE - (THIS WILL ERASE USER MEMORY)? Y
TOTAL AVAILABLE MEMORY:  08592
AUTO/NOAUTO:  N<cr>
# PROGRAMS, # VARIABLES, # LOCATIONS, # PATH KNOTS:  20,30,50,10<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **FREE**, **CTPATH**, **GOPATH**, **NEW**, **@SAVE**, **@RESTORE** and **HIMEM**

TOKEN: /093

FORMAT:
[Line#] AOUT <OUTPUT#>,<VAR_NAME>

DESCRIPTION:

<OUTPUT#>
Can use constants, variable names, arrays or expressions.
<VAR_NAME>
Can only use a variable name.

NOTE: *This command requires the COMBO/32 RAM & I/O expansion option.*

AOUT is an acronym for ANALOG OUTPUT. This command sends the digital number <VAR_NAME> to the selected analog output channel where it is converted to a voltage between -10 and +10 Vdc. The COMBO/32 option provides two analog output channels selected by using a 1 or 2 for the <OUTPUT#> argument.

Both channels have 8-bit resolution so that each step will increase the output voltage by approximately 0.0784 Vdc. Use this formula to calculate output voltage: ((VAR_NAME/255)*20)-10 Vdc.

WARNING:

Drawing more than 10 mA will affect output voltage regulation and **could damage the controller.**

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: ANALOG(), ●●DIAG

OTHER CRS Plus Publications: COMBO/32-UM

APPRO

TOKEN: /002

FORMAT:
[Line#] **APPRO** <LOC_NAME>,<DISTANCE>[,S]

DESCRIPTION:

<LOC_NAME> Can use a location name, a string or an Implied location.
<DISTANCE> Can use constants, variable names, arrays or expressions.

The APPRO command moves the robot to a position which is the programmed distance back along the tool axis from the stored location "LOC_NAME".

The robot moves to that position in either joint interpolated or linear fashion. The location may either be a cartesian point, or a precision point. A positive "distance" implies that the tool will stop in front of the specified location.

The APPRO command enables the programmer to define an approach point without using extra memory resources.

The straight line [,S] specifier defines whether or not the approach should be made in a straight line or not. Typically, in an insertion operation, a non-straight approach is made to the approach point, then a straight move command is used to move to the location required.

APPLICABLE MODES:
{I},{P}

EXAMPLES:

1. Approach point PICK_1 stopping +2 inches away. Decrease speed to 20, then move to point PICK_1 in a straight line and close the gripper.

```
100 SPEED 100
110 APPRO PICK_1,2
120 SPEED 20
130 MOVE PICK_1,S
140 FINISH
150 CLOSE
```

2. Approach point PICK_1 in a straight line stopping +2 inches away.

```
110 APPRO PICK_1,2,S
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **DEPART**, **TOOL** and **MOVE**

TOKEN: /142

FORMAT:

[Line#] **ARM** <ON | OFF>

DESCRIPTION:

Sets the status of the Arm Power Relay (APR). If the "ON" parameter is chosen, the APR is enabled and pressing the ARM POWER switch on the front panel will turn the power on.

If the arm power is on, the ARM OFF command disables the APR and the arm power will turn off. This will result in an ARM POWER error and the robot arm will drop. The ARM POWER switch will be disabled until an ARM ON or ENABLE ARM command is issued.

Robots With Brakes:

When arm power is on the brakes are released and the arm can be moved under program control, MANUAL mode, or LIMP mode. Conversely, when the arm power is off joints 2 to 5 will be locked in position.

APPLICABLE MODES:

{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE**, **DISABLE**, **LIMP** and **NOLIMP**

OTHER CRS Plus Publications: TECHNICAL MANUAL

AXSTATUS

TOKEN: /185

FORMAT:

[Line#] **AXSTATUS** <AXIS#>,<INPUT | STATUS>,<VAR_NAME>

NOTE: *This command is only applicable to A200/400 series robots.*

DESCRIPTION:

<AXIS#>

Can use constants, variable names, arrays or expressions.

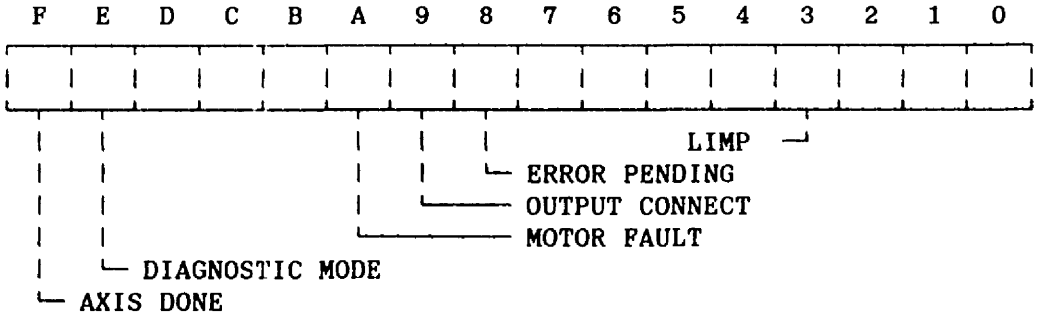
<VAR_NAME>

Can use a variable name or constant.

This command allows the RAPL-II processor to interrogate the status and active inputs of each axis card. The value of the input or status registers is saved in the variable name provided. Axis number can be specified as a variable reference.

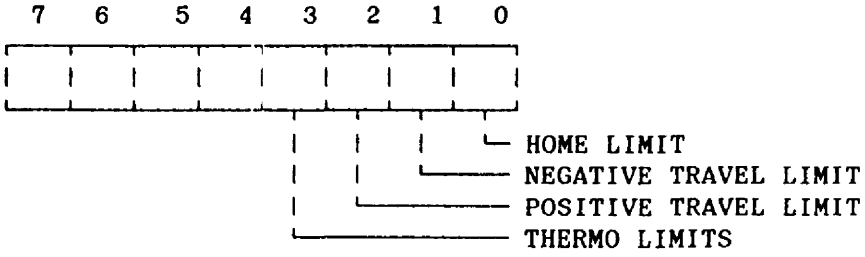
Status:

The status word from the axis card provides current information as to the condition of the card. Not all bits are described here, as they cannot be used by the programmer. Relevant bits will provide information as to whether or not an axis is moving, or whether there are errors occurring on the card.



Input:

The input register is an 8 bit register that monitors the state of the travel limit, home limit and thermo limit switches of the axis in question. The definition of the register is shown here:



APPLICABLE MODES:
{I},{M},{P}

CROSS REFERENCES:

RAPL-II COMMANDS: **STATUS, HOME**

OTHER CRS Plus Publications: TECHNICAL MANUAL

CIRCLE

THIS COMMAND FUNCTIONS ON CRS CARTESIAN ROBOT, MOTION CONTROL SYSTEMS AND A460 ROBOTS ONLY!

TOKEN: /182

TOKEN: /038

FORMAT:
[Line#] **CLOSE** [,Force]

DESCRIPTION:

[,Force]
Can use constants, variable names, arrays or expressions.

With a pneumatic gripper installed, this command closes the gripper by switching the solenoid in the arm. The FORCE argument is ignored.

With the Servo Gripper installed, it closes the gripper at the specified force, which is a percentage of full force. If no force argument is specified, then the previous force value (entered with a previous OPEN or CLOSE command) will be used.

In Manual mode the gripper can be closed by use of the gripper switch on the Teach pendant. For servo grippers the gripping force is determined by the setting of the speed potentiometer.

APPLICABLE MODES:
{I},{P}

EXAMPLE:
1. To close an air gripper:

```
>>CLOSE<cr>
```

2. To close the Servo Gripper at a force of 60% of full:

```
>>CLOSE GRIPPER CLOSING FORCE(%): 60<cr>
```

Note that the message prompting for gripping force shows up only when the system is set up for use of the servo gripper.

3. Grip an object tightly without overloading the gripper (see warning below). In this program, the gripper will close at maximum until it contacts a part. The gripping force is then reduced electronically but the mechanics of the gripper will keep a firm grip on the object.

```
...
1100 MOVE GET_PART
1200 FINISH
1300 CLOSE 100
1400 FINISH
1500 DELAY .2
1600 CLOSE 60
1700 ...
```

CLOSE (Continued)

WARNING:

THE SERVO GRIPPER MUST NOT BE CLOSED AT A FORCE OF MORE THAN 75% FOR EXTENDED PERIODS. The gripper will lock quite well mechanically after initially gripping the part tightly (see example 3 above). Gripping at values above 75% for longer than a few seconds is un-necessary and likely to shorten the life of the gripper.

CROSS-REFERENCES:

RAPL-II COMMANDS: **OPEN**, **MANUAL**, **GRIP** and **GGTYPE**

OTHER CRS Plus Publications: TECHNICAL MANUAL, SGRIP/UM

TOKEN: /110

FORMAT:

[Line#] CONFIG <DEV#>,<BAUD>,<PARITY>,<#DATA>,<#STOP>,<HANDSHAKE>,<ECHO>

DESCRIPTION:

<DEV#>,<BAUD>
Can use constants, variable names or arrays.

This command alters the configuration of the serial ports (DEVICE #0 and DEVICE #1) to match the requirements of other equipment. The following are the only allowable values for each argument:

Device Number:	0,1
Baud rates:	50,75,110,135,150,200,300,600,1200,1800,2400,4800,7200,9600, 19200
Parity Check:	[E] Even, [O] Odd, [N] None
Data Bits:	5,6,7,8
Stop Bits:	[1] 1 bit, [2] 1 1/2 bits, [3] 2 bits
Handshake:	[R] RTS/CTS, [X] XON/XOFF, [N] None, [B] Both
Echo:	[E] ON, [N] OFF

Devices #0 and #1 are initialized on start-up to the configuration last specified by the user.

WARNINGS:

When issuing this command, all arguments must be entered and all must be separated by commas.

APPLICABLE MODES:

{I},{M},{P}

EXAMPLE:

1. Set the configuration of device #1 to 1200 baud, even parity, 8 data bits, 1 stop bit, XON/XOFF handshake, and no echo:

```
>>CONFIG
DEVICE (0,1):1,
BAUD RATE (50 TO 19200) :1200,
PARITY (E)VEN, (O)DD, (N)ONE :E,
NUMBER OF DATA BITS (5,6,7,8) :8,
NUMBER OF STOP BITS (1) = 1, (2) = 1 1/2, (3) = 2 BITS :1,
HANDSHAKE (R)TS/CTS, (X)ON/XOFF, (B)OTH, OR (N)ONE :X,
(E)CHO FOR ON, OR (N)ONE  - THIS IS THE LAST INPUT:N<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: SERIAL
OTHER CRS Plus Publications: TECHNICAL MANUAL

COPY

TOKEN: /099

FORMAT:
COPY <OLD_NAME>,<NEW_NAME>

DESCRIPTION:

<OLD_NAME>,<NEW_NAME>
Can only use program names.

The COPY command duplictes an existing program in RAPL-II memory.

WARNINGS:
An error will result if there is insufficient memory space to contain the new program or if the new program name is the same as a program already present in memory.

APPLICABLE MODES:
{I},{M}

EXAMPLE:
1. Copy program TEST to a new file called TEST1.

```
>>COPY OLD PROGRAM NAME: TEST, NEW PROGRAM NAME: TEST1<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **RENAME**, **DELETE** and **EDIT**

TOKEN: /163

FORMAT:
[Line#] **CPATH** <LOC_NAME,...>

DESCRIPTION:

<LOC_NAME>
Can use location names, strings or Array-type location names.

This command permits the robot to move through a random series of points. In this command, unlike the CTPATH command, the locations specified in the argument list can be any location in memory. All locations in the list must be either cartesian locations, or precision points; no mixing of types is allowed. Up to 15 locations are permitted in the argument list (making a path using 16 locations including the starting location of the arm).

Triggers will be activated during the path execution if the Trigger table includes activities at any of the locations specified in the CPATH argument string and the Trigger flag has been enabled.

WARNINGS:
1. No two consecutive locations in the list may have identical values. This will produce a mathematical error and processing will be aborted.

2. If more than 5 axes are used in the controller, precision points must be used.

APPLICABLE MODES:
{I},{P}

EXAMPLE:
1. To move through a series of points without stopping. Any single location can be used, so long as it does not appear consecutively.

```
>>CPATH
ENTER PATH POINTS, SEPARATED BY COMMAS(UP TO 16)
A,B,C,B,C,A,B<cr>
```

2. This will produce an error, since the location B appears consecutively. Also, the precision point #EE is not allowed since the remainder of the locations in the list are cartesian locations.

```
>>CPATH
ENTER PATH POINTS, SEPARATED BY COMMAS(UP TO 16)
A,B,B,C,S,#EE<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE**, **DISABLE**, **CTPATH**, **GOPATH** and **TRIGGER**
OTHER CRS Plus Publications: TECHNICAL MANUAL, PATH APPLICATION NOTE

CTPATH

TOKEN: /164

FORMAT:
[Line#] **CTPATH** <PATH#>,<TEMPLATE>,<START INDEX>,<END INDEX>,<SPEED>

DESCRIPTION:

<PATH#>
Can use constants, variable names, arrays or expressions.
<TEMPLATE>
Can use template names or strings.
<START INDEX>...<SPEED>
Can use constants, variable names, arrays or expressions.

This command creates and stores a continuous path through an array of consecutive locations stored in memory. The maximum size of an array of taught locations, and therefore the maximum path length, is 256. The path is executed with the GOPATH command. The locations are entered into the command by template, name, and starting and finishing indices. The speed argument determines the path speed and is independent of the current robot speed setting. Path speed is given as a percentage of the current maximum linear speed (set with the @MAXSPD command).

When the GOPATH command is executed, it prompts for a path number. The robot then moves to the first location of the path at current robot speed in Joint Interpolated mode and executes the path at the path speed specified.

The CTPATH command can be used to store up to 8 paths in memory. Memory for the total number of path locations to be used by all stored paths must be allocated before the use of this command. This is done with the ALLOC command.

If the CTPATH command is entered in the Immediate mode with a value of '0' or a <cr> as a response to the path number, the current list of all stored paths is displayed and the prompt for the path number is re-issued.

If a path number is entered for which a path has already been programmed, the old path data is erased and the new path entered WITH NO WARNING.

The CTPATH command utilizes the Trigger table contents in the same fashion as the CPATH command.

If the CARTVEL flag is set (by the ENABLE CARTVEL command) the path will be programmed to maintain a constant linear velocity of the tool tip (TCP). Enabling the CARTVEL flag will slow down the processing somewhat.

WARNINGS:

1. In the location list, no two consecutive locations may be identical. This condition would produce a math error.

2. Due to the high memory requirements of the CTPATH command, it requires the installation of the expanded memory option.

3. CTPATH data is calculated and stored in memory using the currently active base offset and tool offset values. Subsequent changing of this data will not change the actual path stored in memory, and consequently the path may not be valid in the work space.

4. When using the cartesian locations, only those axes involved in the cartesian transformation may be included in the CTPATH All other axes must be locked out using the LOCK command or a RAPL-II error will result.

APPLICABLE MODES:
{I},{P}

EXAMPLES:

1. Having issued the command:

```
>>TEACH A,0<cr>
```

Assuming that locations A____000 through to A____010 were stored using the teach mode, we can now calculate a path through those points, at a speed of 50% of full. Save this path data in the path #1 buffer.

```
>>CTPATH
PATH NUMBER(1..8): 1,
TEMPLATE, :A,
ENTER START POINT INDEX : 0,
ENTER END POINT INDEX : 10,
ENTER PATH SPEED(1-150): 50<cr>
```

The reverse path can also be executed by reversing the indices (prompt messages not shown)

```
>>CTPATH 1,A,10,0,50<cr>
```

Variables may be used in the CTPATH command argument list as below (prompt messages not shown):

```
>>CTPATH PATHNUM,&1,FIRST,LAST,SPEED<cr>
```

The above command will use string #1 as the template, and the variables PATHNUM, FIRST, LAST, and SPEED as arguments in the list.

CROSS-REFERENCES:

RAPL-II COMMANDS: **ALLOC, CPATH, ENABLE, DISABLE, GOPATH, @MAXSPD, LOCK, TRIGGER**
OTHER CRS Plus Publications: TECHNICAL MANUAL, PATH APPLICATION NOTE

CUT

TOKEN: /145

FORMAT:

[Line#] **CUT** <STR_NUM>,<CHAR_INDEX>,<NUM_CHARS>

DESCRIPTION:

<STR_NUM>...<NUM_CHARS>
Can use constants or variable names.

This command takes the specified string, and removes NUM_CHARS of characters from the string, starting with, and including the character indicated by CHAR_INDEX.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. Assign a value to string 1, and remove 3 characters from it starting at index 5 (prompt messages not shown):

```
>>! &1 = 'LET US CHOP THIS LINE'<cr>
>>CUT  STRING NUMBER(1-4) :1, INDEX NUMBER(1-31) :5, # CHARACTERS :3<cr>
>>TYPE &1LET CHOP THIS LINE<cr>
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **PASTE**, **STRPOS**, **!**, **ENCODE**, **DECODE**, **IFSTRING** are other string related commands.

TOKEN: /147

FORMAT:

[Line#] **DECODE** <STR_NUM>,<CHAR_INDEX>,<VAR_NAME>

DESCRIPTION:

<STR_NUM>,<CHAR_INDEX>
Can use constants or variable names.

<VAR_NAME>
Can only use variable names.

This command decodes an ASCII encoded real number from the specified string number, starting at the specified character index (character position within the string). The character index takes on a value from 1 to 31.

WARNING

The letter 'E' may be a legal character as part of a number encoded in a string, as in '1E-6' (one times ten to the minus six). Thus an 'E' following a number without a space will be decoded as if it has an exponent. Use caution with this letter.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. Assigning string 1 to the value below, we can extract the real number 510 and place it into the variable AA

```
>>! &1 = 'VALUE IS 510'<cr>
>>DECODE  STRING NUMBER(1-4) :1, INDEX NUMBER(1-31) :10, VARIABLE: AA<cr>
>>TYPEV AA<cr>
510.0000
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **CUT**, **PASTE**, **STRPOS**, **!**, **ENCODE**, **IFSTRING** are other string related commands.

DELAY

TOKEN: /081

FORMAT:
[Line#] **DELAY** <TIME>

DESCRIPTION:

<TIME>
Can use constants, variable names, arrays or expressions.

This function provides a time delay for the robot. The delay is programmable in seconds, and accuracy of about +/- 30 milliseconds can be expected. A maximum delay of 65.530 seconds can be programmed per command. Longer delays must use successive DELAY commands.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **TIME**, **FINISH** (may be required after move, before **DELAY**)

TOKEN: /100

FORMAT:

DELETE <PRG_NAME>

DESCRIPTION:

<PRG_NAME>

Can only use a program name.

This command deletes the specified program from the RAPL-II program table. No check is done before completing the command.

APPLICABLE MODES:

{I},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **COPY, EDIT, RENAME**

DEPART

TOKEN: /004

FORMAT:
[Line#] **DEPART** <DISTANCE>[,S]

DESCRIPTION:

<DISTANCE>
Can use constants, variable names, arrays or expressions.

The DEPART command moves the robot from its present location by the specified distance along the tool axis. A positive distance moves the Tool Centre Point away from the point, while a negative distance moves the Tool Centre Point towards the point.

The DEPART command with the optional straight line [,S] specifier is useful when removing the tool along a straight line in order to eliminate the possibility of collision in tight spaces.

APPLICABLE MODES:
{I},{P}

EXAMPLES:

1. Approach point PICK_1 stopping +2 inches away. Decrease speed to 20%, then move to point PICK_1 Close gripper, delay for 2 seconds, then depart from PICK_1 stopping +2 inches away.

```
100  SPEED 100
110  APPRO PICK_1,2
120  SPEED 20
130  MOVE PICK_1
140  FINISH
150  CLOSE
160  DELAY 2
170  DEPART 2
```

2. Depart from PICK_1 in a straight line stopping +2 inches away.

```
170  DEPART 2,S
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **APPRO**, **TOOL** (Affects the depart direction)

TOKEN: /112

FORMAT:
[Line#] **DEVICE** <DEV#>

DESCRIPTION:

<DEV#>
Can use constants, variable names, arrays or expressions.

This command selects the device for future terminal or simple ASCII communication operations. This facility is normally used for communicating to an auxiliary serial device in the work cell other than the standard terminal device.

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. This example prints a status report to device #1. This could also be done using PRINT-type commands and no DEVICE command.

```
340 DEVICE 1
350 TYPE 'PRODUCTION REPORT'/
355 TYPE 'WIDGET TRANSFERRED :'
356 TYPEV W_CNT
357 TYPE ' '/
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **CONFIG, SERIAL DIR, LISTL, LISTP, LISTV**

DIR

TOKEN: /102

FORMAT:
[Line#] **DIR** [1]

DESCRIPTION:

The directory command lists the names of all programs currently in the program table and the size of each program in characters (bytes). This information is normally sent to device 0, the optional 1 argument will re-direct the information to device 1.

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **EDIT**, **DELETE**, and **LISTP**

TOKEN: /177

FORMAT:
[Line#] **DISABLE** [Item]

DESCRIPTION:

[Item]
Use actual parameter name from list below or can use a string containing only the parameter name.

This command turns off the corresponding system parameter. The DISABLE command resets one of the system parameters contained in the following list:

TRIGGER	Prevent trigger outputs from being performed.
OUTPUT	Prevent all digital outputs from changing according to the OUTPUT command. TRIGGERS are not affected.
ONSIG	Turn off the ONSIG trigger, but maintain the ONSIG condition.
EDIT	Disable any further modifications to stored programs.
ARM	Disable the Arm power signal, turning off the motor amplifiers. With robot arms equipped with brakes, the brakes will be engaged, and the arm will not move.
TRACE	Disable the Trace mode. Same as NOTRACE command.
FLASH	Turn off the FLASH command. Same as NOFLASH.
APC	Disable the Arm power check. Same as @NAPC monitor command.
HELP	Turn off the Help mode. Same as NOHELP.
TEACH	Turn off the teach facility, but maintain the Teach template, and the current value of the Teach counter. Same as the NOTEACH command.
CARTVEL	Use joint (not cartesian) distances to calculate path knot timing.
SLEW	Turn off SLEW (trapezoidal profile) in Joint-Interpolated motion (see chapter 3-2).
HOLD	Disable the HOLD feature without losing use of the HALT ON <Input> command.
LINENUM	When disabled, RAPL-II programs do not need line numbers except where required for program re-direction destinations.

If no ITEM is entered, then the parameter list is displayed on the terminal, along with the current parameter settings.

Several of the items in the parameter list are covered by other existing RAPL-II commands. The HELP/NOHELP, TEACH/NOTEACH commands are examples. The DISABLE command will be the standard RAPL-II parameter control function for all future versions. Thus instead of typing NOFLASH, the user should type DISABLE FLASH. It is recommended that the RAPL-II programmer follows this convention.

DISABLE (Continued)

APPLICABLE MODES:
{I},{P},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE, NOHELP, NOTRACE, NOFLASH, EDIT, NOTEACH, CTPATH @NAPC**

OTHER CRS Plus Publications: TECHNICAL MANUAL, PATH APPLICATION NOTE

TOKEN: /048

FORMAT:
[Line#] **DLOCN** <LOC_NAME,...>

DESCRIPTION:

<LOC_NAME>
Can use location names, strings containing location names, or array-type location names.

This command deletes stored locations from the location table.

WARNINGS:

If more than one location is entered in the list of locations, each location is deleted as soon as the comma is entered to terminate that entry. Using the Back-Space function to edit the command is not possible after the comma has been typed.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Delete stored location 'A'.

```
>>DLOCN LOCATION(,LOCATION,..) :A<cr>
```

2. Delete stored precision point location '#A'.

```
>>DLOCN LOCATION(,LOCATION,..) :#A<cr>
```

3. Delete stored locations 'A','B','C'.

```
>>DLOCN LOCATION(,LOCATION,..) :A,B,C<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **LISTL, HERE, POINT, @INIT**

DVAR

TOKEN: /067

FORMAT:
[Line#] **DVAR** <VAR_NAME,...>

DESCRIPTION:

<VAR_NAME>
Can only use variable names.

This command delete a variable or a list of references from the variable table. When a variable is cleared, the next time it is referenced it returns with a value of 0.

WARNINGS:

If deleting more than one variable, be aware that a variable is deleted as soon as the comma following its name is entered. Using the Back-Space function to edit the command is not possible after the comma has been typed.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Deletes variable HEIGHT.

```
>>DVAR VARIABLE(,VARIABLE,..) :HEIGHT<cr>
```

2. Deletes variables A,B,C,D,E and F.

```
>>DVAR VARIABLE(,VARIABLE,..) :A,B,C,D,E,F<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: !, LISTV, @INIT

TOKEN: /101

FORMAT:

EDIT [Prg_Name]

DESCRIPTION:

[Prg_Name]
Can only use program names.

RAPL-II provides the programmer with a line editing feature which can be used to create or modify existing programs. The EDIT command activates the Edit mode. When a program name is entered with the EDIT command, that program then becomes the target program for any program lines entered in PLE mode (see section 1-6). If a program name is issued which does not already exist in the program table, RAPL-II creates one. An error will occur if the program table is full. If no program name specifier is used, the edit mode is entered using the last edited program. A program cannot be deleted from within the Edit mode.

The editor and its commands is described in more detail in section 2-9.

All editing features can be prevented by the DISABLE EDIT command. It will stay disabled until either the ENABLE EDIT command is issued or a Teach start is done.

APPLICABLE MODES:
{I},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **LISTP**, **DELETE**, **ENABLE**, **DISABLE**. Also refer to Chapter 2-4 of this manual.

ELBOW

TOKEN: /169

FORMAT:
[Line#] **ELBOW** <UP | DOWN>

DESCRIPTION:

This command specifies the position of the elbow (joint #3) for all successive cartesian motions. Some locations taught in RAPL-II can be accessed by the robot arm with the elbow joint pointing up or down. This command specifies the direction of the elbow if there is a possibility of either approach.

The default condition is ELBOW UP. Inverted robots would use ELBOW DOWN as the default. DOWN would also be used whenever the robot is attempting to reach the work space "over its head". In the latter case, the ELBOW command is used with the REACH command.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

```
10 REACH BACK
15 ELBOW UP
20 MOVE A
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **INVERT, REACH, STATUS**

TOKEN: /176

FORMAT:

[Line#] **ENABLE** [Item]

DESCRIPTION:

[Item]

Can use parameter from list below or can use a string containing only a parameter from the list below.

This command turns on the corresponding system parameter. The ENABLE command sets one of the system parameters contained in the following list:

TRIGGER	Permit trigger outputs to be performed.
OUTPUT	Permit all digital outputs according to the OUTPUT command.
ONSIG	Turn on the ONSIG trigger, using the existing ONSIG condition.
EDIT	Enable all future the line editor (Edit mode).
ARM	Enable the Arm power signal, enabling the motor amplifiers to be turned on with the ARM POWER ON switch.
TRACE	Enable the Trace mode. Same as TRACE command.
FLASH	Turn on the FLASH command using the previous value of the FLASH interval.
APC	Enable the Arm power check. Same as @APC monitor command.
HELP	Turn on the Help mode. Same as HELP.
TEACH	Turn on the teach facility, using the previous Teach template, and the current value of the Teach counter.
CARTVEL	Use cartesian distances to calculate path knot timing.
SLEW	Turn on SLEW (trapezoidal profile) for Joint-Interpolated motion (see Section 3-2).
HOLD	Enable the HOLD feature for previously defined HALT ON <Input> command.
LINENUM	When enabled, RAPL-II programs must use line numbers.

If no ITEM is entered, then the parameter list is displayed on the terminal, along with the current parameter setting.

Several of the items in the parameter list are controlled by other existing RAPL-II commands. The HELP/NOHELP and TRACE/NOTRACE commands are examples. The ENABLE command will be the standard RAPL-II parameter control function for all future versions. Thus instead of typing TRACE the user should type ENABLE TRACE. It is recommended that the RAPL-II programmer follows this convention.

ENABLE (continued)

APPLICABLE MODES:
{I},{P},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **DISABLE, HELP, TRACE, FLASH, EDIT, TEACH, CTPATH, @APC**

OTHER CRS Plus Publications: TECHNICAL MANUAL, PATH APPLICATION NOTE

TOKEN: /148

FORMAT:

[Line#] **ENCODE** <VAR_NAME>,<STRING_NUM>[,I]

DESCRIPTION:

<VAR_NAME>

Can use variable names.

<STRING_NUM>

Can use constants or variable names.

This command encodes a value contained in the specified variable into the specified string number. The string number can be a value from 1 to 4. If the variable name does not exist, it is created, and a value of zero is encoded. A real number is encoded unless the [,I] argument is included, in which case an integer representation is encoded.

APPLICABLE MODES:

{I},{P},{M}

EXAMPLES:

1. Encode the variable A into string number 1 in real format.

```
>>! A = 4<cr>
>>ENCODE VARIABLE: A, STRING NUMBER(1-4) :1<cr>
>>TYPE &1<cr>
+4.0000
>>
```

2. Encode the variable A into string number 2 in integer format using the optional [,I] argument. (prompt messages not shown):

```
>>ENCODE A,2,I<cr>
>>TYPE &2<cr>
+4
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **CUT**, **PASTE**, **STRPOS**, **!**, **DECODE**, **IFSTRING** are other string related commands.

EXECUTE

TOKEN: /186

FORMAT:
[Line#] **EXECUTE** <MEMORY ADDRESS>

DESCRIPTION:

<MEMORY ADDRESS>
Can use constants, variable names, arrays or expressions.

This command executes a machine language program starting at the selected physical address. The program can be any sequence of 8086 assembly language instructions that is loaded in the controller RAM memory. Advanced users can implement complex functions that have been designed on other computers. Using CRS Plus software utilities, this software can be loaded into the robot memory for execution.

To ensure that a proper sequence of instructions is loaded at the specified address, the EXECUTE command examines the contents of the memory byte at the specified address plus 2. The content of this address must correspond to a "password" value. If this byte value is incorrect, the code specified will not be executed and an '018 ILLEG ARGU' error will be displayed. This feature protects the system from running illegal code which could do serious damage to the operating system.

A CRS Plus utility program is available for programmers who wish to develop compatible machine language code to run from this command. This utility generates the correct password code automatically.

WARNINGS:

1. Only users with RAPL-II and 8086 assembly programming language experience should attempt to use this command capability.

2. *USE OF THIS COMMAND WITHOUT COMPLETE KNOWLEDGE OF THE RAPL-II OPERATING SYSTEM IS NOT RECOMMENDED AND SUBSEQUENT DAMAGE WILL NOT BE COVERED UNDER WARRANTY*

CROSS REFERENCES:

Other CRS Publications: UM/TECH, UM/ACI

TOKEN: /010

FORMAT:
[Line#] **FINISH**

DESCRIPTION:

The FINISH flag prevents RAPL-II from proceding to the next instruction in a program until the current motion underway has completed.

RAPL-II normally executes the command following a motion command as soon as the parameters of the motion have been determined; normaly about 100 msec after a motion command was given. Usually, this is a good thing as the program will run much faster. Sometimes however, it is important to synchronize I/O type commands with robot motion. As an example, the robot must usually reach its final destination before the signal to close the gripper is issued. In this case the FINISH command must be used.

If two consecutive Motion commands are issued, there is an implicit FINISH between the commands. Therefore, the first command will be completed before the next is processed. In this case, no FINISH statements are needed between consecutive motion blocks. The only exception is MOTOR: consecutive MOTOR commands for different axes will not wait for each other to FINISH.

APPLICABLE MODES:
{P}

EXAMPLE:

1. In this example, a collision between the gripper and the object at PICK_1 would likely occur. The CLOSE command in line 150 will be processed shortly after the MOVE command in line 140 has started and will be complete before the arm arrives at its destination.

```
110 APPRO PICK_1,2
130 MOVE PICK_1
140 CLOSE
```

2. By inserting a FINISH command between the MOVE and CLOSE commands, the arm will reach the PICK_1 position before the CLOSE command is executed.

```
110 APPRO PICK_1,2
130 MOVE PICK_1
135 FINISH
140 CLOSE
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **OPEN**, **CLOSE**, **DELAY**, **OUTPUT**, **IFSIG** (may require a **FINISH** command).

FLASH

TOKEN: /084

FORMAT:
[Line#] **FLASH** <INTERVAL>

DESCRIPTION:

<INTERVAL>
Can only use constants.

This command permits the use of the "READY" light on the Teach pendant as a general purpose output. This feature can be used to signal an operator in cases where no terminal device is available.

The flash interval number has a valid range of 1 to 255. Each unit of the flash interval is approximately equal to 20 milliseconds.

If no flash interval value is entered, the default is the previous setting

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Flashes the Teach pendant light at an interval of 60 milliseconds

```
>>FLASH 3<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE, NOFLASH, DISABLE**

TOKEN: /023

FORMAT:
[Line#] **FREE** [1]

DESCRIPTION:

The FREE command displays the status of the user memory, along with the remaining memory to be used. The optional argument routes the information to the printer port (Device 1).

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:
1. Display Memory remaining, Enter:

```
>>FREE <cr>
ROBOT MEMORY ALLOCATION:
NUMBER OF EMPTY PROGRAMS  :  00009     MEMORY REMAINING  :  03659
NUMBER OF EMPTY VARIABLES :  00085
NUMBER OF EMPTY LOCATIONS :  00077
TOTAL KNOTS REMAINING     :  00000
RESERVED MEMORY ALLOCATED :  00000
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **ALLOC**, **NEW**

GAIN

TOKEN: /150

FORMAT:
A100 Series Robot Controller:
[Line#] **GAIN** <AXIS#>,<VALUE>

A200/400 Series Robot controller:
[Line#] **GAIN** <AXIS#>,<P | I | D>,<P_GAIN>,<I_GAIN>,<D_GAIN>

DESCRIPTION:

The GAIN command changes the software gain of one or more axis in the system. This allows the response of the robot arm or extra servo axes to be customised to suit a specific application.

The programmer can select any single axis by specifying a number between 1 and 8, or all motors can be selected by entering 0 for the axis number. Pressing ENTER immediately after typing 0 as the axis number will display the current gain settings for all motors.

A100 Series Robots (P-Type Controllers):

The GAIN command allows the operator to control the proportional gain of the servo loop. The proportional gain sets the level of the voltage to the motor amplifiers proportional to the amount of positional error that the controller sees when comparing the commanded position with the actual position as provided by the encoder feedback.

Since increasing the proportional gain can lead to instability of the servo mechanism, and possible mechanical damage, the operator should keep the GAIN value for any of the robot joints to less than or equal to one (1). The range of values acceptable in the command is a REAL value between zero (0) and two (2).

The default value of gain for all axes is 1 and this value should be unchanged for the robot arm axes for optimum robot performance.

A200/400 Series Robots (PID-Type Controllers):

Three gains may be entered into the servo loop: Proportional, Integral, and Derivative. Any combination of the three gain values can be programmed by entering the appropriate initial(s) as the 'PID' argument. Once this is entered, the three gain values must be entered in the order of P, I and D. Even though three arguments must be entered, only those gains that have been selected with the PID argument will actually be read in. This way, any values can actually appear for those gains which are not selected.

The software gains set by this command affect the digital value sent to the D/A converter. This device converts digital values to analog voltage which is sent to the amplifiers. This nominal gain of the D/A converter, DA is:

DA = 10/2048 volts/DIG_ERR (digital error)

The computation of the digital error as used in the above equation comes from the three gain factors (P, I, and D).

Proportional Gain:

The number of digits in the proportional term comes directly from the position error.

$DIG\_ERR_P = K_P \times$ position error

For example, if proportional gain is 2, and a position error of 100 pulses is indicated, the command to offset this error will be:

$$Vcom_P = \frac{2 * 100 \text{ pulse error} * 10 \text{ volts}}{2048 \text{ digits}} = 0.976 \text{ volts}$$

Integral Gain:

Integral Gain operates on a persistent input value. If a position error continues to exist, the Integral term will build up the digital input to the D/A to counteract this steady state position error. The effect of this stage is to build up the command voltage to compensate for static error such as deflection due to gravity or payload. A higher value improves the on-position accuracy of the axis.

The digital error due to the integral term comes from the current error times the integral term plus the previous error which has not been corrected since the last sample period.

$DIG\_ERR_I = (DIG\_ERR_{last}) + K_I \times POS\_ERR$

The output compensation factor applied to the motor would then be this value each elapsed period. If the static error is not reduced, the command keeps building until the maximum voltage output is reached.

Differential Gain:

Differential gain operates on the difference of the input error, and so responds to changes in the input. This type of gain creates more system damping. It is intended to reduce servo over-shoot upon achieving a position, or to smoothen the path motion of an axis. As such, the differential (or, derivative) gain term takes on the units of:

$DIG\_ERR_D = K_D \times (ERR\_POS - ERR\_POS_{last})$

GAIN (continued)

WARNINGS

A low gain will cause the arm to become less stiff. Under load it will droop and paths will not be followed as closely.

For PID-Type controllers, the use of integral gain can effectively reduce steady state servo lag, thereby increasing positioning accuracy. The use of this gain however can cause instability, so careful use is required.

Increasing the gain by more than 0.1 at a time may cause a noticeable jump in the arm position. This effect will be more noticeable under loaded conditions.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLES:
1. To display the default GAIN settings in an A100 series robot:

```
>>GAIN 0<cr>
001 +001.0000
002 +001.0000
003 +001.0000
004 +001.0000
005 +001.0000
>>
```

2. To display the default GAIN settings in an A200/400 series robot:

```
>>GAIN 0<cr>
001 P/I/D  +012.0000  +012.0000  +000.2500  +100.0000
002 P/I/D  +012.0000  +012.0000  +000.2500  +100.0000
003 P/I/D  +012.0000  +012.0000  +000.2500  +100.0000
004 P/I/D  +012.0000  +012.0000  +000.2500  +100.0000
005 P/I/D  +012.0000  +012.0000  +000.2500  +100.0000
>>
```

CROSS-REFERENCES:

OTHER CRS Plus Publications: TECHNICAL MANUAL

TOKEN: /165

FORMAT:

[Line#] **GOPATH** <PATH#>

DESCRIPTION:

<PATH#>

Can use constants, simple variable names and expressions.

This command will execute one of the 8 CTPATHs that have been entered. Once a CTPATH command has been executed, the path remains in memory until the controller is shut off.

When GOPATH command is issued, the robot will perform a Joint Interpolated move to the starting knot of the path at the current robot speed setting and will then start the continuous path at the programmed speed.

If the GOPATH command is entered in the Immediate mode with a value of '0' as a response to the path number, the current list of all stored paths will be provided, and the programmer will be asked to enter another path number selection.

APPLICABLE MODES:

{P},{I}

EXAMPLES:

1. To execute a previously defined path in the Immediate mode:

```
>>GOPATH PATH NUMBER(1..8): 1<cr>
```

2. From a program:

```
100 CTPATH 1,PNTS,1,35,70
....
500 GOPATH 1
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **CTPATH, ENABLE, DISABLE, TRIGGER**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

GOSUB

TOKEN: /074

FORMAT:

[Line#] **GOSUB** <PRG_NAME> [Parameter0,...,Parameter7]

DESCRIPTION:

<PRG_NAME>

Can be a simple program name or a string.

[Parameter0]

Can be a simple variable name, an array, an expression or an array-type location name.

Control is passed to the specified sub-program. If the sub-program is terminated with a RETURN command, program flow will return to the line following the GOSUB call when processing of the subroutine is complete. The RAPL-II processor contains a stack which stores the information needed to return to the calling program. The stack allows up to 10 nested sub-program calls.

The GOSUB command can transfer parameters to the subroutine. These parameters are 8-byte string registers which can contain names of locations, variables etc. for use inside the subroutine. In the subroutine, they are recalled by substituting a special variable name identified as '%0', '%1',..'%7' depending on their position in the list in the calling line.

There can be 8 parameters in all. The programmer may nest subroutines which make use of the same parameters.

The sub-program name can be entered as a reference to a string. This permits a 'BATCH' approach to program execution, where the operator can enter a program name as a response to a query from the system. See example #2.

APPLICABLE MODES:

{P}

WARNING:

Calling a subroutine which uses parameter within another routine using parameters must be done carefully. When a GOSUB command uses parameters, these change the previous value stored in the parameter register. Thus, referring to a parameter in the calling routine aftr the call may result in an error.

EXAMPLES:

1. Program MAIN is the calling program. Sub-program APPROACH will approach a location by some distance with the ONSIG condition activated. Note that Sub-program E_STOP uses the same parameters as APPROACH did without them needing to be re-defined.

```
50 ; PROGRAM MAIN
60 ;
100 GOSUB APPROACH POINT1,4
110 ...

PROGRAM APPROACH:
100 ONSIG 4 E_STOP
110 APPRO %0,%1
120 FINISH
130 IGNORE
140 RETURN

PROGRAM E_STOP:
100 HALT
110 WAIT -4
120 APPRO %0,%1
130 RETURN
```

2. Batch control over program execution can be an effective way to alter robot programming for new jobs. Several tasks can be loaded into the robot memory concurrently, providing decreased down time required to load new data. Program execution can then be branched to the correct job by using a string to refer to the sub-program.

```
10 TYPE 'ENTER THE JOB NAME (SWITCH, PLATE, KNOB) '
10 INPUT &1
16 TYPE &1
17 TYPE ' NOW BEING EXECUTED'/
20 GOSUB &1
99 GOTO 10
```

The above MAIN program will permit branching to one of three sub-programs. They must be called SWITCH, PLATE or KNOB. If the operator enters a name which does not match any of the selections, a 'PROGRAM NOT FOUND' error will be created at line 20. If necessary, testing for correct entry data can be made using the IFSTRING command.

3. The following example illustrates the warning above.

```
From MAIN Program:
...
2500 GOSUB CHECK 5,RESTART
...

PROGRAM CHECK:
1000 IFSIG %0 THEN 1200
1100 GOSUB TURN_OFF 3,2,7
1150 RETURN
1200 ONERR %1
1300 THIS LINE CREATES AN INTENTIONAL ERROR
$

PROGRAM TURN_OFF:
1000 OUTPUT %0,%2,%3
1100 RETURN
$
```

The above will NOT work! The call to TURN_OFF in line 1100 of CHECK changes the value in the %1 register to "2" from "RESTART" as it was before the call. The subsequent reference to %1 in line 1200 of CHECK will be in error. In this case the programmer could put line 1200 before the GOSUB call to get around the problem.

CROSS-REFERENCES:

RAPL-II COMMANDS: **RETURN, ONERR, ONSIG**

TOKEN: /075

FORMAT:
[Line#] GOTO <LINE#>

DESCRIPTION:

<LINE#>
Can be a constant, a simple variable name, an expression or an array.

An unconditional branch to another program line can be performed by this statement.

APPLICABLE MODES:
(P)

EXAMPLE:

1. Program will increment the value X and display the result to the terminal indefinitely.

```
050 ; PROGRAM MAIN
060 ;
090 ! X=0
100 ! X=X+1
110 TYPE! X
120 TYPE ''/
130 GOTO 100
```

2. RAPL-II permits programs to run without line numbers (after entering DISABLE LINENUM). For branching purposes however, line destinations are necessary. The following program illustrates this:

```
10 ! A = 0
   MOVE ^A
   ++ A
   IF A > LAST THEN 20
   GOTO 10
20 RETURN
$
```

CROSS-REFERENCES:

RAPL-II COMMANDS: IF, IFSIG, IFSTART, IFPOWER all also redirect program flow but conditionally.

GRIP

TOKEN: /039

FORMAT:
[Line#] **GRIP** <DISTANCE>

DESCRIPTION:

<DISTANCE>
Can be a constant, a simple variable name, an expression or an array.

The GRIP command sets the position of the fingers of the Servo Gripper. When the argument <DISTANCE> is less than the current finger position the fingers will close. When the argument <DISTANCE> is larger than the current finger position the fingers will open. The argument <DISTANCE> reflects the distance between the fingers and has a range from 0 to 2 inches.

The GRIP command operates at 100% force. By comparison the OPEN and CLOSE commands control the gripper force but not its position.

WARNINGS:

1. *DO NOT USE THIS COMMAND TO HOLD ON TO AN OBJECT. THIS WILL DAMAGE THE GRIPPER!* The GRIP command is intended as a way of positioning the gripper fingers. It operates at 100% force at all times.

2. REQUIRES THE SGRIP/M1 OPTION

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **OPEN**, **CLOSE**, **GTYPE**

TOKEN: /154

FORMAT:
[Line#] **HALT** [ON <[-]INPUT#>]

DESCRIPTION:

<[-]INPUT#>
Can be a constant, a simple variable name, an expression or an array.

The HALT command performs two functions:

1) If issued as a HALT<cr>, it stops any current robot motion that is in progress. This can be used in a program called from an ONSIG command that requires a robot motion to cease. In that mode, HALT would be the first command in the routine called by the ONSIG.

2) If the optional HALT ON <INPUT> is called, all robot motion will cease whenever the <Input#> is in the programmed state. Activation of this HOLD feature requires the ENABLE HOLD command to be issued.

APPLICABLE MODES:
{I},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ABORT**, **DISABLE**, **ENABLE**, **ONSIG**, **PROCEED**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

HELP

TOKEN: /095

FORMAT:
HELP

DESCRIPTION:

Turns the syntax building feature ON. In this mode, the programmer needs only to enter enough of the command to make it unique, then the syntax builder will finish "typing" the command and promt for the required arguments. With the HELP disabled, certain displays are suppressed on the terminal screen. For example, the display of the current speed during the SPEED command.

The syntax builder can also be turned on by a <Ctrl-H> from a standard terminal. This control code will not work from ROBCOMM as it will be interpreted as the Back-Space key.

WARNINGS:

Make sure to type carefully with Help mode enabled since any extra characters that are entered may be mistaken for data required later in the command line. For example typing "**MOVE**" with the HELP enabled would generate this display:

```
>>MOVE TO LOCATION : VE
```

Unless a location called VE existed, an error would result!

APPLICABLE MODES:
{I},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE**, **DISABLE**, **NOHELP**, Also see chapter 1-6 of this manual.

TOKEN: /024

FORMAT:
[Line#] HERE <LOC_NAME>

DESCRIPTION:

<LOC_NAME>
Can use a location name or an array-type location name.

Defines a location as the current commanded robot position (contrast this with the ACTUAL command). The location can be stored in cartesian (X, Y, Z, YAW, PITCH, and ROLL) or precision (motor encoder pulses) form. If ENTER is typed without specifying a location name, the current commanded position is displayed.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. For a cartesian location:

```
>>HERE LOCATION : POINT<cr>
```

2. For a precision point:

```
>>HERE LOCATION : #POINT<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: ACTUAL, DLOC, POINT, LISTL, WO

HIMEM

TOKEN: /187

FORMAT:
HIMEM [Buffer Size]

DESCRIPTION:

[Buffer Size]
Can be a constant, a simple variable name, an expression or an array.

This command permits the programmer to partition a portion of the program buffer memory for reserved use. Once the HIMEM command is entered, the program buffer will be reduced in size by the amount indicated in the HIMEM command line. The command will fail if there is insufficient free space in the program buffer.

Entering 0 as the buffer size will effectively turn off the reserved space, making it accessible to the program buffer once again.

WARNING:

If reserved memory is needed, this command must be issued each time an allocation is performed and must be done immediately after the ALLOC command.

CROSS REFERENCES:

RAPL-II COMMANDS: **FREE**, **ALLOC**

TOKEN: /020

FORMAT:
[Line#] **HOME**

DESCRIPTION:

The HOME command aligns the arm with the mathematical model of the arm stored in the controller memory. When the robot is first turned on, the controller does not know the arm position relative to the world (as defined by the centre of the manipulator base). The operator must Home the robot to provide the synchronization between controller and arm.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE

1. Enter Manual mode:

   >>MANUAL JOINT OR CYLINDRICAL MODE? JOI<cr>

   Move all five joints with the teach pendant to marks on robot starting with joint 1 through to joint 5.

   Enter HOME at keyboard. The controller will ask you to confirm that you have moved the robot arm into the starting Home range. If robot arm is in starting home range then enter Y for YES at keyboard.

   J>HOME LOCATED IN HOME BOUNDS? Y<cr>

   The controller will move each joint to the factory set home position and display the status on the terminal screen.

   If an error occurs in any of the joints return to step (1) and repeat complete procedure. If the error persists refer to the diagnostic chapter in the service manual. If you do not have a service manual or require further assistance, contact your distributor or nearest service depot.

   Visually inspect each Homing marker to ensure that the robot is homed properly. If the arm did not home correctly return to step (1) and repeat complete procedure.

CROSS-REFERENCES:

RAPL-II COMMANDS: **@@CAL**, **@LOCATE**, **XCAL**, **XHOME**

IF

TOKEN: /072

FORMAT:

[Line#] IF <VARIABLE | CONSTANT> <RELATION> <VARIABLE | CONSTANT> THEN <LINE#>

DESCRIPTION:

<VARIABLE | CONSTANT>

Can use variable names and constants without brackets. Expressions and arrays must be placed inside brackets.

<LINE#>

Can be a constant, a variable name, an expression or an array.

The boolean expression between the two arguments is evaluated using one of the six tests listed below. If a true result is obtained, then control passes to the specified line number. Logical statements can be evaluated using the following expressions:

== or EQ	equal to
!= or NE	not equal to
< or LT	less than
> or GT	greater than
<= or LE	less than or equal to
>= or GE	greater than or equal to

WARNINGS:

1. It should be noted that since all values are stored as real numbers, it is possible to obtain confusing results when using the EQ operator, since the numbers may not be exactly equal due to the round-off errors associated with the storage of such elements.

APPLICABLE MODES: {P}

EXAMPLE:

1. While the variable ROW is less than 5 this program will loop between line #90 and line #100.

```
90 ! ROW = ROW + 1
100 IF ROW < 5 THEN 90
110 STOP THE TRAY IS FINISHED
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **GOTO, IFSIG, IFSTART, IFPOWER, IFSTRING**

TOKEN: /189

FORMAT:
[Line#] IFAUX THEN <LINE#>

DESCRIPTION:

<LINE#>
Can be a constant, an array, an expression or a simple variable name.

This command branches program control depending on the state of the auxiliary input. The auxiliary input is a built-in isolated input that can be used for minimum configuration systems. A typical use for the auxiliary input is the sensor-equipped Homing Bracket option. A check can be made that will ensure that the robot is located in the bracket before a Homing sequence is issued to the arm.

APPLICABLE MODES:
{P}

WARNING:

This command requires the AUXILIARY input connection on the back of the controller chassis. This feature is not available on all systems.

EXAMPLES:

1. Utilize the IFAUX as a preliminary step in a Homing sequence:

```
10 ENABLE ARM
20 IFAUX THEN 99
30 TYPE 'ARM NOT LOCATED IN BRACKET'/
40 TYPE 'MOVE ARM INTO BRACKET AND PRESS AUTOSTART TO CONTINUE'/
50 ONSTART
60 GOTO 20
99 GOSUB HOME_ARM
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **IFSIG, IFPOWER, IFSTART, ONAUX, IORD**

IFPOWER

TOKEN: /155

FORMAT:
[Line#] **IFPOWER** THEN <LINE#>

DESCRIPTION:

<LINE#>
Can be a simple variable name, an array, a constant or an expression.

Test the sense of the arm power switch, and if it is turned on, then branch to the line specified by the command.

APPLICABLE MODES:
{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE, DISABLE, IFSIG, IFSTART, IORD, ONPOWER**

TOKEN: /073

FORMAT:
[Line#] IFSIG <[-]INPUT#,...> THEN <LINE#>

DESCRIPTION:

<[-]INPUT#>
Can use constants or simple variable names (ie. no expressions permitted).
<LINE#>
Can use constants, arrays, expressions or simple variable names.

This command branches program flow depending on the state of the specified inputs. If more than one input is entered, the state of all of the input points are logically ANDed. If any of the specified inputs do not match the required state, the test is false and program flow will continue to the next line. Should the test prove to be true, then program flow will proceed to the specified line number. Any number of inputs can be tested by this command, so long that the line length does not exceed the maximum line length permitted (128 characters).

Valid input numbers are from 1 to 96. The standard controller uses 16 user digital input points and up to 64 are available with the COMBO/32 option. The input points between 64 and 96 can be used as virtual Inputs from any 8086 assembly language program.

APPLICABLE MODES:
{P}

EXAMPLES:

1. If input#1 is at a high level, jump to Line 399, otherwise continue to next line.

```
100 IFSIG 1 THEN 399
```

2. If input#3 is at a low level, jump to Line #400, otherwise continue to next line.

```
101 IFSIG -3 THEN 400
```

IFSIG (Continued)

3. If input#1 is at a high level AND input #2 is at a low level AND input #3 is at a high level AND input #4 is at a low level AND input #5 is at a high level AND input #6 is at a low level, jump to Line #101, otherwise continue to next line.

```
120 IFSIG 1,-2,3,-4,5,-6 THEN 101
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **@DIGIO, ONSIG, IFPOWER, IFSTART**

TOKEN: /086

FORMAT:

[Line#] **IFSTART** THEN <LINE#>

DESCRIPTION:

<LINE#>

Can be a constant, an array, an expression or a simple variable name.

This is a specialized version of the IFSIG command. Here, the state of the Auto Start switch is examined. If it is in a high state, then program control is passed to the program number defined in the statement. If the input is low, then control passes to the next block in the program.

APPLICABLE MODES:

{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **IFSIG**, **IFPOWER**, **ONSTART**, **ONPOWER**

IFSTRING

TOKEN: /151

FORMAT:
[Line#] **IFSTRING** <&n> <== | EQ> <&n | 'TEXT'> THEN <LINE#>

DESCRIPTION:

<LINE#>
Any expressions or arrays must be placed in brackets. Can also use constants and variable names without brackets.

This command compares two strings using the "equal" condition only (EQ or ==). If a true result is obtained, then control passes to the specified line number.

The first argument can only be a string variable. The second argument can be a string variable, or a string of text bounded by single quote marks.

APPLICABLE MODES:
{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **GOTO**, **IF**, **IFSIG**, **IFSTART**, **IFPOWER**

TOKEN: /087

FORMAT:

[Line#] **IGNORE**

DESCRIPTION:

This command disables and clears the ONSIG state. In contrast, the DISABLE ONSIG command will disable but not clear the ONSIG state, permitting re-enabling (by ENABLE ONSIG) without re-programming.

APPLICABLE MODES:

{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ONSIG, DISABLE**

INPUT

TOKEN: /068

FORMAT:

[Line#] **INPUT** <VAR_NAME> | <&n>[,1]

DESCRIPTION:

<VAR_NAME>
Can only be a variable name.

This command permits the user to input data into the program at run time. The specified variable or string specifier indicates the destination of the input information. The operator can enter another variable name in response to a variable type INPUT command (see example 1. below).

DEVICE 1 can be used as the source of the input by using the optional ",1" argument.

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{P}

EXAMPLES:

1. Check for program continuation using a variable INPUT:

```
2100 ! N = -1
2200 ! Y =  1
2300 TYPE 'DO YOU WISH TO PROCEED? (Y/N): '
2400 INPUT RESULT
2450 TYPE ''/
2500 IF RESULT EQ Y THEN 3000
2600 IF RESULT EQ N THEN 2800
2700 GOTO 2300
2800 STOP FINISHED AS REQUESTED.
3000 ; SO CONTINUE...
```

2. Check for program continuation using a string INPUT:

```
2300 TYPE 'DO YOU WISH TO PROCEED? (Y/N): '
2400 INPUT &4
2450 TYPE ''/
2500 IFSTRING &4 EQ 'Y' THEN 3000
2600 IFSTRING &4 EQ 'N' THEN 2800
2700 GOTO 2300
2800 STOP FINISHED AS REQUESTED.
3000 ; SO CONTINUE...
```

3. Receive a text string from a serial device hooked up to DEVICE 1:

```
1100 PRINT 'S'/
1200 INPUT &1,1
1400 ; ES IS AN ERROR IN SYNTAX SO RETRY:
1500 IF &1 EQ 'ES' THEN 1000
1600 RETURN
$
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **DEVICE**, !, all string commands

INVERT

TOKEN: /170

FORMAT:

[Line#] **INVERT** <ON | OFF>

DESCRIPTION:

The robot system can be mounted upside down to provide a clear table top work space. To facilitate programming the robot in this configuration, the Z axis of the robot coordinate system may be inverted. This leaves the X and Y coordinate axes the same a before. Note that JOInt MANUAL mode and JOINT commands are not inverted. Also, when using the robot in the inverted position, the normal arm configuration is ELBOW DOWN.

As an aid to programming, the Z base OFFSET parameter can be set to equal the distance between the robot base and the work surface. This way, the origin of the inverted coordinate system will be on the work surface, which is usually more convenient.

WARNING:

Ensure that any locations taught with the INVERT ON condition are used in that fashion as well. Otherwise, unexpected results could occur. It is good programming practice to put the INVERT command at the start of any program using inverted locations.

APPLICABLE MODES

{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ELBOW, REACH**

TOKEN: /085

FORMAT:
[Line#] **IORD** <BYTE | WORD>,<ADDRESS>,<VARIABLE>

DESCRIPTION:

<BYTE | WORD>
Must type either BYTE or WORD.
<ADDRESS>
Can use constants, variable names, arrays or expressions.
<VARIABLE>
Can be a variable name or an array.

Read the byte or word value of a NEC V-30 (8086) input port. The value will be truncated by modulus 256 for byte inputs, or modulus 65536 for word values, and placed into the specified variable. Typical addresses are 0x70 for digital inputs 1 to 8, and 0x71 for digital inputs 9 to 16.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL II COMMANDS: **IOWR**, TECHNICAL MANUAL Appendix G

IOWR

TOKEN: /091

FORMAT:
[Line#] **IOWR** <BYTE | WORD>,<ADDRESS>,<VARIABLE | CONSTANT>

DESCRIPTION:

<BYTE | WORD>
Must type either BYTE or WORD.
<ADDRESS>
Can be an array, an expression, a constant or a variable name.
<VARIABLE>
Can be a constant, an array or variable name.

Write the byte or word value of the constant or variable to the NEC V-30 (8086) output ports. The value will be truncated by modulus 256 for byte outputs, or modulus 65536 for word values. Typical addresses are 0x500 for digital outputs 1 to 8, and 0x501 for digital outputs 9 to 16.

WARNING:
This command writes directly to the output circuitry, which is updated by RAPL-II every 40 mSEC. Unless the I/O interrupt is disabled any data written with this command may be lost within 40 mSEC.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: IORD, TECHNICAL MANUAL Appendix G

TOKEN: /005

FORMAT:
[Line#] **JOG** <dX>,<dY>,<dZ>

DESCRIPTION:

<dX>...<dZ>
Can use constants, variable names, arrays or expressions.

The JOG command permits the operator to move the robot by a specified cartesian increment in inches (or millimeters). The move is completed using straight line motion with the gripper flange ending with the same orientation wherever possible.

This command is useful when positioning the robot at a desired position. Once close proximity has been achieved in the Manual mode, the JOG command can be used to perform final positioning. The JOG command will execute at the last setting of the SPEED command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Move arm by +0.5 inches along the Y axis.

```
200  JOG 0,0.5,0
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **DEPART, SHIFT, SHIFTA**

JOINT

TOKEN: /006

FORMAT:
[Line#] **JOINT** <JOINT#>,<DEGREES>

DESCRIPTION:

<JOINT#>
Can use constants, variable names, arrays or expressions.
<DEGREES>
Can use constants, variable names, arrays or expressions.

The joints of the robot will move the specified distance in degrees. Optional extra axes will move the specified number of units from their current location. The desired units are selected by the @XPULSES and @XRATIO commands were executed.

Robot joints will move in a joint de-coupled fashion. The joint number de scribes which joint is moved by the following legend:

1. Waist
2. Shoulder
3. Elbow
4. Wrist Bend (pitch)
5. Wrist swivel (Roll)

The JOINT command will execute at a speed as specified in the last SPEED command. The sense of each joint can be seen in FIGURE 2-2.

WARNINGS:

1. Each robot joint has a limiting travel which should not be exceeded. This limit is checked prior to the execution of the JOINT command when the robot has been Homed.

2. Joint Interpolated moves can be done before the robot has been homed, but limits will not be checked. Care must be taken with this command until the robot is Homed.

3. The INVERT command does not affect the joint polarities.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Move Joint #1 (Base) by +45 degrees.

```
200 JOINT 1,45
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **@NOA, @ACCEL, @XPULSES, @XRATIO, @XLIMITS, @MAXVEL, LOCK, UNLOCK, INVERT**

LIMP

TOKEN: /007

FORMAT:
[Line#] **LIMP** [Axis#]

DESCRIPTION:

[Axis#]
Can use constants, variable names, arrays or expressions.

The LIMP command disengages all or some of the positional servos which maintain robot position. The robot joints can then be moved by hand. Entering 0 for axis number will LIMP all motors.

A Limp condition is terminated by the NOLIMP command. At termination of a Limp condition, the commanded position of each axis will be their current position.

Robots Without Brakes:
LIMP mode is entered automatically whenever arm power is turned OFF. The arm can then be easily moved by hand. It will also drop due to gravity, so care must be taken not to damage the arm or surrounding equipment. In this case LIMP mode will be cancelled when arm power is turned ON. The robot servos will then hold position.

Robots With Brakes:
LIMP mode can only be entered with the LIMP command while arm power is turned ON. This is because the brakes engage when arm power is OFF.

WARNINGS:

1. During Limp motions, care must be taken not to over-stress the robot arm when back-driving the motors.

2. If any joint is NOLIMPed outside its soft-stop (close to a harstop) RAPL-II will issue AXIS OUT errors for that joint whenever you try and move it or the arm.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **NOLIMP**, **ENABLE**, **DISABLE**, **@APC**, **@NAPC**, **ARM**

TOKEN: /049

FORMAT:

[Line#] **LISTL** [Loc_Name][,0 | 1]

DESCRIPTION:

[Loc_Name]
Can only be a location name.

This command lists a location stored in the robot memory. If no name is supplied, then a complete list of all locations along with their values will be provided in a tabular format.

The optional [1] argument will specify output to the printer port.

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. List all locations to video display.

```
>>LISTL LOCATION : <cr>
```

2. List location POINT to printer port.

```
>>LISTL LOCATION : POINT,1<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **HERE**, **POINT**, **DLOC**

LISTP

TOKEN: /103

FORMAT:
[Line#] **LISTP** <PRG_NAME>[,0 | 1]

DESCRIPTION:

<PRG_NAME>
Can only be a program name.

This command lists a program to the selected device. If no program name is specified, then the current program being edited will be listed.

The optional [1] argument will specify output to the printer port.

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **EDIT**

TOKEN: /069

FORMAT:

[Line#] **LISTV** [,0 | 1]

DESCRIPTION:

This command lists all the variables stored in memory and displays their values.

The optional [1] argument will specify output to the printer.

NOTE: *Output cannot be re-directed to device 1 until the ACI has been disabled with the @@RD command.*

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. List all variables to video terminal.

```
>>LISTV<cr>
```

2. List all variables to printer.

```
>>LISTV ,1<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **!, DVAR, TYPEV, TYPEI, PRINTV, PRINTI**

LOCK

TOKEN: /009

FORMAT:
[Line#] **LOCK** <JOINT#,...>

DESCRIPTION:

<JOINT#>
Can use constants, variable names, arrays or expressions.

Any joints specified in the LOCK command are excluded from any further motion commands. This allows the controller to control two separate machines which must perform non-synchronized tasks. LOCK is cancelled by UNLOCK, Ctrl-C, or the occurance of a RAPL-II error.

This command is useful if uncoordinated motion is required as in the case where the robot controller is used to control the robot and a separate piece of equipment in a work cell. If at some time during the operation, this external piece of equipment had to perform an independent motion, it can be performed without affecting the timing of the robot motions. The programmer commands the equipment to move, LOCKs those axes of motion, then commands the robot to perform its task.

WARNING:

The locked status does not apply to Manual mode.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. This program commands the robot to move to a position including a TRACK axis (axis #6) coordinate and then moves the arm only after locking the track axis:

```
...
170 MOVE MACHINE1
175 ; LOCK OUT THE TRACK FROM THE NEXT MOVE COMMANDS
180 LOCK 6
200 APPRO PICK_1,2
210 MOVE PICK_1
...
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **UNLOCK, @NOA, @ACCEL, @XPULSES, @XRATIO, @XLIMITS, @MAXVEL**

OTHER CRS Plus Publications: TECHNICAL MANUAL

MA

TOKEN: /132

FORMAT:
[Line#] **MA** <j1>,<j2>,<j3>,<j4>,<j5>

DESCRIPTION:

<j1>...<j5>
Can use constants, variable names, arrays and expressions.

This command moves the robot to the pose determined by the absolute joint angles specified, in Joint Interpolated motion. The joint angles are entered in radians.

WARNINGS:

All five angles must be entered, each separated by a comma (preferred) or a space.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **MI**

TOKEN: /042

FORMAT:
[Line#] **MAGGRIP** <%MAXIMUM>

DESCRIPTION:

Specifies the amount of force to be applied by the optional magnetic gripper. The value must be between 0 and 100.

APPLICABLE MODES:
{I},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **GGTYPE**

MANUAL

TOKEN: /045

FORMAT:
[Line#] **MANUAL** [JOI | CYL]

DESCRIPTION:

The Manual mode allows the operator to move the robot using the 8 joint selector switches on the Teach pendant, along with the gripper toggle switch and the speed selector knob.

During Manual mode, all non-motion commands remain active. Note that the system prompt changes from the usual >> after the **MANUAL** command is entered. The system console may operate slightly slower in the Manual mode due to the extra scanning which is required for the Teach pendant.

There are two Manual modes of robot control. In the Joint Manual mode, individual joints are controlled with each toggle switch on the Teach pendant. In Cylindrical Manual mode, the motion of joints 2 and 3 are coordinated to provide a motion of the wrist that is radial, with constant elevation, or vertical (in the Z axis), with constant radial distance from the robot base. The base rotate and the two wrist axes are controlled the same as in Joint Manual mode. **CYLindrical MANUAL mode cannot be used until the robot has been Homed.**

In addition to the arm motions, the arm can be limped and aligned from the Teach Pendant if less than 7 axes are installed. The following table illustrates the functions of the 8 switches at different robot configurations:

SWITCH #	JOINT mode 5 - 6 axes	CYL mode 5 - 6 axes	JOINT mode 7 - 8 axes	CYL mode 7 - 8 axes
1	JOINT 1	JOINT 1	JOINT 1	JOINT 1
2	JOINT 2	RADIUS	JOINT 2	RADIUS
3	JOINT 3	VERTICAL	JOINT 3	VERTICAL
4	JOINT 4	JOINT 4	JOINT 4	JOINT 4
5	JOINT 5	JOINT 5	JOINT 5	JOINT 5
6	JOINT 6	JOINT 6	JOINT 6	JOINT 6
7	ALIGN	ALIGN	JOINT 7	JOINT 7
8	LIMP/NOL.	LIMP/NOL.	JOINT 8	JOINT 8

TABLE 4-1 Switch functions in Manual mode

APPLICABLE MODES:
{I},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **NOMANUAL**, Section 1-6 of this manual

TOKEN: /104

FORMAT:

[Line#] **MEMRD** [Item].[Address],[Var_Name]

DESCRIPTION:

[Item]
Must type in one of the parameters from the table below.

[Address]
Can be a constant, variable name, array or expression.

This command reads the contents of memory at the physical address location specified. This address can be of the general type, where a specific physical memory address is given, or of the parameter type which refers to the RAPL-II memory parameter list. The address can be issued as a constant, or a variable name. A general memory address must be within the physical address limits of the 8086 processor (0-1048575) and can be entered in decimal or hex notation. The parameter memory type address refers to the parameter position in the parameter list for that memory type.

The contents of the memory location is returned and loaded into the variable name that is provided. The type of memory access is specified by the [Item] argument, which is a the name of one of the types listed in the table below.

General Memory Address	Parameter Memory Address	Data range
BYTE	PBYTE	0 to 255
WORD	PWORD	0 to 65535
DWORD	PDWORD	0 to 4.295×10^{9}
INTEGER	PINTEGER	-32768 to 32767
REAL	PREAL	1×10^{-38} to 1×10^{38}
DINTEGER	PDINTEGER	-2.148×10^{9} to 2.148×10^{9}
POINTER	PPOINTER	Mem. Address Pointer

WARNING:

Altering information stored in the parameter memory list may cause damage to the robot system. It should be used with extreme caution. Because of this, the parameter list is not generally available. It may be obtained from a CRS Plus distributor. To obtain the release of this information, the user may be required to sign a CRS non-disclosure agreement.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. In an program entered by the ONERR command, determine the error number knowing the address of the "Alarm Number":

```
1000 ; PROGRAM ERR_RECV:
1100 DISABLE ARM
1200 MEMRD BYTE,ERR_ADDR,ERR_NUM
1300 TYPE '*** ERROR ENCOUNTERED - '
1400 TYPEI ERR_NUM
1500 TYPE ' ***'/
1600 STOP
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **MEMWR**

OTHER CRS Plus Publications: TECHNICAL MANUAL, OPNL-UM

TOKEN: /118

FORMAT:

[Line#] **MEMWR** [Item],[Address],[Var_Name | Constant]

DESCRIPTION:

[Item]
Must type in one of the parameters from the list below.
[Address]
Can be a constant, variable name, array or expression.

This command writes a variable or constant to the contents of the memory at the address location specified. This address can be of the general type, where a specific physical memory address is given, or of the parameter type which refers to the RAPL-II memory parameter list. The address can be issued as a constant, or a variable name. A general memory address must be within the physical address limits of the 8086 processor (0-1048575) and can be entered in decimal or hex notation. The parameter memory type address refers to the parameter position in the parameter list for that memory type.

The contents of the memory location is overwritten by the value provided without querying. The type of memory access is specified by the [Item] argument, which is a the name of one of the types listed in the table below.

General Memory Address	Parameter Memory Address	Data range
BYTE	PBYTE	0 to 255
WORD	PWORD	0 to 65535
DWORD	PDWORD	0 to 4.295×10^9
INTEGER	PINTEGER	-32768 to 32767
REAL	PREAL	1×10^{-38} to 1×10^{38}
DINTEGER	PDINTEGER	-2.148×10^9 to 2.148×10^9
POINTER	PPOINTER	Mem. Address Pointer

WARNING:

Altering information stored in the parameter memory list may cause damage to the robot system. It should be used with extreme caution. Because of this, the parameter list is not generally available. It may be obtained from a CRS Plus distributor. To obtain the release of this information, the user may be required to sign a CRS non-disclosure agreement.

MEMWR (continued)

APPLICABLE MODES:
{I},{P},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **MEMRD**

OTHER CRS Plus Publications: TECHNICAL MANUAL

TOKEN: /131

FORMAT:
[Line#] **MI** <j1>,<j2>,<j3>,<j4>,<j5>

DESCRIPTION:

<j1>...<j5>
Can use constants, variable names, arrays or expressions.

This command moves the robot joints by the incremental joint angles specified, in Joint Interpolated motion. The joint angles are entered in radians.

WARNINGS:

All five angles must be entered, each separated by a comma (preferred) or a space.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **MA**

MOTOR

TOKEN: /011

FORMAT:
A100 series robot (P-Type Controller):
[Line#] **MOTOR** <MOTOR#>,<PULSES>

A200/400 series robot (PID-Type Controller):
[Line#] **MOTOR** <MOTOR#>,<PULSES>[,INPUT PORT | ONHOME | OFFHOME]

DESCRIPTION:

<MOTOR #>
Can use constants, variable names, arrays or expressions.
<PULSES>
Can use constants, variable names, arrays or expressions.

The MOTOR command is used to turn a motor shaft by the specified number of encoder pulses.

A200/400 Series Robot Controllers:

The MOTOR command has been improved to take advantage of the new axis card capabilities. The programmer may specify that a motor move terminate prematurely if the specified input attains the programmed level, or if the axis moves onto or off from the Home limit switch (using the standard HOME input to the axis card). For proper operation, it is suggested that the switch state should be stable for at least 100 milliseconds. If a momentary switch contact is used, then this will determine the maximum approach speed of the axis. Also, for such Homing sequences, it is recommended that a hardware limit switch be active on the controller. In the event that the Home switch fails to detect the travel of the axis, the hardware switch will disconnect all power to the drive motors before the hardstop is reached.

WARNING:

The MOTOR command does not observe softstop limits, and will not coordinate with other MOVE commands.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLES:
1. Execute a motor move of Motor #2 by 10000 pulses

```
>>MOTOR MOTOR (1-8) #2, (PULSES) : 10000<cr>
```

2. Execute a Motor move of the axis specified in variable N, by the number of pulses specified in variable P (prompt messages not shown):

```
>>MOTOR N,P
```

3. Generate a motor move sequence that will move the motor onto its Home switch, then will execute a Homing sequence. Issue a motor move with sufficient motor pulses to reach the Home switch.

```
10 MOTOR 2, 100000, ONHOME
20 XHOME 2
30 STOP
```

4. Move axis #2 until input #1 goes to a high state. Approach this switch at a high speed, then backup up off of the switch at a lower speed and home the axis. This sequence is a typical Homing type of sequence when large axis motions may be involved.

```
10 SPEED 100
20 MOTOR 2,100000,INPUT 1
30 SPEED 10
40 MOTOR 2, -10000, INPUT -1
50 XHOME 2
60 STOP
```

MOVE

TOKEN: /012

FORMAT:
[Line#] **MOVE** <LOC_NAME>[,S]

DESCRIPTION:

<LOC_NAME>
Can use a location name or a string.

Moves the robot to the specified location. The location can be either a cartesian coordinate, or a precision point.

A MOVE command utilizes Joint Interpolated motion so that individual robot joints are commanded to start and stop at the same time.

The straight line [,S] argument instructs the robot to move so that the tool tip follows a straight line path to the destination. The MOVE command proceeds at the previously defined speed setting.

MOVE will normally coordinate only the five axes of the robot. It will coordinate the extra axes under the following conditions:

1. More than the 5 axes are in the system and a precision point location is specified.

2. A @TRACK or @GANTRY command has been issued prior to the move. In this case all locations, cartesian and precision point include extra axis coordinates.

APPLICABLE MODES:
{I},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **APPRO, DEPART, FINISH, HERE, POINT, TOOL**

TOKEN: /021

FORMAT:

NEW

DESCRIPTION:

This command clears user memory without changing memory partitioning. A fail-safe prompt (ARE YOU SURE?) is provided to prevent accidental erasure of user memory. All programs, variables, locations and strings will be erased if a Yes response is given.

APPLICABLE MODES:

{I},{M}

EXAMPLE:

1. Clear all memory:

```
>>NEW
ARE YOU SURE?Y<cr>
>>_
```

CROSS REFERENCES:

RAPL-II COMMANDS: **ALLOC, FREE, DIR, HIMEM, LISTL, LISTV, DLOCN, DVAR, DELETE**

NEXT

TOKEN: /105

FORMAT:

NEXT [Prg_Name],[Line#]

DESCRIPTION:

[Prg_Name]
Can be a program name or a string containig a legal program name.
[Line#]
Can be a constant, variable name, array or expression.

This command executes the next step in a RAPL-II program or can be used to position the program counter at a specific point in a program. This "single-stepping" through a program can be a useful tool for debugging programs. Once a program and line count has been specified with a NEXT command, they need not be entered again, since the program line number is automatically incremented to the next line. The NEXT command can also be used after a <Ctrl-A> program abort. The PROCEED command can be used to continue program execution after a NEXT command.

NEXT can be used with TRACE enabled to provide single-stepping with a display of program status.

APPLICABLE MODES:
{I},{M}

EXAMPLE:

1. To start a program (MAIN in this example) at a point other than the first line (line 1350 in this example) type:

   >>NEXT MAIN,1350<cr>

   After that program line has been executed (check using STATUS), type:

   >>PROCEED<cr>

CROSS-REFERENCES:

RAPL-II COMMANDS: **RUN, PROCEED, TRACE, NOTRACE, ABORT**

TOKEN: /088

FORMAT:
[Line#] **NOFLASH**

DESCRIPTION:

This command turns off the FLASH command, and reinstates the error condition output to the READY light.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **FLASH**, **ENABLE**, **DISABLE**

NOHELP

TOKEN: /096

FORMAT:
NOHELP

DESCRIPTION:

This command turns the syntax building feature (Help mode) OFF. When controlling the robot through a host computer interface, the syntax builder should be turned off.

APPLICABLE MODES:
{I},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **HELP, ENABLE, DISABLE**

TOKEN: /008

FORMAT:
[Line#] NOLIMP [Axis#]

DESCRIPTION:

[Axis#]
Can use constants, variable names, arrays or expressions.

This command re-establishes closed loop servo control after a LIMP command has been issued. (Refer to description of LIMP command.) Entering a 0 for axis number will NOLIMP all motors.

When the robot arm power is off, the robot is put in the Limp mode. This is so that whenever the arm power is turned back on, the robot will assume this current position under closed loop control.

APPLICABLE MODES:
{I},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: LIMP, ENABLE, DISABLE

NOMANUAL

TOKEN: /046

FORMAT:
[Line#] **NOMANUAL**

DESCRIPTION:

This command turns off the function of the Teach Pendant. This will return the system prompt to the >> symbol. (Refer to description of MANUAL command.) Issuing any RAPL-II motion command automatically executes a NOMANUAL command.

APPLICABLE MODES:
{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **MANUAL**

TOKEN: /043

FORMAT:
[Line#] **NOTEACH**

DESCRIPTION:

This command deactivates the Teach mode. (Refer to description of TEACH command.)

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **TEACH, ENABLE, DISABLE, CTPATH, WITH**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

NOTRACE

TOKEN: /080

FORMAT:
[Line#] **NOTRACE**

DESCRIPTION:

This command disables the Trace mode. (Refer to description of TRACE command.)

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **TRACE, ENABLE, DISABLE**

TOKEN: /047

FORMAT:
[Line#] **OFFSET** [<dX>,<dY>,<dZ>,<dO>]

DESCRIPTION:

<dX>...<dO>
Can use constants, variable names, arrays or expressions.

This command permits re-definition of the origin of the world coordinate system. In the default condition, the centre of the base of the robot defines the cartesian origin. The OFFSET command sets the new origin in absolute terms relative to the default origin.

Each of the three cartesian coordinates can be changed with the OFFSET command, along with the YAW angle. All further robot commands which reference the world coordinate system will be made relative to this new coordinate system.

Issuing this command without arguments and with HELP enabled will display the current setting of the OFFSET.

OFFSET can be used to set the "Z=0" plane at the work surface in the case where the robot is suspended from a gantry frame and the INVERT command has been issued.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Two arrays of components to be picked are +2" in the X dimension, +2" in the Y dimension, and at a 45 degree to each other. A program to pick parts from both fixtures would be:

```
....
100 GOSUB GET_PRTS
150 OFFSET 2,2,0,45
200 GOSUB GET_PRTS
....
```

2. A robot is inverted on a gantry frame 28 inches above the work surface. To set the world coordinate to a Z value of 0 at the table the following is recommended:

```
100 INVERT ON
200 OFFSET 0,0,-28,0
....
```

CROSS-REFERENCES:
RAPL-II COMMANDS: **MOVE**, **SHIFT**, **INVERT**

ONAUX

TOKEN: /190

FORMAT:
[Line#] **ONAUX**

DESCRIPTION:

This command holds program flow and waits for the state of the auxiliary input to go high. The program will remain in this state until:

1) The auxiliary input state becomes true
2) An ONSIG condition is realized
or 3) Ctrl-C or ABORT is pressed.

APPLICABLE MODES:
{I},{P}

WARNING:

This command requires the AUXILIARY input connection on the back of the controller chassis. This feature is not available on all systems.

EXAMPLES:

1. Utilize the ONAUX command as a preliminary step in a homing sequence:

```
10 TYPE 'ARM NOT LOCATED IN BRACKET'/
20 TYPE 'MOVE ARM INTO BRACKET AND TURN ON ARM POWER TO CONTINUE'/
30 ONAUX
40 ENABLE ARM
50 ONPOWER
60 GOSUB HOME_SEQ
...
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **ONSTART**, **WAIT**, **ONPOWER**, **IFAUX**

TOKEN: /143

FORMAT:
[Line#] **ONERR** <PRG_NAME>

DESCRIPTION:

<PRG_NAME>
Can use a program name or a string.

This command branches program control to the start of the specified program when a RAPL-II error occurs. Normally when a error occurs during the execution of a program, the robot is halted and the program terminates. By using the ONERR command, this will cause program control to jump to the named program (PRG_NAME). This can be useful in the case of an arm power error due to the use of the ARM OFF command in a program.

APPLICABLE MODES:
{P}

EXAMPLES:

1. If an error occurs during a program, it is useful to determine what the error is. If no terminal is in the system, the message will not be displayed, and once the arm power is turned off, the last error STATUS line will always show error "040 - ARM POWER". Program GET_ERR will read the last error number from a memory address, and store it as a variable which can be examined later by a programmer. The location in memory where the error number is stored can be determined using ROBCOMM. It can be displayed using the Utilities/Memory/View menu item. In program GET_ERR, the address must be stored in a variable called ERR_ADDR:

```
100 ; IN MAIN:
200 ONERR GET_ERR
300 ....
```

```
1000 ; PROGRAM GET_ERR:
1100 MEMRD BYTE,ERR_ADDR,ERR_NUM
1200 TYPE '** ERROR ENCOUNTERED -'
1300 TYPEI ERR_NUM
1400 TYPE ' **'/
1500 DISABLE ARM
```

2. The ONERR command can be used to reset a program halted from some unpredictable point in a program operation. Due to problems with the use of the RUN command in programs (see RUN), and limits of nesting of subroutines, it is difficult to restart a main program without terminal intervention or turning the system off and re-starting from an AUTO-START procedure. One way to get around this is using the ONERR command and an intentional error.

   For example, a program called ABORT is called from an ONSIG condition when an emergency stop push button is pressed. The system waits for an operator to clear the system and reset the E-Stop switch, and then moves slowly to a SAFE location before running program MAIN:

```
PROGRAM MAIN:
...
1200 ONSIG -E_STOP,ABORT
...

PROGRAM ABORT:
1000 WAIT E_STOP
1100 SPEED SLOW
1200 DEPART SAFEDIST
1300 MOVE SAFE
1400 ONERR MAIN
1500 THIS LINE WILL CAUSE A SYNTAX ERROR!
$
```

   Note that the emergency stop button being referenced in this example is NOT the EMERGENCY STOP button on the front panel, remote panel, or REMOTE E-STOP on the rear panel. Pressing any of these will immediately turn OFF arm power. This example refers to a digital input that is being used with an ONSIG command to provide a software interrupt. This example appears in greater detail in the PATH Application note and the OPERATOR PANEL user manual.

CROSS-REFERENCES:

RAPL-II COMMANDS: **GOSUB, ONSIG, @APC, @NAPC, ENABLE, DISABLE, MEMRD**

TOKEN: /156

FORMAT:
[Line#] **ONPOWER**

DESCRIPTION:

This command holds program execution until the arm power is turned on. Unlike the ONSTART command, this command does not look for a transition of the arm power state. Thus if the arm power is on when this command is entered, control will pass to the next line.

This command is useful for auto-start routines (AUTO_ST) where the program must wait for the operator to turn on the arm power.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. The following program EXIT can be used to exit a homing bracket and HOME the robot:

```
PROGRAM EXIT:
1000 ONAUX
1100 ONPOWER
1200 SPEED SLOW
1300 JOINT 3,30
1400 HOME
1500 RETURN
$
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **IFPOWER**

OTHER CRS Plus Publications: TECHNICAL MANUAL (Chapter 9)

ONSIG

TOKEN: /090

FORMAT:
[Line#] **ONSIG** <[-]PORT#>,<PRG_NAME>

DESCRIPTION:

<[-]PORT#>
Can use constants, variable names, arrays or expressions.
<PRG_NAME>
Can use program names only.

This command sets up RAPL-II to react to an input immediately by branching to the specified sub-program.

Once the ONSIG condition has been activated, the specified input will be checked approximately every 40 milliseconds. When the input matches the specified state, program flow branches to the specified subroutine.

The input used can be any user digital input. If the input is a pulse, the pulse must be at least 100 mSEC long to be sure it is captured reliably by the system.

APPLICABLE MODES:
{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **IGNORE**, **ENABLE**, **DISABLE**, **GOSUB**, **RETURN**, **IFSIG**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

TOKEN: /092

FORMAT:
[Line#] **ONSTART**

DESCRIPTION:

This command holds program flow until the Auto Start switch is pressed. When the state of the switch changes from low to high, control is passed to the next line of the program. Note that this command waits for the transition not just for the state to be high. Thus if the switch is held pressed when the command is executed, it must be released and pressed again for program flow to continue. This prevents one trip of the switch from passing through multiple ONSTART commands in the same program.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **ONPOWER, IFSTART**

OPEN

TOKEN: /040

FORMAT:
[Line#] **OPEN** [%FORCE]

DESCRIPTION:

[%FORCE]
Can use constants, variable names, arrays or expressions.

With a pneumatic gripper installed, this command opens the gripper by switching the solenoid in the arm. The FORCE argument is ignored.

With the Servo Gripper installed, it opens the gripper at the specified force, which is a percentage of full force. If no force argument is specified, then the previous force value (entered with a previous OPEN or CLOSE command) will be used.

In Manual mode the gripper can be opened by use of the gripper switch on the Teach pendant. For servo grippers the gripping force is determined by the setting of the speed potentiometer.

APPLICABLE MODES:
{I},{P}

EXAMPLE:

1. To open an air gripper:

```
>>OPEN <cr>
```

2. To open the SERVO gripper at a torque of 60%:

```
>>OPEN GRIPPER OPENING FORCE(%): 60<cr>
```

Note that the message prompting for gripping force shows up only when the system is set up for use of the servo gripper.

WARNING:

THE SERVO GRIPPER MUST NOT BE OPENED TO ITS LIMITS OR TO GRIP A PART FROM INSIDE AT A FORCE OF MORE THAN 75% FOR EXTENDED PERIODS. Gripping at values above 75% for longer than a few seconds is un-necessary and likely to shorten the life of the gripper.

CROSS-REFERENCES:

RAPL-II COMMANDS: **CLOSE**, **GRIP**, **@@GTYPE**

TOKEN: /089

FORMAT:

[Line#] **OUTPUT** <[-]OUTPUT#,...>

DESCRIPTION:

<[-]OUTPUT#>

Can use constants, variable names, arrays or expressions.

This command sets the state of the programmable digital outputs of the controller.

Any number of output points can be turned on or off by this command. The acceptable output number ranges from 1 to 112. Numbers 1 to 16 are standard user programmable physical outputs. Numbers 17 to 64 are available with the optional COMBO/32. Output numbers from 65 to 112 are not physical outputs but can be used as virtual outputs to an 8086 assembly language program.

A sign before each output number determines the desired state of the output. A negative sign before an output channel number will cause that output to go low. The positive sign is unnecessary.

APPLICABLE MODES:

{I},{M},{P}

EXAMPLES:

1. Sets output #1 to a high level.

```
100 OUTPUT 1
```

2. Sets output #2 to a low level

```
110 OUTPUT -2
```

3. Sets outputs #1, #3 and #5 to a high level and #2 and #4 to a low level.

```
120 OUTPUT 1,-2,3,-4,5
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **TRIGGER, ENABLE, DISABLE, @DIGIO, IFSIG**

PASSWORD

TOKEN: /141

FORMAT:
PASSWORD <PASSWORD>

DESCRIPTION:

This command with the correct PASSWORD value permits access to the monitor, set-up commands and the diagnostic command level (refer to APPENDIX A of the Technical Manual for a complete list of these command). After issuing the correct PASSWORD, these commands will also be displayed when the ? command is issued.

This mode may be disabled by entering an incorrect PASSWORD.

WARNING:

After entry of the command and before the <cr> to terminate the command line, the terminal echo is disabled so that a private password entry can be made. If the operator aborts the PASSWORD command using an ABORT sequence, the terminal echo will remain disabled until re-enabled by the <Ctrl-E> or by the use of the CONFIG command.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: ?

OTHER CRS Plus Publications: TECHNICAL MANUAL

TOKEN: /146

FORMAT:

[Line#] **PASTE** <STRING1 | 'Text'>,<STRING2>,<CHAR_INDEX>

DESCRIPTION:

<CHAR_INDEX>

Arrays and expressions must be contained within brackets. Can use constants and variable names without use of brackets.

This command takes the first string, and inserts, or pastes it into the second string at the character index provided. Notice that the command line requires the string symbol and number, not just the string index.

APPLICABLE MODES:

{I},{P},{M}

EXAMPLE:

1. Assign values to strings 1 and 2. Insert string 2 into string 1 starting at character index 8.

```
>>! &1 = 'INSERT><HERE'<cr>
>>! &2 = 'SOMETHING'<cr>
>>PASTE SOURCE STRING :&2, DESTINATION STRING :&1, INDEX NUMBER(1-31) :8<cr>
>>TYPE &1
INSERT>SOMETHING<HERE
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **CUT**, **STRPOS**, **!**, **ENCODE**, **DECODE**, **IFSTRING** are other string related commands.

PAUSE

TOKEN: /076

FORMAT:

[Line#] **PAUSE** [Message String]

DESCRIPTION:

This command halts program flow, and displays the desired message on the terminal. A PROCEED command from the terminal will resume program control at the next program line.

APPLICABLE MODES:

{P}

EXAMPLE:

1. In this example, the program will halt after the MOVE command and the message 'INSERT NEW PART, THEN TYPE "PROCEED"' will appear on the screen of the terminal. When the operator enters the PROCEED command, the program will continue.

```
920 MOVE AWAY
930 PAUSE INSERT NEW PART, THEN TYPE "PROCEED"
940 APPRO PART,3
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **STOP, RUN, PROCEED**

TOKEN: /050

FORMAT:

POINT <LOC_NAME>

DESCRIPTION:

This command defines a location, and permits its current value to be changed from the terminal. A cartesian location requires six values, the X,Y,Z coordinates and Yaw, Pitch and Roll angles of the tool. The orientation angles are provided in degrees. A precision point value requires pulse encoder unit inputs for each axis of motion.

Can use constants, variable names, arrays or expressions when changing coordinates.

APPLICABLE MODES:

{I},{M}

EXAMPLES:

1. For a cartesian location (after a @GANTRY command has been issued):

```
>>POINT ENTER LOCATION NAME: PART7<cr>

NAME      X/TRACKX  Y/TRACKY      Z        YAW      PITCH     ROLL
PART7     +000.0000 +000.0000 +000.0000 +000.0000 +000.0000 +000.0000
          +000.0000 +000.0000

CHANGE? Y
X,Y,Z,Y,P,R,GANTRY-X,GANTRY-Y :
12,15.75,10.875,0,90,0,0,2<cr>
>>_
```

2. For a precision point:

```
>>POINT   ENTER LOCATION NAME: #AA<cr>
NAME       AX#1/6      AX#2/"      AX#3/8      AX#4        AX#5
#AA      +0000001200 +0000001000 +0000002000 +0000000000 +0000000212

CHANGE? Y
(M1),(M2),(M3),(M4),(M5),(M6),(M7),(M8) :
1200,1000,-2000,0,212<cr>
>>_
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **LISTL**, **HERE**, **ACTUAL**

PRINT

TOKEN: /055

FORMAT:

[Line#] **PRINT** <'text' | &n>[Argument]

DESCRIPTION:

This command sends the specified text to the printer port, device #1. The optional arguments provide additional characters to be sent to the device after the string is complete. The following are valid arguments:

/	Include a carriage return/line feed combination after the line
*	Include only a carriage return after the text.
B	Output the BELL character to the display.
F	Output a form feed to the device.

WARNINGS:

Attempting to use this command with device 1 configured as an ACI port will cause a 'Reserved I/O' error #59.

APPLICABLE MODES

{I},{M},{P}

EXAMPLE:

1. Prints "TEST" on printer.

```
>>PRINT 'TEST'<cr>
```

2. Prints "TEST" on printer with a carriage return and line feed.

```
>>PRINT 'TEST'/<cr>
```

3. Prints "TEST" on printer with a carriage return.

```
>>PRINT 'TEST'*<cr>
```

4. Print a string on the printer

```
>>PRINT &1<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **TYPE**, **PRINTI**, **PRINTV**

SET

TOKEN: /051

FORMAT:
[Line#] **SET** <LOC_NME1> = <LOC_NAME>

DESCRIPTION:

<LOC_NME1>
Can use a location name or a string.
<LOC_NAME>
Can use location names, strings or implied locations.

The SET command equates one location to another. This command is useful when a location must be modified, but its original value must also be maintained.

The SET command can be used to assign user defined locations to reserved internal robot locations. These locations refer to the current commanded position, the actual position, and the current end point of the robot. These specific locations are accessible through reserved location names. The reserved locations can be determined in cartesian or precision point values. These reserved names are:

ACTUAL, #ACTUAL -	Actual robot position registers
POSCOM, #POSCOM -	Commanded position registers
ENDPNT, #ENDPNT -	End point or destination registers for any motion. For PATH motion, this contains the last knot of the path.

Never attempt to write to these registers. If any operations are to be performed on this data, it is best assigned to a working location that is defined by the user using the SET command.

APPLICABLE MODES:
{I},{M},{P}

RUN

TOKEN: /109

FORMAT:
[Line#] **RUN** [Prg_Name][,Loop Count | F]

DESCRIPTION:

[Prg_Name]
Can use program names and strings.
[,Loop Count | F]
Can use constants, variable names, arrays or expressions.

This command executes a program from the robot memory, starting with the first statement in the program. If no program name is given, then the program that was last executing is run. If that program is a subroutine normally called by a main program, the return statement will result in an error.

A program loop counter can be included in the statement. If no loop counter is supplied, then it is assumed to be 1. The maximum loop count is 65535. Entering an 'F' (Forever) as the loop count will result in infinite looping.

WARNINGS:

The RUN command may be used in a program with the following limitations:

1) The command must not be executed as many as 50 times without causing a stack error in the system computer and causing a shutdown.

2) Returning to the Immediate mode may cause an unexpected error: probably error '012 COMMand ERROR'. The system will also have disabled the Help mode when it returns.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **RETRY**, **PROCEED**, **PAUSE**, **STOP**

OTHER CRS Plus Publications: M1A/TUTORIAL

RUN

TOKEN: /109

FORMAT:
[Line#] **RUN** [Prg_Name][,Loop Count | F]

DESCRIPTION:

[Prg_Name]
Can use program names and strings.
[,Loop Count | F]
Can use constants, variable names, arrays or expressions.

This command executes a program from the robot memory, starting with the first statement in the program. If no program name is given, then the program that was last executing is run. If that program is a subroutine normally called by a main program, the return statement will result in an error.

A program loop counter can be included in the statement. If no loop counter is supplied, then it is assumed to be 1. The maximum loop count is 65535. Entering an 'F' (Forever) as the loop count will result in infinite looping.

WARNINGS:

The RUN command may be used in a program with the following limitations:

1) The command must not be executed as many as 50 times without causing a stack error in the system computer and causing a shutdown.

2) Returning to the Immediate mode may cause an unexpected error: probably error '012 COMMand ERROR'. The system will also have disabled the Help mode when it returns.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **RETRY**, **PROCEED**, **PAUSE**, **STOP**

OTHER CRS Plus Publications: M1A/TUTORIAL

SET

TOKEN: /051

FORMAT:
[Line#] **SET** <LOC_NME1> = <LOC_NAME>

DESCRIPTION:

<LOC_NME1>
Can use a location name or a string.
<LOC_NAME>
Can use location names, strings or implied locations.

The SET command equates one location to another. This command is useful when a location must be modified, but its original value must also be maintained.

The SET command can be used to assign user defined locations to reserved internal robot locations. These locations refer to the current commanded position, the actual position, and the current end point of the robot. These specific locations are accessible through reserved location names. The reserved locations can be determined in cartesian or precision point values. These reserved names are:

ACTUAL, #ACTUAL -	Actual robot position registers
POSCOM, #POSCOM -	Commanded position registers
ENDPNT, #ENDPNT -	End point or destination registers for any motion. For PATH motion, this contains the last knot of the path.

Never attempt to write to these registers. If any operations are to be performed on this data, it is best assigned to a working location that is defined by the user using the SET command.

APPLICABLE MODES:
{I},{M},{P}

RTCLOCK

TOKEN: /191

FORMAT:
[Line#] **RTCLOCK**

DESCRIPTION:

NOTE: *This command requires the real time clock option to function.*

This command reads the current state of the MK48T12 real time clock option. The result is displayed on the terminal if the command was entered in Immediate mode and returned in string #1 if entered in a program.

The complete date, time and day of the week are returned in the format as follows:

"MO/DD/YY HH:MI:SS (DA)"

MO - month code 01-12
DD - date code 01-31
YY - year code 00-99
HH - hours code 00-23
MI - minutes code 00-59
SS - seconds code 00-59
DA - Day code 01-07 (representing Monday thru Sunday)

To use this information, the DECODE command can be used on string 1.

The clock is started and stopped and the time is set in the diagnostic mode (see @@DIAG command in APPENDIX A of the Technical Manual).

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. A certain task must start at 7:45 am of the next day. The folowing program will hold program execution until this time:

```
PROGRAM SEVEN_45:
1000 RTCLOCK
1100 DECODE 1,10,HOUR
1200 IF HOUR <> 7 THEN 1000
1300 DECODE 1,13,MINUTE
1400 IF MINUTE <> 45 THEN 1000
1500 RETURN
$
```

PRINTV

TOKEN: /057

FORMAT:
[Line#] **PRINTV** <VAR_NAME,...>

DESCRIPTION:

<VAR_NAME>
Can use constants, variable names, arrays or expressions.

This command sends the specified variables to the printer port, device #1 in Real number format. Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

WARNINGS:

Attempting to use this command with device 1 configured as an ACI port will cause a 'Reserved I/O' error #59.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Prints variable value RADIUS to printer.

```
>>PRINTV  VARIABLE(,VARIABLE,..) :RADIUS<cr>

>>_
```

2. Prints variable values RADIUS and CIRCUM to printer (prompt messages not shown):

```
>>PRINTV RADIUS,CIRCUM<cr>

>>_
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **TYPEV**, **PRINT**, **PRINTI**

RETURN

TOKEN: /077

FORMAT:
[Line#] **RETURN** [Skip#]

DESCRIPTION:

[Skip#]
Can use constants, variable names, arrays or expressions.

This command returns program control to the calling program at the end of subroutine. By using the optional line skip counter, the return can include skipping over one or more lines when returning to the calling program.

Up to 255 lines can be skipped with this function. Omitting the skip count is the same as programming a skip count of 0. That is, the next line in the main program immediately following the call to the sub-program will be executed next.

Note that the skip number refers to numbers of lines and not to the numeric value of the line numbers in the calling program.

There is a special case of the skip number which is useful when calling a routine from an ONSIG condition. A value of -1 can be used in this case to ensure a return to the exact line which was being executed at the time of the ONSIG condition being satisfied. For instance if an ONSIG is anticipated in a motion command, follow this command with a explicit FINISH command. If the ONSIG occurs, a RETURN -1 at the end of the subroutine will return to the FINISH command.

APPLICABLE MODES:
{P}

PROCEED (continued)

2. If a continuous path move is interrupted by a HALT command within a program, the PROCEED command will cause it to resume from where it stopped.

```
PROGRAM: MAIN
10 ONSIG 1,ERR
20 CPATH A,B,C,D,E,F,G,H,J,K,L,M,N
30 FINISH
40 STOP
$

PROGRAM: ERR
10 HALT
20 DEPART 2
30 MOVE SAFE
40 WAIT -1
45 ; MOVE BACK TO THE PATH AND CONTINUE:
50 PROCEED
60 RETURN -1
$
```

The ERR sub-program will be triggered when input #1 goes high. The robot will immediately stop, depart from the path, and move to a safe location. There it will wait for the signal that everything is clear, do a Joint Interpolated move back to the location at which it was interrupted and resume the path.

When programming this type of sequence, ensure that the location SAFE is accessible from any point along the path. If this is not the case, a collision may result during the retreat or the Joint Interpolated move back to the path. Also ensure that no path-type moves are executed before the PROCEED command or the path pointer will be cleared. This will prohibit the PROCEED from functioning as desired. NOTE THAT ANY STRAIGHT MOVE IS A PATH-TYPE COMMAND.

CROSS-REFERENCES:

RAPL-II COMMANDS: **PAUSE, RETRY, CPATH, GOPATH, NEXT**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

RENAME

TOKEN: /107

FORMAT:

RENAME <OLD PRG_NAME>,<NEW PRG_NAME>

DESCRIPTION:

<OLD PRG_NAME>,<NEW PRG_NAME>
Can only use program names.

This command changes the name of a program already stored in memory. This function can be used when loading a new program through ROBCOMM. If an existing program has the same name, an error will result. Renaming the old program beforehand will eliminate this problem.

APPLICABLE MODES:
{I},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **COPY**, **EDIT**, **DELETE**

READY

TOKEN: /015

FORMAT:
[Line#] **READY**

DESCRIPTION:

This command moves arm to READY position shown in FIGURE 4-1 below.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Move arm to READY position shown in FIGURE 4-1.

```
>>READY <cr>
```

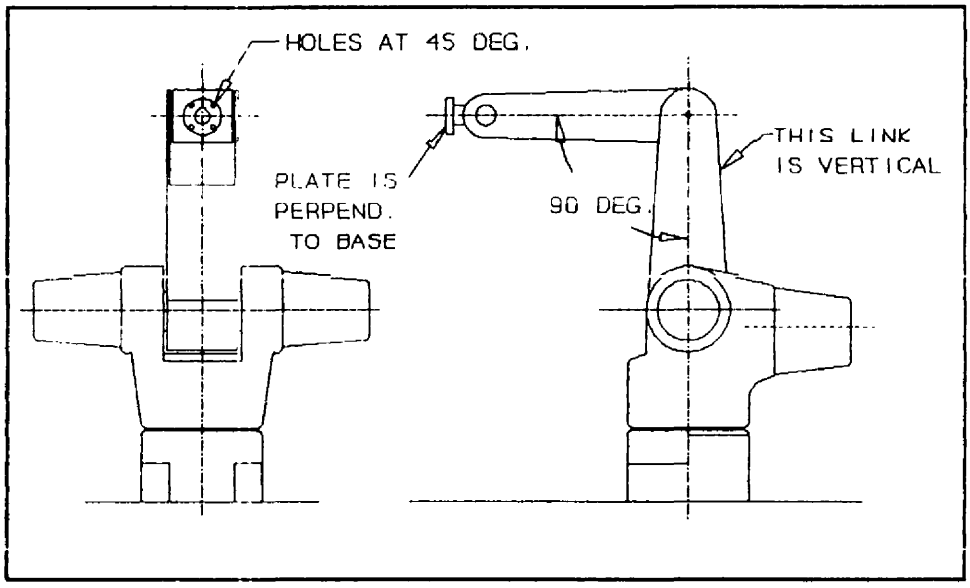


FIGURE 4-1 Arm at READY Position

CROSS-REFERENCES:

RAPL-II COMMANDS: **HOME**, **CAL**, **XREADY**

OTHER CRS Plus Publications: TECHNICAL MANUAL

READY

TOKEN: /015

FORMAT:
[Line#] **READY**

DESCRIPTION:

This command moves arm to READY position shown in FIGURE 4-1 below.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Move arm to READY position shown in FIGURE 4-1.

```
>>READY <cr>
```

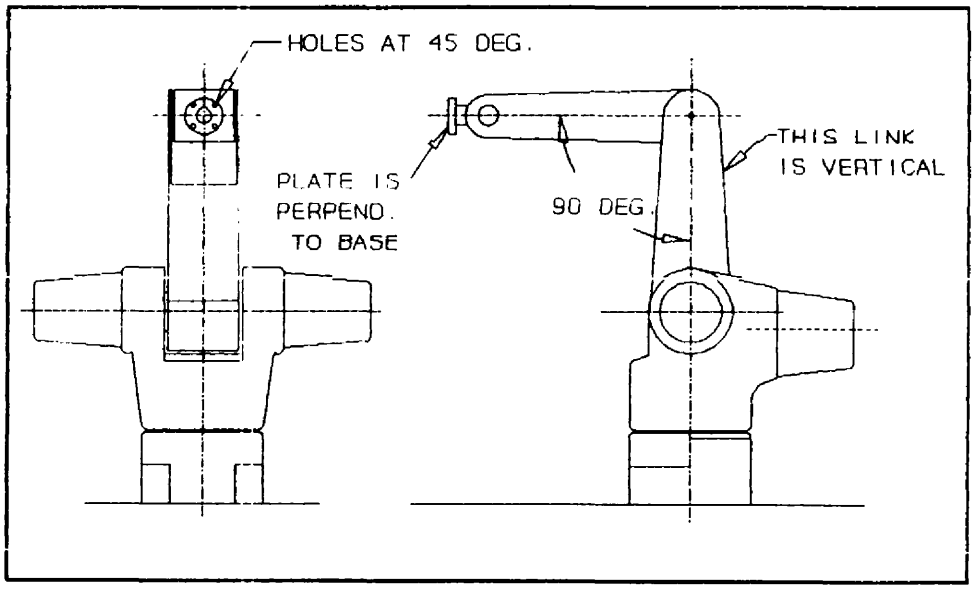


FIGURE 4-1 Arm at READY Position

CROSS-REFERENCES:

RAPL-II COMMANDS: **HOME**, **[redacted]CAL**, **XREADY**

OTHER CRS Plus Publications: TECHNICAL MANUAL

RENAME

TOKEN: /107

FORMAT:

RENAME <OLD PRG_NAME>,<NEW PRG_NAME>

DESCRIPTION:

<OLD PRG_NAME>,<NEW PRG_NAME>
Can only use program names.

This command changes the name of a program already stored in memory. This function can be used when loading a new program through ROBCOMM. If an existing program has the same name, an error will result. Renaming the old program beforehand will eliminate this problem.

APPLICABLE MODES:
{I},{M}

CROSS-REFERENCES:

RAPL-II COMMANDS: **COPY**, **EDIT**, **DELETE**

PROCEED (continued)

2. If a continuous path move is interrupted by a HALT command within a program the PROCEED command will cause it to resume from where it stopped.

```
PROGRAM: MAIN
10 ONSIG 1,ERR
20 CPATH A,B,C,D,E,F,G,H,J,K,L,M,N
30 FINISH
40 STOP
$

PROGRAM: ERR
10 HALT
20 DEPART 2
30 MOVE SAFE
40 WAIT -1
45 ; MOVE BACK TO THE PATH AND CONTINUE:
50 PROCEED
60 RETURN -1
$
```

The ERR sub-program will be triggered when input #1 goes high. The robot will immediately stop, depart from the path, and move to a safe location. There it will wait for the signal that everything is clear, do a Joint Interpolated move back to the location at which it was interrupted and resume the path.

When programming this type of sequence, ensure that the location SAFE is accessible from any point along the path. If this is not the case, a collision may result during the retreat or the Joint Interpolated move back to the path. Also ensure that no path-type moves are executed before the PROCEED command or the path pointer will be cleared. This will prohibit the PROCEED from functioning as desired. NOTE THAT ANY STRAIGHT MOVE IS A PATH-TYPE COMMAND.

CROSS-REFERENCES:

RAPL-II COMMANDS: **PAUSE, RETRY, CPATH, GOPATH, NEXT**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

RETURN

TOKEN: /077

FORMAT:
[Line#] **RETURN** [Skip#]

DESCRIPTION:

[Skip#]
Can use constants, variable names, arrays or expressions.

This command returns program control to the calling program at the end of subroutine. By using the optional line skip counter, the return can include skipping over one or more lines when returning to the calling program.

Up to 255 lines can be skipped with this function. Omitting the skip count is the same as programming a skip count of 0. That is, the next line in the main program immediately following the call to the sub-program will be executed next.

Note that the skip number refers to numbers of lines and not to the numeric value of the line numbers in the calling program.

There is a special case of the skip number which is useful when calling a routine from an ONSIG condition. A value of -1 can be used in this case to ensure a return to the exact line which was being executed at the time of the ONSIG condition being satisfied. For instance if an ONSIG is anticipated in a motion command, follow this command with a explicit FINISH command. If the ONSIG occurs, a RETURN -1 at the end of the subroutine will return to the FINISH command.

APPLICABLE MODES:
{P}

PRINTV

TOKEN: /057

FORMAT:
[Line#] **PRINTV** <VAR_NAME,...>

DESCRIPTION:

<VAR_NAME>
Can use constants, variable names, arrays or expressions.

This command sends the specified variables to the printer port, device #1 in Real number format. Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

WARNINGS:

Attempting to use this command with device 1 configured as an ACI port will cause a 'Reserved I/O' error #59.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Prints variable value RADIUS to printer.

```
>>PRINTV  VARIABLE(,VARIABLE,..) :RADIUS<cr>

>>_
```

2. Prints variable values RADIUS and CIRCUM to printer (prompt messages not shown):

```
>>PRINTV RADIUS,CIRCUM<cr>

>>_
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **TYPEV, PRINT, PRINTI**

RTCLOCK

TOKEN: /191

FORMAT:
[Line#] **RTCLOCK**

DESCRIPTION:

NOTE: *This command requires the real time clock option to function.*

This command reads the current state of the MK48T12 real time clock option. The result is displayed on the terminal if the command was entered in Immediate mode and returned in string #1 if entered in a program.

The complete date, time and day of the week are returned in the format as follows:

"MO/DD/YY HH:MI:SS (DA)"

MO - month code 01-12
DD - date code 01-31
YY - year code 00-99
HH - hours code 00-23
MI - minutes code 00-59
SS - seconds code 00-59
DA - Day code 01-07 (representing Monday thru Sunday)

To use this information, the DECODE command can be used on string 1.

The clock is started and stopped and the time is set in the diagnostic mode (see @@DIAG command in APPENDIX A of the Technical Manual).

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. A certain task must start at 7:45 am of the next day. The folowing program will hold program execution until this time:

```
PROGRAM SEVEN_45:
1000 RTCLOCK
1100 DECODE 1,10,HOUR
1200 IF HOUR <> 7 THEN 1000
1300 DECODE 1,13,MINUTE
1400 IF MINUTE <> 45 THEN 1000
1500 RETURN
$
```

EXAMPLE:

1. A program used to halt any Joint Interpolated motion is below. Typically this would be used with a light curtain input (denoted LC here). It uses the SET command to determine where the robot is headed when interrupted:

```
PROGRAM STOP:
100 SET DEST = ENDPNT
200 HALT
300 WAIT -LC
400 MOVE DEST
500 RETURN -1
$
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **HERE, POINT, SHIFT, SHIFTA**

OTHER CRS Plus Publications: PATH APPLICATION NOTE, PALLET APPLICATION NOTE

SHIFT

TOKEN: /052

FORMAT:
[Line#] **SHIFT** <LOC_NAME> BY <dX>,<dY>,<dZ>

DESCRIPTION:

<LOC_NAME>
Can use location names, strings or implied locations.
<dX>...<dZ>
Can use constants, variable names, arrays or expressions.

This command permits altering of the X, Y and Z coordinates of a cartesian location by the specified amounts. A precision point location cannot be changed using this command.

The word "BY" must appear in the command when used in a program. In immediate mode with HELP enabled, RAPL-II prompts the user only for the name of the location and the coordinates. It is not necessary to type the word "BY".

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. SHIFT point 'P' by 1,1,0

```
120 SHIFT P BY 1,1,0
```

2. SHIFT point 'A' by .002,1.2,-0.5

```
>>! DX = 0.002<cr>
>>! DY = 1.2<cr>
>>! DZ = -.5<cr>
...
160 SHIFT A BY DX,DY,DZ
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **SET**, **SHIFTA**, **HERE**

OTHER CRS Plus Publications: PALLET APPLICATION NOTE

TOKEN: /053

FORMAT:

[Line#] **SHIFTA** <LOC_NAME> BY <dX>,<dY>,<dZ>,<dO>,<dA>,<dT>[,<DTrkX>,<DTrkY>]

DESCRIPTION:

<LOC_NAME>
Can use location names, strings or implied locations.
<dX>...<dT>
Can use constants, variable names, arrays or expressions.

This command is similar to the SHIFT command, but permits all coordinates of a cartesian location to be changed. For cartesian locations, the X, Y, and Z coordinates are in inches, while the tool angles are in degrees. A precision point location cannot be changed using this command.

The word "BY" must appear in the command when used in a program. In the immediate mode with HELP enabled, RAPL-II prompts the user only for the name of the location and the coordinates. It is not necessary to type the word "BY".

WARNINGS:

All coordinates must be entered. If a @TRACK or @GANTRY has been specified, the corresponding axis values for these must also be entered in the units which are specified for the axis in question.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **SHIFT**, **SET**

OTHER CRS Plus Publications: PALLET APPLICATION NOTE

SPEED

TOKEN: /016

FORMAT:
[Line#] **SPEED** <VALUE>

DESCRIPTION:

<VALUE>
Can use constants, variable names, arrays or expressions.

This command changes the speed for all subsequent motions (except the GOPATH command).

The value of the speed corresponds to a percentage of full speed. The range permitted is from 1% to 150% of full speed. For Joint Interpolated motion, 100% is the maximum speed at which the robot will move in a coordinated fashion from any one programmable point to any other programmable point. For straight line motion, 100% is the linear speed entered by the @MAXSPEED command and displayed in the STATUS display.

In immediate mode, with HELP on, the current speed is displayed when the SPEED command is entered. This is the easy way of determining the current speed setting.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **STATUS**, All motion commands

OTHER CRS Plus Publications: TECHNICAL MANUAL

TOKEN: /025

FORMAT:

[Line#] **STATUS** [1]

DESCRIPTION:

This command outputs a status display of many important robot operation parameters to the current default device.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Display status, Enter:

```
>>STATUS <cr>
```

response:

```
RAPL-2 VERSION 1.00     M1A ROBOT   M2 CONTROLLER
LAST ERROR CODE: 040-ARM POWER
SPEED (% OF FULL) :   +100
MAX LINEAR SPEED: +012.0000
CURRENT PROGRAM EXECUTING: MAIN       LINE NUMBER: 01000
PROGRAM LOOPS EXECUTED : 65535  PROGRAM LOOPS REMAINING : 00001
CURRENT PROGRAM EDITING: MAIN2
JOINT LIMIT CHECK IS ON
CONTROLLER IS HOMED
TOOL TRANSFORM
NAME          X         Y         Z        YAW      PITCH     ROLL
NULL      +000.0000 +000.0000 +000.0000 +000.0000 +000.0000 +000.0000
GRIPPER TYPE:  AIR
AXIS#         1 2 3 4 5
LOCK          N N N N N
DONE          Y Y Y Y Y
LIMP          N N N N N
HOMED         Y Y Y Y Y
OK            N N N N N

ROBOT CONFIGURATION IS UPRIGHT/REACH FORWARD/ELBOW UP
MANUAL MODE STATUS IS OFF
ENGLISH UNITS
```

CROSS-REFERENCES:

OTHER CRS Plus Publications: TECHNICAL MANUAL

STOP

TOKEN: /078

FORMAT:
[Line#] **STOP** [Message String]

DESCRIPTION:

This command terminates program operation, permitting robot motions to be completed normally. It must be located at the end of a program, unless an ABORT command is used. If it is not present, then an error will occur.

The message string is useful for informing the operator when a task is complete

APPLICABLE MODES:
{P}

EXAMPLE:

1. The result of this portion of code is to terminate the program and print the message 'THAT IS ALL' on the terminal screen.

```
970 MOVE AWAY
990 STOP THAT IS ALL
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **PAUSE, RUN, RETRY, PROCEED**

TOKEN: /149

FORMAT:

[Line#] **STRPOS** <STR_NUM>,<'TARGET STRING'>,<VAR_NAME>

DESCRIPTION:

<STR_NUM>
Can use constants or variable names.
<VAR_NAME>
Can use constants or variable names.

This command returns the character index of the start of the TARGET STRING in the string identified by STR_NUM if it appears there. The TARGET STRING must be an assigned text string, bounded by single quotes. If no match exists for the TARGET STRING, a value of 0 is returned in the variable. Otherwise, a value of from 1 to 32 is returned.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. Assign string 1 with the following data. Pick the index for the target string 'KK' and assign it to variable AA. Print it out.

```
>>! &1 = 'FIND THE KK STRING'<cr>
>>STRPOS STRING NUMBER(1-4) :1, SOURCE STRING :'KK', VARIABLE: AA<cr>
>>TYPE! AA<cr>
+10
>>
```

2. Refer to the second example under the RTCLOCK command.

CROSS-REFERENCES:

RAPL-II COMMANDS:**CUT**, **PASTE**, **!**, **ENCODE**, **DECODE**, **IFSTRING** are other string related commands.

TEACH

TOKEN: /044

FORMAT:
[Line#] **TEACH** <TEACH TEMPLATE>[,Index,Number,Insert | Pack]

DESCRIPTION:

<TEACH TEMPLATE>
Must be a template name.
[Index,Number]
Can use constants, variable names, arrays or expressions.

This command enables the function of the Teach button on the pendant for recording robot positions. Location names will consist of the template and index merged into an 8-character string. The index is incremented by one each time the Teach button is pressed. Pressing the TEACH button creates an array of locations which are "related" by their templates.

Entering an [index] in the command string permits entry of a new starting index. If not included, the index will continue from the previous index used. The maximum index value is 255. After 255, the index will reset to 0. If previous locations exist with the same effective name as the template and the 'index' value, they will be overwritten when the Teach button is pressed.

Entering a value for the 'Number' of teach points will limit the number of points that will be recorded, and the Teach mode will automatically turn off afterwards. If this option is not included, then the programmer will be able to record points until the index reaches 255, after which it will "wrap around" back to 0.

The INSERT option will permit new locations to be inserted into an existing location array. The 'index' entered will be the starting index and the 'number' argument is used to specify the number of location entries to insert. All locations in the array with indeces greater than or equal to 'index' will have their indeces shifted upwards by 'number'.

The PACK option ignores the value of 'index' and 'number' and packs the existing location array into contiguous index numbers. This way, a location array that has had locations deleted after the Teach session can be PACKed, and then used for CTPATH operations. The PACK operation is automatically performed when the INSERT option is provided. This can be a useful function when teaching locations for a CTPATH command as no "gaps" in the path array will be permitted.

The Teach mode remains active until a NOTEACH or DISABLE TEACH command is issued. ENABLE TEACH will re-establish Teach button function but not permit changing the template or index values.

WARNINGS:

1. Pressing the Teach button in the Teach mode causes a message string to be displayed. Since the Teach button has a higher priority than all other normal screen update functions, this message will interrupt dynamic displays such as the one generated by the W1 command.

2) Using the TEACH button without the use of the 'INSERT' option, will overwrite previously taught location data if the template and counter are the same. Care should be taken to disable the TEACH button when not in use.

3) The WITH command uses the same internal register for storage of the template as the TEACH command does. Thus the use of either command will overwrite the contents placed in that register by the other command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLES:

1. Teach three points, starting at index #10. Enter:

```
>>TEACH TEMPLATE, :PICK, COUNT :10,
NUMBER OF LOCATIONS: 3<cr>
```

Press Teach button at the desired three locations. Response:

```
>>
POINT SAVED :
PICK_010
>>
POINT SAVED :
PICK_011
>>
POINT SAVED :
PICK_012
TEACH MODE OFF
```

2. Then enter a new template:

```
>>TEACH TEMPLATE, :NEW<cr>
```

Press Teach button at the desired location. Response:

```
>>
POINT SAVED :
NEW__013
>>
POINT SAVED :
NEW__014
>>
```

3. Assume that the complete location array NEW__000 to NEW__010 exists. We wish to insert 2 locations, starting at index 6 into this array. Enter:

```
>>TEACH TEMPLATE, :NEW,COUNT :6,
NUMBER OF LOCATIONS: 2,
INSERT/PACK INSERT<cr>
INSERTING...
>>
```

There will be a short time delay while the "INSERTING..." message is displayed. During this time, the system is renaming locations in the array. Now press the Teach button at the desired locations. Response:

```
>>
POINT SAVED :
NEW__006
>>
POINT SAVED :
NEW__007
TEACH MODE OFF
>>
```

The location array will now consist of locations NEW__000 to NEW__012 inclusive. The old locations that were NEW__006 to NEW__010 are now renamed to NEW__008 to NEW__012.

4. Assume now that the location array NEW__000 to NEW__012 contains one bad location NEW__002. This is removed with the DLOCN command, then the array must be PACKed so that subsequent CTPATHs can be used (prompt messages not shown):

```
>>DLOCN NEW__002<cr>
>>TEACH NEW,0,0,PACK<cr>
```

response will be:

```
PACKING....
>>
```

The data is now packed into the array NEW__000 to NEW__011 inclusively.

CROSS-REFERENCES:

RAPL-II COMMANDS: **WITH, DLOCN, LISTL, FREE**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

TOKEN: /082

FORMAT:
[Line#] **TIME** <VAR_NAME>

DESCRIPTION:

<VAR_NAME>
Can use variable names only.

This command reads the system timer clock register. The value returned will be stored in real format in the prescribed variable register. The system timer clock ticks roughly every 40 milliseconds.

The counter is reset only by a TEACH START (see Service Manual).

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. To determine the time taken for a certain move:

```
...
440 FINISH
450 TIME START
460 MOVE PLACE
470 FINISH
480 TIME FINISH
490 ! TIME = FINISH - START
495 ; TO GET THE VALUE IN SECONDS, MULTIPLY BY 0.041 SECONDS/TICK
500 ! TIME = TIME * .041
510 TYPE 'IT TOOK '
520 TYPEV TIME
530 TYPE ' SECONDS FOR THE MOVE.'/
540 ...
```

For realistic numbers, the tokenized version of the code would be better as the time to interpret the code would be much less.

CROSS-REFERENCES:

RAPL-II COMMANDS: **DELAY**

OTHER CRS Plus Publications: TECHNICAL MANUAL

TOOL

TOKEN: /054

FORMAT:
[Line#] **TOOL** <TRANSFORM>

DESCRIPTION:

<TRANSFORM>
Can use location names or strings.

This command allows users to program the robot to use different tools with varying geometries (see section 2-4).

Because of varying tool geometries RAPL-II includes a TOOL command. The TOOL command is a transform which indicates where the end of the tool that is being used is in relation to the origin of the tool coordinate system (see chapter 2-4). The TOOL command uses a six variable transform (X, Y, Z, YAW, PITCH, ROLL) to define this relationship. Thus, the transform for example 1 below, would indicate that the Tool Centre Point (the end of the tool being used) is located on the X-axis, 2 units (inches or millimetres) away from the origin.

For initiation of the TOOL transform, there is a standard non-defined transform called NULL, which has coordinates: 0,0,0,0,0,0. Thus to clear the tool transform, enter "TOOL NULL".

In immediate mode, with HELP on, the current tool transform is displayed when the TOOL command is entered. This is the easy way of determining the current setting. Otherwise, this information can be found in the STATUS display.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. To enter a TOOL transform for a gripper with a Tool Centre Point 2 inches from the mounting flange. Use the POINT command to set up a location (defined and stored in the same way as a tool transform) called GRIP_1 to 2,0,0,0,0,0 (prompt messages for POINT command not shown). Then issue the TOOL command for that transform:

```
>>POINT GRIP_1 2,0,0,0,0,0<cr>
>>TOOL TRANSFORM :GRIP_1<cr>
```

2. To check the current transform:

```
>>TOOL TRANSFORM :<cr>
TOOL TRANSFORM
NAME          X         Z         Z        YAW      PITCH     ROLL
GRIP_1    +002.0000 +000.0000 +000.0000 +000.0000 +000.0000 +000.0000

>>_
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **POINT, SHIFT, OFFSET, APPRO, DEPART**

OTHER CRS Plus Publications: RAPL-II MANUAL section 2-4, TECHNICAL MANUAL

TRACE

TOKEN: /079

FORMAT:

TRACE <LINES | NOLINES | TOKENS | NOTOKENS | LEVEL | NOLEVEL>

DESCRIPTION:

This command causes the program and line number being executed to be displayed at the current default device. This is useful when tracing through programs and subroutines for debugging purposes.

Due to the ability of the controller to process a non-motion commands during a move, the line number on the screen is usually the next line after the one being executed. When two consecutive motion commands occur, the line of the second will appear during the execution of the first. This may lead to some confusion.

The arguments available through the TRACE command will provide the programmer with a variety of tools to debug program execution. Selecting the LINES option, the data in each program line will be displayed along with the program name and line number. NOLINES will turn this feature off. Selecting the TOKENS option, any tokenized commands will be displayed as full commands in the Trace display. This makes debugging of tokenized programs much easier. Selecting no options in the TRACE command will retain the currently active settings of these options. The NOTRACE command turns the Trace mode off, but does not change the setting of these options.

Selecting the LEVELS option will display a two digit code at the left of the display that will indicate the current nesting level of the RAPL-II program. This is a useful feature when debugging programs that utilize subroutine operations extensively.

APPLICABLE MODES:

{I},{M}

EXAMPLE:

The program TEST looks like this. Remember that the token number of 078 refers to the STOP command.

```
PROGRAM TEST:
10 MOVE A
20 MOVE B
30 /78
```

1) To trace the program execution for TEST, Enter:

```
>>TRACE <cr>
>>RUN TEST<cr>
```

the RAPL-II response will be:

```
TEST#00010
TEST#00020
TEST#00030
>>
```

2) To trace the program execution and also to show line numbers, enter:

```
>>TRACE LINES<cr>
>>RUN TEST<cr>
```

the RAPL-II response will be:

```
TEST#00010 MOVE A
TEST#00020 MOVE B
TEST#00030 /78
>>
```

3) To provide de-tokenized commands to clarify the program execution:

```
>>TRACE TOKENS<cr>
>>RUN TEST<cr>
```

the RAPL-II response will be:

```
TEST#00010 MOVE A
TEST#00020 MOVE B
TEST#00030 STOP
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **NOTRACE, ENABLE, DISABLE, NEXT**

TRIGGER

TOKEN: /175

FORMAT:
[Line#] **TRIGGER** [<TRIG NUM>,<[+/-]OUTPUT NUM>,<LOCATION>]

DESCRIPTION:

<TRIG NUM>,<[+/-]OUTPUT NUM>
Can use constants, variable names, arrays or expressions.
<LOCATION>
Can use location names, strings or implied locations.

This command sets up or displays the system Trigger table. Triggers are used in conjunction with CPATH and CTPATH commands, activating outputs when the robot passes through points in a path. This feature permits digital output states to be changed without interrupting motion. Using the - character before the output number will specify a low level for that particular output, similar to the OUTPUT command. Due to the internal architecture of RAPL-II, timing of the output is only accurate to within 40 milliseconds of passing the specified location.

The TRIGGER table can contain up to 8 entries. A complete entry in the table consists of location name, output number and the state to which the output will be changed at the specified location. Locations can be precision points or cartesian locations.

On power up, the TRIGGER table is cleared and the TRIGGER flag is disabled. It is necessary to set up the trigger table in any program which makes use of this function. Triggers will not be active until after the ENABLE TRIGGER command has been executed. The TRIGGER table must be set up before calling the path command intending to use the triggers; the CTPATH or CPATH commands.

Entering a 0 for the TRIGGER number will place the new entry in the first vacant spot in the TRIGGER table. Entering a 0 for the output number will clear that TRIGGER table entry. Entering a '0<cr>' after the command will provide a display of the current TRIGGER settings.

APPLICABLE MODES:
{P},{I},{M}

EXAMPLES:

1. To set a TRIGGER table entry:

```
>>TRIGGER 1,3,POINT1<cr>
```

   This will turn output 3 high when the robot passes POINT1 in any path in which it appears.

2. To set a trigger at the next available spot in the table (prompt messages not shown):

```
>>TRIGGER
TRIGGER #(1-8), 0 FOR NEXT ENTRY, OR <CR> FOR TRIGGER LIST: 0,
OUTPUT #+/-(1-16) OR 0 FOR CLEAR ENTRY: -3,
TRIGGER LOCATION: POINT2<cr>

>>CPATH POINT1,POINT2,POINT3<cr>
```

   This will turn output 3 off at POINT2.

3. To display the current TRIGGER table:

```
>>TRIGGER
TRIGGER #(1-8), 0 FOR NEXT ENTRY, OR <cr> FOR TRIGGER LIST: <cr>
********* TRIGGER LIST ***********
TRIG# OUTPUT# LOCATION  STATE
001   003     POINT1      1
002   003     POINT2      0
003**** NO VALID TRIGGER LOADED *****
004**** NO VALID TRIGGER LOADED *****
005**** NO VALID TRIGGER LOADED *****
006**** NO VALID TRIGGER LOADED *****
007**** NO VALID TRIGGER LOADED *****
008**** NO VALID TRIGGER LOADED *****
```

4. To clear the first entry (prompt messages not shown):

```
>>TRIGGER 1,0,POINT1<cr>
```

5. To enable the trigger for any subsequent path moves (prompt messages not shown):

```
>>ENABLE TRIGGER<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **CTPATH, OUTPUT, ENABLE, DISABLE**

OTHER CRS Plus Publications: PATH APPLICATION NOTE

TYPE

TOKEN: /058

FORMAT:
[Line#] TYPE <'text' | &n>[Argument]

DESCRIPTION:

This command sends the specified text to device 0 (TERMINAL PORT). The optional arguments provide additional characters to be sent to the device after the string is complete. The following are valid arguments:

/	Include a carriage return/line feed combination after the text
*	Include only a carriage return after the text.
B	Output the BELL character to the display.
F	Output a form feed to the device.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

The TYPE command is not normally used in immediate mode. These examples are as seen in programs:

1. Type TEST on terminal

```
100 TYPE 'TEST'
```

2. Type TEST on terminal with a carriage return and line feed.

```
102 TYPE 'TEST'/
```

3. Type TEST on terminal with a carriage return.

```
125 TYPE 'TEST'*
```

4. Type TEST on terminal with a bell.

```
345 TYPE 'TEST'B
```

5. Type a string on the terminal

```
1100 TYPE &1
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **TYPEI, TYPEV, PRINT, PRINTI, PRINTV**

TOKEN: /059

FORMAT:
[Line#] **TYPEI** <VAR_NAME,...>

DESCRIPTION:

<VAR_NAME>
Can use constants, variable names, arrays or expressions.

This command sends the specified variables to device 0 in Integer format. It is often useful to display variables in an integer format, since a variable may only represent an integer value. A counter which must be displayed each loop looks far more informative when shown as a '+1', instead of '+1.0000' as a real variable would be displayed.

If the value does not fit within the integer limits (+/- 32764) then the display will be shown in real format.

Each typed variable will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **TYPE**, **TYPEV**, **PRINT**, **PRINTI**, **PRINTV**

TYPEV

TOKEN: /060

FORMAT:
[Line#] **TYPEV** <VAR_NAME,...>

DESCRIPTION:

<VAR_NAME>
Can use constants, variable names, arrays or expressions.

This command sends the specified variables to device #0 in Real number format. Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Types variable value RADIUS to video terminal.

```
>>TYPEV VARIABLE(,VARIABLE,...) :RADIUS<cr>
+002.7500
```

2. Types variable value RADIUS and CIRCUM to video terminal.

```
>>TYPEV VARIABLE(,VARIABLE,...) :RADIUS, +002.7500 CIRCUM<cr>
+008.6393
>>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **TYPE, TYPEI, PRINT, PRINTI, PRINTV**

TOKEN: /017

FORMAT:
[Line#] UNLOCK <JOINT#,...>

DESCRIPTION:

<JOINT#>
Can use constants, variable names, arrays or expressions.

This command restores control over the specified joint, undoing the LOCK command.

The FINISH command will be able to synchronize any robot command with a motion of a previously locked joint. It is recommended that the LOCK command be used only until both sets of motion have been commanded. Issuing the UNLOCK command will then re-establish full communication with all robot joints. Issuing the UNLOCK command will not disturb any motion in progress.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLES:

1. The following example is an excerpt from a program which commands the robot to load a rotary table (Joint #6). The rotary table then moves to a new position, while the robot proceeds to pick up a new part.

```
...
115 ; MOVE TO THE ROTARY TABLE
120 APPRO TABLE,2
130 MOVE TABLE,S
140 OPEN
150 DEPART 2
155 ; COMMAND THE TABLE TO ROTATE AT SPEED 20 BY 180 DEGREES
160 SPEED 20
165 UNLOCK 6
170 JOINT 6,180
175 ; LOCK OUT THE TABLE FROM THE NEXT MOVE COMMANDS
180 LOCK 6
200 APPRO PICK,2
...
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **LOCK, JOINT**
OTHER CRS Plus Publications: TECHNICAL MANUAL Appendix J

VIA

TOKEN: /194

FORMAT:
[Line#] **VIA** <LOC_NAME,...>

NOTE: ***This command is only applicable to A200/400 Series Robots.***

DESCRIPTION:

<LOC_NAME>
Can use locations names, strings or array-type location names.

This command calculates a continuous path motion through up to 14 locations. Unlike the CPATH or CTPATH commands the robot is not forced to pass through each intermediate knot. Each knot will be approached as closely as motion speed permits. All locations in the list must be either cartesian locations or precision points; no mixing of types is allowed.

Triggers will be activated during the path execution if the Trigger table includes activities at any of the locations specified in the VIA argument string and the Trigger flag has been enabled.

The VIA command is currently only capable of joint interpolated motion.

WARNINGS:

1. Using a VIA command while CARTVEL is enabled will cause a RAPL-II error.

2. If precision points are specified in the argument list of the VIA command, then the CARTVEL flag is ignored and interpolation proceeds in JOINT space.

3. If more than 5 axes are used in the controller, precision points must be used.

4. If a specific location appears consecutively within the argument list, the VIA command will force the robot to stop at this location before continuing the path.

EXAMPLES:

1. To do a pick and place operation without stopping at the approach locations:

```
>>VIA
ENTER PATH POINTS, SEPARATED BY COMMAS(UP TO 16)
DEP1,APP2,PLACE<cr>
```

2. This command will move along a path near locations A, AB, B, C, and D. The robot will pause at location AB before continuing to location D. (prompt messages not shown):

```
>>VIA A,AB,AB,B,C,D<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **ENABLE, DISABLE, TRIGGER, CPATH, CTPATH**

WO

TOKEN: /027

FORMAT:
[Line#] **WO**

DESCRIPTION:

This command displays the current robot position in the motor, joint and world coordinate systems. If more than the standard 5 axes are installed, the extra axes will be displayed under the first, second and third values in the MOTOR and JOINT displays. In the WORLD display, they will be displayed only if a @TRACK or @GANTRY command has been issued.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. In the case where a @GANTRY or @TRACK command has been entered:

```
>>WO<cr>

 COMMANDED POSITION :
NAME       AX#1/6      AX#2/7      AX#3/8      AX#4        AX#5
PULSES   +0000000000 -0000018000 +0000000000 +0000000000 +0000000000
         +0000000000 -0000010000
NAME       JT#1/6    JT#2/7    JT#3/8    JT#4      JT#5
JOINTS   +000.0000 +090.0000 +000.0000 +000.0000 +000.0000
         +000.0000 -002.0000

NAME       X/TRACKX  Y/TRACKY      Z         YAW      PITCH     ROLL
WORLD     +013.5000 +000.0000 +020.0000 +000.0000 +000.0000 +000.0000
          +000.0000 -002.0000
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W1, W2, W3, W4, W5, WE1, WE3, WE5, LISTL, HERE**

TOKEN: /029

FORMAT:
[Line#] W1

DESCRIPTION:

This command continually displays the actual robot position in motor coordinates. A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the Teach button to register a location during a dynamic display may cause the screen to lock up. Use caution with this command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>W1<cr>

ACTUAL POSITION (MOTOR PULSES):
+0000000001 -0000000003 +0000000031 +0000000000 +0000000000
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W2, W3, W4, W5, WE1, WE3, WE5, LISTL, HERE**

W2

TOKEN: /031

FORMAT:
[Line#] **W2**

DESCRIPTION:

This command displays the current actual robot position in motor, joint and cartesian coordinates. If more than the standard 5 axes are installed, the extra axes will be displayed under the first, second and third values in the MOTOR and JOINT displays. In the WORLD display, they will be displayed only if a @TRACK or @GANTRY command has been issued.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>W2<cr>

CURRENT ACTUAL POSITION:
NAME       AX#1/6      AX#2/7      AX#3/8      AX#4        AX#5
PULSES   +0000000000 -0000018000 +0000000000 +0000000000 +0000000000
         +0000000000 -0000010000
NAME       JT#1/6    JT#2/7    JT#3/8    JT#4      JT#5
JOINTS   +000.0000 +090.0000 +000.0000 +000.0000 +000.0000
         +000.0000 -002.0000

NAME       X/TRACKX  Y/TRACKY      Z         YAW      PITCH     ROLL
WORLD     +013.5000 +000.0000 +020.0000 +000.0000 +000.0000 +000.0000
          +000.0000 -002.0000
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W1, W3, W4, W5, WE1, WE3, WE5, LISTL, HERE**

TOKEN: /033

FORMAT:
[Line#] **W3**

DESCRIPTION:

This command continually displays the commanded robot position in motor coordinates. A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the Teach button to register a location during a dynamic display may cause the screen to lock up. Use caution with this command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>W3<cr>

COMMANDED POSITION (MOTOR PULSES):
+0000000001 -0000000003 +0000000031 +0000000000 +0000000000
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W1, W2, W4, W5, WE1, WE3, WE5, LISTL, HERE**

W4

TOKEN: /035

FORMAT:
[Line#] **W4**

DESCRIPTION:

This command displays the current motion end point in motor, joint and cartesian coordinates. If more than the standard 5 axes are installed, the extra axes will be displayed under the first, second and third values in the MOTOR and JOINT displays. In the WORLD display, they will be displayed only if a @TRACK or @GANTRY command has been issued.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>W4<cr>

NEXT END POINT:
NAME      AX#1/6      AX#2/7      AX#3/8      AX#4        AX#5
PULSES   +0000000000 -0000018000 +0000000000 +0000000000 +0000000000
         +0000000000 -0000010000
NAME       JT#1/6    JT#2/7    JT#3/8    JT#4      JT#5
JOINTS   +000.0000 +090.0000 +000.0000 +000.0000 +000.0000
         +000.0000 -002.0000

NAME       X/TRACKX  Y/TRACKY      Z         YAW      PITCH     ROLL
WORLD      +013.5000 +000.0000 +020.0000 +000.0000 +000.0000 +000.0000
           +000.0000 -002.0000
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W1, W2, W3, W5, WE1, WE3, WE5, LISTL, HERE**

TOKEN: /037

FORMAT:
[Line#] W5

DESCRIPTION:

This command continually displays the robot velocity command. A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the Teach button to register a location during a dynamic display may cause the screen to lock up. Use caution with this command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>W5<cr>

VELOCITY COMMAND (MOTOR PULSES):
   +0    +0    +0    +0    +0
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W1, W2, W3, W4, WE1, WE3, WE5, LISTL, HERE**

WAIT

TOKEN: /094

FORMAT:
[Line#] **WAIT** <[-]INPUT#>

DESCRIPTION:

<[-]INPUT#>
Can use constants, variable names, arrays or expressions.

This command halts program flow until the value of the specified input point matches the required state. The input value can range from 1 to 40 with the extended I/O option, but 1 to 16 is standard.

The sense of the input is denoted by including a sign with the input number. If the input number is negative, then a low level on the input is requested. Likewise, a positive sign, or none at all, implies that a high state on the input is required.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Wait until input #2 is at a low level, then continue.

```
100 WAIT -2
```

2. Wait until input #2 is at a high level, then continue.

```
120 WAIT 2
```

CROSS-REFERENCES:

RAPL-II COMMANDS: IFSIG, ONSIG

OTHER CRS Plus Publications: APP/SAFETY

TOKEN: /028

FORMAT:
[Line#] **WE1**

DESCRIPTION:

This command continually displays the actual position of the extra axes in motor coordinates.

A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the Teach button to register a location during a dynamic display may cause the screen to lock up. Use caution with this command.

APPLICABLE MODES:
{I}.{M}.{P}

EXAMPLE:

1. Enter:

```
>>WE1<cr>

ACTUAL POSITION (MOTOR PULSES):
+000002789    +123456789    -987654321
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **WO, W1, W2, W3, W4, W5, WE3, WE5, LISTL, HERE**

WE3

TOKEN: /032

FORMAT:
[Line#] **WE3**

DESCRIPTION:

This command continually displays the commanded position of the extra axes in motor coordinates.

A <cr> or an <ABORT> is needed to terminate the display.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the Teach button to register a location during a dynamic display may cause the screen to lock up. Use caution with this command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>WE3<cr>

COMMANDED POSITION (MOTOR PULSES):
+000002789    +123456789    -987654321
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W1, W2, W3, W4, W5, WE1, WE5, LISTL, HERE**

TOKEN: /036

FORMAT:
[Line#] **WE5**

DESCRIPTION:

This command continually displays the velocity commands to the extra axis motors.
115 is the maximum command allowed.

A <cr> or an <ABORT> will terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the Teach button to register a location during a dynamic display may cause the screen to lock up. Use caution with this command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. Enter:

```
>>WE5<cr>

VELOCITY COMMAND (MOTOR PULSES):
   +5      -12      +65
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **W0, W1, W2, W3, W4, W5, WE1, WE3, LISTL, HERE**

WITH

TOKEN: /153

FORMAT:
[Line#] **WITH** <TEMPLATE>

DESCRIPTION:

<TEMPLATE>
Can use a template name or a string.

This command permits the operator to set up an array-type addressing scheme for locations stored in memory with the TEACH command. The template name entered in this command follows the same programming rules as the template in the TEACH command (see chapter 1-5).

Once the WITH command has specified a template, then the programmer need only to use the ^ character to address the range of locations stored using that template. Since an index number must be associated with a template before any location can be uniquely defined, the ^ operator permits the use of a number or variable for this purpose.

WARNING:

The WITH command uses the same internal register for storage of a template as the TEACH command. Use of either command will overwrite the contents placed in that register by the other command.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. A program may use the WITH command as follows:

```
100 WITH POINT
120 MOVE ^0
130 ! C = 0
140 MOVE ^C
```

Lines 120 and 140 both create a MOVE command to location POINT000 stored in memory. The advantage of using a variable as an index (as in line 140) is that the same MOVE command can be used to move the robot to any one of the taught points that use the template specified by the WITH command. The value of the variable can be calculated using logic.

CROSS-REFERENCES:

RAPL-II COMMANDS: **TEACH**, **NOTEACH**, **CTPATH**
OTHER CRS Plus Publications: PATH APPLICATION NOTE

TOKEN: /172

FORMAT:

[Line#] **WRIST** <NOFLIP | FLIP>

NOTE: ***This command is only applicable to A400 Series Robots.***

DESCRIPTION:

This command determines the wrist orientation when moving to locations. NOFLIP forces the robot to use a POSITIVE wrist pitch (joint 5) angle. FLIP forces the robot to use a NEGATIVE wrist pitch angle.

APPLICABLE MODES:

{I},{P}

CROSS-REFERENCE:

RAPL-II COMMANDS: **REACH, ELBOW, INVERT**

XCAL

TOKEN: /157

FORMAT:
[Line#] **XCAL** <Axis#>

DESCRIPTION:

<Axis#>
Can use constants, variable names, arrays or expressions.

This command permits the programmer to create and store a Home position for any axis (1 to 8). Subsequent HOME commands will use the absolute position stored by this command. The end of the Homing mark (the pointed end) is the next encoder zero-cross detection. The motor will be moved in a positive rotation (counter-clockwise) until the mark is reached.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. Using XZERO command, zero the axis in question.

2. Using the Manual mode, or a JOINT or MOTOR command, move the axis to within one (1) revolution of the Homing mark.

3. Enter the XCAL command.

4. The axis is now calibrated.

CROSS-REFERENCES:

RAPL-II COMMANDS: **XZERO**, **XHOME**, **XREADY**

OTHER CRS Plus Publications: TECHNICAL MANUAL Appendix J

TOKEN: /158

FORMAT:
[Line#] **XHOME** <Axis#>

DESCRIPTION:

<Axis#>
Can use constants, variable names, arrays or expressions.

This command is used to HOME an axis (Axis 1 to 8) independently. In contrast, the HOME command homes all 5 robot axes simultaneously. When the Homing point has been reached, the value of the calibrated position will be loaded into the position registers.

APPLICABLE MODES:
{I},{P},{M}

EXAMPLE:

1. Enter Manual mode:

```
>>MANUAL <cr>
```

Move the extra axis to within 1 motor revolution of the homing position.

Enter the XHOME command, say, for axis #6:

```
>>XHOME AXIS #: 6<cr>
  LOCATED IN ITS HOME BOUNDS? Y<cr>
```

The controller will home the specified axis.

CROSS-REFERENCES:

RAPL-II COMMANDS: **XZERO**, **XCAL**, **XREADY**, **HOME**

OTHER CRS Plus Publications: TECHNICAL MANUAL Appendix J

XREADY

TOKEN: /014

FORMAT:
[Line#] **XREADY** <Axis#>

DESCRIPTION:

<Axis#>
Can use constants, variable names, arrays or expressions.

This command moves the specified extra axis to its zero position.

APPLICABLE MODES:
{I},{M},{P}

CROSS-REFERENCES:

RAPL-II COMMANDS: **XZERO**, **XCAL**, **XHOME**

OTHER CRS Plus Publications: TECHNICAL MANUAL Appendix J

TOKEN: /159

FORMAT:
[Line#] **XZERO** <Axis#>

DESCRIPTION:

<Axis#>
Can use constants, variable names, arrays or expressions.

This command zeros the position registers for the specified axis (axis 1 - 8). This is useful when doing a calibration sequence for an axis, as it allows a known position to be loaded into the position registers.

APPLICABLE MODES:
{I},{M},{P}

EXAMPLE:

1. To enter a value of zero (0) into the current position register for axis six (6):

```
>>XZERO AXIS #: 6<cr>
```

CROSS-REFERENCES:

RAPL-II COMMANDS: **XZERO**, **XHOME**, **XREADY**

OTHER CRS Plus Publications: TECHNICAL MANUAL Appendix J

????????	(/000)	- RAPL-II Command directory.
;	(/098)	- Comment statement.
!	(/070)	- Assign the value of the simple expression to the variable specified on the left side of the =.
++	(/192)	- Increment a variable by 1.
--	(/193)	- Decrement a variable by 1.
ABORT	(/071)	- Terminate existing program.
ACTUAL	(/152)	- Stores the actual position of the arm.
ALIGN	(/001)	- Align the tool axis with the nearest major axis (X, Y or Z).
ALLOC	(/018)	- Allocate robot memory according to an automatic format, or a user defined format. In this way, robot memory can be divided between locations, variables and robot programs as the programmer chooses.
AOUT	(/093)	- Send a value to an analog output point (included in the COMBO card option). The digital value is from -5 to +5 corresponding to digital 0 to 255.
APPRO	(/002)	- Move to a position a specified distance away from a user defined location. The approach path can be defined as a straight line by using the optional [,S] argument.
ARM	(/142)	- Enables or disables the arm power. Can be used to turn arm power off but not back on.
AXSTATUS	(/185)	- Display status of each axis card.
CIRCLE	(/182)	- Move the robot in a user defined circular path.
CLOSE	(/038)	- Close the gripper. Force argument used only for optional servo gripper.
CONFIG	(/110)	- Set the configuration of one of the two serial input channels to the required baud rate, parity, etc.
COPY	(/099)	- Copy one program to another.
CPATH	(/163)	- Execute a continuous path through the points specified in the argument list. Up to 16 locations can be specified. Cartesian and precision points cannot be mixed.

CTPATH	(/164)	- Calculate a continuous path through a specified series of points. The points used to calculate the path must have previously been stored via Teach mode. They are identified by a template name, and a three digit number. The path is calculated using cubic-spline interpolation through the points. The path is executed using a **GOPATH** command.
CUT	(/145)	- Cut characters out of a string.
DECODE	(/147)	- Decode a real value from the string, starting at the character index, and load the value into the variable.
DELAY	(/081)	- Specify a time delay, in seconds. The resolution of this command is in milliseconds.
DELETE	(/100)	- Delete program.
DEPART	(/004)	- Depart from the present location (along the tool axis) by a specified amount. The optional [,S] argument will specify a straight line motion.
DEVICE	(/112)	- Select either device 0 or device 1 as the future default input/output device for user interaction.
DIR	(/102)	- Send a list of all programs, and the corresponding memory requirements to the output device. The optional argument will send the output to the printer port.
DISABLE	(/177)	- Turn off a software feature.
DLOCN	(/048)	- Delete location(s) from the robot's memory.
DVAR	(/067)	- Delete variable(s) from the robot's memory.
EDIT	(/101)	- Assign the specified program name for all future edit commands and enter Edit mode. The program will be created if it does not already exist.
ELBOW	(/169)	- Specify the position of the elbow (joint #3) for future moves. Allows elimination of singularity locations.
ENABLE	(/176)	- Turn on a software feature.
ENCODE	(/148)	- Encode a value into the string, in either real or integer format.
EXECUTE	(/186)	- Causes execution of an 8086 machine language program.
FINISH	(/010)	- Complete current motion command before continuing.

FLASH (/084) - Flash the READY lamp on the Teach pendant at the specified frequency. Valid range is 1 to 255, 1 is fastest.

FREE (/023) - Display the current status of robot memory.

GAIN (/150) - Permits the user to change the position gain of each motor servo system. A motor number of 0 will set all motors.

GOPATH (/165) - Execute a path calculated with the **CTPATH** command.

GOSUB (/074) - Send program control to the specified subroutine. If a list of arguments exist, then replace the current set of 8 macro parameters with the contents of the list.

GOTO (/075) - Send program control to the specified line number in the current program.

GRIP (/039) - Commands the opening or closing of the servo gripper to a specified distance.

HALT (/154) - Halt robot motion or set up a **HALT** ON <input#>.

HELP (/095) - Enables syntax building feature (Help mode).

HERE (/024) - Stores the current robot commanded position as a precision point or cartesian location in the location table.

HIMEM (/187) - Permits the programmer to partition a portion of the program memory for reserved use.

HOME (/020) - Send the robot to its Home position, assuming that the robot is already within bounds of the Home position. Replace the robot command registers with the calibrated Home position when it is reached.

IF (/072) - Evaluate the logical expression according to one of 6 operators. If the expression is true, then send program control to the specified line number.

IFAUX (/189) - Branch program control depending on the state of the auxiliary input.

IFPOWER (/155) - This a special case of **IFSIG**. Here the arm power status is sampled.

IFSIG (/073) - Examine the states of the specified inputs. If all conditions are true, then send program control to the given line number.

IFSTART	(/086)	- Examine the state of the Auto Start button on the front panel. If it is set, then branch to the specified line number.
IFSTRING	(/151)	- Compare two strings and branch on result.
IGNORE	(/087)	- Cancel **ONSIG** command.
INPUT	(/068)	- Accept input from the current terminal device.
INVERT	(/170)	- Invert the coordinate system of the robot.
IORD	(/085)	- Read value of byte or word at the output port of the 8086.
IOWR	(/091)	- Write a constant or variable to the output port of 8086.
JOG	(/005)	- Move the robot end effector by the specified cartesian offsets in a straight line fashion.
JOINT	(/006)	- Move the specified joint by the given number of units.
LIMP	(/007)	- Limp specified robot joint. No 'joint#' entered will limp all joints. Terminated by **NOLIMP** command.
LISTL	(/049)	- List all location values in a tabular format on the output device. The 1 argument will send output to the printer.
LISTP	(/103)	- List the program on the output device. Send it to the printer if the 1 argument is specified.
LISTV	(/069)	- List the variables on the output device. Send it to the printer if the 1 argument is specified.
LOCK	(/009)	- Prevents selected joints from moving.
MA	(/132)	- Move all 5 joints to an absolute radian position.
MAGGRIP	(/042)	- Adjust the magnetic strength.
MANUAL	(/045)	- Activate Manual mode. The Teach pendant can be used to control robot motion either in Joint or Cylindrical mode.
MEMRD	(/104)	- Read contents of a specified memory address.
MEMWR	(/118)	- Write a constant or variable to a specified memory address.
MI	(/131)	- Move all 5 joints by an incremental radian value.
MOTOR	(/011)	- Command an individual motor to move by a specified number of pulses.

MOVE	(/012)	- Move the robot to the specified cartesian or precision point location using the speed as given in the **SPEED** command. The optional [,S] argument will command a straight line move to the point.
NEW	(/021)	- Erase all robot user memory. All programs, variables and locations will be erased.
NEXT	(/105)	- Execute the next program line. Program name and line number may optionally be specified.
NOFLASH	(/088)	- Cancel **FLASH** command. READY lamp on the Teach pendant indicates robot alarm status.
NOHELP	(/096)	- Disables syntax builder (Help mode).
NOLIMP	(/008)	- Re-engages positional servo for specified joint. If no joint is specified then all joints are Re-engaged.
NOMANUAL	(/046)	- Disables Teach pendant.
NOTEACH	(/043)	- Turn off the Teach mode.
NOTRACE	(/080)	- Turn off the Trace mode.
OFFSET	(/047)	- Set the robot base offset to the specified displacements and yaw angle. All future locations will be made relative to this new base offset.
ONAUX	(/190)	- Wait condition until an auxiliary input's state becomes true, or an ONSIG condition is realized.
ONERR	(/143)	- Program flow will enter subroutine "PRG_NAME" instead of going to the prompt in the case of an error.
ONPOWER	(/156)	- This a special case of **WAIT**, here the arm power status is sampled.
ONSIG	(/090)	- Set the specified input line as a special input which will be monitored at a regular interval. When the input changes to the desired state, program flow is re-directed to the specified program. Returning from this program will place the program pointer at the first program line after the one in which the signal was detected.
ONSTART	(/092)	- Sample the state of the Auto Start button. When it is switched high, program control will continue to the next line. Wait at this point until the condition is met.

OPEN	(/040)	- Open the gripper. Force argument used only for optional servo gripper.
OUTPUT	(/089)	- Set the output lines to the specified states.
PASSWORD	(/141)	- Enters Supervisory mode.
PASTE	(/146)	- Paste string 1 into string 2, starting at the specified index.
PAUSE	(/076)	- Pause the program flow and display the message on the output device. The program can then be continued by entering the **PROCEED** command.
POINT	(/050)	- Defines a location.
PRINT	(/055)	- Send the text string to the printer. Use a set of special characters to enhance the display. See **TYPE**.
PRINTI	(/056)	- Print variable(s) to the printer in an Integer format.
PRINTV	(/057)	- Print variable(s) to the printer in Real format.
PROCEED	(/106)	- Continue program execution after a **PAUSE** command or a <Ctrl-A> has been used to halt program flow. In a program, this command will restart a path once interrupted.
REACH	(/171)	- Cause to robot to access a singularity location either in forward mode (traditional) or by reaching "over the head".
READY	(/015)	- Moves arm to the READY position, and any extra axes to their zero positions. Also resets any LOCK conditions.
REMOTE	(/195)	- Enable Remote manual control from Host.
RENAME	(/107)	- Rename an existing program in the robot memory.
RETRY	(/108)	- Retry any command which caused an error, halting program flow. That statement will be attempted again and the program will continue normally.
RETURN	(/077)	- Return to calling program.
RTCLOCK	(/191)	- Reads current state of MK48T12 real time clock.
RUN	(/109)	- Execute a program in memory. Execute it for a given number of cycles. Execute it only once if no loop counter is present.
SERIAL	(/111)	- Display the current serial interface parameters.
SET	(/051)	- Assign the value of a location to that of an existing one.

SHIFT	(/052)	- Shift the cartesian displacements of a cartesian location by the specified vector.
SHIFTA	(/053)	- Shift the values of all components of the cartesian location by the specified amounts.
SPEED	(/016)	- Set the speed of future robot moves.
STATUS	(/025)	- Display the current robot status on the default output device.
STOP	(/078)	- Terminate program flow. Print the optional string on the output device.
STRPOS	(/149)	- Match the occurrence of the target string in the specified string number, and return the character index in the variable. Returns a value of zero if no match was made.
TEACH	(/044)	- Turn on the Teach mode of operation. Specify a Teach template which will be used to identify all future teach points. Each time the Teach button is pressed, a new location will be stored in memory. Locations can be stored according to the format described by the name; that is either a precision point or a cartesian location.
TIME	(/082)	- Extract the system timer clock value. Store the value in real format in the given variable.
TOOL	(/054)	- Set the Tool transform to a value described by a stored 'location'.
TRACE	(/079)	- Turn on the Trace mode of operation. Each new program line which is executed will be displayed on the output device in terms of program name and line number.
TRIGGER	(/175)	- Set up a table of outputs to be turned on and/or off during path execution.
TYPE	(/058)	- Send a text string output to the default output device. Use special characters to enhance the display.
TYPEI	(/059)	- Print the list of variables to the output device in an integer format.
TYPEV	(/060)	- Print the list of variables to the output device in a real format.
UNLOCK	(/017)	- Disables **LOCK** command.
VIA	(/194)	- Move through a series of locations without stopping.

WO	(/027)	- Display the current commanded robot position.
W1	(/029)	- Display the actual robot position continuously.
W2	(/031)	- Display the actual current robot position.
W3	(/033)	- Displays the robot commanded position continuously on the output device.
W4	(/035)	- Display the end coordinates of the current path.
W5	(/037)	- Continuously displays the robot velocity command.
WAIT	(/094)	- Test the condition of the input line and wait here until the condition is met.
WE1	(/028)	- Display the actual position of the extra axes continuously.
WE3	(/032)	- Displays the commanded position of the extra axes on the output device continuously.
WE5	(/036)	- Displays the extra axes velocity commands continuously.
WITH	(/153)	- Permits numerical access to all locations stored with the same template.
WRIST	(/172)	- Determine position of wrist for next move(s).
XCAL	(/157)	- Calibrate an extra axis.
XHOME	(/158)	- Home an extra axis.
XREADY	(/014)	- Move an extra axis to its zero position.
XZERO	(/159)	- Zero the position registers of the specified axis.

APPENDIX B - ERROR CODES

B-1 RAPL-II Error Codes

All robot operations are monitored for error conditions. Detectable errors are listed in this appendix. When an error condition exists, a message will be displayed at the default device, and the ready-light on the teach pendant will turn off.

To clear the error condition, any correct command may be entered. The command will clear the error detection flag, and will then turn the ready-light back on. A STATUS command will list the previous error condition.

ERROR	MESSAGE	DESCRIPTION

000 AXIS#1 OUT — AXIS #1 IS OUT OF RANGE:
This error means that there has been a Loss Of FeedBack error for joint 1. This error could be due to a collision with a stationary object, a blown fuse in the motor circuit, or a true loss of feedback signal from the encoder.

001 AXIS#2 OUT — AXIS #2 IS OUT OF RANGE:
Same as error 000, but for axis 2.

002 AXIS#3 OUT — AXIS #3 IS OUT OF RANGE:
Same as error 000, but for axis 3.

003 AXIS#4 OUT — AXIS #4 IS OUT OF RANGE:
Same as error 000, but for axis 4.

004 AXIS#5 OUT — AXIS #5 IS OUT OF RANGE:
Same as error 000, but for axis 5.

005 AXIS#6 OUT — AXIS #6 IS OUT OF RANGE:
Same as error 000, but for axis 6.

006 AXIS#7 OUT — AXIS #7 IS OUT OF RANGE:
Same as error 000, but for axis 7.

007 AXIS#8 OUT — AXIS #8 IS OUT OF RANGE:
Same as error 000, but for axis 8.

008 ILLEG OPER — ILLEGAL OPERAND:
An operand in the command line is not acceptable.

B-1 RAPL-II Error Codes (continued)

ERROR	MESSAGE	DESCRIPTION

009 EOL NOT FO — END OF LINE NOT FOUND:
The end of line descriptor (a <cr>) was not found, causing an incorrect read of the data. Command lines are limited to 128 characters in length including the EOL character.

010 SYMB UNDEF — SYMBOL UNDEFINED:
A necessary variable was not present in the symbol table. To correct, manually, or through the program logic, create a symbol in the table.

011 LOCN UNDEF — LOCATION UNDEFINED:
The robot location referenced was not previously defined. Check the location table for the location in question, and create if necessary.

012 COMM ERR?? — COMMAND ERROR:
The command entered was not found in the command list. A spelling mistake, or the lack of a space between the line number and the command descriptor could have caused this problem. Either re-type the line, or inspect the program line for the mistake.

013 ARG MISSIN — ARGUMENT MISSING:
An argument was expected, but not read. Check the command syntax

014 BAD SYNTAX — BAD SYNTAX:
A general error which occurs when the command line could not be decoded. Check for a spelling error in the command name.

015 PP NOT LEG — PRECISION POINT NOT LEGAL:
A precision point was specified as a location in the command line, but was not permitted in this situation.

016 I/O ADDR?? — I/O ADDRESS ERROR:
The I/O number which was entered in the command line was beyond the permissable range. Standard digital I/O is from 1 to 16, while optionally, 1 to 40 can be used.

017 PROG NOT F — PROGRAM NOT FOUND:
The program called was not found in the program directory. Create the program, or check the spelling of the missing program call command.

018 ILLEG ARGU — ILLEGAL ARGUMENT:
An argument was entered, but not permitted.

019 LINE NOT F — LINE NOT FOUND:
The program line number called in a branching statement is not present in the program.

020 ARG TOO BI — ARGUMENT TOO BIG:
The value of a specified argument is too big for the command.

ERROR MESSAGE DESCRIPTION

021 NO ROOM LE NO ROOM LEFT:
The program buffer is full, and cannot accept any further information. Delete unwanted programs, or re-allocate memory with the ALLOCATE command.

022 TABLE FULL TABLE FULL:
The robot controller maintains program names, location and variable references in tables. This error means that there is no room left to store a new item. Delete all unwanted information, or re-allocate memory.

023 BAD LINE N BAD LINE NUMBER:
The line number called could not be found, and that a value of zero was returned after its search. The value of zero is an illegal line number.

024 ILL A/D CH ILLEGAL A/D CHANNEL:
An illegal analog to digital conversion channel was specified. Channel numbers 1 to 25 are valid. Channel 9 is the speed selector knob on the teach pendant.

025 LINE TOO L LINE TOO LONG:
The command line exceeded the maximum number of characters permitted, 128 characters.

026 EOF NOT FO END OF FILE NOT FOUND:
An end-of-file character was not found. This implies that a program buffer error has occurred. To remedy clear all memory and re-load all information. If the problem persists, then a fault exists in user memory, or a problem with storing the information on an external device caused invalid storage of information. When this information was re-loaded, the EOF character was not found, thus causing the error.

027 PROG EXIST PROGRAM EXISTS:
An attempt to create a duplicate file name, or an attempt to load into an existing file name was discovered. Either delete the existing file, or define a new program name for the new function.

028 LINE EXIST LINE EXISTS:
An attempt to insert an existing line number was found. Either define a new line number, or delete the old one first.

029 XFRM ILLEG TRANSFORM ILLEGAL
A move to an impossible robot coordinate was attempted and the coordinate transformation for the location generated a mathematically impossible result.

030 RESERVED

031 RESERVED

ERROR	MESSAGE	DESCRIPTION

032 HOME FAIL1 JOINT #1 HOME FAILURE:
The home sequence failed during the motion of axis 1. The axis was commanded a motion of just over 1000 pulses, or one motor revolution. During this motion, a zero-crossing marker pulse was expected but not seen. This could be due to an encoder failure, a wiring problem or just a lack of power to motor 1. Check for servo action and retry the HOME command. If it fails again, check encoder function and connection.

033 HOME FAIL2 JOINT #2 HOME FAILURE:
Follow the same procedure as error 032.

034 HOME FAIL3 JOINT #3 HOME FAILURE:
Follow the same procedure as error 032.

035 HOME FAIL4 JOINT #4 HOME FAILURE:
Follow the same procedure as error 032.

036 HOME FAIL5 JOINT #5 HOME FAILURE:
Follow the same procedure as error 032.

037 HOME FAIL6 JOINT #6 HOME FAILURE:
Follow the same procedure as error 032.

038 HOME FAIL7 JOINT #7 HOME FAILURE:
Follow the same procedure as error 032.

039 HOME FAIL8 JOINT #8 HOME FAILURE:
Follow the same procedure as error 032.

040 ARM POWER ARM POWER:
The arm power switch was turned off, which means that no power is going to the servo motors. In this condition, the robot cannot move. The robot has been placed in a LIMP state.

041 NOT HOMED NOT HOMED:
The robot cannot perform the requested motion because it has not been homed after power up.

042 JNT #1 OUT JOINT #1 OUT OF RANGE:
The next motion will send joint 1 out of range. Inspect the destination point for validity.

043 JNT #2 OUT JOINT #2 OUT OF RANGE:
Follow the same procedure as error 042.

044 JNT #3 OUT JOINT #3 OUT OF RANGE:
Follow the same procedure as error 042.

ERROR MESSAGE DESCRIPTION

045 JNT #4 OUT JOINT #4 OUT OF RANGE:
Follow the same procedure as error 042.

046 JNT #5 OUT JOINT #5 OUT OF RANGE:
Follow the same procedure as error 042.

047 CHECKSUM F CHECKSUM FAILURE:
The desired item of memory (program, location or variable) has been changed in some unpredictable way. The use of this memory item could cause unexpected results. Check the item and change or edit it to the correct value.

048 RESERVED

049 MEM NOT AL MEMORY NOT ALLOCATED:
RAPL-II checked user memory and believes it has not been allocated. This could be an indication of battery failure. Use the ALLOC command to partition and clear the robot memory before use.

050 AMBIG CMD AMBIGUOUS COMMAND:
The user specified too few characters in a command specifier so that it is not yet a unique command choice. Re-enter the command. This error will occur when pressing a <cr> immediately after the prompt.

051 #0 TXD TIM SERIAL CHANNEL #0 TIMEOUT:
A timeout in serial device 0 indicating a hardware communication failure. When you see this message, the condition should already be cleared, but check the cable and its connections to be safe. If this fails, then turn system off and on.

052 #0 CTS TIM SERIAL CHANNEL #0 CTS TIMEOUT:
A timeout in the serial device 0 handshake control signal. This will appear only if the operator has selected a CTS/RTS handshake format. A timeout error would seem to indicate that the external device has failed, or a connection has come loose.

053 #1 TXD TIM SERIAL CHANNEL #1 TIMEOUT:
Follow the same procedure as error 051 but for device 1.

052 #1 CTS TIM SERIAL CHANNEL #1 CTS TIMEOUT:
Follow the same procedure as error 052 but for device 1.

053 RESERVED

054 RESERVED

055 RESERVED

B-1 RAPL-II Error Codes (continued)

ERROR MESSAGE DESCRIPTION

056 CAL CKSUM CALIBRATE CHECKSUM ERROR:
The robot arm calibration values stored in memory, which are used in the homing sequence, have been corrupted. Re-Calibrate the robot or re-load the calibration values from disk using the ROBCOMM utility LOADCAL.

057 RESERVED

058 NO ACCESS NO ACCESS:
User requires supervisory access to this command. Enter the correct PASSWORD and retry the command.

059 RSERVED IO RESERVED I/O ERROR:
Attempt to use serial device #1 when it has been reserved for ACI use.

060 ACI ERROR! REMOTE COMMUNICATION ERROR:
Contact your distributor.

061 RESERVED

062 NO RET LEV NO RETURN LEVEL
Either a RETURN instruction was used when the program was not called with a GOSUB (ie. trying to RUN a subroutine), or a attempted GOSUB call would result in a subroutine nesting level greater than 10.

063 BAD AXIS N BAD AXIS NUMBER
An extra axis command was issued to a nonexistent axis. Use the @NOA command to set the number of axes.

064 NOT AVAIL NOT AVAILABLE
An option cannot be installed in the current system, or a function has been rendered unavailable with the current software.

065 SLOT USED AXIS SLOT ALREADY USED
Either the programmer was attempting to set up the controller for 8 axes while the servo or magnetic gripper option was installed, or vice versa. The programmer must deactivate the option which uses slot 8 first before assigning the new function.

066 STR ERROR STRAIGHT LINE ERROR
A straight line path was halted since it hit a joint limit. It may not be possible to move the arm out of this position using a straight line, so use a JOINT move.

067 NOT INSTAL EXPANSION MEMORY NOT INSTALLED
The continuous path command CTPATH was attempted, but the extra memory option was not installed. Could also indicate an attempt to write to a write-protected or missing EEPROM memory board.

ERROR MESSAGE DESCRIPTION

068 NO PATH LO NO PATH LOADED
When requesting a continuous path execution with GOPATH, the command interpreter could not find any relevant path data loaded. Re-teach the path using the CTPATH command.

069 #0 FRA ERR DEVICE 0 FRAMING ERROR
Serial device #0 framing error. A hardware communication error has been determined. Retry the communication.

070 #0 PAR ERR DEVICE 0 PARITY ERROR
Serial device #0 parity error. A hardware communication error has been determined. Retry the communication.

071 #0 OVR ERR DEVICE 0 OVERRUN ERROR
Serial device #0 overrun error. A hardware communication error has been determined. Retry the communication.

072 #1 FRA ERR DEVICE 1 FRAMING ERROR
Serial device #1 framing error. A hardware communication error has been determined. Retry the communication.

073 #1 PAR ERR DEVICE 1 PARITY ERROR
Serial device #1 parity error. A hardware communication error has been determined. Retry the communication.

074 #1 OVR ERR DEVICE 1 OVERRUN ERROR
Serial device #1 overrun error. A hardware communication error has been determined. Retry the communication.

075 UNDEF ERRO UNDEFINED ERROR
An undefined RAPL error has been detected. This is an internal software failure which should be reported to a qualified CRS service centre.

076-083
IAC WRIT N IAC WRITE ERROR 1-8
This error range will be displayed if there has been a write error to a particular smart axis card. This error may occur during an emergency stop procedure, but should never occur during proper operation. Any occurrence of this error should be reported to a qualified CRS service centre.

084-091
IAC ERRO N IAC ERROR 1-8
This error range will be displayed if there has been an internal error in a particular smart axis card. This error may occur during an emergency stop procedure, but should never occur during proper operation. Any occurrence of this error should be reported to a qualified CRS service centre.

B-2 Parser Error Codes

CODE:	PARSER ERROR DESCRIPTION:
01	Missing '('
02	Missing or invalid operator
03	Stack overflow at unary operation
04	Missing argument or extra operation: two consecutive operations
05	----------
06	Illegal format in scientific notation
07	Illegal format (non-digit) in scientific notation
08	Illegal format (non-digit) in exponent
09	Stack overflow at scientific notation constant
10	Stack overflow at symbolic variable name
11	Missing operation before '('
12	Illegal operation unrecognized operation character
13	Stack overflow at end of parentheses, exponent or function
14	Extra ')' in function
15	Missing parameter in function
16	Missing '(' in line
17	Stack overflow: no room for function
18	Unknown function or missing operator
19	Stack overflow: no room for '('
20	Stack overflow: no room for array index expression
21	Stack overflow at operation
22	Stack overflow: no room for * or / operation
23	Stack overflow: no room for * or / operation

B-2 Parser Error Codes (continued)

CODE:	PARSER ERROR DESCRIPTION:
24	Stack overflow at * or / or function
25	Stack overflow: no room for + or -
26	Stack overflow: no room for operation
27	Unexpected end of line
28	Stack overflow in function parameter expression
29	Missing ')' at end of function
30	Too many parameters in function
31	Illegal characters in expression
32	Missing operation
33	Illegal operation: unrecognized character
34	Symbolic name too long: maximum 8 characters
35	Missing argument in expression
36	Non alphabetic in lead character of symbolic name
37	Invalid character in symbolic name
38	Expecting '(' in component functions and call
39	Argument list too long in component function
40	Missing or unknown delimiter in component function
41	Illegal character in component number of component function
42	Expecting component number in component function
43	Component number out of range in component function
44	Stack overflow at component function
45	Stack overflow at evaluation
46	Extra '[' found at evaluation

CODE:	PARSER ERROR DESCRIPTION:
47	Stack overflow at evaluation
48	Division by zero
49	Undefined power expression: the non-integer power of a negative number.
50	Analog input out of range
51	Fatal in evaluator
52	Missing expression
53	Array variable name too long: maximum of 5 characters
54	Array index out of range: $0 \leq \text{index} \leq 1000$
55	Illegal character in array name
56	----------
57	----------
58	----------
59	Illegal text substitution value
60	Line too long with text substitutions
61	Expecting numeric constant for text substitutions
62	----------
63	----------
64	----------
65	----------
66	----------
67	----------
68	----------
69	----------

B-2 Parser Error Codes (continued)

CODE:	PARSER ERROR DESCRIPTION:
70	Missing ']'
71	Expecting hex digit
72	Stack overflow at numeric constant
73	Expect numeric or e ± xx: not a number
74	Missing ')': expecting zero arguments in function
75	Missing array name
76	Extra ']'
77	Stack overflow
78	Missing arguments in function
79	Missing ')'
80	Missing '[' or ']' for array index

C-1 Introduction:

RAPL-II contains terminal control codes which can affect robot operation.

To enter a control code, the "control" (usually labelled "Ctrl") key must be pressed while the corresponding UPPERCASE alphabetic key is pressed.

C-2 Control Characters:

<Ctrl-A> The robot will stop after the current block is finished. If a motion is in progress, the next program line is processed as soon as the motion profile has been determined. Stopping a program and motion simultaneously will likely leave the arm out of sync with program execution. <Ctrl-A> is useful since it will terminate program execution but permit the robot to finish motion.

<Ctrl-C> The robot current operation is terminated immediately. <Ctrl-C> will stop motion, terminate program execution or command line entry, and reset communication parameters. This command is the equivalent of pressing the <ABORT> button on the Teach pendant.

<Ctrl-E> Resume echo mode of terminal operation. The serial channel that is affected is the one that received this code.

<Ctrl-H> Turn on the Help mode. This will activate the syntax building feature of the operator interface *[NOT EFFECTIVE WHEN USING THE ROBCOMM PROGRAM].*

<Ctrl-I> Turn off the Help mode. This will deactivate the syntax building feature of the control.

<Ctrl-Q> Resume terminal output. Essentially, this is an XON code that the user can manually issue to resume output to the terminal device.

<Ctrl-R> The echo mode of terminal operation will be turned off. The serial channel that is affected is the one that received this code.

<Ctrl-S> Stop terminal output. This code is an XOFF command to the controller that stops further output to the terminal.

<Ctrl-X> This command is the same as the <Ctrl-C> command. It is intended for use where users utilize a personal computer instead of a terminal to communicate with the robot controller. In this case, often the personal computer responds internally to a <Ctrl-C> character issued from the keyboard.

D-1 Introduction:

Before an Autostart operation is executed the system must have a program called **AUTO_ST**. This special program is executed automatically when the system MAIN POWER is turned on with the Auto Start switch depressed. If this type of start up is attempted and a program with this name is not found, an error will result.

D-2 Example 1:

This example is an application for the Autostart procedure using the Manual mode to Home the robot during the start-up.

In this example, the Autostart procedure takes care of Homing the robot by prompting the user to enter Manual mode. The **IFSTART**, **ONSTART** and **ONPOWER** commands are used to synchronize the Homing procedure with the user. After Homing, program flow will branch to the application program.

Any output to a terminal device will be ignored if no terminal is connected when the system is powered up. The **FLASH** command can be used to signal the operator in this case. In this case, the Teach pendant must be plugged in to the front panel.

AUTOSTART PROCEDURE - EXAMPLE 1

STEP DESCRIPTION

1. Start with the Controller in the OFF condition.
2. Press and HOLD Auto Start button.
3. Turn on Controller.
4. Release Auto Start button after about three (3) seconds.
5. Watch for Teach pendant Indicator flashing (Slow).
6. Turn arm power on.
7. Press START switch again.
8. Watch for Teach pendant Indicator flashing (Fast).
9. Move robot to Home boundaries with Teach pendant.
10. Press Auto Start button.
11. Homing sequence executes.
12. Application program, MAIN_PRO, now executes.

Example 1 (continued):

EXAMPLE 1 AUTO_ST PROGRAM LISTING

```
10 TYPE "/
20 TYPE 'Release START Switch.'/
30 TYPE "/
50 FLASH 5
60 TYPE 'Turn ARM POWER on.'/
70 TYPE "/
80 ONPOWER
90 FLASH 3
100 MANUAL
110 TYPE 'Now in Manual mode. Move to within HOME'/
120 TYPE 'bounds and press Auto Start switch.'/
130 TYPE "/
140 ONSTART
150 NOFLASH
160 HOME
170 TYPE 'Program will run when Auto Start switch pressed.'/
180 FLASH 1
190 ONSTART
200 NOFLASH
210 GOSUB MAIN_PRO
220 STOP Finished for the day!
```

D-3 Example 2:

This example assumes the use of the Homing Bracket to position the robot for automatic Homing. It also assumes that no terminal will be used for output. In this case, the only operator input will be to turn on the ARM POWER. (For more information on the Homing Bracket, refer to CHAPTER 9 of the Technical Manual.)

AUTOSTART PROCEDURE - EXAMPLE 2

STEP DESCRIPTION

1. Start with the Controller in the OFF condition.
2. Press and hold Auto Start switch.
3. Turn on Controller.
4. Release Auto Start button after three (3) seconds.
5. Watch for Teach pendant Indicator flashing (Slow).
6. Turn arm power on.
7. The robot will Home itself and run the application procedure.
8. After completion of the application, the robot will return to the nest and turn off the arm power.

Example 2 (continued):

EXAMPLE 2 AUTO_ST PROGRAM LISTING

```
50 FLASH 5
80 ONPOWER
90 NOFLASH
100 ; Exit the Homing Bracket:
110 SPEED 20
120 JOINT 3,30
130 HOME
140 MOVE SAFE
145 ; Run the application program:
150 GOSUB MAIN_PRO
155 ; When finished, move back to the Homing Bracket:
160 MOVE SAFE
170 MOVE ABOVE
180 SPEED 20
190 MOVE BRACKET
200 FINISH
205 ; Turn off the arm power:
210 ARM OFF
220 STOP Finished for the day!
```

D-3 Example 3:

Another possibility for an AUTO_ST program is a "Pure executive" program. In such a program, all activities are contained in subroutines. For instance, an Autostart program could look like this:

```
100 GOSUB EXIT
200 GOSUB INIT
300 GOSUB MAIN_PRO
400 GOSUB GOODNITE
500 STOP
```

In this case, EXIT waits for arm power, exits the Bracket and HOMEs the robot. INIT initializes several system parameters (gripper status, I/O status, robot starting speed, tool transform, **WITH** command etc.), MAIN_PRO runs the application in a loop form, and GOODNITE puts the robot back into the bracket and turns off arm power.

This program format could be used for any industrial robot application.

APPENDIX E - A151/A251 BRAKE FEATURES

Introduction

The A151 and A251 robots have an additional feature not found in the A150 and A250 robots. They have electric brakes on joints 2, 3, 4 and 5. All of the joints which will react under the influence of gravity during a motor power interruption have now been protected.

The main advantage of brake equipped robot systems is that when power is interrupted the brakes hold the robot in position, even if the robot is carrying its rated payload. The reaction of the brakes to a loss of arm power is less than 50 milliseconds. This results in a minimal droop of position of the arm when the arm power is removed.

The brakes are located in bell-shaped housings in line with the joint 2 and 3 motors. For joints 4 and 5, they are contained in a cylindrical extension housing adjacent to the motors. The brake mechanisms are hidden within the plastic covers around the robot arm shoulder. The additional brake wires are contained within the same Hiroshi connectors as used in A150/250 robots.

Operation

The operation of the brakes is invisible to the operator in most circumstances. The brakes are activated entirely separately from the controller software. The control of the arm power determines the state of the brakes. When the arm power is applied, the brakes are released; when the arm power is OFF, the brakes are engaged. This mode of operation makes them fail-safe. In other words, if a condition arises that will remove arm power, then the brakes will always be applied. Such conditions are:

i) Software failure creating a loss of the watchdog signal, which removes arm power.

ii) Pressing the E-STOP mushroom pushbutton on the controller front panel.

iii) Opening the remote E-STOP circuit that removes arm power.

iv) Using the software commands DISABLE ARM, or ARM OFF which removes arm power.

Since the brakes are linked to the arm power, only the front panel switch which turns on the arm power can disengage the brakes.

Operational Differences

The only notable differences in operation of the robot will be in the LIMP mode teaching. Typically in this mode with the A150/250 robots, the operator could turn off arm power, then physically move the robot by manually supporting the arm and moving it through many positions. Turning the arm power off removes the resistance of the motors during this sequence. Now, the operator cannot remove arm power since the arm will lock into position (except joint #1). The operator must put the robot in LIMP mode, which does not affect the state of the arm power.

Physically, the A151/251 robots have the same swing radius as the A150/250 robots. The swing radius is the distance swept by the robot shoulder assembly as joint 1 is moved. This may impact applications where items are positioned close to the robot base.

Service Considerations

The brake assemblies of the A151/251 robots are not included in any routine maintenance schedules. The controller has a 2 amp fuse that protects the brake circuitry. Failure of the brakes to disengage when the arm power is applied may indicate a faulty fuse. This fuse can be found on the Encoder Connector board. Refer to section 7-11 of the Technical manual for a description of this board.

The brake release signal is carried through a conductor in the encoder cable. The return for each brake is the same conductor as the motor return wire which is part of the power cable. For this reason, the brakes cannot be released if either cable is disconnected between the controller and the arm.