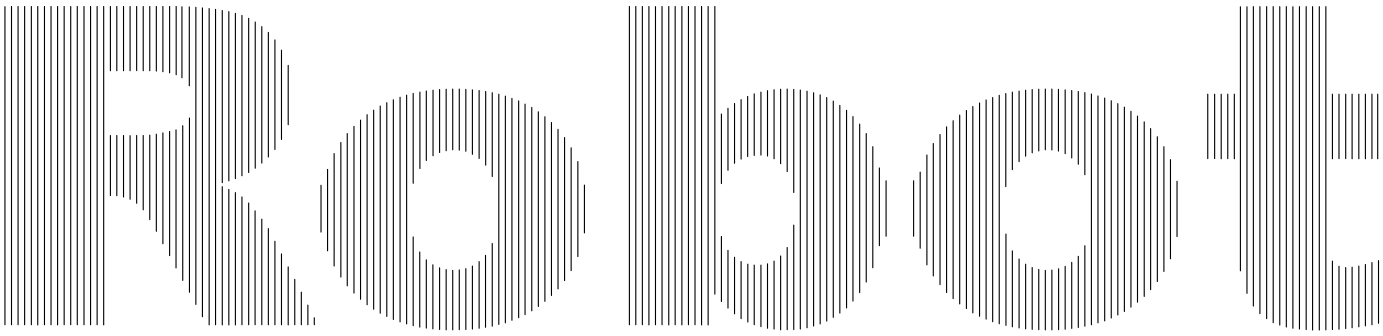
Simple and friendly

Kawasaki Robot

KAWASAKI ROBOT
C Series Controller

AS LANGUAGE PROGRAMMING MANUAL



KAWASAKI HEAVY INDUSTRIES, LTD.

90209－1006DEC

Preface

This manual is subject to change without prior notice and without our legal responsibility.

Kawasaki pursues a policy of continuing improvement in design and specification of the product. The right is therefore, reserved to vary this manual without prior notice.

This manual is prepared with the best care but anyway it is impossible to exclude any misinterpretation by the user or any spelling mistakes.
In case of doubts or if the robot behavior is different from that reported in this manual please would you get in touch with the nearest Kawasaki or Kawasaki's agent.

KHI cannot anticipate every possible circumstance that might involve a potential hazard. If a procedure, work method or operating technique not specifically recommended by KHI is used, you must satisfy yourself that it is safe for you and others. In this case you are completely responsible for eventual damages on human bodies or things.

If you discover physical defects in the manual, KHI will replace the manual at no charge during 90 days period after you purchased the robot.

If you do not observe warnings and/or caution to be taken in the manual as described below, it might result in the serious injuries or damages on human bodies or robot.
Therefore you must take extreme care to observe these warnings and cautions.

[SYMBOLS]

Whenever you see the symbols shown below in this manual or on the machine, read and understand their messages.

⚠ DANGER

This danger symbol identifies special warnings or procedures which, if not strictly observed, will result in serious injury or death.

⚠ WARNING

This warning symbol identifies special warnings or procedures which, if not strictly observed, could result in serious injury or death.

⚠ CAUTION

This caution symbol identifies special instructions or procedures which, if not correctly followed, may result in injury.

[NOTE]

This note symbol identifies supplementary information, or to emphasize a point or procedure, or to give a tip for easier operation.

This Manual has been described for the installation, maintenance and inspection of KAWASAKI ROBOTS Traverse Unit Z-Series.
We sincerely hope that you will please understand the contents of this Manual well and strictly observed the safety at all times. After that, start operation.
In addition to the above, in the Robot arm section, carefully read the “Installation and Connection =Robot Arm=” edition
and in the control section, also read the “Installation and Connection =Controller=” edition
and read the “Maintenance and Inspection” edition.

[SAFETY NOTICE]

If you do not observe warnings and/or cautions to be taken in the manual as described below, it might result in the serious injuries or damages on human bodies or Robots.
Therefore, you must take extreme care to observe these warnings and cautions.


MEMO

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

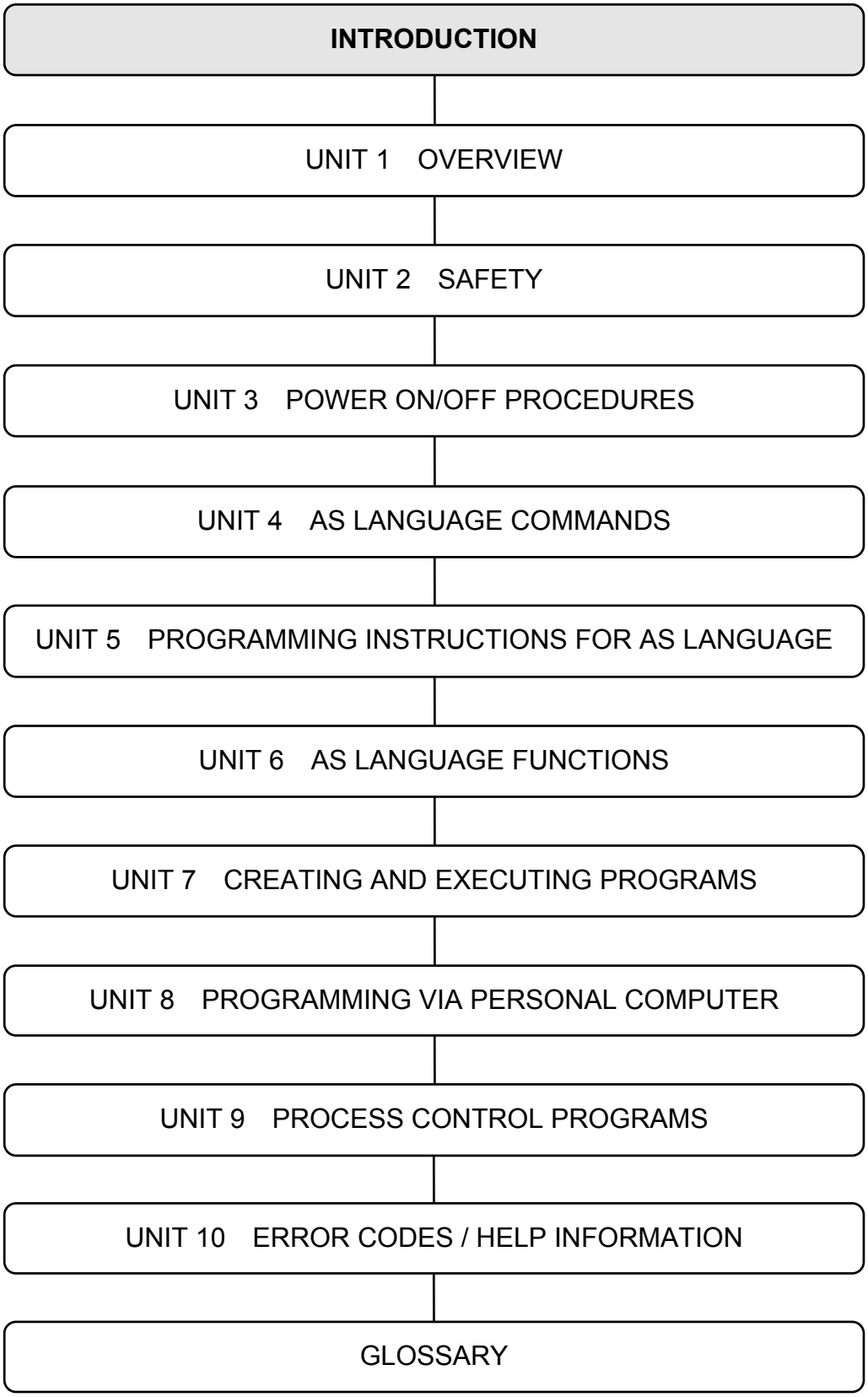
TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS


MEMO


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION
GLOSSARY

INTRODUCTION

I.0. INTRODUCTION

The *AS Language Programming Manual* is designed to assist the user whose primary responsibility includes programming and operating Kawasaki industrial robots on a daily basis. AS language is a computer control language designed specifically for use with Kawasaki robot controllers. This text provides information on creating, running and editing programs using AS language commands. The use of intuitive keywords, syntax sequences, and interface commands makes the AS Language relatively easy to learn.

AS language provides the programmer with the ability to precisely define the task a robot is to perform. Programming the robot with a computer control language (AS) also lends the ability to integrate peripheral components into the program. Components typically interfaced with AS language programs include: programmable logic controllers (PLCs), lasers, weld controllers, gray scale vision, and remote sensing systems.

AS language programs also provide outstanding performance in terms of robot trajectory control. Program location points can be stored and played back as either joint angles representing the manipulator configuration (precision points) or geometrically defined locations in the work envelope (transformations). Transformation locations can also be defined based on their relative position to one another (compound transformations). These capabilities allow program locations to be shifted/moved based on parameters and variables identified in the AS language program.

INTRODUCTION

I.1. DESIGN SPECIFICATIONS FOR C CONTROLLER

Control System:	32 bit RISC main CPU 32 bit RISC CPU for multi function panel unit 32 bit RISC servo CPU controller (one per 3 axes) Software controlled AC servo drive system utilizing PWM (pulse width modulation) circuitry	
Number of Axes:	6 standard; 7th optional	
Motion Control:	Teach mode	- Joint Base Tool
	Repeat mode -	Joint movement Linear movement Circular movement (optional)
Memory:	CMOS RAM	
Memory Capacity:	Standard - 1,024 KB (approx. 5,000 steps) Optional - 4,096 KB (approx. 35,000 steps)	
Accuracy:	Four levels of accuracy for block step programs:	

UX/UT-series
Adjustable between 0.5 mm ～ 5,000 mm

UZ-series
Adjustable between 0.3 mm ～ 5,000 mm

JC5/JW5/JS5/10/30/40
Adjustable between 0.1 mm ～ 5,000 mm

FS30L/FS45N

INTRODUCTION

Adjustable between +/-0.15 mm

FS10E/FS20N
Adjustable between +/-0.1 mm

FS06N/FS03
Adjustable between +/-0.05 mm

FS02/FC02
Adjustable between +/-0.03 mm

Speed: 10 levels of speed for block step programs
(adjustable between 0% ～ 100%)

Limitlessly adjustable speed via AS language
(adjustable between 0% ～ 100%)

Absolute speed adjustment in mm per second via AS language

Data Editing: Step insertion, deletion, and modification
Rewriting or modifying positional information

Software Features: Continuous path motion control - CP ON/OFF
Time delays
Coordinate modification
Process control programs (3)
Peripheral equipment control
Interrupt signal control
Error interrupt control
Input of real, string, and integer variables
Local variables
Subroutine calls with arguments (maximum stack = 20)
Program weld schedules
Servo shutdown timer
Auto Start function

INTRODUCTION

I/O Signals:	1FR I/O board	32 inputs/32 outputs (96 including dedicated signals)
	1FS RI/O board (option)	
	Robot I/O	128 I/O (including dedicated signals)
	Robot internal	128
	Relay circuit	32 I/O
	A-B PLC	64 I/O
	Weld controller	32 I/O
	Non-retentive	128
	Retentive	16
	Timers	16
	Counters	16
	Message display	64
	Slogic status	16
	Control Net (optional)	

INTRODUCTION

Dedicated Signals: Outputs - Motor Power ON
Error Occurrence
Automatic
CYCLE_START
Teach Mode
HOME1
HOME2
Power ON
RGSO
Ext. Program Select (RPS) Enabled

Dedicated Signals: Inputs - Ext. Motor Power ON
Ext. Error Reset
Ext. Cycle Start
Ext. Program
Select Start (JUMP)
JUMP_ON
JUMP_OFF
JUMP_ST
Ext. Program Select Start (RPS)
RPS_ON
RPS_ST
Number of RPS Code Signal
First Signal Number of RPS Code
Program Reset
Ext. Hold (EXT_IT)
Ext. Condition Wait (EXT_WAIT)
Ext. SLOW REPEAT MODE

Error Messages: Error code messages, self-diagnosis, error logging, and operation logging

Special Features: Program check mode
Adjustable restriction of JT1
Terminal box on robot arm (optional)

INTRODUCTION

Interface panel for robot applications (optional)
Over travel limit switch - JT1 (JT2, JT3 option)
Power lockout (C3X,C4X,C5X,C80 controller)
Ethernet (option)

INTRODUCTION

Multi Function Panel: (optional)	Deadman safety switches 8 inch color LCD Touch panel Teach-lock function Emergency stop switch Pen for touch panel Section for inserting PC card
Teach Pendant: (optional)	Deadman safety switches Teach-lock function Emergency stop switch Membrane switch keypad Alphanumeric LCD
Supplemental Data Storage:	PC SRAM Memory card 1, 2 or 4 MB, PCMCIA 2.1 Slot (optional) Floppy disk drive (optional) Personal computer (optional)
Power Requirements:	Standard Spec.: 3-phase 200/220 VAC North Am Spec.: 3-phase 400/440/460/480/515/575 VAC European Spec.: 3-phase 380/400/415/440/460/480 VAC C70 Spec.: Single-phase 200-240 VAC C80 Spec.: Tolerance: +/- 10% Frequency: 50/60 Hz Rated Load: 10.5 kVA / C70 1.5kVA / C80 3kVA

INTRODUCTION

Ground: welder	Less than 100 ohm ground line separated from power ground

INTRODUCTION

Dimensions: Standard Spec.: W x D x H, 460.8mm x 430mm x 1240mm (inches) (18.1 x 16.9 x 48.8)

North Am. Spec.: WxDxH, 550mm x 500mm x 1150mm (inches) (21.7 x 19.7 x 45.3)

European Spec.: WxDxH, 550mm x 500mm x 1150mm (inches) (21.7 x 19.7 x 45.3)

C5X Spec.: WxDxH, 550mm x 600mm x 1150mm

C70 Spec.:WxDxH, 270mm x 500mm x 470mm

C80 Spec.:WxDxH, 470mm x 350mm x 850mm

Weight: Standard Spec.: approx. 80 kg (176 lbs)

North Am. Spec.: 250 kg (550 lbs)

European Spec.: 250 kg (550 lbs)

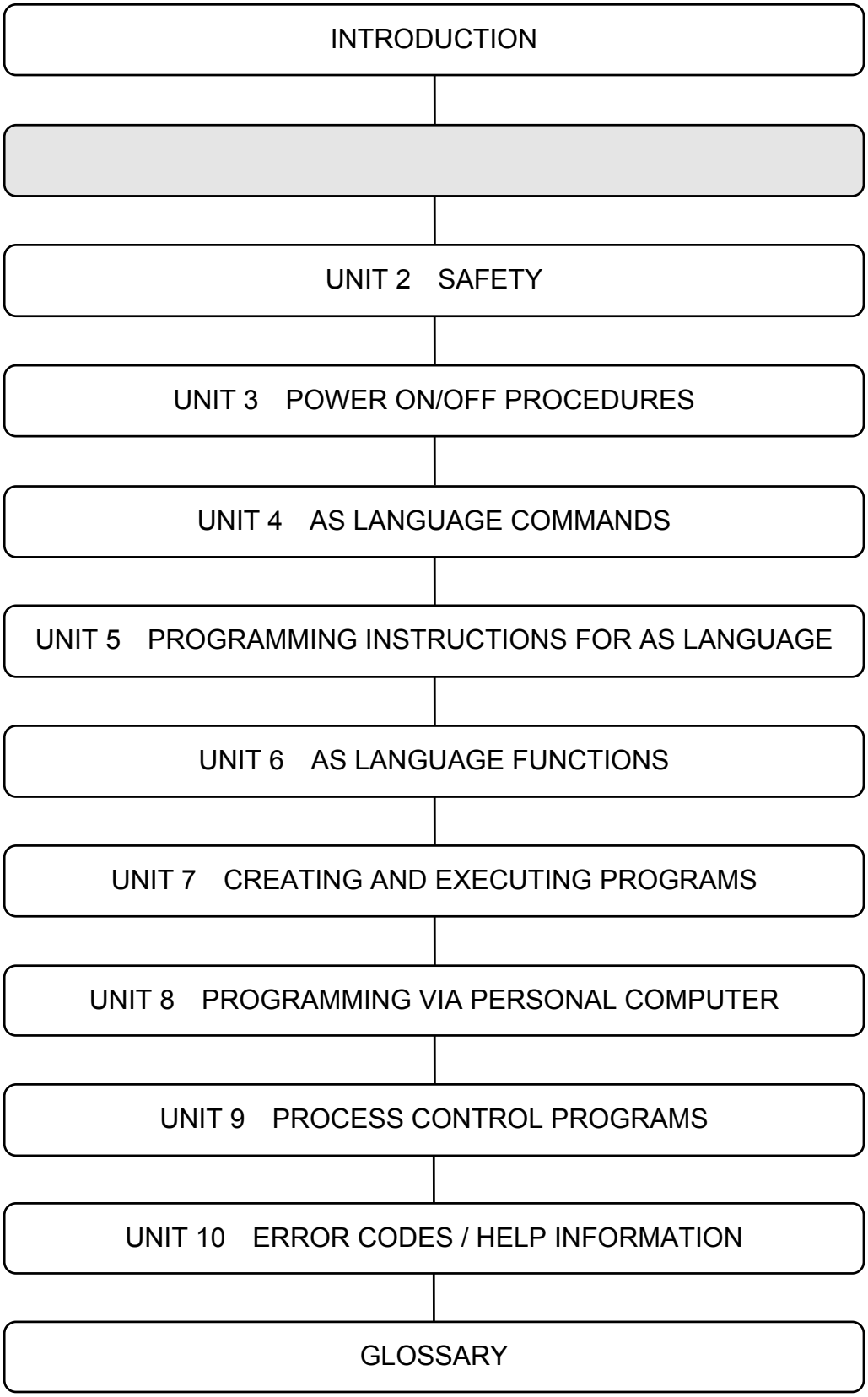
C5X Spec.: 260 kg

C70 Spec.:30 kg

C80 Spec.:60 kg

INTRODUCTION

MEMO


INTRODUCTION
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION
GLOSSARY

OVERVIEW

1.0. OVERVIEW

AS language is a computer control language that is resides in the controller's nonvolatile memory. Programs in AS are available as soon as power is supplied to the controller, and the initial startup process is complete, and accessible by Personal Computer or Multi-Function Panel.

There are three basic modes for the AS language program to accept commands and instructions, with each mode determining the types of commands/instructions available. The three modes are described below:

1. **Monitor Mode.** Available commands and instructions do not become part of a specific program, but are instead used to change system parameters and execute selected programming instructions. This win affect the status of programs currently underway or stored in the system memory.

2. **Editor Mode.** This is used to create new programs and modify existing ones. Editor mode uses a unique set of commands/instructions to work with specific programs.

3. **Playback Mode.** The system is in playback mode whenever a program is being executed in this mode, the controller is continuously computing robot motion control and executing program instructions, for which reason certain commands/instructions are available only when the processor is not otherwise occupied, and some monitor commands are not available at all. Not that in playback mode existing programs can be edited and new programs created, but the program currently being executed cannot be modified.

1.1. SYNTAX AND CONVENTIONS

Though AS language keywords and instructions can be entered in either upper or lower case without causing a syntax error, this manual shall represent such information only in capital letters. The reason for this is to be consistent with the AS language method of automatically processing keywords and instructions in uppercase. For example, if the AS language keyword “random” were entered in lowercase, it would be accepted, converted and stored as “RANDOM.”

In addition to keywords and commands, programs in AS contain elements freely identifiable by the programmer. Freely-defined elements can be entered in either uppercase or lowercase without causing a syntax error, but will be automatically converted to lowercase during processing. For example, if a location variable were entered in uppercase letters as “FIRST_WELD,” it would be accepted, converted, and stored as “first_weld”.

Many AS language keywords and commands have abbreviations that can be used in place of the entire word(s). For example, the command EXECUTE can be abbreviated to EX.

Many AS language commands also have an element called an “argument.” An argument is additional information for the program instruction provided by the programmer. For example, in the instruction SPEED 50 mm/sec, 50 is the programmer-provided argument. Some commands have optional arguments that may (or may not) be identified according to the preference of the programmer. At least one space is required between program instructions and arguments. If the space is not present to act as separator between the instruction and argument, a syntax error will be displayed when the Return key is pressed. If more than one space is entered between the instruction and argument the entry will be accepted and the extra spaces automatically removed from the line of code. If the instruction format calls for more than one argument, a comma must be used as a delimiter to separate arguments.

AS language instructions are processed according to conventions established by the American Standard Committee for Information Interchange, ASCII. All ASCII

characters have a numeric value assigned for their identification. The ASCII format used by AS language is compatible with all standard word processing a programs, processor making it possible to write AS language programs with a word processor and then download them to the controller.

1.1.1. MONITOR MODE SYNTAX

Monitor commands are entered at the > prompt. The acceptable format for individual monitor commands is identified with each command (covered in detail in unit 4 of this manual). The Multi Function Panel or laptop display will show the characters as they are selected from the keypad. Pressing the ↵ or Return key will cause the controller to process the monitor command, at which time (depending on the command) the robot will begin to move, or additional monitor prompts will appear.

Figure 1-1 is an example of how monitor mode entries appear on screen. DIR/P is a command to display the names of programs stored in system memory. Pressing the ↵ key will display the program names. The monitor command RENAME is used to assign a new name to an existing program. In this example, the program sample is being renamed to `samp1`. The DIR/P monitor command is used again to confirm that the name change occurred.

```
>Dir/p
  Programs
    arc   sample  zz      pg0
    pg1   pg2     pg10    pg555
>rename samp1=sample
>Dir/p
  Programs
    arc   samp1   zz      pg0
    pg1   pg2     pg10    pg555
```

Figure 1-1 Syntax of Monitor Commands

1.1.2. EDITOR MODE SYNTAX

Editor mode is used for creating or modifying specific programs. The example in figure 1-2 shows the EDIT command with an argument of test entered from the monitor prompt. This takes the programmer into the editor mode of a program named “test.”. The format of the program, as it is displayed in the editor mode, has a ? and a blank space after each line of code. Editing is done from the lines that begin with the ?. In this example, a command was given at line 3 to move the display to show line 6. At line 7 the command was given to return to the first step of the program.

```
>Edit test
  .PROGRAM test()
  1   LMOVE aa
  1?
  2   LMOVE bb
  2?
  3   JMOVE cc
  3? s6
  6   LMOVE dd
  6?
  7   DEPART 200
  7? s
  1   LMOVE aa
  1?
```

Figure 1-2 Editor Mode Syntax

1.2. LOCATION VARIABLES

AS language instructions can be used to identify two types of positional information: precision points and transformations. Each type has different characteristics and can be selected by the programmer to meet specific program needs. Locations are identified with names and stored in system memory as location variables.

The Cartesian coordinate system is used to identify locations in three dimensional geometric space. The axes are identified as X, Y, and Z, with the intersection being the origin of the coordinate system. In Kawasaki robot applications, the origin of the Cartesian coordinate system is located at the base of the robot. This coordinate system is also referred to as the base coordinate system.

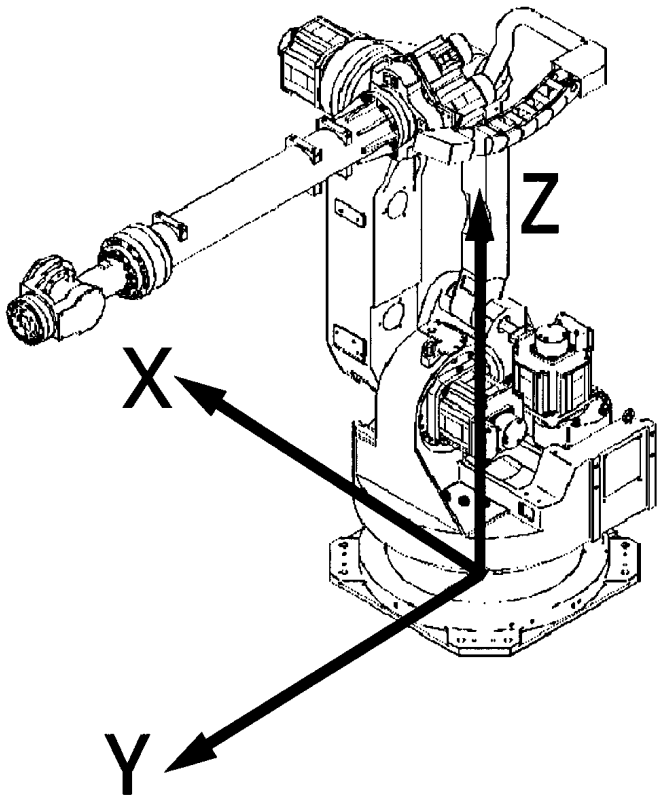


Figure 1-3 Cartesian Coordinate System

OVERVIEW

An easy way to determine the orientation and direction of the base coordinate system is illustrated by the “left hand rule,” shown in figure 1-4. When using the left hand rule, orient your left hand so that your arm represents the cables entering the base of the robot. From this orientation, the middle finger is held 90 degrees from the arm and points in the direction of positive X. The index finger points in the direction of positive Y, and the thumb to positive Z.

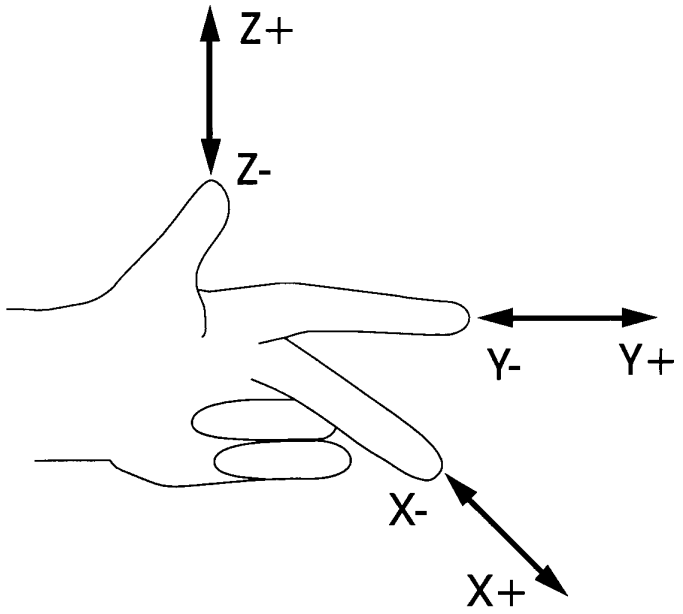


Figure 1-4 The left Hand Rule

In addition to the base coordinate system, Kawasaki robots utilize a second (tool) coordinate system to define the orientation of the tool center point. The tool coordinate system’s X, Y, and Z directions change as the robot wrist is moved; the base coordinate system’s orientation is fixed and does not change with robot movement. Figure 1-5 shows the orientation of the tool coordinate system when joints 4, 5, and 6 are in the zero or aligned position.

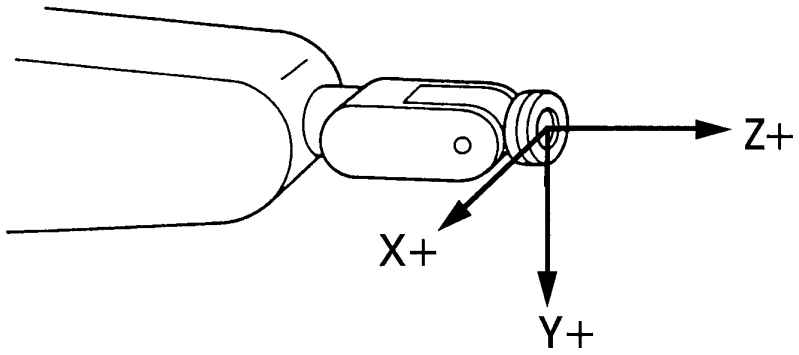


Figure 1-5 Tool Coordinate System

1.2.1. PRECISION LOCATIONS

When a precision location is recorded, the position can be represented in degrees by the exact position of the robot's individual joints. When precision points are played back, actual joint positions at the time of recording are repeated.

An advantage of programming precision locations is that the robot configuration/posture will always be the same as when the position was taught. Block step programming uses precision locations.

A disadvantage of precision locations is that they do not take into account robot work space as defined by the Cartesian coordinate system. The relocation of precision locations based on a dimensional shift is not possible because of the way location information is stored (joint angles). Another disadvantage of precision locations is that if a tool is installed with a different dimension, all program location points must be re-taught.

1.2.1.1. NAMING PRECISION LOCATIONS

Precision points, in order to be identified as location variables, must be assigned names preceded by the pound sign. Location names are limited to fifteen characters in length. For example, `#weld` identifies a precision point location named `weld`. They must begin with an alphabetic character, and may contain only letters, numbers, periods, and underlines. Care must be taken when assigning names to location variables to ensure

that the name is not used as an AS language command or keyword, and has not been previously assigned to another variable.

1.2.2. TRANSFORMATION LOCATIONS

A transformation location is represented by defining the positional location in terms of a Cartesian coordinate system (XYZ) having its origin in the base of the robot. The transformation position of the tool center point is defined with X, Y, and Z coordinates, and the tool orientation is defined by the three angles O, A, and T, measured in degrees from the coordinate axes.

OVERVIEW

One advantage of transformation locations is the ability to manipulate taught positions based on dimensional movements in the Cartesian coordinate system. Secondary coordinated systems for framing and work coordinates become available to the programmer when transformation locations are defined. Replacing a tool with one having different dimensions, does not require transformation locations to be re-taught (assuming that the dimensions of the new tool have been correctly registered). Another advantage, and a powerful feature of transformation locations, is the ability to define locations as combinations of location values. This type of location is called a compound or relative transformation. Such values are used to define location relative to another location. When compound transformation locations are defined and the original location is moved, all locations defined relative to the original location will also be moved.

A disadvantage of transformation locations is that the specific robot configuration or posture at the time of teaching may not be the same as when the robot travels to that location as part of a program. When using transformations and compound transformations, programmers must understand the properties of these types of locations to prevent error messages or unwanted robot motion.

1.2.2.1. NAMING TRANSFORMATION LOCATIONS

Transformation location points must be assigned names to identify them as location variables. Names are limited to fifteen characters in length, and must begin with an alphabetic character. They may contain only letters, numbers, periods, and underlines. Care must be taken when assigning names to location variables to ensure that the name is not used as an AS language command or keyword, and has not been previously assigned to another variable.

1.2.2.2. COMPOUND TRANSFORMATION LOCATIONS

Compound transformations are represented by defining robot locations and orientations relative to other transformation locations. This is useful in situations when several locations are defined relative to a reference location. To change the location of all points relative to a defined reference point, only the transformation location of the

reference must be updated. All locations defined relative to the reference point are automatically revised to reflect the location change of the reference point.

Figure 1-6 illustrates how compound transformations locations are identified relative to other transformations. The + sign is used to create compound transformations by making the location on the right of the + relative to the location on the left of the + In the example, location “A” has compound transformations A+B, A+C, and A+D defined relative to it. When location A is moved, A+B, A+C, and A+D are moved to maintain their positional relationships with A.

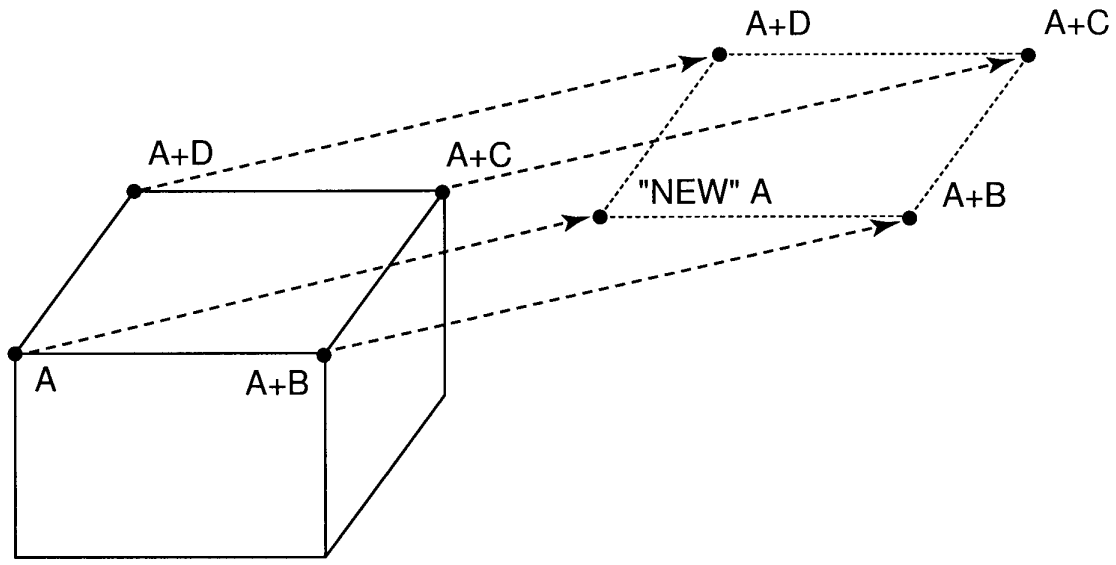


Figure 1-6 Compound Transformations

1.3. NUMERIC INFORMATION

Numerical expressions are used in AS language programs to provide a variety of information. These expressions consist of a combination of numerals, variables, operators, and functions which return numeric values. Numerical expressions can be used for mathematical calculation and as part of program instructions. Four types of

numerical information are available to the AS language programmer.

1.3.1. INTEGERS

Integers are numeric values without a fractional part (whole numbers). In AS, the acceptable range for integers is from -16,777,216 to +16,777,216. Values exceeding this range are rounded to seven significant digits. Thought integers are usually entered as whole numbers in decimal notation, they may also be expressed in binary or hexadecimal notation. The following are examples of integers:

572 Decimal Notation
^B101 Binary Notation of 5
-^H1000 Hexadecimal Notation of –4096

1.3.2. REAL NUMBERS

Real numbers have both an integer and fractional part. Like integers, real numbers are positive, zero or negative. Real numbers differ from integers in that they can be represented in scientific notation; In other words, using an exponent. The exponent may be either negative (power of 1/10) or positive (power of 10). Real number values are stored with an accuracy of approximately 7 digits.

1.3.3. LOGICAL VALUES

Logical values have only two states, ON and OFF. These two states are also referred to as *true* and *false*. True values are stored as -1 and false values as 0. The AS language uses the reserved words *true* and *false* to indicate these two states. For example, if the real variable `abc` is assigned a logical true value (`abc=TRUE`), a -1 is stored as the value of abc.

1.3.4. ASCII VALUES

The American Standard Code for Information Interchange (ASCII) is an 8-bit code that allows keyboards to communicate with computers. Each key is assigned a unique value recognizable by the computer. ASCII values are identified by apostrophe symbol (’). For example, `'a` has a value of 97 and `'A` has a value of 65. (See the appendix for a complete list of ASCII values).

1.3.5. NUMERICAL OPERATORS

Expressions used in AS are described and defined using, arithmetic, relational, logical, and binary operators. Most applications use a numeric operator with two numeric values to produce a result or *answer*. Two numerical operators, NOT and COM, require a single numeric value only. (Numerical operators are described in table 1-1).

The relational operator == is the symbol for a mathematical equal, while = is an assignment symbol. Logical operators are not used for calculating numeric values, but for determining the logical state (TRUE, -1 or FALSE, 0) of a conditional expression.

OVERVIEW

Table 1-1 Numeric Operators

Type of Operator	Symbol or Format	Description
Arithmetic Operators	+	Addition
	_	Subtraction
	*	Multiplication
	/	Division
	^	Power (exponential)
	MOD	Remainder
Relational Operators	<	Less Than
	<=, (=<)	Less Then or Equal To
	==	Equal To
	<>	Not Equal To
	>=, (=>)	Greater Than or Equal To
	>	Greater Than
Logical Operators	AND	Logical AND
	NOT	Logical Complement
	OR	Logical OR
	XOR	Exclusive Logical OR
Binary Operators	BAND	Binary AND
	BOR	Binary OR
	BXOR	Binary XOR
	COM	Binary Complement

1.3.6. ORDER OF OPERATIONS

The order in which numeric operations are calculated is determined by a set of priorities. Programmers can change this order, however, by placing operations to be performed first within parentheses. Operations including parentheses are calculated such that operations within parentheses are evaluated first, evaluation then continuing outward in sequence. Inside the parentheses, or when no parentheses exist, numeric operations are evaluated in the following order:

1. Evaluate functions and arrays.
2. Process relational operators of pair character strings.
3. Process power operators "^ "
4. Process unary operators "-"
5. Process multiplication "*" and division "/" operators from left to right.
6. Calculate residues (MOD operators) from left to right.
7. Process addition "+" and subtraction "-" operators from left to right.
8. Process relational operators from left to right.
9. Process COM operators from left to right.
10. Process BAND operators from left to right.
11. Process BOR operators from left to right.
12. Process BXOR operators from left to right.
13. Process NOT operators from left to right.
14. Process AND operators from left to right.
15. Process OR operators from left to right.
16. Process XOR operators from left to right.

1.4. REAL VARIABLES

Real variables are used in AS to assign a numerical value to a variable name. The = symbol is used to assign the value on the right hand side of the expression to the variable on the left. The expression `hold=73`, for example, will assign the integer value of 73 to the real variable “hold”. A variable is automatically defined the first time it is assigned a value. Attempting to use an undefined variable (a variable not yet assigned a value) will result in error.

The numeric value to the right of the = may be a constant, an existing variable, or a numeric expression. When an assignment instruction is processed, the value on the right is computed first, and then the value assigned to the variable on the left. For example, the assignment `space = 12+5` assigns the value 17 to the real variable “space.” If a real variable named “space” ware already in memory, the new value of 17 would replace the old value.

The = symbol used to assign values to real variables, and is not the mathematical equals sign. In AS, the expression `x = x+1` would assign a value to `x` increased by 1 each time the expression is evaluated. In a mathematical calculation, the expression `x = x+1` would not be acceptable because it violates the mathematical principle of communicative properties.

Real variable names must begin with an alphabetical character and can contain only letters, numbers, periods, and underscore characters. Their length is limited to fifteen characters. The characters can be entered in either upper or lower-case, but are always displayed in lowercase. AS language commands and keywords cannot be used for the names of real variables, and care should be taken to ensure that the selected name is not already defined and in use by another program. Because a real variable is available to an programs in the system once it has been defined, changing the value of a real variable may inadvertently affect otter programs.

1.5 STRING VARIABLES

String variable information available for use in the AS language is a sequence of ASCII characters enclosed by quotation marks. Since these indicate the beginning and end of a character string, they cannot be used within the string itself. ASCII control characters (CTRL, CR, and LF) cannot be used in a string either. For example, a command for printing would be entered as:

```
PRINT "Kawasaki"      Kawasaki will be displayed on the status screen.
PRINT "5+7 =", 5+7    5+7 = 12 will be displayed on the status screen.
```

Character strings can also be defined by using the assignment instruction =. String variables are identified by a $ preceding the string variable name. The format for assigning a character string variable is :

```
$name_of_ variable    = "string expression"
```

When an assignment instruction for a string variable is processed, first the value to the right of the = is computed, and then the value assigned to the variable on the left. The value on the right side may be a constant, string variable, string expression, or numeric expression. If the variable on the left side has never been used, it is defined automatically', if it has already been used, its old value is lost and a new value assigned. For example:

`$First = "Kawasaki"`	The character string `Kawasaki` is assigned to the string variable `$First`.
`$Next = "Robotics"`	The character string `Robotics` is assigned to the string variable `$Next`.
`$Name = $First + $Next`	String variables `$First` and `$Next` are assigned to the string variable `$Name`.

In the above example, the string variable `$Name` is assigned the sum of `$First` and `$Last`. When the command to `PRINT` the character string `$Name` is entered, the display

will return `Kawasaki Robotics`

1.6. ARRAY VARIABLES

An array is a group of values that share a single name. Each value in an array is called an *element* of the array. An element of a location array is specified in exactly the same way as an element of a numeric array by appending an index enclosed in brackets to the array name. A program example using an array variable is shown below:

PROGRAM	OUTPUT
`HERE edge`	`edge[1] = 120.45`
`DECOMPOSE edge[1] = edge`	`edge[2] = 145.67`
`FOR i = 1 TO 6`	`edge[3] = -95.43`
`Type "edge[",I,"] = "edge[i]`	`edge[4] = 90.45`
`END`	`edge[5] = 45.02`
	`edge[6] = 10.58`

In the above example, the current location of the robot is defined as `edge`. The `DECOMPOSE` instruction extracts the location component values (1 through 6) of `edge`. The program instructions between the `FOR` and `END` statements will be executed six times, while the `TYPE` instruction will display the location component values of `edge` individually.

SAFETY

2.0. SAFETY

2.1. INTRODUCTION

Safety is an important consideration in the use of automated and robotic equipment in the industrial environment. All operators, maintenance personnel, and programmers must be aware of all automated, peripheral, and robotic equipment occupying the work cell, and their associated operational and maintenance procedures. For this reason, it is recommended that all personnel who operate, maintain, and program Kawasaki robots' attend the appropriate Kawasaki-approved training course pertinent to the specific responsibilities of each job.

The following safety sections in this text are designed to support and augment existing safety guidelines that may be in use in your plant and/or are provided by municipal, state, or country; they are NOT designed to supplant or supersede any existing rules, regulations, or guidelines. Because safety is the primary responsibility of the user, owner, and/or employer, Kawasaki recommends that specific safety guidelines and recommendations be adopted from groups or individuals that are professionals in safety design and implementation.

All safety-related issues and descriptions, either presented in written or oral form from any representative of Kawasaki, Inc., are intended to provide general safety precautions and procedures, and by no means address all safety measures necessary for the protection of personnel in the work environment.

Kawasaki robots are considered safe for use in industrial environments when all safety guidelines are adhered to. Adherence to the guidelines for safe robot operation, as well as the protection of personnel and equipment, is the responsibility of the end user.

2.2. PERSONAL SAFETY

2.2.1. CATEGORIES OF PERSONNEL SAFETY

The issue of personnel safety can be divided into four separate categories:

- Personal safety
- Safety during operation
- Safety during programming
- Safety during inspection and maintenance

A description of each follows in this section.

2.2.1.1. PERSONAL SAFETY

Safety procedures are essential for operating, programming, and maintenance personnel. These procedures, which must be followed explicitly and regularly, are a part of everyday operational policy designed to protect the user. A plan for personal safety should in clued the following cautions:

- All maintenance, operating, and programming procedures should be fully understood, and all safety-related precautions taken before attempting to operate/maintain the robot or robot controller.
- Avoid wearing loose clothing, scarves, wrist watches, rings, and jewelry when working on the robot or robot. It is also recommended that if ties must be worn in your shop environment, that they be of the clip-on variety.
- Always wear safety glasses or goggles and approved safety shoes . Follow all applicable local, state, country, and plant safety specifications/procedures.

SAFETY

- Know the entire work cell or area that the robot occupies.
- Be aware of the entire work envelope of the robot and any peripheral devices.
- Locate all emergency stop buttons or switches.

SAFETY

- Avoid trap points in which personnel could become caught between moving and stationary devices.
- Prohibit personnel from entering the work envelope during automatic operations.
- Ensure that all personnel are clear of the work envelope before initiating any motion commands to the robot.
- Know beforehand how the robot will perform before initiating any motion commands.
- Be sure that the entire work area is free of debris, tools, fixturing, lubricants, and cleaning equipment before attempting to operate the robot.
- Have personal report unsafe working conditions to their supervisor or plant safety coordinator *immediately*.
- Be sure that all personal can identify, by name and function, all switches, indicators, and control signals capable of initiating robot motion.
- Never defeat, render useless, jumper out, or bypass any safety related device, whether mechanical or electrical in design.
- To ensure personal safety, properly install and maintain all safety devices approved for use.
- NEVER attempt to stop or brake the robot during operation with your body or person. Utilize Emergency-Stops only to stop robot motion.

2.2.1.2. SAFETY DURING OPERATION

- Be sure to identify the *work envelope*, being the maximum reach of the robot wall directions during robot operations.

- Always keep the work area clean and free of debris. This includes, but is not limited to, oil, water, tool, fixturing, and electronic test equipment.

- During teach operations, ensure that the only persons allowed into the work envelope are the teacher or the person operating the teach pendant. Make note of Teach Pendant's provisions to protect the operator, which include an Emergency-Stop, trigger switch, and deadman switch.

- Never block the operator's path of retreat.

- Always have a path of retreat planned during teach operations.

- AVOID pinch points.

2.2.1.3. SAFETY DURING PROGRAMMING

- Be sure to identify the *work envelope*, being the maximum reach of the robot in all directions, during robot operations.

- During teach operations, ensure that the only persons allowed into the work envelope are is the teacher or the person operating the teach pendant. Make note of the teach pendant's provisions to protect the operator, witch inclued an Emergency-Stop, trigger switch, and deadman switch.

- AVOID pinch points.

- During point-to-point playback operations, be aware that the robot is cognizant only of its present location and the next point it is requested to move to. It will execute this move with total disregard to what may lie in its path when the move is executed.

- Because playback accuracy and speed can affect the geometry of path coordinates, when changing accuracy or speed, always test run the program at a slow speed (or in point-to-point mode) before attempting continuous path operation in repeat mode.

- ALWAYS test run a new path program at a reduced speed (or in point-to-point mode) before attempting a high-speed playback operation in repeat mode.

2.2.1.4. SAFETY DURING INSPECTION AND MAINTENANCE

Before entering the work envelope to perform either inspection or maintenance procedures, be sure to the turn off the three-phase power, and tag and lock-out the disconnect switch.

> **⚠ WARNING**
>
> The input side (top) of the controller disconnect may still be live when the controller disconnect is turned OFF, if work is to be performed at the controller disconnect switch, turn the three phase power OFF at the source, and tag and lock-out

Additional safety measures

- Be aware that, because the brake assembly is in the servo drive motor, the axis of the robot will be unsupported and fall if the axis motor is removed.
- When using the axis brake release switches in the controller, be aware that the axis may fall if left unsupported.
- Turn off supply pressure and purge all lines to remove any residual pressure before working on pneumatic or high pressure water supplies.
- Assign only qualified personnel to perform maintenance procedures.
- Consult all available documentation before attempting any repair or service procedures.
- Use only Kawasaki-approved replacement parts.
- Before attempting to adjust or repair a device in the robot controller that may have

yellow interlock control circuit wires attached, be sure to locate and disconnect it's power source.

- If your installation is equipped with safety fences plugs, remove and hold the safety plug during inspection and maintenance procedures.

2.2.2. SAFETY FEATURES

The Kawasaki robot system is equipped with several features designed to safeguard the user. Some of these include:

- All Emergency-Stops are hardwired.
- The Multi-Function Panel, Teach Pendant, and Operation Panel are all equipped with Emergency-Stop push buttons.
- Robot velocities are software-monitored. Should velocity exceed set parameters, the robot will default into a velocity error state.
- Teach and check mode velocities are limited to a maximum of 250 mm/sec.
- All robot axes have preset software limits.
- All J-series mechanical units have overtravel hardstops on JT1 and JT2. All U-series mechanical units have overtravel hardstops on the JT1, JT2, JT3, and JT5 axes.
- All robot axes are monitored by the robot controller for velocity and deviation errors.
- All robot axes are equipped with 24 VDC electromechanical brakes. Should the robot lose line power, the robot arm will not drop because the brakes engage when power is off at the robot controller.

2.2.3. SCHEMATICS OF THE WORK ENVELOPE

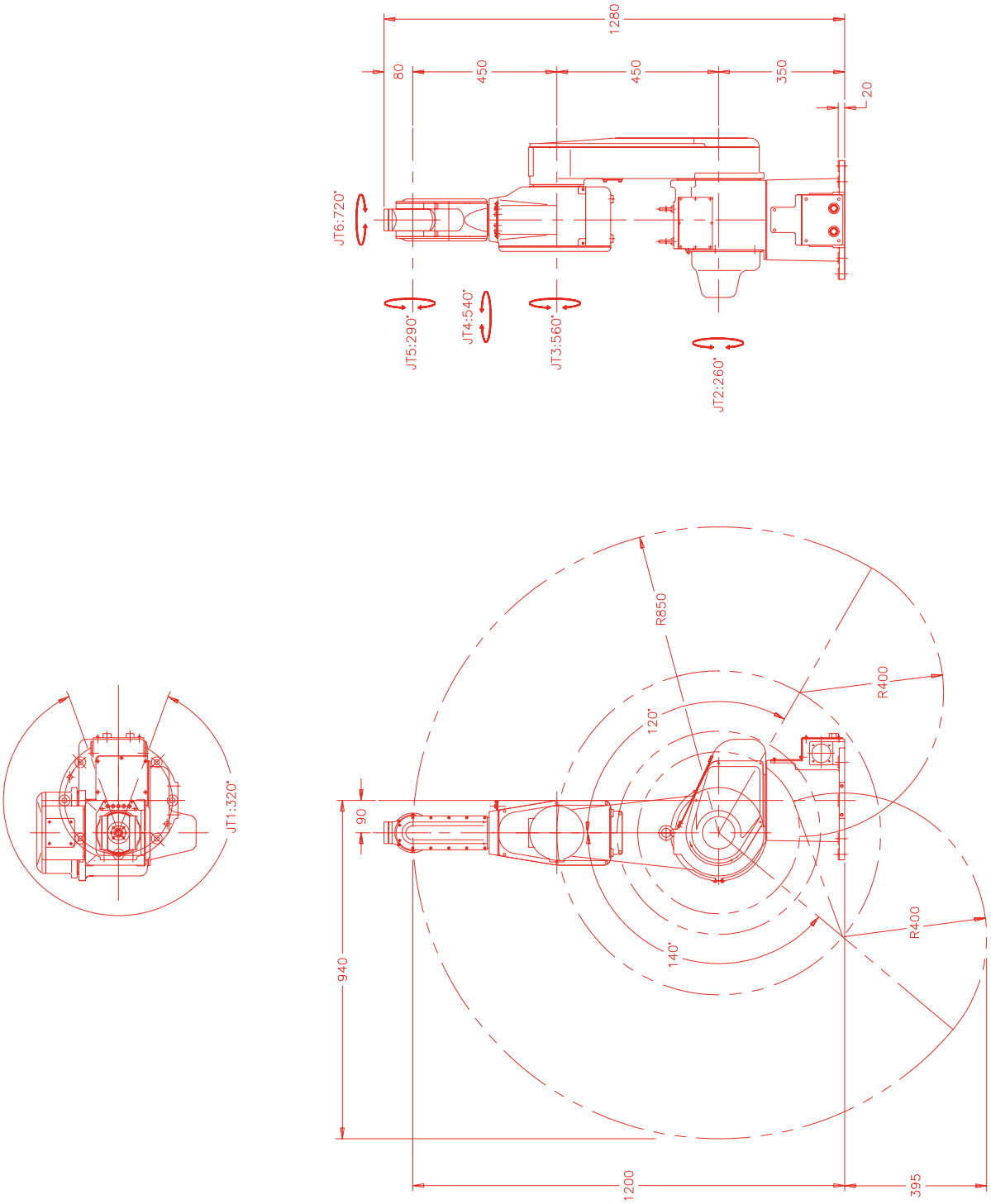


Figure 2-1 Work Envelope for models JA5/JC5/JS5/JW5

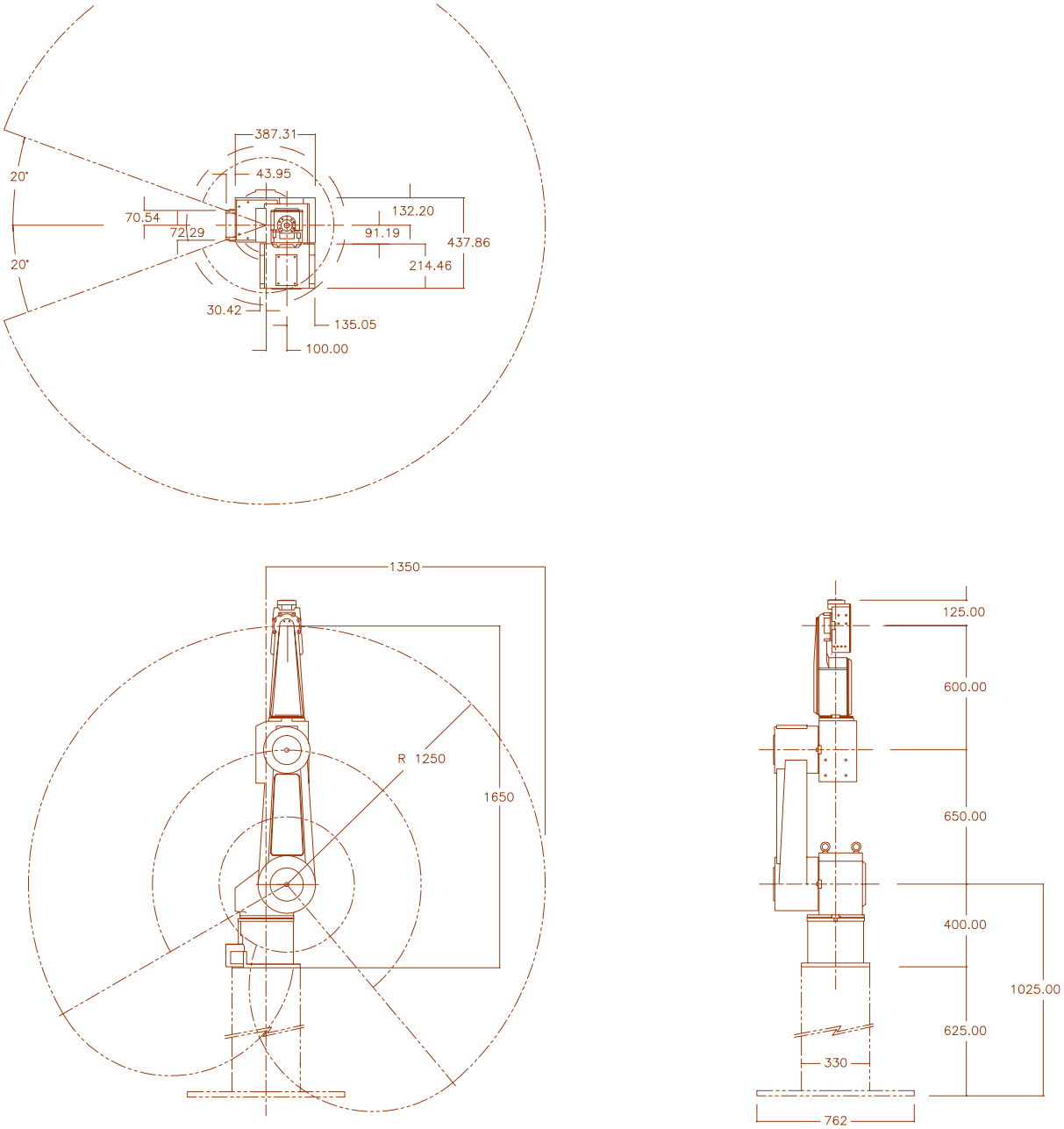


Figure 2-2 Work Envelope for models JA10/JS10

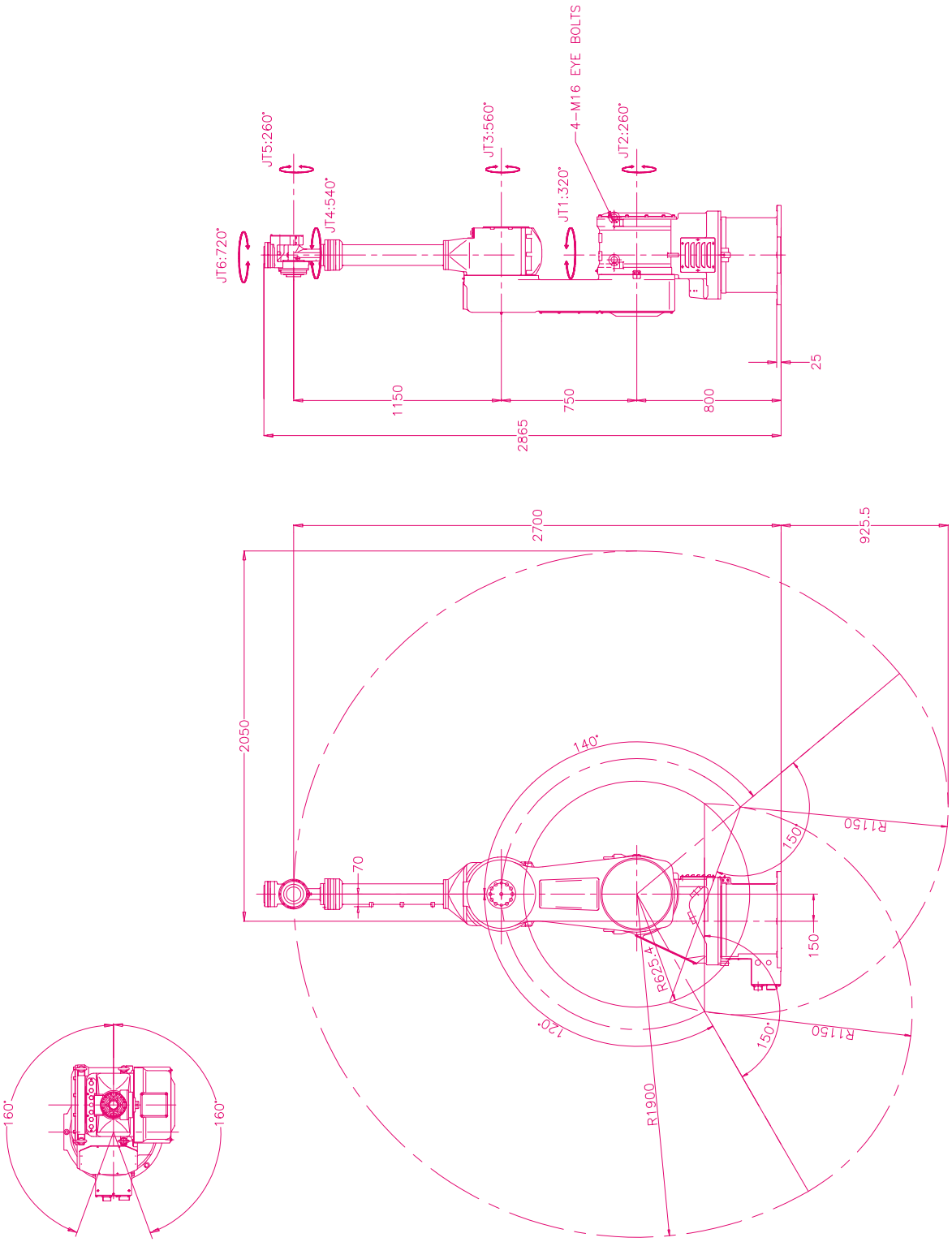


Figure 2-3 Work Envelope for model JS30

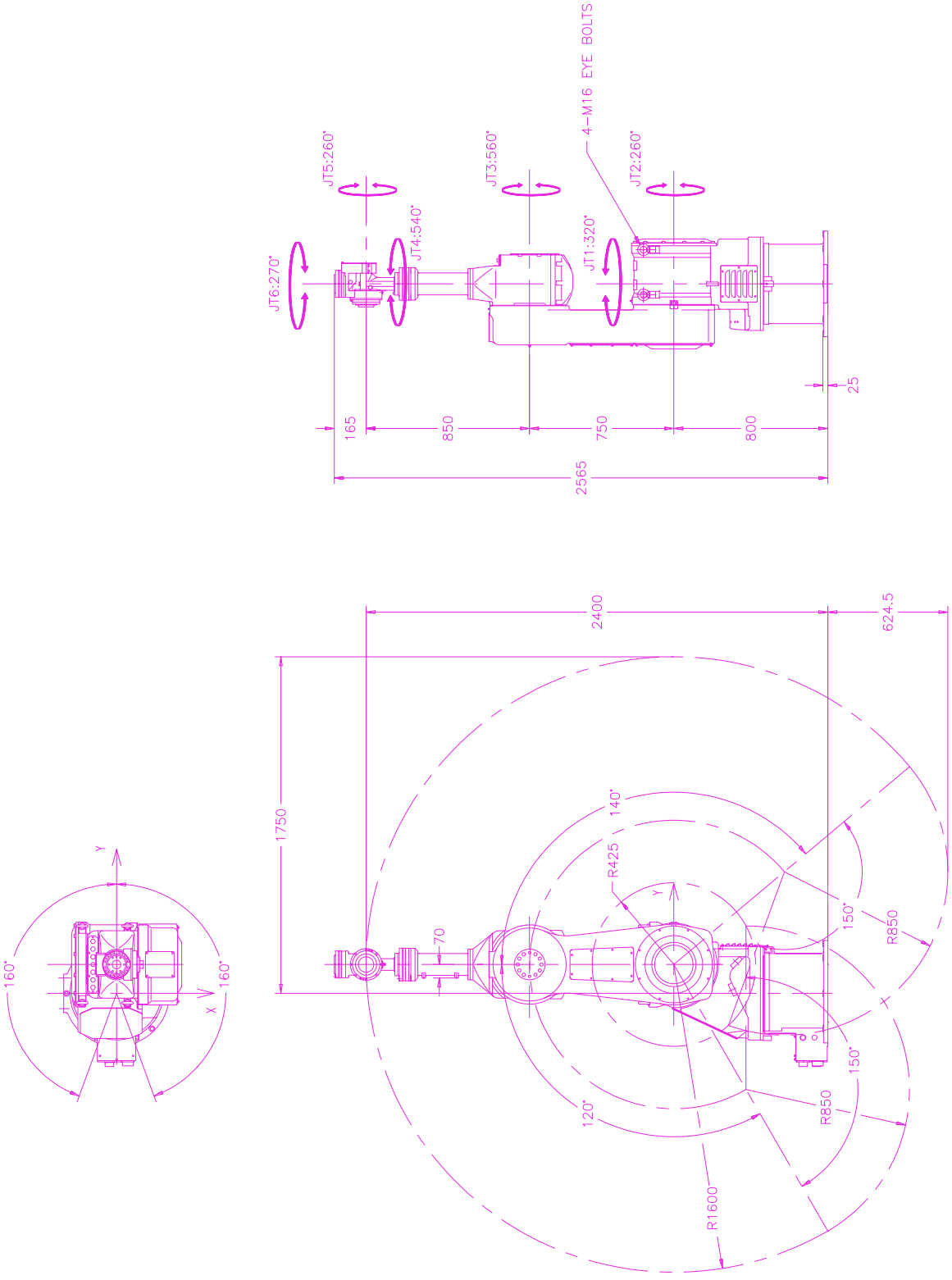


Figure 2-4 Work Envelope for model JS40

SAFETY

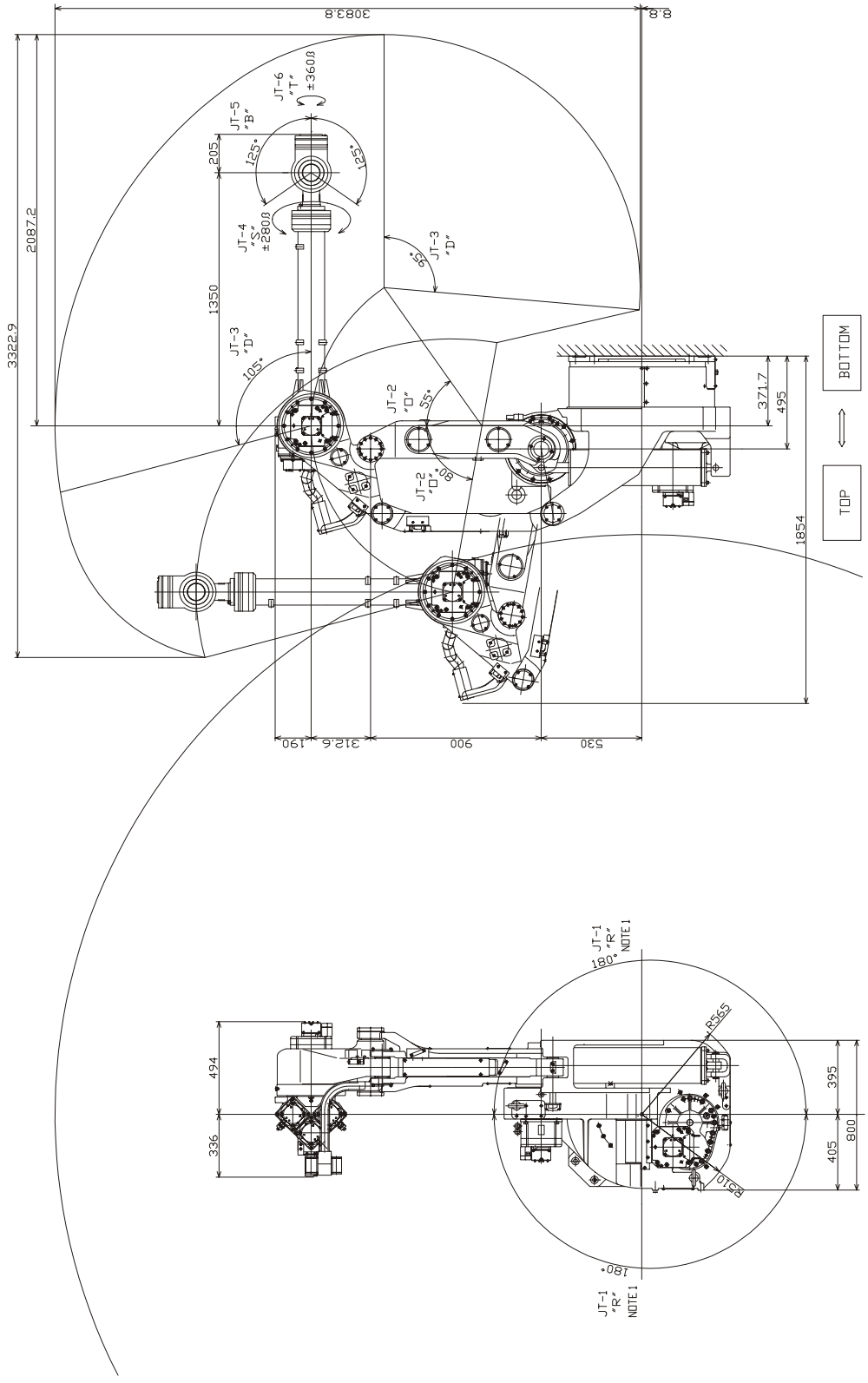


Figure 2-5 Work Envelope for model UT-Series

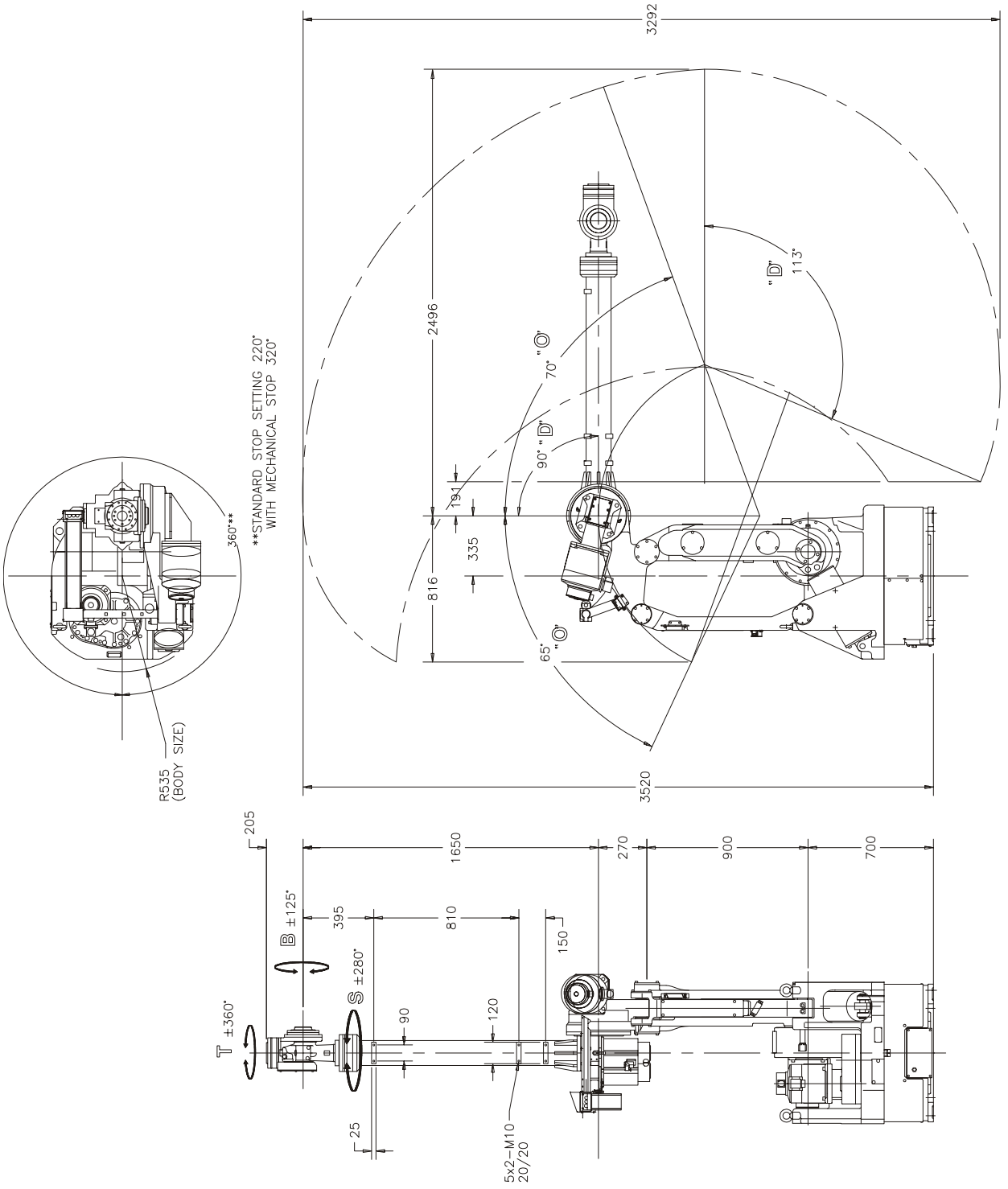


Figure 2-6 Work Envelope for model UX70

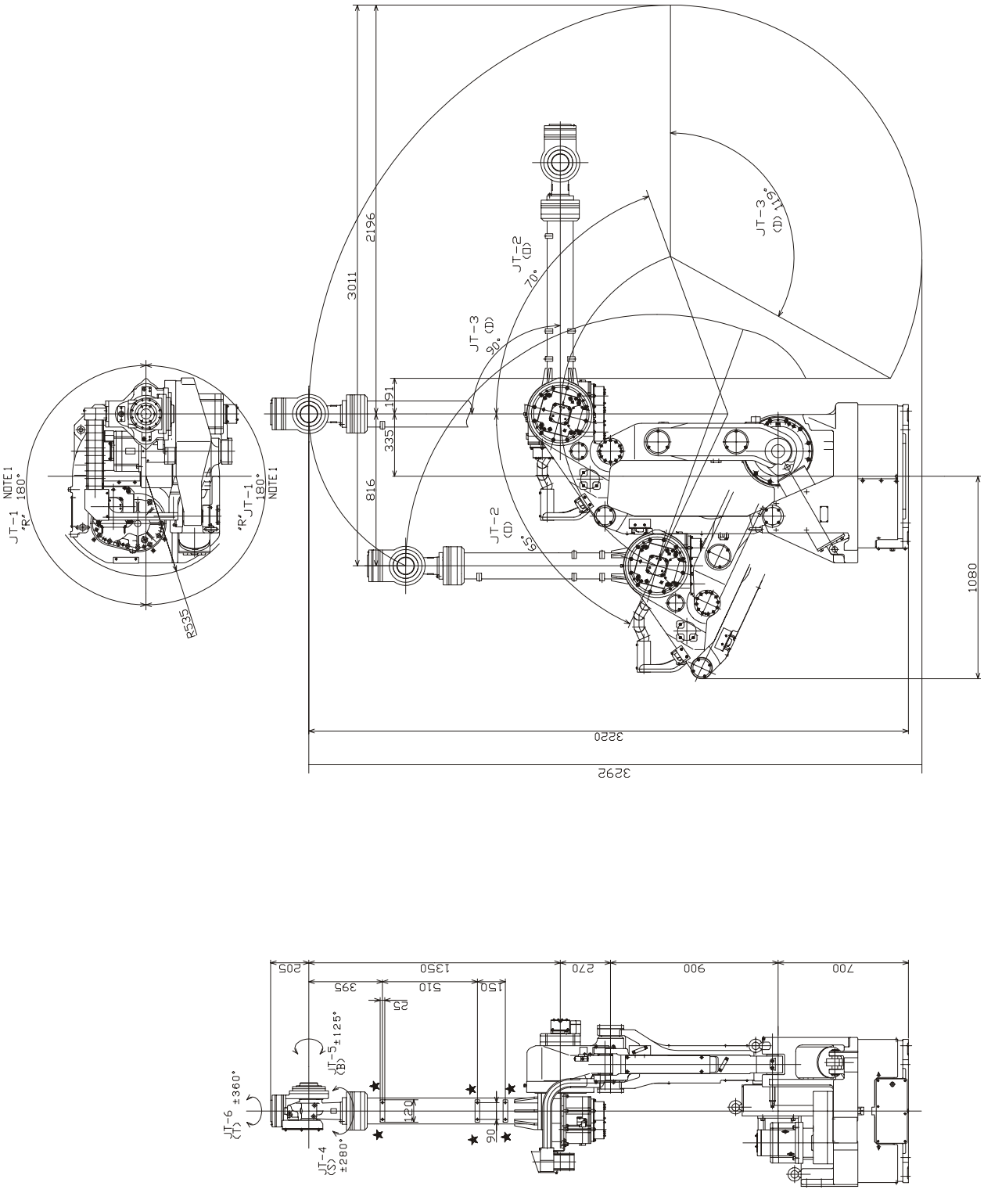


Figure 2-7 Work Envelope for models UX100/120/150

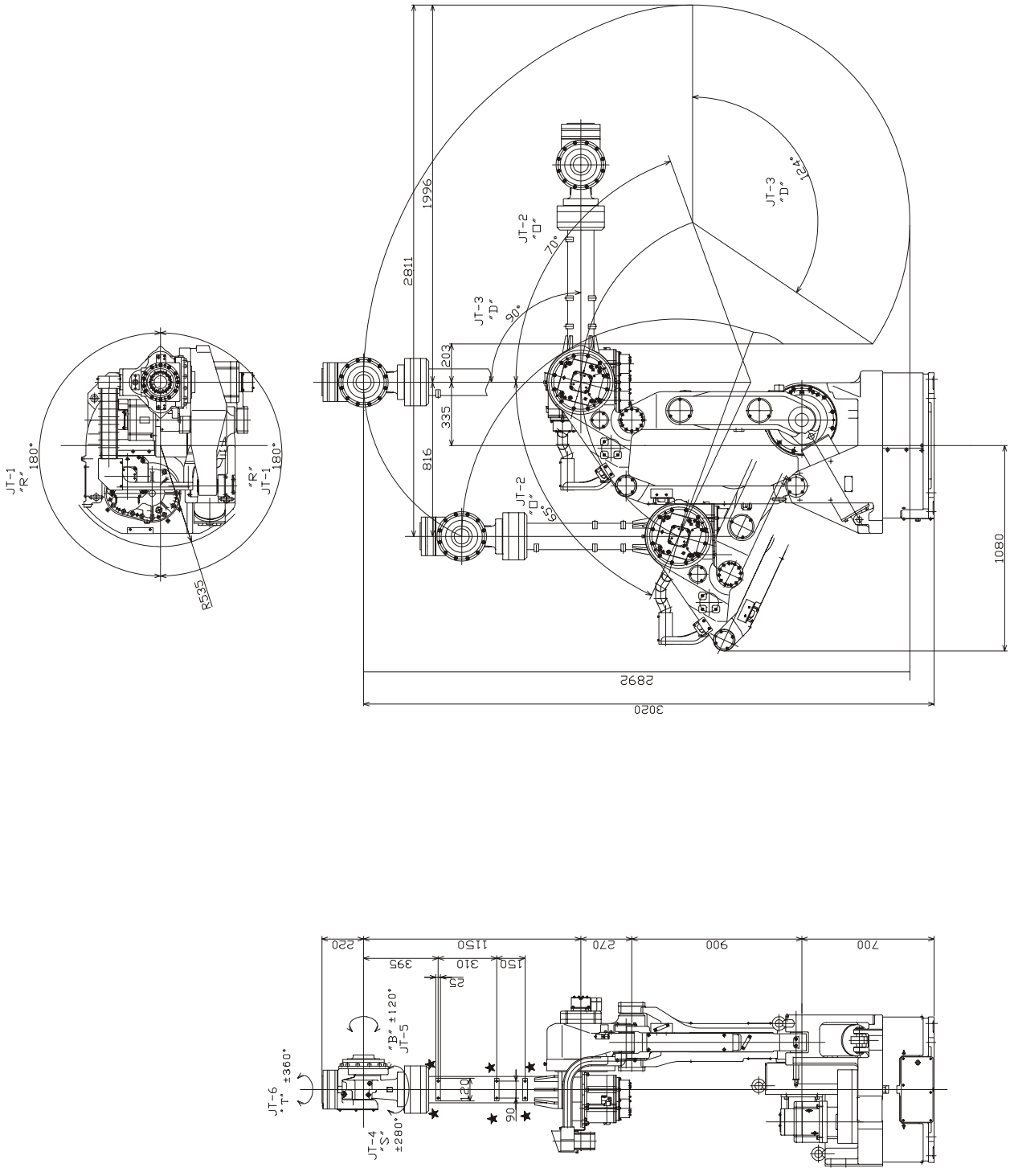


Figure 2-8 Work Envelope for modes UX200

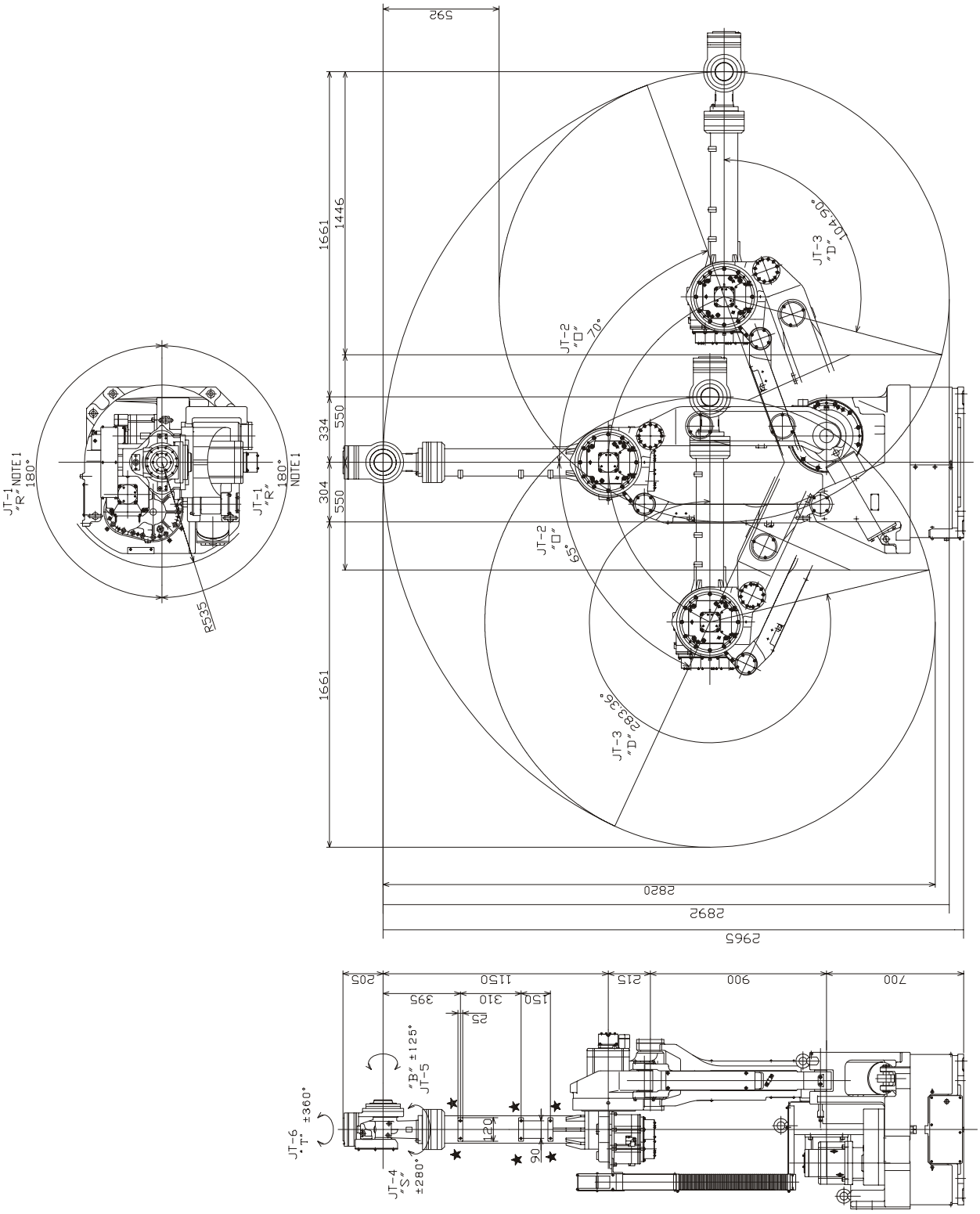
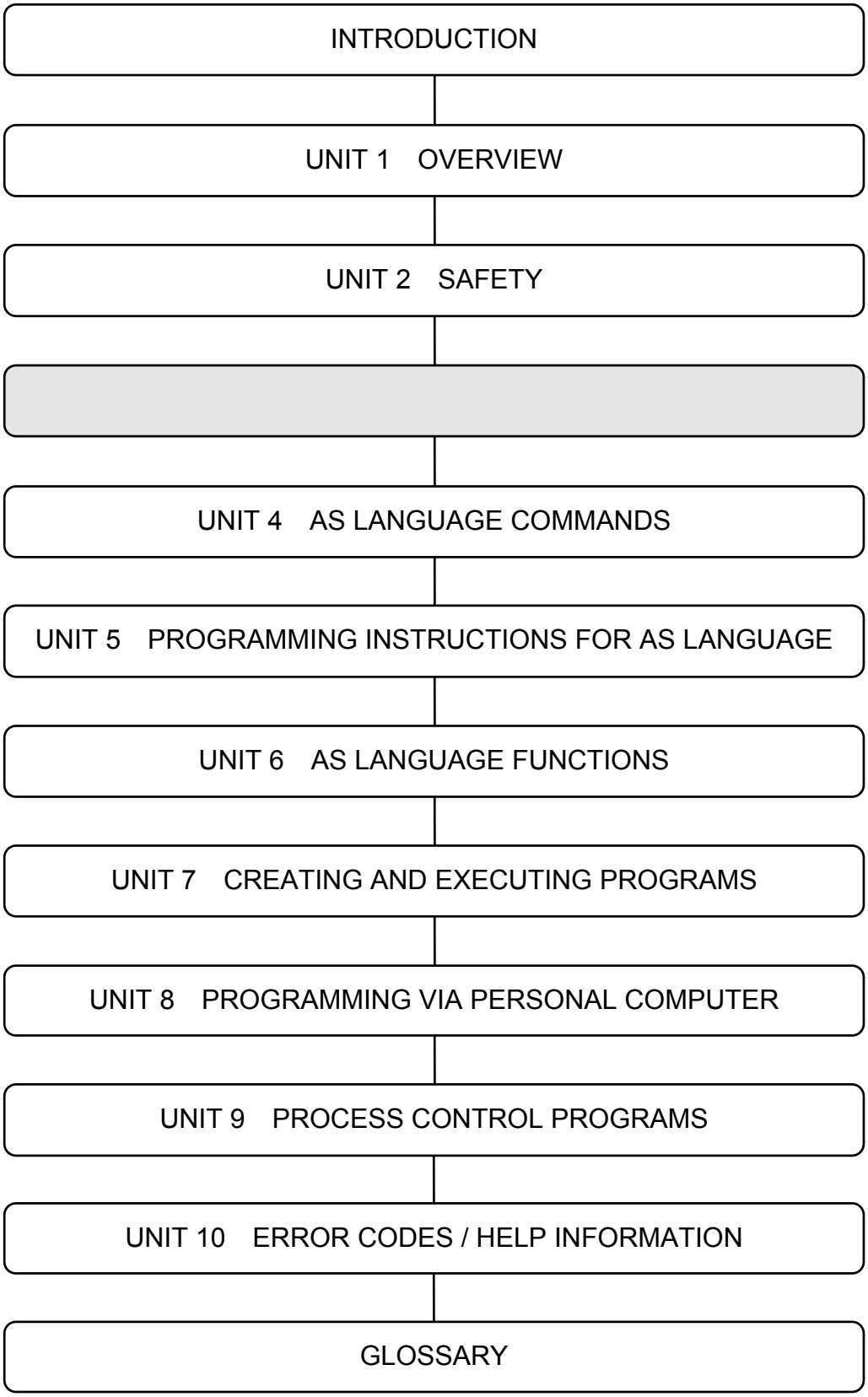


Figure 2-9 Work Envelope for models UZ100/200/150


MEMO


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION
GLOSSARY

POWER ON/OFF PROCEDURES

3.0. POWER ON/OFF PROCEDURES

This unit details power ON/OFF procedures for the robot controller and servo motors. Refer to figures 3-1 through 3-12 during these procedures.

3.1. CONTROLLER POWER ON/OFF PROCEDURES

3.1.1. CONTROLLER POWER ON

1. Ensure that all personal are clear of the work cell, and that all safety devices are operational and in place.

2. Turn the HOLD/RUN switch to the HOLD position.

3. Turn the controller's main disconnect switch to ON. The CONTROL POWER indicator lamp should light up.

3.1.2. CONTROLLER POWER OFF

1. Turn the HOLD/RUN switch to the HOLD position. The robot will decelerate and stop.

2. Press the EMERGENCY STOP switch. The CYCLE START and MOTOR POWER lamps should turn off.

3. Turn the controller's main disconnect switch to OFF.

POWER ON/OFF PROCEDURES

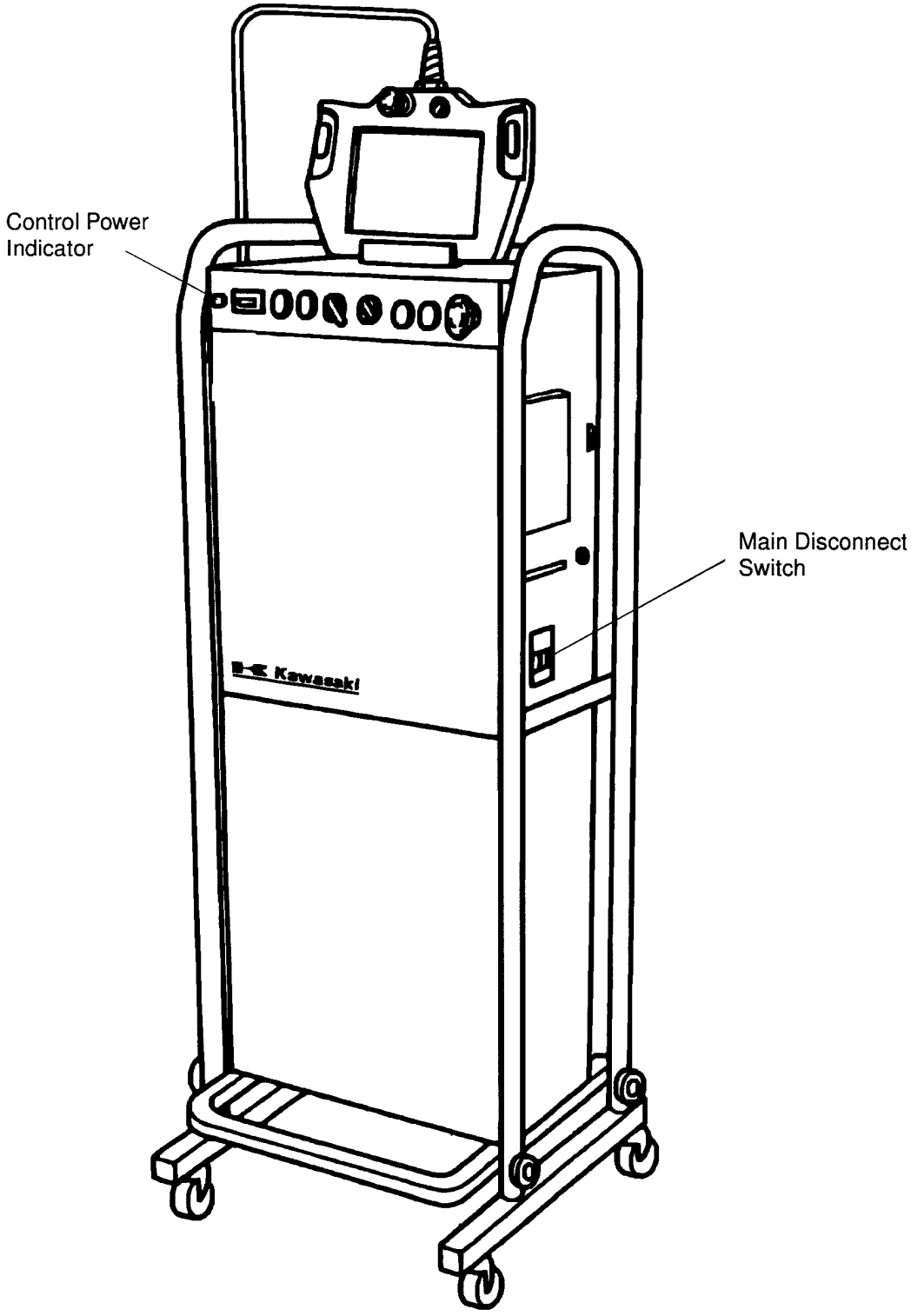


Figure 3-1 Standard C Controller

POWER ON/OFF PROCEDURES

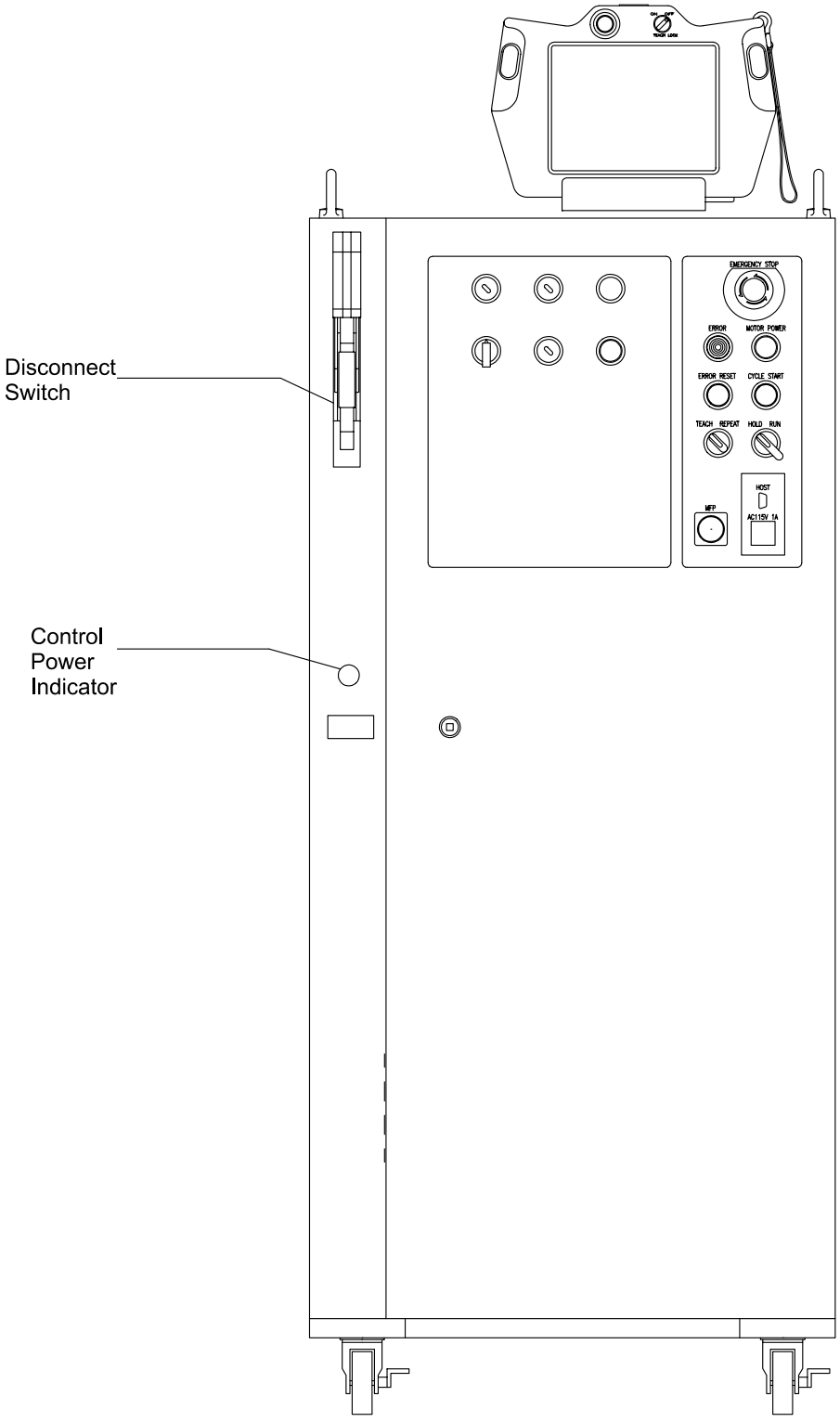


Figure 3-2 North American C Controller

POWER ON/OFF PROCEDURES

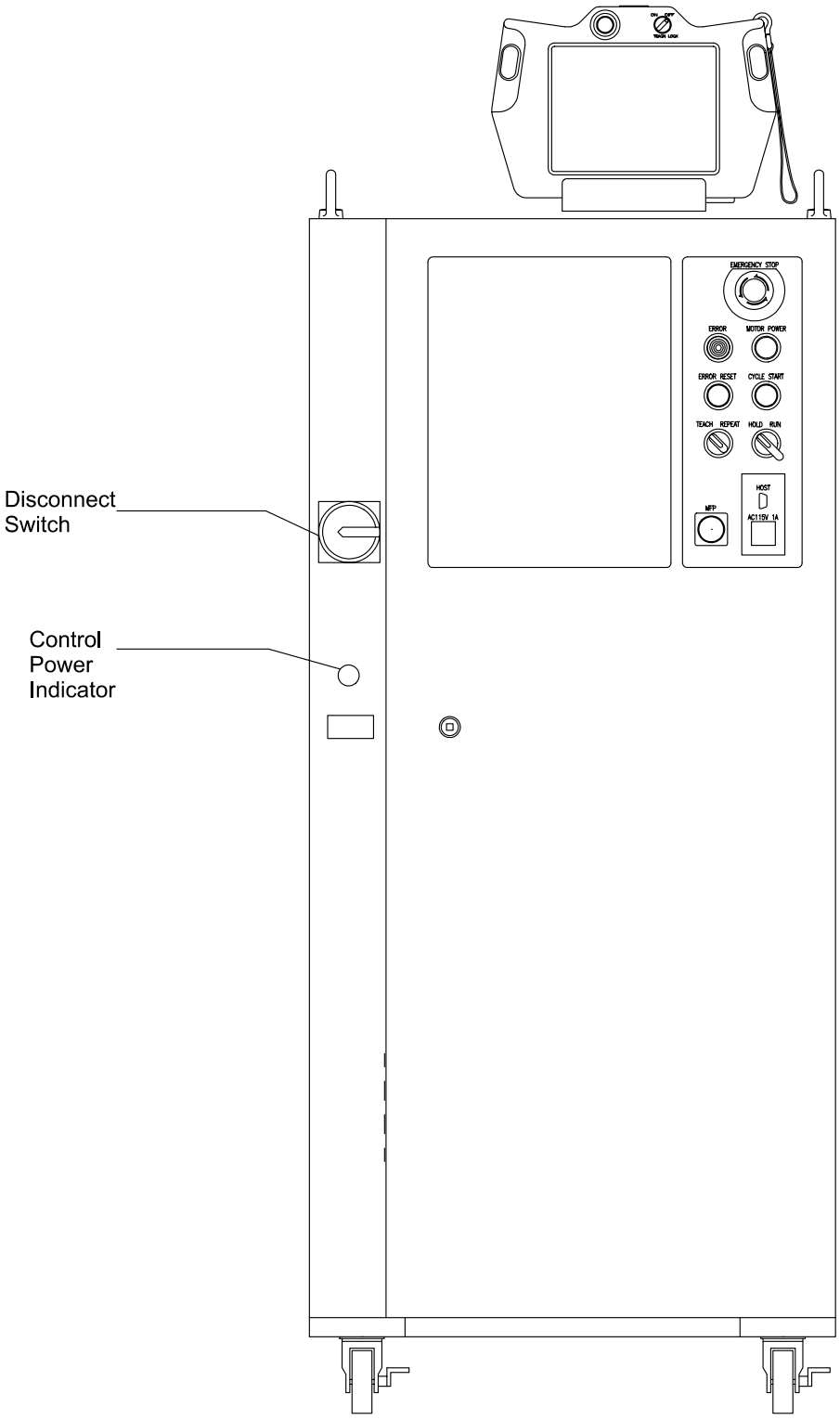


Figure 3-3 European C Controller

POWER ON/OFF PROCEDURES

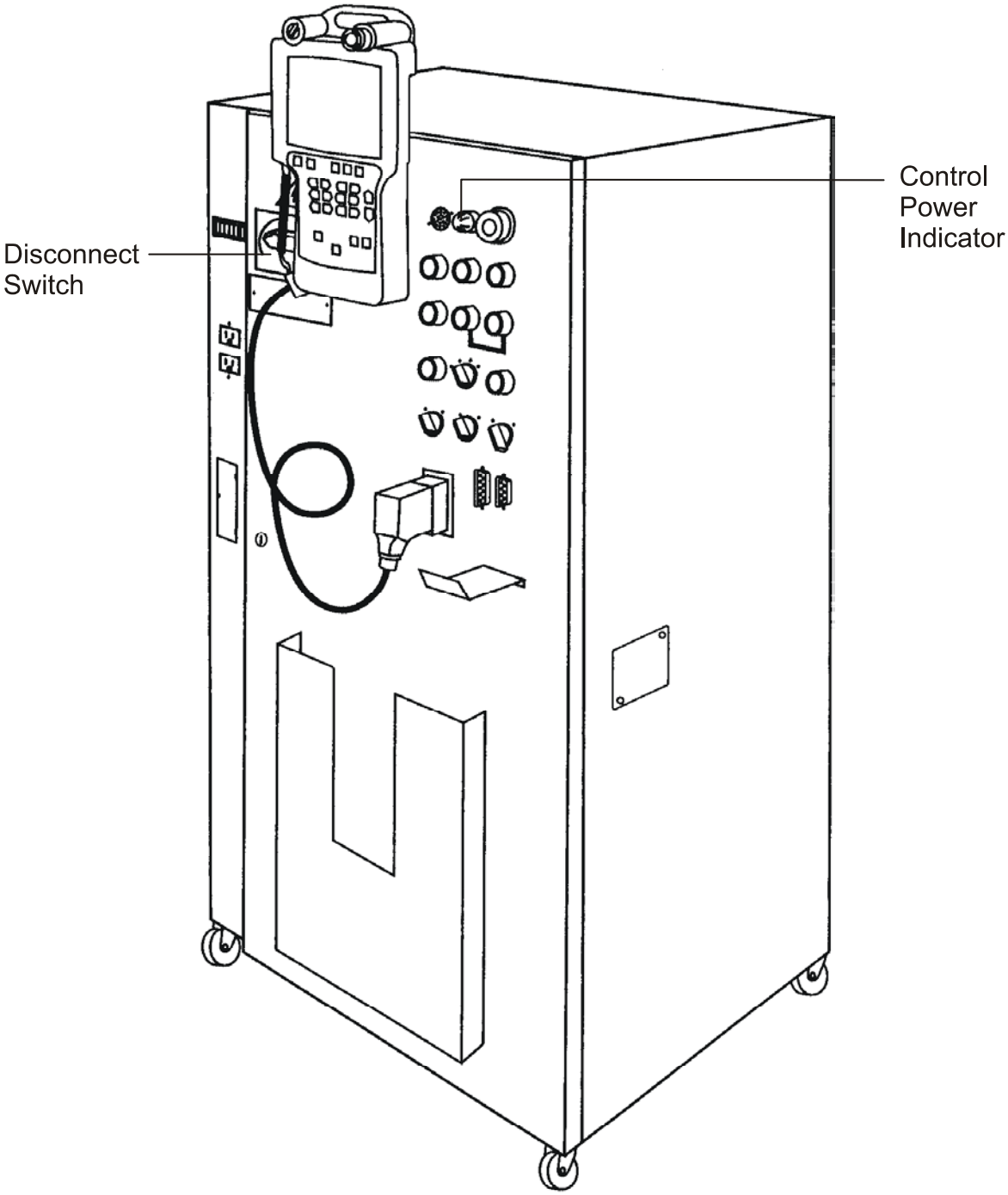


Figure 3-4 C5X Controller

POWER ON/OFF PROCEDURES

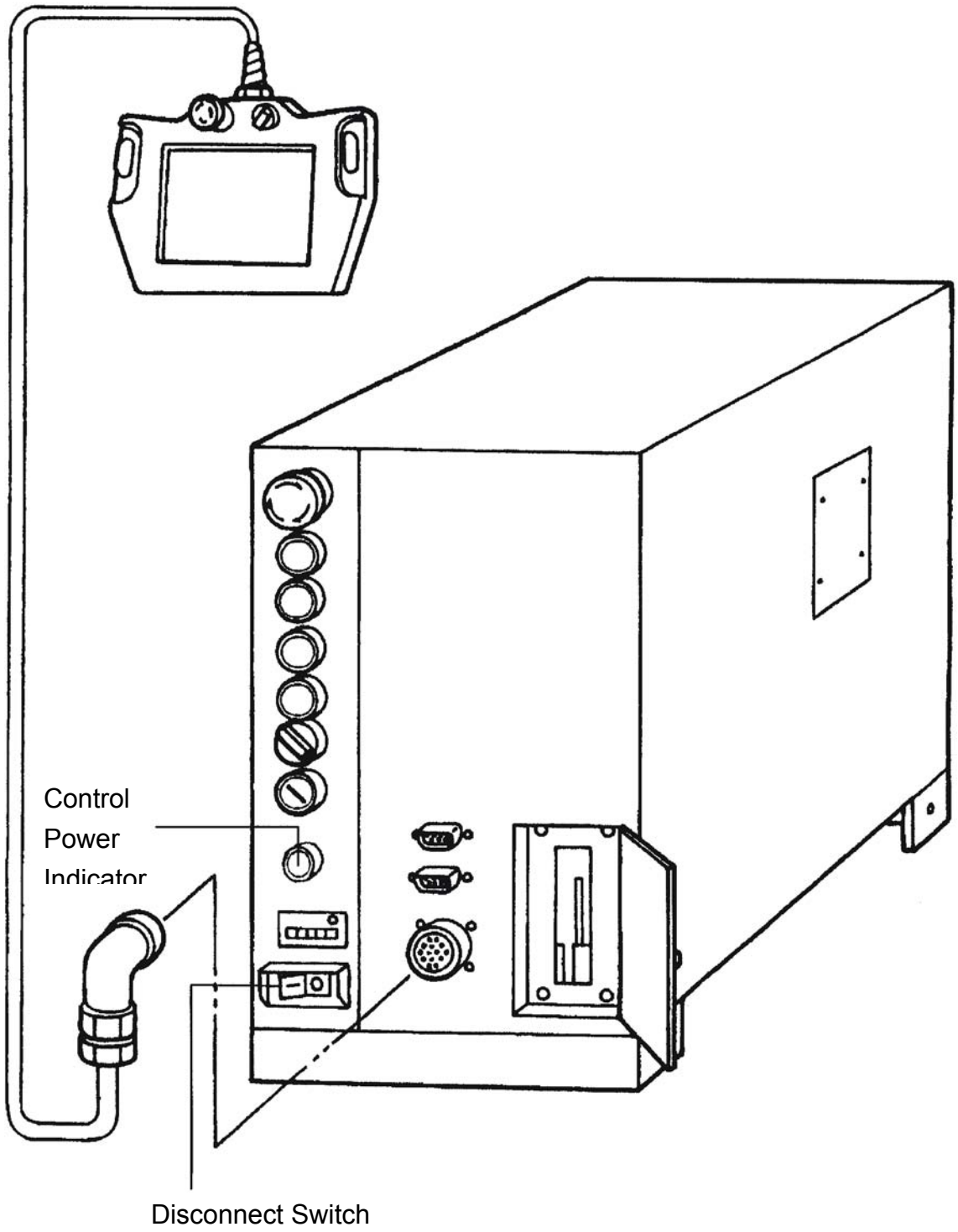


Figure 3-5 C70 Controller

POWER ON/OFF PROCEDURES

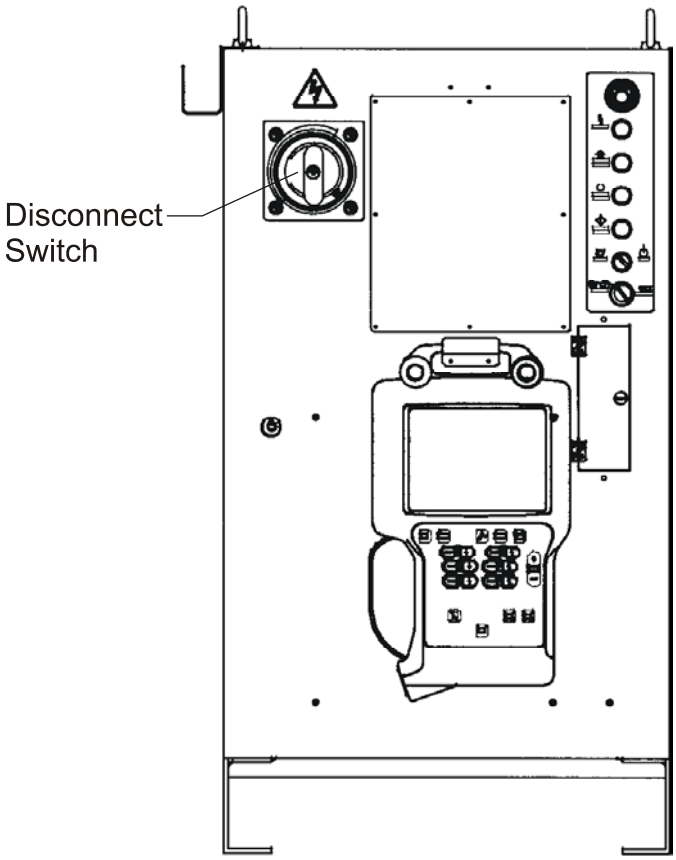


Figure 3-6 C80 Controller

POWER ON/OFF PROCEDURES

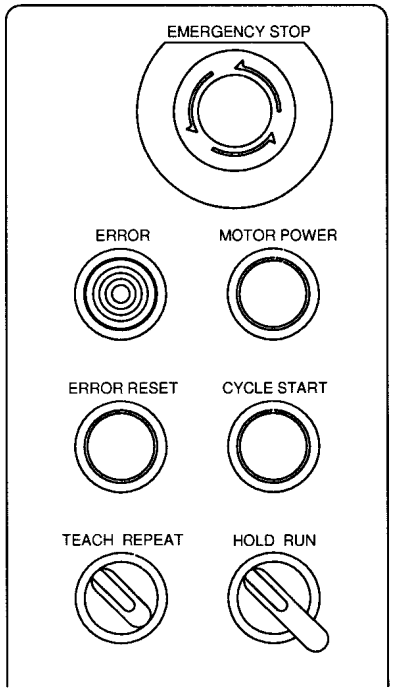


Figure 3-7 Switch Panel for North American, European C Controller

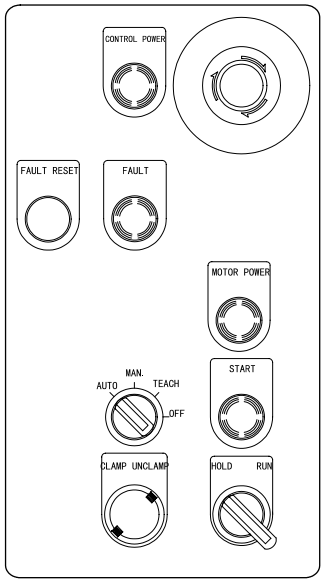


Figure 3-8 Switch Panel for C5X Controller

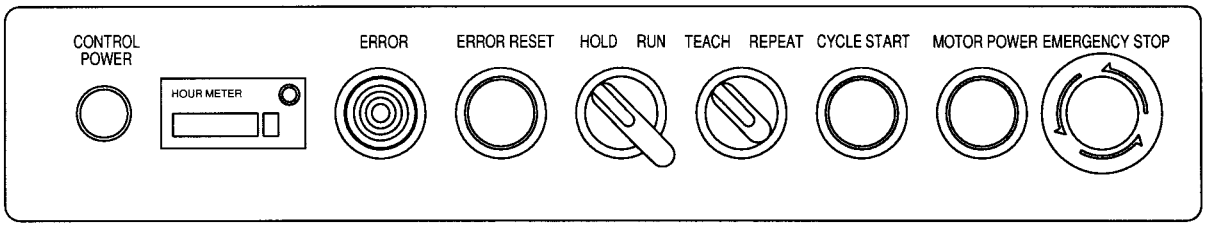


Figure 3-9 Switch Panel for standard C Controller

POWER ON/OFF PROCEDURES

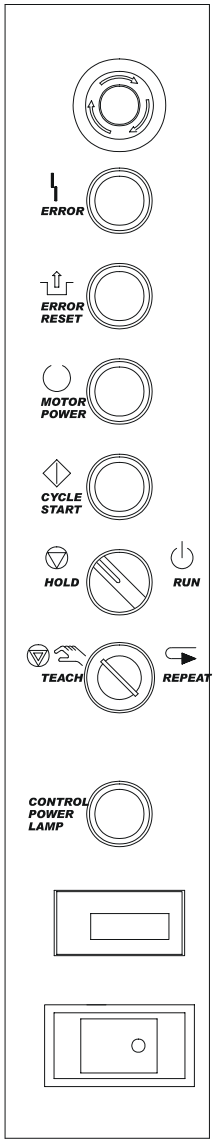


Figure 3-10 Switch Panel for C70 Controller

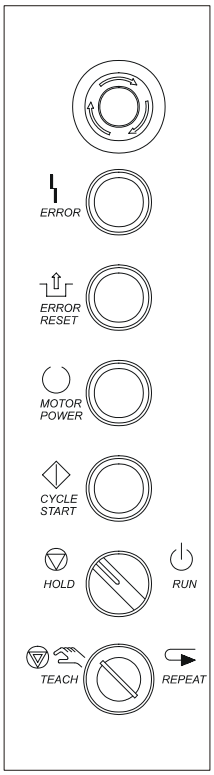


Figure 3-11 Switch Panel for C80 Controller

3.2. SERVO MOTOR POWER ON

3.2.1. SERVO MOTOR POWER ON IN REPEAT MODE

1. Turn the TEACH/REPEAT switch to the REPEAT position.

2. Turn the HOLD/RUN switch to HOLD.

3. Push the MOTOR POWER button. The MOTOR POWER lamp should illuminate.

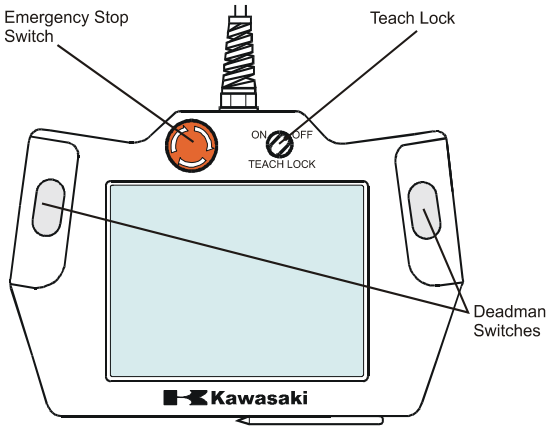
4. Turn the HOLD/RUN switch to RUN. The robot is now ready to execute a program.

3.2.2. SERVO MOTOR POWER-ON IN THE TEACH MODE

1. Turn the TEACH/REPEAT switch to the TEACH position.

2. Turn the TEACH LOCK to ON.

Press either deadman switch and push the MOTOR POWER button. The MOTOR POWER lamp should illuminate.



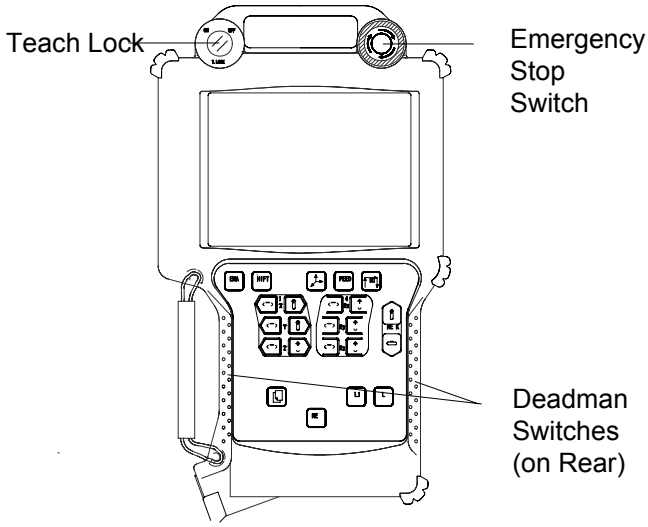


Figure 3-12 Multi Function Panel (Type 1) Figure 3-13 Multi Function Panel (Type 2)

3.3. METHODS FOR STOPPING THE ROBOT

One of three methods can be used to stop robot motion. Each of these methods is described below, and illustrated in figure 3-14.

3.3.1. EMERGENCY STOP SWITCH

Pressing the EMERGENCY STOP switch will turn motor power off and immediately apply the brakes. Note that this method places considerable strain on the robot and is only recommended for emergency situations. Refer to section 3.3.2 for method of stopping the robot in non-emergency situations.

3.3.2. HOLD/RUN SWITCH

Turn the HOLD/RUN switch to the HOLD position to apply the brakes and have the robot decelerate smoothly to a stop. This will place the robot in a state of temporary stop (CYCLE START and MOTOR POWER indicator lamps remain on); Turn the HOLD/RUN switch back to RUN to continue robot operation. To create a state of permanent stop, press the EMERGENCY STOP button, or turn the TEACH/REPEAT switch to TEACH in either case, CYCLE START and MOTOR POWER indicator lamps should turn off.

3.3.3. TEACH/REPEAT SWITCH

Turning the TEACH/REPEAT switch to the TEACH position will turn motor power off and immediately apply the brakes. As this places considerable strain upon the robot, it is only recommended for emergency situations. Refer to section 3.3.2 for method of stopping the robot in non-emergency situations.

POWER ON/OFF PROCEDURES

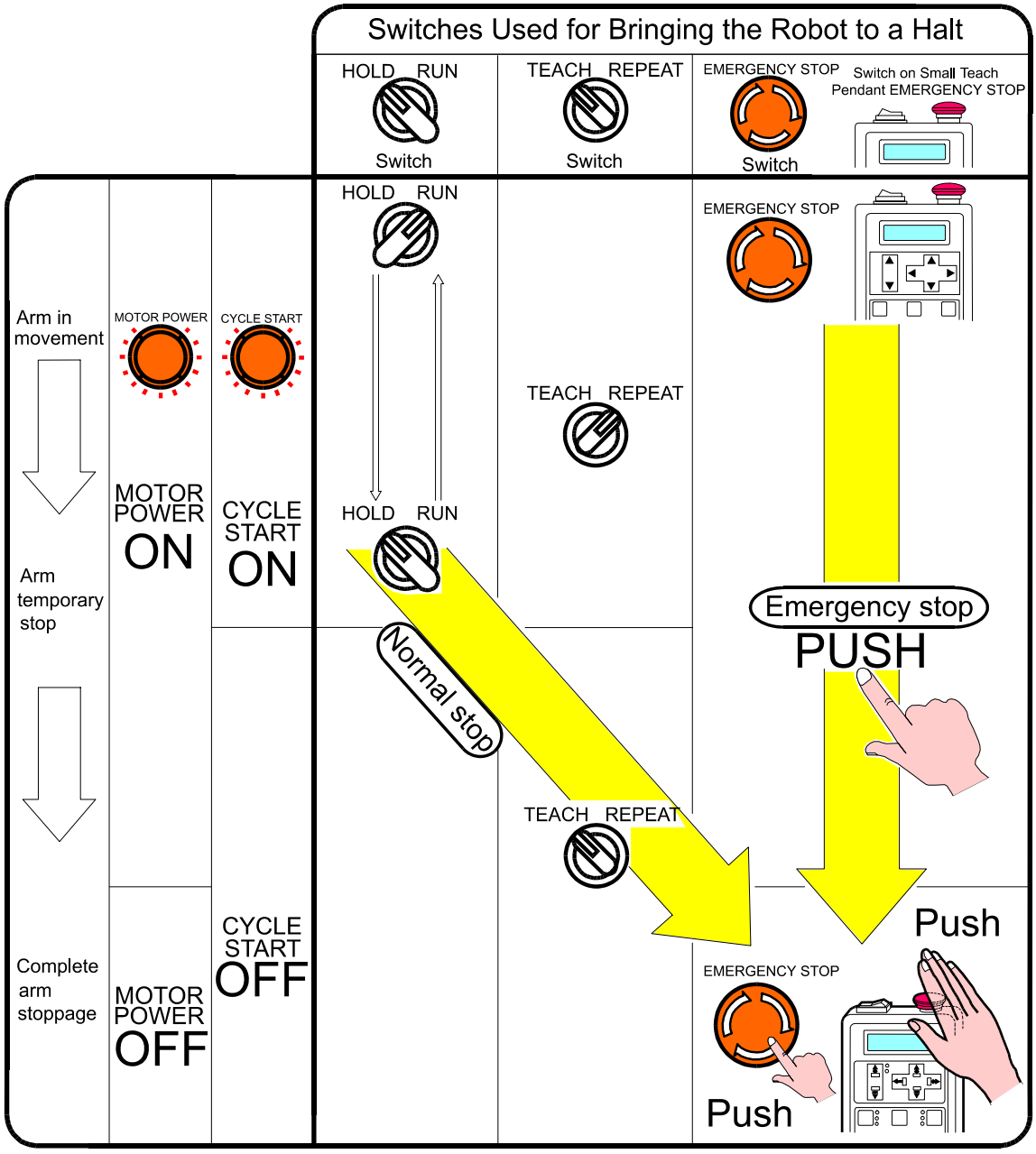


Figure 3-14 Methods of Bringing the Robot to a Halt

POWER ON/OFF PROCEDURES


MEMO

AS LANGUAGE COMMANDS

4.0. AS LANGUAGE COMMANDS

4.1. TERMINAL CONTROL

The following terminal control commands, also known as *control characters*, enable the programmer to control the display of output information. Control characters are used by entering control *character key* (note that character keys can be ether upper or lower case). Unlike other AS language commands, the ENTER key does not need to be pressed to process the command.

The following terminal control commands are available for AS language programming:

CTRL C Similar to pressing the EXIT key on the Multi-Function Panel, this command is used to cancel the current input line; however, it cannot terminate a program currently being executed.

CTRL L Displays a previously-entered line of code on the current input line. This operation can be used up to four times to recover previously entered data.

CTRL N Used in conjunction with the CTRL L command, CTRL N displays the input line directly following a previously-entered line displayed via CTRL L. This operation is only effective after CTRL L is pressed more than once.

CTRL Q Resumes the scrolling of terminal output paused via CTRL S command.

CTRL S Pauses the scrolling of terminal read out, allowing for content confirmation. Output is resumed by pushing CTRL Q.

4.2. EDITING COMMANDS

Editing commands are used to both modify existing programs and create new ones. These commands are exited from the program editor, entered, not surprisingly, via the command `edit`.

[NOTE] Throughout the manual, the following command notation will be used:
All caps: The required command keyword. Enter is as shown.
Italic: Required values to be entered by the user.
Italic + Bold: Optional values that may be entered or omitted by the user.
For example, the command ***EDIT program name, step*** shown below will be entered as `EDIT spray, 10` `EDIT spray` or simply `EDIT`.

4.2.1. EDIT COMMAND

The format for the EDIT command is `EDIT` *`program name, step`*. EDIT is the processor-recognized keyword; *program name and step* are the optional arguments.

Program name is the name of the program to be edited. If that program does not exist, a new program with that name will be created. If a program name is not specified, the last program edited will be displayed.

The optional *step* number designates the point in the program from which editing is to begin. If the program being edited was not the last program edited and step is not specified, editing will begin at the first step of the program. If the program was the last edited and step is not specified, editing will begin at the last step edited. Should the last program run have incurred an error message, editing will automatically begin at the step where the error occurred.

[NOTE] A program in progress can be neither edited nor deleted.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
```

```
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? ↵
```

Figure 4-1 EDIT Command

The above example shows the `EDIT` command used at the `$` monitor prompt. The program to be edited is named “test,” and editing is to begin at step 5.

4.2.2. S (STEP)

The format for the S command is `S` *`step number`*. S is the processor-recognized keyword, and *step number* the optional argument.

Step number designates number of the step to be edited. Editing will begin at the first program step if *step number* is not specified. Should the step number entered be greater than the number of steps in the program. *Step number* will become the value of the last program step + 1.

```
$EDIT test, 5 ↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? S 123↵
123  JMOVE weld3
123$ ↵
124  JMOVE weld4
124?s ↵
1  HOME
1? ↵
```

Figure 4-2 S Command

The above example shows the S command used to advance the editing display to step 123. The S command is also used at step 124, this time without designated a step

number as a result, the display returns to the first step of the program.

4.2.3. P (PRINT)

The format for the P command is `P step count`. P is the processor-recognized keyword, and *step count* the optional argument.

The P command does not output information to a printer; rather it is used to display command steps. *The Step count* value designates the number of consecutive steps to be output. Omitting step count will cause only a single command line to be displayed. Regardless of whether step count is used or not, display allays begins from the current step.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? P 4↵
7  SPEED 50 always
8  JMOVE pounce
9  LMOVE begin
10 LAPPRO pickup
11 LMOVE pickup
11? ↵
12 LDEPART
12? ↵
```

Figure 4-3 P Command

The example above shows the `P` command, with a *step count* of `4` used at step 7 to display steps 7, 8, 9, and 10.

4.2.4. L (LAST)

The **L** command is used to display and edit the previous program step. Such editing will not affect, the current program step.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? L↵
6  JMOVE CC
6?
```

Figure 4-4 L Command

The example above shows the `L` command entered at step 7. Resulting in the previous line (6) being displayed.

[NOTE] The S command could also be used to perform the same operation.

4.2.5. I (INSERT)

The I command is used to insert lines of code between existing steps. Use this command at the step number just below where you wish to insert the new lines. All steps following the newly-inserted line will be renumbered to reflect the new step order.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? I
7I JMOVE weld1↵
8I JMOVE weld2↵
9I JMOVE weld3↵
10I ↵
10  SPEED 50always
10? ↵
```

Figure 4-5 I Command

In the example above, the `I` command is used at step 7 to insert three new steps. Step 7 is renumbered as step 10, and all following step will be renumbered accordingly. Press the ENTER key at the `I` prompt to discontinue step insertion.

4.2.6. D (DELETE) COMMAND

The format of the D command is `D` *`step count`*. `D` is the processor-recognized keyword, and the optional argument step count is the number of steps to be deleted, (beginning with the current step). If the step count is not specified, only the current step will be deleted. Figure 4-6 shows an example of the D command.

[NOTE] Deleting a step will cause all following program steps to be re-numbered accordingly.

Original Program

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? ↵
8  JMOVE weld1
8? ↵
9  JMOVE weld2
9? ↵
10 JMOVE weld3
10? ↵
11  HOME
11?
```

Deletion Process

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? D3↵
7  JMOVE weld3
7? ↵
8  HOME
8? ↵
9  JMOVE pounce
9?
```

Program After Editing

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7 JMOVE weld3
7? ↵
8  HOME
8? ↵
9  JMOVE pounce
9?
```

Figure 4-6 D Command

The example above shows the command `D 3` entered at step 7 as a result, steps 7, 8, and 9 are deleted, and the step previously numbered `10` is renumbered `7`.

4.2.7. F (FIND)

The format for the F command is `F` *`character string`*. F is the processor-recognized keyword, and *character string* is the group of characters to be located. The F command searches the current program from beginning to end for the specified *character string*, and displays the first step that includes that string.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? ↵
8  JMOVE weld1
8? ↵
9  JMOVE weld2
9? F xyz ↵
163  JMOVE xyz
163 ↵
```

Figure 4-7 F Command

The example above shows the command `F xyz` entered at step 9. The processor searches the program “test” for a character string that includes “xyz,” and displays its first use (step 163).

4.2.8. M (MODIFY)

The M command is used to substitute the character string from a selected command line with a new staring. The format for the M command is (`M/existing characters/new characters`.)

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  POINT pounce=SHIFT(s_cartframe+star BY 0,0,,clearance)
6? M/star/start ↵
6  POINt pounce=SHIFT(s_cartframe+start BY 0,0,clearance)
6? ↵
```

Figure 4-8 M Command

The example above shows the command, `M/star/start` entered at step 6. The *existing character* `star` in step 6 is replaced with the *new character* `start`.

4.2.9. O (OVERWRITE)

The **O** command utilizes the arrow and backspace keys to edit an existing program line. To replace an entire program line, enter the new line at the `?` prompt and press ENTER. This procedure eliminates the need to DELETE and INSERT when a step is replaced.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  POINT pounce=SHIFT(s_cartframe+star BY 0,0,,clearance)
6? 0 ↵
6  POINt pounce=SHIFT(s_cartframe+start BY 0,0,clearance)
6  POINT pounce=SHIFT(s_cartframe+star BY 0,0,,clearance)
6? ↵
```

Figure 4-9 O Command

The example above shows the O command used at step 6 to re-display the program line. The arrow keys are used to navigate the cursor to the point in the line that requires editing. Backspace is then used to delete line information (in this case, change `star` to `start`).

4.2.10. R (REPLACE)

This command is used to replace part of an existing program line. The format for the R command is `R` *`new`*, figure 4-10 shows an example of the R command in use.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? ↵
7  SPEED 50 always
7? R75 ↵
7  SPEED 75 always
7? ↵
8  JMOVE weld1
8?
```

Figure 4-10 R Command

In the example above, the space bar is used at line 7 to place the cursor under the characters to be replaced. `R` command is entered, followed by new information to replace the old. The revised line is then shown, followed by the `?` prompt.

4.2.11. C (CHANGE)

This command is used to cease editing of the current program, and begin editing a second program.

The format for the c command is `C` *`program name, step number`*. If no step number is specified, editing of the second program will begin from the first command step. This command is useful when programs call subroutine that also require editing.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? C tested,37 ↵
.PROGRAM tested()
37  JMOVE weld1
37? ↵
38  JMOVE weld2
38?
```

Figure 4-11 C Command

In the example above, the command `C tested,37` is entered at step 6 of the program "test." The display then changes to show step 37 of the program "tested."

4.2.12. E (EXIT)

The **E** command is used to exit Editor mode and return to Monitor mode.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? E ↵
$
```

Figure 4-12 E Command

4.2.13. XD

The XD command is used to remove (cut) a specified number of lines from the program being edited and store them in a paste buffer. The format for the XD command is `XD` *`number of lines`*. To use the XD command, move the cursor to the first program line to be removed to the paste buffer, enter `XD` plus the number of lines to be cut, and press ENTER. The current line and all subsequent specified lines will be placed in the paste buffer and the program steps will be renumbered accordingly. The contents of the paste buffer are overwritten whenever a new item is placed in the paste buffer.

4.2.14. XY

The XY command is used to copy a specified number of lines from the program being edited and store them in a paste buffer. The format for the XY command is `XY` *`number of lines`*. To use the XY command, move the cursor to the first, program line to be copied, enter `XY` plus the number of lines to be copied, and press ENTER. The current line and all subsequent lines specified will be copied to the paste buffer. XY differs from the XD Command in that XD removes (cuts) the selected line, while XY does not.

4.2.15. XP

The **XP** command is used to place the contents of the paste buffer into the program being edited. Step copied from the paste buffer are placed into the program before the line where the XP command is executed remaining program steps are renumbered accordingly.

4.2.16. XQ

The **XQ** command takes the contents of the paste buffer and places them *in reverse order* into the program being edited. Steps copied from the paste buffer are placed into the program before the line where the XQ command is executed. Remaining program steps are re-numbered accordingly.

4.2.17. XS

The **XS** command is used to display the contents of the paste buffer. Figure 4-13 shows an example of the XS command in use. The `XS` command at step 6 shows the three steps currently in the paste buffer.

```
$EDIT test, 5↵
.PROGRAM test()
5  LMOVE bb
5? ↵
6  JMOVE cc
6? XS ↵
---Paste Buffer---
[1]>   JMOVE #a
[2]>   JMOVE #b
[3]>   JMOVE #c
6  JMOVE cc
6?
```

Figure 4-13 XS Command

4.2.18. T COMMAND

The **T *command with variable name*** utilizes the Teach Pendant to program movement commands such as `JMOVE` and `LMOVE` in the editor mode, beginning with the step at which the `T` command is executed.
Variable name designates the positional variable to be programed.
This command is executed by entering the following (when edit is in run mode):
`XX ? T variable name` followed by RETURN.
Entering the above will being up the T command screen, where motion instructions can be input into the program.
Attempting to execute this command at the location of a previously-entered step will cause a new step to be created.
Also, designating a variable name such as A[ ] will enable it to be used as a positional array variable.
The constant for each axis can be set if no variable name is assigned, you can insert the step by using together with I command.

NOTE
This function cannot be executed using the small model Teach Pendant; it will be Small Teach Pendant. Necessary to connect a Multi Function Panel.

4.3. DATA CONTROL COMMANDS

This set of commands allows the user to access program and variable data in system memory, as well as delete, transfer, copy, and view save data.

4.3.1. DIRECTORY COMMAND

The DIRECTORY group of commands is used to view programming data in the system memory. They can be utilized a number of different ways to access desired information in its most usable form.

4.3.1.1. **DIRECTORY** COMMAND

The `DIRECTORY` command, commonly shortened to `DIR` lists all programs and variables in system memory. Entering a program name particular after `DIR` will list only the location, real, and string variables used in that program. Should the program specified contain subroutines, the names of the subroutine programs and their location, real, and string variable names will also be displayed. `DIR` used without a program name will list all programs locations, real variable, and string variables in system memory. Should this information span more than a single page, readout will pause once the screen has become filled. Press the spacebar to resume the listing, or press enter to discontinue.

The asterisk (*) is a wildcard character compatible with all program and data control commands with the exception of `RENAME`. It can be used in place of any other character.

Figure 4-14 shows the asterisk being used to get specific information. In that example, the command `DIR s*` is entered to retrieve all programs beginning with “s.” This command with also list all subroutines, as well as location, rear and string variables associated with the specified program(s).

```
$DIRECTORYs* ↵
Program
alternate body_side  pg01    sample01  sample02   sealing01 sealing02 sealing03
side1     side2      side_welds
Location
a_pillar  b_piller   c_piller          spot1      spot2     spot3     sopt4 spot5
sopt6     spot7      sopt8
#a1       #a2        #b1     #b2       #c1        #c2       #weld1
Real
bundel    consign    d_countfixt       frmoffset  hoppercnt st1       vr4
vr5       vr7        wwf
Sting
#count    $oh1       $pwr
```

Figure 4-14 DIRECTORY Command

4.3.1.2. DIRECTORY/P COMMAND

The `DIRECTORY/P` command is used to display the names of all programs in system memory. Only the names of the programs will be displayed with this command. In the example shown in figure 4-15, the names of eight programs are displayed.

```
$DIRECTORY/P ↵
Program
alternate body_1    body_r    body_side  get_pt    pg01    pg02    pg03
sample01 sample02  sealing01 sealing02  sealing03 side1   side2   side_welds
zsihft
```

Figure 4-15 DIRECTORY/P Command

4.3.1.3. DIRECTORY/L COMMAND

The DIRECTORY/L command displays the names of all location variables in system memory. Only the names of the location variables are displayed with this command. The example in Figure 4-16, shows the names of eight location variables.

```
$DIRECTORY/L ↵
Location
a_pillar b_pillar c_pillar  door1    door2    door3    hrdtop1   spot1    spot2
spot3    spot4    spot5     spot6    spot7    spot8
#a1      #a2      #b1       #b2      #c1      #c2       #weld1
```

Figure 4-16 DIRECTORY/L Command

4.3.1.4. DIRECTORY/R COMMAND

The DIRECTORY/R command displays the names of all real variables in system memory. Only the names of these variables will be displayed. The example in Figure 4-17, shows the names of nine real variables.

```
$DIRECTORY/R ↵
Real
bundel       consign  d_count    d_count   fixt    fixt1   fixt2     frmoffset
hopper_cont  st1      vr1        vr2       vr4     vr5     vr7       wwf
```

Figure 4-17 DIRECTORY/R Command

4.3.1.5. DIRECTORY /S COMMAND

The DIRECTORY/S command display the names of all string variables in system memory. Only the names of the string variables will be displayed. The example in Figure 4-18, shows the names of three string variables.

```
$DIRECTORY/S ↵
Sting
$count   $drops    #field   $goals    $ohl     $pwr     $qulity   $stamp
$trips
```

Figure 4-18 DIRECTORY/S Command

4.3.2. LIST COMMANDS

The LIST group of commands is used to view programming data and values in system memory. They can be utilized a number of different ways to access desired information in its most usable form. Information displayed via LIST commands is more detailed than that available with DIRECTORY commands.

4.3.2.1. LIST COMMAND

The command **`LIST`** displays program steps and variable values residing in system memory. Entering `LIST` *`program name`* will display real variables and their values, location variables and values, string variables, and program steps for the designated program. If no program name is entered, the above information will be displayed for *all* programs in memory. Should this information span more than a single page, readout will pause once the screen has been filled. Press the spacebar to resume the listing, or press enter to discontinue.

4.3.2.2. LIST/P COMMAND

The `LIST/P` command is used to display the names and steps of all programs in system memory. Entering `LIST` *`program name`* will display steps for the designated program only. In the example below, the command `LIST/P` ***test*** is used to display all steps for the program "test."

```
$LIST/P test ↵
.PROGRAM TEST()
1  count=0
2  parts_max=75
3  SPEED 85 ALWAYS
4  ACCURACY 10 ALWAYS
5  JAPPRO pick
6  LMOVE pick
7  CLOSE
8  LDEPART
9  JMOVE place
10 OPEN
.END
```

Figure 4-19 LIST/P Command

4.3.2.3. LIST/L COMMAND

The `LIST/L` command is used to display information on all location variables in system memory this information includes.

- Variables names and values
- Location names
- XYZ dimensions
- OAT angles

In the example below, `LIST/L` *`part*`* is entered to display all location names and values containing the character string "***part***."

```
$LIST/L part* ↵
Location
part1 -130.35    390.49    378.91    155.67     93.11    -28.94
```

```
part2  137.82    582.62    981.07    -33.21    187.51    261.14
part3  492.44   -390.12    236.72     91.29    101.28     47.89
```

Figure 4-20 LIST/L Command

4.3.2.4. LIST/R COMMAND

The `LIST/R` command is used to display the names and values of all real variables in system memory. In the example below, the command `LIST/R r*` is used to display the value of all real variables beginning with *r*.

```
$LIST/R r*↵
Real
r1=      100, r2      =      347, r3      =     582.1, r4      =        0.0
```

Figure 4-21 LIST/R Command

4.3.2.5. LIST/S COMMAND

The **`LIST/S`** is used to display all string variables and the character strings assigned to them. See the example below for the `LIST/S` command in use.

```
$LIST/S ↵
String
$count="Number of parts processed"
$name="Kawasaki Robotics Inc."
```

Figure 4-22 LIST/S Command

4.3.3. DELETE COMMANDS

The `DELETE` group commands is used to delete programming data and values in system memory. These commands can be used a number of different ways to delete the desired information in the most efficient manner.

4.3.3.1. DELETE COMMAND

The command for deleting specific programs `DELETE` *`program name`*. This procure also erases all location, real, and string variables related to the specified program. All subroutine programs will similarly deleted, unless they are in use by another program.

The DELETE command is commonly shortened to DEL.

The asterisk command may also be used with DELETE.

For example, typing DEL S* will delete all programs beginning with the letter “S”, as well as all related variables also subroutine.

4.3.3.2. DELETE/P COMMAND

DELETE/P *program name* is used to remove the specified program and all steps from system memory. DELETE/P will *not* remove subroutines or location, real, or string, variables associated with the specified program.

4.3.3.3. DELETE/L COMMAND

The DELETE/L *location variable name* command is used to delete specific location variables from system memory.
For example, DELETE/L *weld1* will remove the location variable *weld1* and all components.

4.3.3.4. DELETE/R COMMAND

The DELETE/R *real variable name* command is used to delete specific real variables from system memory.
For example, DELETE/R row.max will delete the real variable row.max and its assigned value.

4.3.3.5. DELETE/S COMMAND

The DELETE/S *string variable name* command is used to delete specific string variables from system memory.
For example, DELETE/S *$title_words* will delete the string variable *$*title_words and its assigned characters.

4.3.4. RENAME COMMAND

This command is predictably enough, used to **RENAME** a program in system memory. The format is RENAME *new program name=old program name*. If the name entered for the new program name is already in system memory, the RENAME operation will be aborted and an error message displayed.

In the example below, the command RENAME *samp1=sample* is entered, followed by DIR/P to confirm that renaming took place. Note that program “sample” is new displays as samp1.

```
$DIRECTORY/P ↵
Program
alternate body_12    body_r    body_side  get_pt     pg01    pg02    pg03
sample    sample02   sealing01 sealing02  sealing03  side1   side2   side_welds
$RENAMEsamp1=sample ↵
$DIRECTORY/P ↵
Program
alternate body_1     body_r    body_side  get_pt     pg01    pg02    pg03
samp1     sample02   sealing01 sealing02  sealing03  side1   side2   side_welds
```

Figure 4-23 RENAME Command

4.3.5. XFER (TRANSFER) COMMAND

The XFER command copies steps from one program to another, or from one place to another within the same program. The format for this command is XFER *new program, start step=old program*. The old program steps will be neither replaced nor deleted by this command.

The example in Figure 4-24 shows the command XFER *pg01,2=pg02,5,4*. This tells the computer to go to program pg02 and copy four lines of code beginning at steps. The computer then takes that information and inserts it into program pg01 beginning at step 2. Program *pg02* is in no way changed by this operation.

AS LANGUAGE COMMANDS

$XFER *pg01,2 = pg02,5,4*

BEFORE		AFTER
pg01	pg01	pg01
1 JMOVEa	1 JMOVEa	1 JMOVEa
2 JMOVEb	2 JMOVEb	2 JMOVEaaa
3 JMOVEc	3 JMOVEc	3 JMOVEbbb
4 JMOVEd	4 JMOVEd	4 JMOVEccc
	5 JMOVEaaa	5 JMOVEddd
	6 JMOVEbbb	6 JMOVEb
	7 JMOVEccc	7 JMOVEc
	8 JMOVEddd	8 JMOVEd

Figure 4-24 XFER Command

4.3.6. COPY COMMAND

The **COPY** command is used to transfer the entire contents of a program (or programs) to a second program. The format for this command is `COPY destination program name = source program name + source program name`. Only one source program needs to be identified, though additional programs can be named and copied to the destination program. The name entered for the destination program cannot currently be in use.

4.3.7. TRACE COMMAND

The **TRACE** command enables or disables trace on robot or PC programs for witch logging is desired. The format for this command is `TRACE stepper_number : ON/OFF`.

The stepper number designates program numbers of either the robot or PC. Using the values of real numbers. For example:

- 1 : Robot program
- 1001:PC program 1
- 1002:PC program 2

- 1003:PC program 3

When stepper number is omitted, the TRACE command will affect *all* program in system memory.
Entering `TRACE ON` without first securing user memory via the SETTRACE command will result in the following error message: error (-391) Undefined memory domain.

[NOTE] Be sure to enter the command to set trace as one word, `SETTRACE`.

4.3.8. SETTRACE COMMAND

The SETTRACE command secures, within user memory, all memory reserved for logging. This information includes the total number of logging steps. As well as the number of steps in all robot and PC programs.
The format for the SETTRACE command is `SETTRACE` *`step number`*. Step number is designated as a real number value of the number of logging steps. The number range is 1-9999, though this value will automatically be set to 100 steps should no value be entered.

[NOTE] Entering `TRACE ON` without first securing user memory via the SETTRACE command will result in the following error message:

```
ERROR (-391) UNDEFINED MEMORY DOMAIN
```

Similarly, entering the SETTRACE command while logging is underway will result in the following error message:

```
ERROR (-390) LOGGING IN PROGRESS
```

Following this error message, all traces on robot and PC programs will automatically shut off.

4.3.9. RESTRACE COMMAND

Entering `RESTRACE` frees the area of user memory reserved by the SETTRACE command.

[NOTE] Executing the RESTRACE command while logging is underway will result in the following error message:

```
ERROR (-390) LOGGING IN PROGRESS
```

4.3.10. LSTRACE COMMAND

The LSTRACE command displays logging data for all steps of the designated robot or PC program. The format for this command is `TRACE` *`stepper number: logging number`*.
The stepper number designates program numbers of either the robot or PC using the values of real numbers. For Example:

- 1 : Robot program
- 1001:PC program 1
- 1002:PC program 2
- 1003:PC program 3

When stepper number is omitted, LSTRACE will affect robot program 1.
Logging number designated by the real number value of the number first used for logging.
If no logging number is entered, the value automatically becomes logging number 1.

Executing the LSTRACE command will display step logging data and put the computer into temporary “stand by” more until a button is pressed. At this time, the following commands may also be entered (all commands are followed by the RETURN key):

N: Display jumps down 9 lines

L: Display jumps back 9 lines

S *no*: Shows 7 lines of information for the specified logging number. If no number is entered, information is displayed for logging number 1.

F *word*: Display jumps to each occurrence of the specified word, showing the line in which the word appears. As well as the four preceding and following lines (9 lines total). If no word is entered search is executed with the last-used entry.

E: Closes the screen and returns to the AS software.

Pressing the return key with no command will display the next 9 lines of program information.

[NOTE] Attempting to run LSTRACE while logging is underway will result in the following error message:

```
ERROR (-390) LOGGING IN PROGRESS
```

4.4. COMMANDS FOR PROGRAM AND DATA STORAGE

4.4.1. FORMAT COMMAND

The FORMAT command is used to ready floppy disks or PC SRAM memory cards for storing data. This process is essential, as information ***cannot*** be saved to unformatted disks or PC cards. To use the FORMAT command, insert a PC card into the Multi-Function Panel (or floppy into the computers disk drive) and type FORMAT at the > prompt, followed by the RETURN key. The prompt Are you sure (Yes:1,No:0): will appear at which time the appropriate response should be entered.

[NOTE] Formatting will permanently remove all information currently saved on the disk or PC card .

4.4.2. FDIRECTORY COMMAND

The `FDIRECTORY` command displays information about files stored on a diskette or PC card. The name of the file is listed first, followed by the file extension. The DIRECTORY command also displays the file size (in bytes) and the date and time of file creation. The example below shows a typical FDIRECTORY command display.

```
$FDIRECTORY ↵
File01.AS220108        98-01-27 09:51
File02.AS 13678        98-01-25 09:59
Ldoor.PG  16104        98-01-18 11:53
Hood.PG   14352        98-01-18 11:16
Spots.LC   5897        97-10-25 14:21

869453 bytes free
$
```

Figure 4-25 FDIRECTORY Command

The extensions and their meanings are:

.AS system
.PG program

AS LANGUAGE COMMANDS

.LC location variables
.RV real variables
.ST string variables
.SY system data
.EL error log

4.4.3. SAVE COMMAND

The SAVE group commands is used to preserve controller system data on a floppy disk or PC card. There are a number of ways to use these commands in order to store the desired information in its most usable form.

4.4.3.1. SAVE COMMAND

The formats for the SAVE command are `SAVE` *`file name = program name`* and **`SAVE/SEL`** ***`file name = program name`***. The two differ in what they SAVE to the designated file as follows:

SAVE Records (1) robot and system data, including transformational values for base and tool, (2) all subroutines contained within and called by the designated program, and (3) all locational, real and character string variables reference by the program and its subroutines.

SAVE/SEL Records (1) robot and system data, (2) the designated program, *excluding* subroutines, and (3) all locational, real, and character staring variables referenced by the program *only*.

SAVE will record the specified information onto either floppy disk or PC card. When *program name* is not specified, all programs, variables, and system information will be saved; Including *program name* will record the specified program(s) and any subroutines along with the system information. Only the variables used by the specified program(s) and subroutines will be saved.

Extensions are added to file_names automatically, based upon the information they contain. Files created with the SAVE command will have an extension of AS.

If the specified final name already exists on the destination diskette or PC card, the processor will modify the file name by adding. `B` (“backup”) to the extension of the existing file. For example, if a file named TEST.AS. exists on disk and *file_name* is specified using the same name, the existing file will be renamed TEST.BAS and a new file TEST.AS will be created.

[NOTE] Only one backup version can be kept in memory via this process attending to save, yet another file named TEST.AS will overwrite the earliest version of the file. This process applies to all file extensions.

Care should be taken when selecting file names and the type of information contained within those files proper management can be the key to recovering stored information when system trouble occurs. Also reminder that only file name(s) can be selected when transferring information from diskette or PC card to the processor.

4.4.3.2. SAVE/P COMMAND

The format for the SAVE/P command is `SAVE/P` *`file name = program name`*.
For example, the command `>SAVE/P Fender = pg01, weld, pg02` will save programs *pg01, weld, and pg02* to a file named Fender.PG. Location, real, and string variables used by these programs will not be saved. Nor will any subroutines called by these programs.

An extension of .PG will be added to files saved with this command.

[NOTE] If no program name is specified, information for *all* programs in memory will saved to the designated file.

4.4.3.3. SAVE/L COMMAND

The format for the SAVE/L command is `SAVE/L` *`file name = program name`*. This will record all location values used by the specified programs and its subroutine to a designated file.

An extension of .LC will be added to files saved with this command.
For example, the instruction `>SAVE/L Where = pg01, weld, pg02` will save the location information from programs pg01, weld, and pg02 to a file named Where.LC.

[NOTE] If no program name is specified, information for *all* programs in memory will saved to the designated file.

4.4.3.4. SAVE/R COMMAND

The format for the SAVE/R command is `SAVE/R` *`file name = program name`*. This will record all real variables used by the specified program(s) and its subroutine to designated file.
For example, the command `>SAVE/R Values = pg*` will save the real variable information from all programs whose filename being with start `pg` to a file named Values.RV.

An extension of .RV will be added to files saved with this command.

4.4.3.5. SAVE/S COMMAND

The format for the SAVE/S command is `SAVE/S` *`file name = program name.`* This will records all string variables used by the specified program(s) and its subroutines to a designated file.
For example, the instruction `$SAVE/S Messages = pg01, weld, pg02` will save the string variables from programs pg01, weld, and pg02 to a file named Messages.LC.

An extension of .ST will be added to files saved with this command.

4.4.3.6. SAVE/SYS COMMAND

The format for the SAVE/SYS command is `SAVE/SYS` *`file name`*. This command will save all system information to the specified file and add the .SY extension.

4.4.3.7. SAVE/ELOG COMMAND

The format for the SAVE/ELOG command is `SAVE/ELOG` *`file name`*. All error log information will be saved to the specified file, and an extension of .EL added.

4.4.3.8. SAVE/A COMMAND

The format for the SAVE/A command is `SAVE/A` *`file name`*. Auxiliary data will be saved to the specified file and an extension of .AU added.

4.4.3.9. SAVE/ROB COMMAND

The format for the SAVE/ROB command is `SAVE/ROB` *`file name`*. Data specific to the robot will be saved to the designated *file,* and an extension of .RB added.

4.4.3.10. SAVE/MWLOG COMMAND

The format for the SAVE/MWLOG command is `SAVE/MWLOG` *`file name`*. The mechanical warning log will be saved to the designated file in a CSV format. This command cannot be used with other modifiers (for example, **X** SAVE/MWLOG/P).

4.4.4. LOAD COMMAND

The formats for the LOAD command are `LOAD file name` and `LOAD/Q file name`. `LOAD` will transfer data from a file on a PC card or floppy disk to system memory, where as `LOAD/Q` prompt the user before loading each type of data. The load prompt will appear as follows: `LOAD this data? (1:Yes, 0:No, 2:load all, 3:exit).`

An error message will appear if a program within the file being loaded is already in memory, and the program in memory must be renamed deleted or before loading can command. Conversely the name of if a locational or real variable being loaded is already in memory, the existing variable will be replaced by the variable being loaded. For example, system memory contains a real variable named shift_distn with an assigned value of 317; the file being loaded also (in memory) has a real variable named shift_distn, this time with an assigned value of 2,842. In this scenario, the existing value because other information such as for the variable shift_dist will be replaced *without* a message being displayed. Care should be tatken whenever loading in outside data. Because other information such as location variables, system settings, and tool dimensions will similarly be this affected.

Attempting to LOAD a program containing syntax errors will generate a prompt asking the line containing the error to be corrected or loading to be aborted.

4.4.5. FDELETE COMMAND

The FDELETE command is used to *permanently* remove files from a PC card or floppy disk, its format being `FDELETE file name, extension.` Note that these is no recycle bin or undo function available to recover data once deleted the information is *gone*.

Before executing an FDELETE command. The processor will display the following prompt: `Are you sure ? (Yes:1, No:0)`

4.5. PROGRAM CONTROL COMMANDS

Program control commands are monitor commands used to manage program operation. Using these command, a programmer can start and stop programs, as well as control the speed at witch they operate.

4.5.1. SPEED COMMAND

The SPEED command is used to effect changes, in percentage to monitor speed. The format for command is `SPEED` *`amount`*. The acceptable range from *0.01*% to *100*%, and monitor speed can be set in increments of 0.01%. Changes to speed will not become effective until the robot completes its current step and begins to processing the next motion. Note that action robot speed is determined by the product of monitor and program speed instructions within the program.

Figure 4-26 shows the SPEED command in use, and how monitor and program speed instructions interact to determine robot speed. In program pg01, (below) program speed is 1000 mm/s and monitor speed is 50, therefore, the robot actual speed will be 500 mm/s (50% of 1000 mm/s). In program pg02, program speed for steps 1 and 3 is SP9 and steps 2 and 4 is SP6. Monitor speed is 60, so therefore, the robot is repeat speed for steps 1 and 3 will be 60% of SP9, while repeat speed for steps 2 and 4 will be 60% of SP6.

```
$LIST/P pg01 ↵              $LIST/P pg02 ↵
.PROGRAM PG01()             .PROGRAM pg02()
1 SPEED 1000mm/s ALWAYS     1 JOINT   SP9   ACCU1  TIMO  TOOL1  OX   WX
2 LMOVE aa                  2 LINEAR  SP6   ACCU1  TIMO  TOOL1  OX   WX
3 LMOVE bb                  3 JOINT   SP9   ACCU1  TIMO  TOOL1  OX   WX
4 LMOVE cc                  1 LINEAR  SP6   ACCU1  TIMO  TOOL1  OX   WX
.END                        .END
$SPEED 50 ↵                 $SPEED 60 ↵
$                           $
```

Figure 4-26 SPEED Command

4.5.2. PRIME COMMAND

The PRIME command is used to prepare the system to execute a program. Its entry format is `PRIME` *`program name, execution cycles, step number`*, with PRIME being the processor-recognized keyword and all remaining arguments being optional.

The PRIME command does not actually cause a program to run; it is used to select the program name to be run and designate the number of execution cycles it will complete. The step number from which program execution is to begin can also be specified.

If No *program name* is entered, the program previously named in a prime or execute command will be selected.

Execution cycles variable sets the number of times the program is to be executed. If omitted, the program will run for single cycle only will allow the program to loop an infinite number of times. Though maximum number of recognized cycles is 32,767, entering a negative value.

To see the prime command in action, consider the example instruction `PRIME` *`prog01,5,3`*. The result would be program "*prog01"* being loaded and, once the cycle start button is pushed or the command `CONTINUE` typed, the program beginning at step 3 and running five consecutive times.

4.5.3. EXECUTE COMMAND

The EXECUTE STEP command is used to start a robot control program. The format for the command is `EXECUTE` *`program name, execution cycles, step number,`* with `EXECUTE` being the processor-recognized keyword and all remaining arguments being optional.

Program name identifies the program to be run. If omitted, the program list named in an EXECUTE, PRIME, MSTEP, or STEP command will be selected. *Execution cycles* sets the number of times to program is to be run. As above, the maximum number of recognized cycles is 32, 767, though entering a negative value will allow the program to loop an infinite number of times. *Step number* sets the step at witch the program is to

begin, Omitting this will default the program to begin from step 1.

4.5.4. STEP COMMAND

The STEP command is used to execute a single program step. The format for this command is `STEP` *`program name, repeat count, step number`*, with STEP this being the processor-recognized keyword and all remaining arguments being optional.

Program name denotes of the program containing the step to be executed. Omitting this will cause the last executed program or the program currently running to be selected by default.

Repeat count is the number of times the program is to be repeated is selected. If omitted, the default setting of 1 is selected. A “Program completed” message will be displayed following executing of the cast step, at which point the step command will no longer be effective. In order to continue stepping through the program completion, after the user must specify a repeat count greater than 1.

S*tep number* is the number of the program step to be executed. If omitted, step 1 is automatically selected. For example, the command `STEP pg01,1,5` will execute step 5 of program pg01 just once. If the command `STEP` is entered at this point, step 6 will be executed.

4.5.5. MSTEP COMMAND

The MSTEP command is used to execute the next motion step of a program. The format for this command is `MSTEP` *`program name, repeat count, step number`*. With the MSTEP command, non motion-related steps within the program are executed, but the program will advance to and execute the next step that initiates robot motion.

Program name denotes the program containing the step to be executed. If omitted, the last executed program will be automatically selected.

Repeat count is the number of times the program is to be repeated. If omitted, the default setting of 1 is selected. A “Program completed” message will be displayed following execution of the last step, at which point the step command will be no longer be effective. In order to continue stepping through the program after completion, the

user must specify a repeat count greater than 1.

Step number is the number of the program step to be executed. If omitted, step 1 is automatically selected. Figure 4-27 shows an example of the MSTEP command in use.

```
$EDIT test,5 ↵
.PROGRAM TEST()
5  LMOVE bb
5? PRINT 5↵
6  JMOVE cc
7  SPEED 50 always
8  SIGNAL 11
9  SIGNAL -3,-5,-7
10 JMOVE xyz
10? E
$MSTEP ↵
```

Figure 4-27 MSTEP Command

In the above example. If the program named “test” was stopped at step 5 and the MSTEP command entered from the monitor prompt, the robot would set speed to 50% (step7), output 11 to on (step 8), outputs 3, 5, and 7 to off (step 9), and move in a joint interpolated path to location “xyz.” If a STEP command has been used in the same example, only the robots speed would be set; `STEP` command would need to be entered three more times before the robot would move to location “xyz.”

4.5.6. ABORT COMMAND

The ABORT command is used to halt execution of a robot control program. If a step is currently in progress, the program will halt after the step has been completed. Similarly, a robot in motion will continue to its next programmed location, decelerate, and then at that point. MOTOR POWER remains on and brakes will not be applied. The cycle start light will not be illuminated.

Program execution is resumed by entering `CONTINUE` or `EXECUTE`.

4.5.7. HOLD COMMAND

The HOLD command stops a robot in motion faster than ABORT. Entering `HOLD` begins immediate robot deceleration, after which the robot will come to a complete stop.

MOTOR POWER and CYCLE START lamps remain ON.

Program execution is resumed by entering `CONTINUE` or `EXECUTE`.

4.5.8. CONTINUE COMMAND

CONTINUE command is used to resume the robot motion after being interrupted by (1) system error or (2) the PAUSE, ABORT, or HOLD commands. The format for this command is **`CONTINUE`** *`next`*, with the argument *next* being optional. The CONTINUE command cannot be used to resume programs that completed their assigned cycles, or were stopped with HALT or KILL commands.
The optional argument *next* is available to specify will resume after the current step. If the argument *next* is omitted, execution will begin from the current step.
In the example in figure 4-28, the robot moves to location cc (step 6) and waits for signal 1009 (step 7) before processing any additional steps. Once the CONTINUE next command is entered, the SWAIT instruction is bypassed and the robot will move to the location “pounce” (step 8) and continue to the program.

```
$EDIT test,5 ↵
.PROGRAM TEST()
5  LMOVE bb
5? PRINT 5↵
6  JMOVE cc
7  SWAIT 1009
8  JMOVE pounce
9  SIGNAL -3,-5,-7
10 JMOVE xyz
10? E
$CONTINUE next ↵
```

Figure 4-28 CONTINUE Command

4.5.9. STPNEXT COMMAND

The STPNEXT command is used when the system switch is turned to STEP_ONCE (STEP_ONCE is ON). The controller will process the next step of the program each time the STPNEXT command is entered.

4.5.10. KILL COMMAND

The KILL command removes a current program from active status. It is used when the

program has halted in mid-operation because of error or PAUSE/ABORT commands. Entering `KILL` will display a prompt asking the user to confirm. If the program stack is initialized and no program name will appear in the status display.

The KILL command does not delete the program.

4.5.11. DO COMMAND

The DO command is used to execute a single program instruction from the monitor prompt. The format for this command is `DO` *`program instruction`*. If ***program instruction*** is omitted, the last instruction named in a DO command win be repeated.

Generally, program instructions are processed only when the programs containing those instructions are executed. Using the DO command monitor, however, a single instruction can be executed without having to create a program specifically for that instruction.

For example, entering `DO JMOVE safe` will direct the robot, in joint-interpolated motion, to the position designated by the variable “safe.”

4.6. COMMANDS FOR DEFINING LOCATION VARIABLES

The AS language affords the programmer considerable flexibility in defining a number of location variables, including:

- Setting/modifying the robot's current position
- Assignment of other locations (or their X, Y, Z, O, and A component).
- Decamped elements from existing locations.

4.6.1. HERE COMMAND

The format for the HERE command is `HERE` *`location name`*, with *location name* defining the current_robot location. The variable defined with the HERE command can be a transformation location, precision location, or compound transformation location.

The HERE command is entered, the display will prompt the programmer for locational dimensions or angles, and ask if the information entered is correct. Pressing the ENTER key will close the screen and record the new locational data.

Precision locations are defined by preceding *location name* with the # symbol. Compound transformations are defined using the + symbol to make the location on the far right of the expression relative to another location. Figure 4-29 shows an example of the HERE command in use.

AS LANGUAGE COMMANDS

```
$HERE fixt.1 ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  -487.8  1272.81  981.40   163.37   -88.97   27.11
CHANGE? (if not hit RETURN only)
,,,,-90 ↵
   -487.8  1272.81  681.40   163.37   -90.00   27.11
CHANGE? (if not hit RETURN only) ↵
$HERE #fixt.1 ↵
    JT1      JT2      JT3      JT4      JT5      JT6
  -21.12   149.20   30.33   -60.63   71.40    55.65
CHANGE? (if not hit RETURN only) ↵
$HERE dog ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  -1169.21 1254.61  970.75   87.96    91.06    41.08
CHANGE? (if not hit RETURN only) ↵
$HERE dog+cat
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  -186.59  171.53   184.87   -73.69    0.04    73.71
CHANGE? (if not hit RETURN only) ↵
$LIST/L ↵
       X[mm]    Y[mm]     Z[mm]    O[deg]   A[deg]    T[deg]
cat   -186.59   171.53    184.87    -73.69     0.04     73.71
fixt.1-487.8  1272.81    581.40    163.37   -90.00     27.11
dog -1169.21  1254.61    670.75     87.96    91.06     41.08
        JT1      JT2       JT3       JT4      JT5       JT6
#fixt.1-21.12  149.20     30.33    -60.63    71.40     55.65
$
```

Figure 4-29 HERE Command

In the above example the HERE command is used to define two transformational locations (fixt.1 and dog), a precision point (#fixt.1), and a compound transformation (dog + cat). The `LIST` command is also shown to display the stored values for the location variable names. The values for location “cat” must be used in a compound transformation with location “dog.” (A `Destination out of range` error would result if the robot were instructed to travel simply to location “cat.”)

If the HERE command defines a location variable already in system memory, the existing value(s) will be replaced. Care must be taken that location names are not repeated.

4.6.2. POINT COMMAND

The POINT command is used to assign an identical value to two location values. The format for this command is `POINT` *`location variable = location value`*. If the location variable to the left of the assignment sign has not been defined and the location value on the right is omitted, all component values are assumed to be zero. The zeroes will be displayed, along with the prompt `CHANGE?`.

See Figure 4-30 for an example of the POINT command in use.

```
$POINT fixt.3 ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
    0.00     0.00     0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only)
1000,1000,1000,,90 ↵
   1000.0   1000.0   1000.0    0.00    90.00   1000.0
CHANGE? (if not hit RETURN only) ↵
$POINT #fixt.2 ↵
    JT1      JT2      JT3      JT4      JT5      JT6
   -21.12   149.20   30.33   -60.63   71.40    55.65
CHANGE? (if not hit RETURN only) ↵
$POINTlstspt=cixt.3 ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
   1000.0   1000.0   1000.0    0.00    90.00   1000.0
CHANGE? (if not hit RETURN only) ↵
$POINTlststp
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
   1000.0   1000.0   1000.0    0.00    90.00   1000.0
CHANGE? (if not hit RETURN only) ↵
$
```

Figure 4-30 POINT Command

In the above example, the location fixt.3 does not yet exist (see first line); accordingly, its values are returned as `O` followed by CHANGE? Values for fixt.3 are then defined as `1000, 1000, 1000,, 90` (the `,,` referents the fact that it is not necessary to enter values of O in the AS language).

Next, the precision point #fixt.2 is confirmed as being in system memory via the POINT command. The programmer does not wish to change its values, so ENTER is pressed lastly, the location variable lststp is assigned the same valles as fixt.3.

[NOTE] If a variable named lststp had previously existed, it will now be overwritten.

The location variable to the left of the assignment sign and the location value to the right should both be of the same type; that is, both either transformations or precision points. When the POINT command is used with the location variable named “lststp,” the location values are displayed and are the same as those for fix.3.

4.6.2.1. VARIATIONS OF THE POINT COMMAND

While the POINT command is used to define the same value to two location variables, there are a number of variations to the POINT command that assign only *part* (or parts) of a value. These variations are POINT/X, POINT/Y, POINT/Z, POINT/OAT, POINT/O, POINT/A, POINT/T, and POINT/7. With each of these commands, the specified component values to the right side of the assignment sign are assigned to the location variable on the left, Figure 4-31 illustrates some of these examples.

```
$POINT xyz.abc ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
    0.00     0.00     0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only)
$POINT/X xyz.abc=loc.a ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  1583.21   0.00     0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only) ↵
$POINT xyz.abc=loc.b ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  1583.21 1378.11   0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only) ↵
$POINT/Z xyz.abc=loc.c ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  1583.21 1378.11  583.71    0.00     0.00     0.00
CHANGE? (if not hit RETURN only) ↵
$POINT/Z xyz.abc=loc.x ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  1583.21 1378.11  583.71   90.00   180.00   90.00
CHANGE? (if not hit RETURN only) ↵
$
```

Figure 4-31 Variations of the POINT Command

In the above example, the POINT/X command is used to assign the X component value (1583.21mm) of location loc.a to the X component of location variable *xyz.abc*. `POINT/Y` is then entered to assign the Y component value `(1378.11mm) of location loc.b` to the Y component of location variable *xyz.abc*. The POINT/Z command assigns the Z component value (583.71mm) of location loc.c to the Z component of location variable *xyz.abc*. Lastly, `POINT/OAT` is used to assign the OAT component angles (90.00, 180.00, and 180.00) of location loc.d to OAT component angles of location variable xyz.abc.

4.6.3. TEACH COMMAND

The format for the TEACH command, which is used in conjunction with the small Teach Pendant, is `TEACH` *`location variable name`*. The TEACH COMMAND allows. The programmer to record transformation locations without having to exit the work cell for each new location, and the pendant jogs the robot to the locations within the specified program **using the Teach Pendant**. Pressing the RECORD key places the location into system memory, with the specified variable name followed by a `0`. This number increases by 1 each time RECORD is pressed for example, the command `THECH SPTWELD` is entered at the monitor prompt. The first time record is pressed, a location named sptweld 0 is saved in system memory. Pressing RECORD again will store the location under the name sptweld 1.

4.6.4. TRHERE COMMAND

The TRHERE command sets a transformation all value as each to the robot's current position. The format for the command is `TRHERE` *`location_variable`*.
Location_variable specifies the name of the transformation or the compound transformation.
The current position of the robot should be defined as a transformational value assigning a compound transformation will define only those elements to the farmost-right within the compound transformation. Error occurs when variables other than those elements to the right of the compound transformation have not yet been defined.
When not erroneously defined as a compound transformation, `TRHERE` is exactly the same as the HERE command.

In the case of compound transformation, motion elements are added to those of the X axis, and the specified relative operation is performed when this operation is complete, motion elements are again subtracted from X-axis elements. The resulting value of these motion elements are then used as the current position value. For the motion (traverse) axis.
Use the monitor-based TRHERE command when performing this type of relative designation lone which takes into account elements of the motion axis.
When a variable is assigned a new value, the value will appear on screen along with the prompt `CHANGE?` The value may then be amended. If desired note that each element should be separated by commas, and that values not to changed maybe omitted.
The `CHANGE?` will repeatedly appear until the screen is closed by pressing the RETURN key (only).

```
>TRHERE pick ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[mm]
    1300     200      1000      90        0      1100
CHANGE? (if not, hit RETURN key)
↵
>
```

The current position of the robot is defined as “Pick,” and movement (traverse) axis values are displayed.

```
>TRHERE plate+object ↵
```

The robot’s current position is defined as “Object,” itself defined relative to position “plate.”

4.6.5. BSHIFT COMMAND

The BSHIFT command shifts only the input XYZOAT elements of designated transformational values to the base direction. Using monitor commands, setting values such as POINT commands can be displayed, and command values revised.
The format for this command is `BSHIFT transformation , X, Y, Z, O, A, T.`

4.6.6. TSHIFT COMMAND

The TSHIFT command shifts only the input XYZOAT elements of designated transformational values to the base direction. The format for this command is `TSHIFT transformation , X, Y, Z, O, A, T`. Using monitor commands, setting values such as POINT command can be displayed, and command values revised.

4.7. SYSTEM CONTROL COMMANDS

System control commands provide the user the flexibility to: check the status, of various items, set system limits and parameters, define tools, identify home positions, set the time, view error and operations logs, and define system input and output configurations.

4.7.1. STATUS COMMAND

The STATUS command displays information on the system and the robot control program being executed. Figure 4-32 shows an example of the status display.

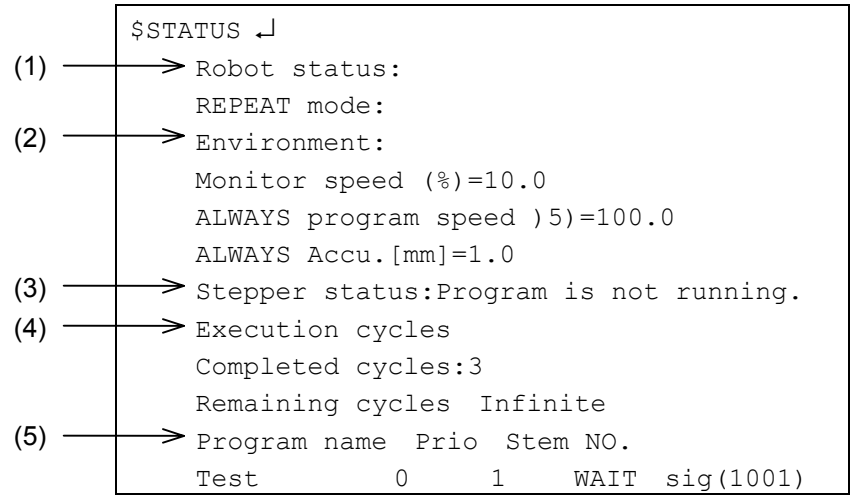


Figure 4-32 STATUS Command

(1) The `Robot status` section provides information on the condition of the robot. Output is in the from of the six following messages:

Error :	An error has occurred attempt to reset the error.
Motor power off:	Power to the motor has been turned off.
Teach mode:	The robot is ready to be controlled using either the Multi Function Panel or small teach pendant. Motor power is on.
Repeat mode:	Motor power is on, and the robot control program is running.
Repeat mode cycle start on:	The robot is reads to be controlled by the robot control program. Motor power is on.
Program wait:	The robot control program is running and in wait more

(executing a WAIT, SWAIT or TWAIT instruction).
Motor power is on

(2) The `settings` section of the status display contains the following information:

Monitor Speed (%)=10.0
ALWAYS Program Speed (%)= 100.0
ALWAYS Accuracy [mm]=1.0

(3) The Stepper status section of the status display contains the information about the present state of step execution.

(4) The `Execution Cycles` section of the status display contains the following information:

Completed cycles: The number of execution cycles already completed. (This number can range follows 0~32,767.

Remainder cycles: Remaining execution cycles. If you specified -1 as execution cycles in the EXECUTE command, “Infinite” is displayed. In the event that execution cycles was set to -1 via the EXECUTE command, `Infinite` is displayed here.

(5) The Program name section of the display shows the name of the program currently running, the step number being processed, and the required input number if the program is in a wait condition.

4.7.2. WHERE COMMAND

The **WHERE** commands are used to display robot positional information. There are seven WHERE commands each with a varying output (see below):

WHERE	Displays information on the robot’s current location in terms of (1) base coordinates (XYZOAT) and (2) values for the angle of each axis.
WHERE 1	Displays current angle values for each joint in degrees.
WHERE 2	Displays current XYZOAT base coordinates in millimeters and degrees.
WHERE 3	Displays the current command values.

WHERE 4 Displays deviations in bits from command values.
WHERE 5 Displays an encoder value in bits for each joint.
WHERE 6 Displays the speed of each joint in degrees per second

The WHERE command rovides a “snap_shot” of positional information while `WHERE number` are an active display of scrolling current information. Press the ENTER key to pause the flow of information.

Figure 4-33 shows an example of the displays available with the WHERE commands.

AS LANGUAGE COMMANDS

```
$WHERE ↵
     JT1        JT2        JT3        JT4        JT5        JT6
   11.998     12.734     -49.549    27.190     -24.714    128.959
    X[mm]      Y[mm]      Z[mm]      O[deg]     A[deg]     T[deg]
  -270.379   1424.420   525.474    110.981    158.083    93.324

$WHERE 1 ↵
joint value
     JT1        JT2        JT3        JT4        JT5        JT6
   11.998     12.734     -49.549    27.190     -24.714    128.959


$WHERE 2 ↵
Transformation value
  -270.379   1424.420   525.474    110.981    158.083    93.324


$WHERE 3 ↵
Joint command
   11.998     12.734     -49.549    27.190     -24.714    128.959


$WHERE 4 ↵
Joint position error
      0          0          0          0          0          0


$WHERE 5 ↵
Joint encoder value
     JT1        JT2        JT3        JT4        JT5        JT6
  268464606 268487546  268142745  268576237  268577166  268840020


$WHERE 6 ↵
Joint speed
     JT1        JT2        JT3        JT4        JT5        JT6
      0          0          0          0          0          0
```

Figure 4-33 WHERE Commands

4.7.3. IO COMMAND

The IO/E number command displays the current state of all internal and external inputs/output signals. There are four selections available with the IO command:

IO 1 : Displays the current state of signals 1-32, 1001-1032, and 2001-2032.

IO 2 : Displays the current state of signals 33-64, 1033-1064, and 2033-2064.

IO 3 : Displays the current state of signals 65-96, 1065-1096, and 2065-2096.

IO 4 : Displays the current state of signals 97-128, 1097-1128, and 2097-2128.

If no number is specified with the IO command, the current state of signals 1-32, 1001-1032, and 2001-2032 is displayed.

Should the DISPIO_01 system switch be turned to OFF, "○" is displayed to represent an ON signal, and "x" for an OFF signal. Dedicated signals are denoted by uppercase, letters.
If the DISPIO_01 system switch is ON, on signals are represented by and ○ a signal that is OFF. Uninstalled external output signals are represented by **-**.
The optional IE allows for status of the 3001 and up signal range to displayed, as well as the normal 1-32/1001-1032/2001-2032 ranges.
The IO signal status display will continue to be updated until the RETURN key is pressed.

```
$IO ↵
32-    1    XxXx   Oxxx   xxxx   xxxx   xxxx  xxxx    xxxx   xxxx
1032-1001   xxxx   xxxx   xxxx   xxxx   xxxx  xxxx    xxxx   xxxx
2032-2001   xxxx   xxxx   xxxx   xxxx   xxxx  xxxx    xxxx   xxxx
↵
$IO2 ↵
64-   33    - - - -- - - -- - - -- - - -- - - -       - - - -- - - -
 - - - -
1064-1033   - - - -- - - -- - - -- - - -- - - -       - - - -- - - -
 - - - -
2064-2033   xxxx   xxxx   xxxx   xxxx   xxxx  xxxx    xxxx   xxxx
```

Figure 4-34 IO Command

In the example, above IO command signal numbers 7, 11, and 13 are general purpose signals that are ON. Signal 28 is a dedicated signal that is ON, and signals 30 and 32 are dedicated signals that are OFF. The `IO 2` command display shows that external IO signals 33 - 64 and 1033 -1064 are not currently configured in this controller. Both examples are with DISPIO_01 system switch OFF.

See section 4-7-14 is for more information on the DISPIO_01 system switch.

4.7.4. FREE COMMAND

The FREE command is displays the amount of available memory in both a percentage and in a number bytes. Figure 4-35 shows an example of the display when the FREE command is used.

```
$FREE ↵
All memory size 490584 bytes.
Available memory size 458680 bytes (93%).
$
```

Figure 4-35 The FREE Command

4.7.5. TIME COMMAND

The TIME command is used to set or display the current date, time, and day of the week. To change the data, enter `TIME year-month-day`, all values are in pairs (for example, enter February 16, 2001 as `01-02-16`). Clock time is entered as time hours: minutes: seconds. As above, time values are entered as pairs, using zeroes when necessary.

See Figure 4-36 for the time command in use.

```
$TIME ↵
98-01-03 Sat) 13:35:50
Change? (If not, hit RETURN key) ↵
$TIME 98-02-11 10:35:30 ↵
$TIME ↵
98-02-11 (Wed) 10:35:38
Change? (If not, hit RETURN key) ↵
$
```

Figure 4-36 TIME Command

In the example, date and time are changed and then `TIME` entered to verify the change. The day of the week is added by the processor, we see that eight seconds have elapsed since the new time was entered.

4.7.6. ULIMIT COMMAND

The ULIMIT command is used to set (and display) the uppermost limit on the range of motion for each robot joint. Limits are specified in degrees of travel from the zero reference point. Limits are set according to user-specified parameters, or using the location values of a precision point. Figure 4-37 belows shows an example of the ULIMIT command in use.

```
#ULIMIT ↵
          JT1      JT2     JT3      JT4      JT5      JT6
Maximum   120.000  60.00   60.00    190.00   115.00   270.00
Current   109.20   45.23   52.11    190.00   111.40   250.00
Change? (If not, hit RETURN key)
115.10,,58,175 ↵
          JT1      JT2     JT3      JT4      JT5      JT6
Maximum   120.000  60.00   60.00    190.00   115.00   270.00
Current   115.10   45.23   58.00    175.00   111.40   250.00
Change? (If not, hit RETURN key) ↵
$ULIMIT#endmove ↵
          JT1      JT2     JT3      JT4      JT5      JT6
Maximum   120.000  60.00   60.00    190.00   115.00   270.00
Current   103.88   24.57   38.21    155.24   107.66   179.31
Change? (If not, hit RETURN key) ↵
$
```

Figure 4-37 ULIMIT Command

In the example above, limits are first set by the user, and then set to equal locational values of precision point `#endmove`.

4.7.7. LLIMIT COMMAND

The LLIMIT command is used to set (and display) the lowermost limit on the range of motion for each robot joint. Limits are specified in degrees of travel from the zero reference point. Limits can be set to user specified-parameters, or using the location values of a precision point. The – sign is not assumed when setting lower limits and must be added to avoid limits from being to restrictive.

Figure 4-38 below shows an example of the LLIMIT command in use.

```
#ULIMIT ↵
          JT1      JT2     JT3      JT4      JT5      JT6
Maximum   -120.000 -60.00  -60.00   -190.00  -115.00  -270.00
Current   -109.20  -45.23  -52.11   -190.00  -111.40  -250.00
Change? (If not, hit RETURN key)
-115.10,,-58,-175 ↵
          JT1      JT2     JT3      JT4      JT5      JT6
Maximum   -120.000 -60.00  -60.00   -190.00  -115.00  -270.00
Current   -115.10  -45.23  -58.00   -175.00  -111.40  -250.00
Change? (If not, hit RETURN key) ↵
$ULIMIT#bottomspt ↵
          JT1      JT2     JT3      JT4      JT5      JT6
Maximum   -120.000 -60.00  -60.00   -190.00  -115.00  -270.00
Current   -103.88  -24.57  -38.21   -155.24  -107.66  -179.31
Change? (If not, hit RETURN key) ↵
$
```

Figure 4-38 LLIMIT Command

In the example above, limits are first set by the user, and then set to equal locational values of precision point `#bottomspt`.

4.7.8. BASE COMMAND

The **BASE** command is used to change the origin of the base coordinate system. This will affect the position of all transformational location variables recorded before use of the BASE command was used, but precision locations will remain unaffected.

Entering the BASE command will display the current base location along with the prompt `CHANGE?`. The command `BASE NULL` can be used to set the position of the base to all zeros, and a transformation location used to assign the values of that location to the base. Figure 4-39 below shows an example of the BASE command in use.

```
>BASE ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
  -200.0    0.00     0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only) ↵
$BASE NULL ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
   -0.00    0.00     0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only) ↵
$BASE lineshift ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
    0.00   200.00   250.00    0.00     0.00     0.00
CHANGE? (if not hit RETURN only) ↵
$
```

Figure 4-39 BASE Command

Figure 4-39 begins with the BASE command entered to display current values. `BASE null` is then entered to clear all values to zero, and finally base given the values of transformational location “lineshift”. The new values are shown, and return pressed to conclude the process. See Figure 4-40 for a graphical representation of this process.

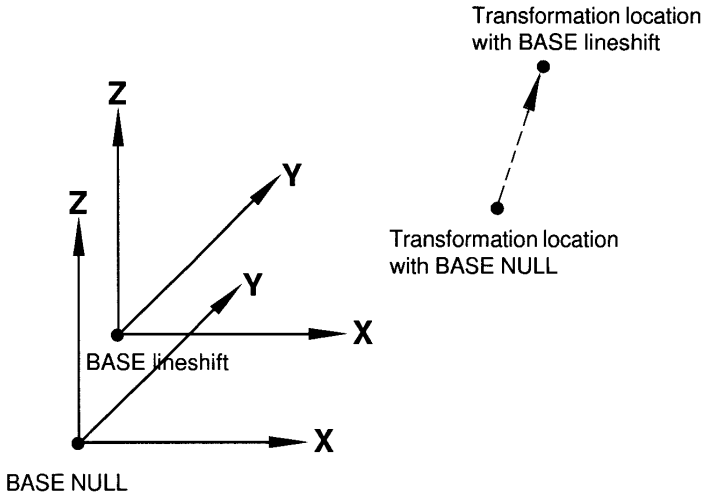


Figure 4-40 Movement of BASE Location

4.7.9 TOOL COMMAND

The TOOL command is used to assign a transformation value to define the tool center point (TCP) relative to the tool mounting flange. The format for this command is `TOOL transformation value`. When the robot is instructed to travel to transformation locations, it is the TCP that will be at the specified values in the coordinate system (assuming system switch QTOOL is OFF and the TCP has been correctly dimensioned and assigned). In the example shown in figure 4-41, the transformation value *little_tool* is defined and the TOOL command used to assign the dimensional values *of little_tool* to the TCP.

```
$POINT little_tool ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
    0.00     0.00     0.00     0.00     0.00     0.00
CHANGE? (if not hit RETURN only)
,,1500,,45 ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
   -0.00     0.00   1500.0    0.00    45.00     0.00
CHANGE? (if not hit RETURN only) ↵
$TOOL little_tool ↵
   X[mm]    Y[mm]    Z[mm]   O[deg]   A[deg]   T[deg]
    0.00     0.00   1500.0    0.00    45.00     0.00
CHANGE? (if not hit RETURN only) ↵
$
```

Figure 4-41 TOOL Command

4.7.10. SETHOME COMMAND

The HOME position is a special location in AS language programming where dedicated output signals can be assigned, and are generated whenever the TCP is within the accuracy range of the HOME position. The format for setting the home position is `SETHOME` *`accuracy, HERE`*. If the *accuracy* is not specified, the default value of 1.00 mm is used. The program instruction HOME will cause the robot to perform a joint interpolated move to the HOME position.

Figure 4-42 shows the `SETHOME 10, HERE` command used to set the HOME position at the current robot position with an accuracy of 10.0 mm. `SETHOME` is then entered to display location values set for the HOME position. No changes are made with this instruction. `SETHOME 5` is used to change the accuracy range of the HOME position from 10.0 mm to 5.0 mm. When the prompt to change the positional data is displayed, all angles are all reset to 0.

```
$SETHOME 10, HERE ↵
    JT1      JT2      JT3      JT4      JT5      JT6    Accuracy[mm]
  -21.12   149.20   30.33   -60.63   71.40    55.65       10.00
CHANGE? (if not hit RETURN only) ↵
$SETHOME ↵
    JT1      JT2      JT3      JT4      JT5      JT6    Accuracy[mm]
  -21.12   149.20   30.33   -60.63   71.40    55.65       10.00
CHANGE? (if not hit RETURN only) ↵
$ SETHOME 5 ↵
    JT1      JT2      JT3      JT4      JT5      JT6    Accuracy[mm]
  -21.12   149.20   30.33   -60.63   71.40    55.65        5.00
CHANGE? (if not hit RETURN only)
0,0,0,0,0,0 ↵
    JT1      JT2      JT3      JT4      JT5      JT6    Accuracy[mm]
   0.00     0.00     0.00     0.00     0.00     0.00        5.00
CHANGE? (if not hit RETURN only) ↵
$
```

Figure 4-42 SETHOME Command

4.7.11. SET2HOME COMMAND

The C-series controller SET2HOME command allows for a second HOME position to be identified. HOME2 location shares all of the characteristics of the HOME position, and is defined via the `SET2HOME` command. Typical locations for HOME and HOME2 include a position with all of the alignment scribe marks set, a manual tip dress position, a gun purge location, or a tool changer station.

4.7.12. ERRLOG COMMAND

The ERRLOG command displays the error history stored in system memory. The log contains information on the last 100 errors incurred, beginning with the most recent. The user can alternate between previous and next screens, using keys F4 and F5 on the keyboard or Multi Function Panel. Press the space bar to view additional log entries, or enter to exit.

The format of the error log is shown in Figure 4-43.

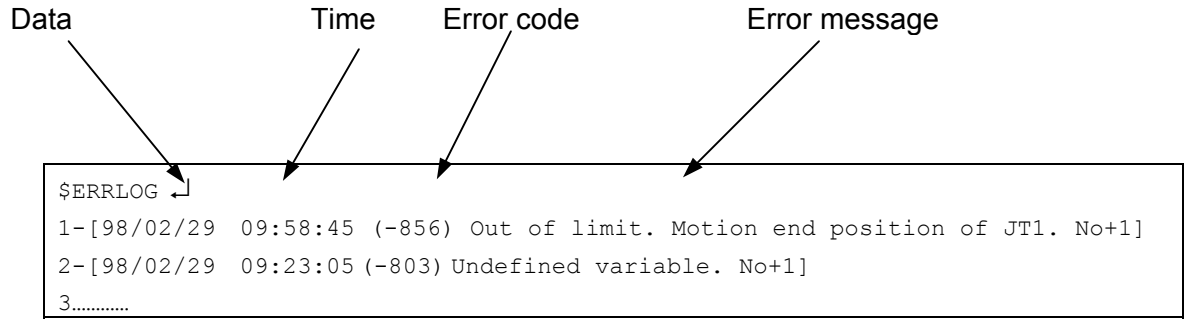


Figure 4-43 ERROR Command

4.7.13. OPLOG COMMAND

The OPLOG command is used to display a history of the last 100 operations performed to the controller, and any messages relating to those operations. As with the ERRLOG command, messages are displayed ten at a time and order of most recent to oldest. Press the space bar to view additional log entries, or enter to exit.

The format of the operations log is shown in Figure 4-44.

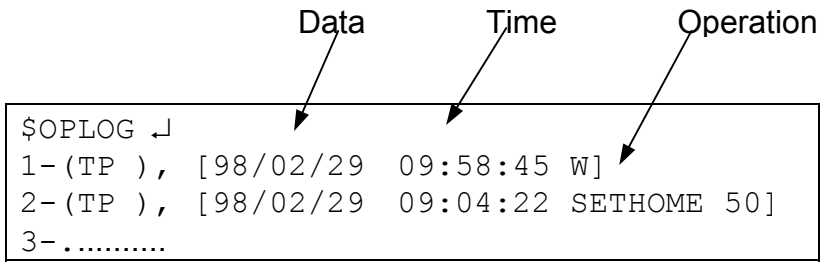


Figure 4-44 OPLOG Command

4.7.14. SWITCH COMMAND

The Switch command is used to display the setting for all system switches. The format for turning a single switch on or off is `SWITCH` *`switch name ON/OFF`*. Simply enter `SWITCH` to display the status of an individual switch, to change the status of system switches. Figure 4-45 shows an example of the SWITCH command display.

```
$SWITCH ↵
*POWER              ON      *REPEAT          ON
*RUN                ON      *CS              OFF
*RGSO               OFF     *ERROR           OFF
*TRIGGER            ON      *TEACH LOCK      OFF
CHECK.HOLD          OFF     CP               ON
CYCLE.STOP          OFF     OX.PREOUT        ON
PREFETCH.SIGINS     OFF     QTOOL            OFF
REP ONCE            OFF     RPS              OFF
STP ONCE            OFF     AFTER.WAIT.TMR   ON
MESSAGES            ON      SCREEN           ON
AUTOSTART.PC        OFF     AUTOSTART2.PC    OFF
AUTOSTART3.PC       OFF     ERRSTART.PC      OFF
DISPIO 01           OFF
```

Figure 4-45 SWITCH Command

Note: Switches preceded by a `*` cannot be altered using the SWITCH command.
For example the switch labeled `*POWER` will only indicate `ON` when motor power is on.
The command `SWITCH POWER OFF` will not turn motor power off.

The default setting for switches is shown in table 4-1.

AS LANGUAGE COMMANDS

Table 4-1 System Switch Default Settings

Switch	Setting
CHECKHOLD	OFF
CP	ON
CYCLE.STOP	OFF
OX/PREOUT	ON
PREFETCH.SIGINS	OFF
QTOOL	ON
REP_ONCE	OFF
RPS	OFF
STP_ONCE	OFF
AFTER.WAIT TIMER	OFF
MESSAGES	ON
SCREEN	ON
AUTOSTART.FC	OFF
AUTOSTART2.PC	OFF
AUTOSTART3.PC	OFF
ERRSTAR.PC	OFF
DISPIO_01	OFF
FLOWRATE	OFF
HOLD.STEP	OFF
WS_COMPOFF	OFF
SPOT_OP	OFF

4.7.14.1. CHECK.HOLD SWITCH

The CHECK.HOLD switch is used with the commands EXECUTE, DO, STEP MSTEP and CONTINUE. When the CHECK.HOLD switch is ON, these commands are only available for use if the HOLD/RUN switch is in the hold position. The controller will accept these commands with HOLD/RUN set to HOLD, but robot motion will not begin until the switch is manually turned to RUN.

The default setting for the CHECK.HOLD switch is off.

4.7.14.2. CP (CONTINUOUS PATH) SWITCH

The CP switch is used to activate the continuous path function. When this switch is ON and accuracy ranges are large enough, the robot will be capable of smooth point-to-point transitions of the type indicated in Figure 4-46. When the CP switch is OFF, the robot will decelerate and stop at each recorded point regardless of the accuracy of those points.

The default setting for the CP switch is ON.

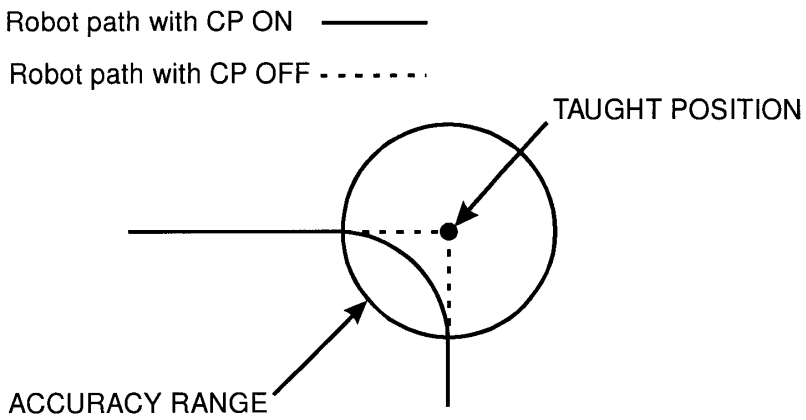


Figure 4-46 CP Switch

4.7.14.3. CYCLE.STOP SWITCH

The CYCLE.STOP switch is used in conjunction with an external input signal to stop robot motion. With the switch ON, the robot will stop and the cycle start light turn OFF, when the input signal is received. When re-started the program will not resume from where it left OFF.

With the switch OFF, the robot will stop and the cycle start light remain ON, when the input signal is received. The robot is places into a hold position meaning that a program re-started will continue where it left.

The default setting for the CYCLE STOP switch is OFF.

4.7.14.4. OX.PREOUT SWITCH

The OX.PREOUT switch affects the timing of output signal generation in block step

programs. With the switch ON, an output programmed for a given point will turn ON as soon as the robot begins motion to ward that point. With the OX.PREOUT switch OFF, an output programmed for a given point will not be turned ON until the robot reaches the accuracy range of the point. Figure 4-47 shows the different effects the OX.PREOUT switch has on signal timing.

The default setting for the OX.PREOUT switch is ON.

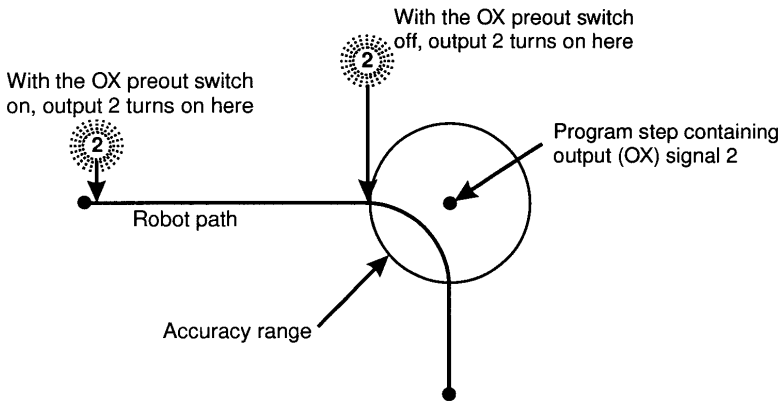


Figure 4-47 OX.PREOUT Switch

4.7.14.5. PREFETCH.SIGINS SWITCH

The PREFETCH.SIGINS switch is used in conjunction with AS language instructions and has the same effect on signal timing that the OX.preout switch has with blockstep instructions. The default setting for the prefetch.sigins switch is on.

4.7.14.6. QTOOL SWITCH

The QTOOL switch allows the user to identify tools to be used in block step or AS language programming. With QTOOL switch ON, the programmer has up to nine tools available for use with programming and jogging. The unique tool dimensions are recorded and assigned a tool number using auxiliary function number 48. When the QTOOL switch is ON, the selected tool dimensions will be in effect for jogging and linear playback of block step programs; with the switch OFF, the tool identified with AS language instructions is utilized.

The default setting for the QTOOL switch is ON.

4.7.14.7. REP_ONCE (REPEAT ONCE) SWITCH

With the REP_ONCE switch ON programs will run only once; With the switch OFF, the program will run continuously. The default setting for the switch is off.

4.7.14.8. RPS (REMOTE PROGRAM SELECTION) SWITCH

This switch enables the processor to scan for binary input from peripheral devices, and select which program is to be executed. The AS language instructions EXTCALL, JUMP, and END combined with the input signal and RPS function determine, which program is to run.

The RPS switch default setting is on.

4.7.14.9. STP_ONCE (STEP ONCE) SWITCH

When the STP_ONCE switch is ON, the repeat function of progressing through a program one step at a time becomes active. The step forward key must be used to progress through a program. When the switch is OFF, programs will run continuously. The default setting for this switch is OFF.

4.7.14.10. AFTER.WAIT TIMER SWITCH

When the AFTER.WAIT TIMER switch is in the ON position, timers will not begin their timing at a specified step until all wait conditions are satisfied. With the switch in the OFF position timers will begin when the robot reaches coincidence of the taught point. The default setting for the switch is OFF.

4.7.14.11. MESSAGES SWITCH

The MESSAGES switch allows the display of PRINT and TYPE information that is part of programs to be displayed for operator viewing. If the messages switch is OFF this information will not be displayed. The default setting for this switch ON.

4.7.14.12. SCREEN SWITCH

The SCREEN switch enables or disables the scrolling of the screen when the amount of information being sent by the processor is larger than the screen can display. The default setting for the screen switch is ON (scrolling enabled).

4.7.14.13. AUTOSTART.PC SWITCH

The AUTOSTART.PC, AUTOSTART2.PC, and AUTOSTART3.PC switches will automatically start the associated PC program when control power is applied. The default setting for these switches is OFF.

4.7.14.14. ERRSTART.PC SWITCH

When the ERRSTART.PC switch is in the on position, a PC program is run as soon as specified errors (errors assigned dedicated signals) occur. The default setting for this switch is OFF.

4.7.14.15. DISPIO_01 SWITCH

The **DISPIO_01** switch allows the user to select the type of display available to view the status of input/output signals. When the switch is ON, a signal are represented by `1`, and off, signals by `O`. If the switch is OFF, ON signals are OFF signals represented by `O`, OFF signals by `X`. See Figure 4-34 for a sample display. Dedicated signals are represented by an uppercase X and O.

The default setting for the dispio_01 switch is OFF.

4.7.14.16. HOLD.STEP SWITCH (OPTIONAL)

When the HOLD.STEP switch is set to HOLD and provided there are no as yet uncompleted program steps, display will show the current stepper step in lieu of the present operational step. If the switch is set to RUN, the operational step is displayed or if there is no active step, the stepper step/hold step.

When the switch is ON, the stepper step of SWAIT command appears (Note that SWAIT can be skipped). Robot motion instruction A (see diagram) appears when the switch is set to OFF.

The default value for this switch is ON.

4.7.14.17. WS_COMPOFF SWITCH (OPTIONAL)

This switch changes between output timing conditions of the weld schedule. When this switch is ON, the *welding complete* input signal from the memory change is output. When OFF, output is from one memory change to the next. The default setting for this switch is OFF.

4.7.14.18. FLOWRATE SWITCH (OPTIONAL)

The FLOWRATE switch toggles between flowlate control (ON) speed output (OFF) modes. The default setting is OFF.

4.7.14.19. SPOT_OP SWITCH (OPTIONAL)

Turns SPOT learning mode ON or OFF. This switch cannot be used if the dedicated spot learning signal has not been set.

4.7.14.20. SWITCH (TRIGGER)

This displays whether wither the trigger switch on either the Teach Pendant or Multi-Function Panel is ON or OFF. The SWITCH cannot be manually set. Using the SWITCH function will display. `1` is if the trigger is ON. `0` if it is OFF.

4.7.14.21. SWITCH (CS)

Displays whether or not a cycle start is in progress. The switch cannot be manually set. Using the switch function will display `-1` if a cycle start is in process, `0` if not.

4.7.14.22. SWITCH (POWER)

Displays whether or not power is being supplied to the motor. The switch cannot be manually set to ON or OFF. Using the switch function will display `-1` if motor power is ON, `0` if it is not.

4.7.14.23. SWITCH (RGSO)

Display whether or not the servoing command to the motor is ON or OFF. The switch cannot be manually set to ON/OFF. Using the switch function will display -1, servoing is being performed, 0 if not.

4.7.14.24. SWITCH (TEACH LOCK)

Displays whether the TEACH_LOCK switch is ON or OFF. The switch cannot be manually set to ON or OFF. Using the switch function will display `-1` if the TEACH_LOCK is ON, `0` if not.

4.7.14.25. SWITCH (ERROR)

Displays whether or nut an error has occurred the switch cannot be manually set to ON or OFF. Using the switch function will display -1 if an error has occurred, or 0 if not.

4.7.14.26. SWITCH (REPEAT)

Displays witch of the two modes the TEACH/REPEAT switch of the operation panel is set to. The switch cannot be manually set to ON or OFF. Using the switch function will display `-1` if the operation panel is set to repeat `0` if to teach.

4.7.14.27. SWITCH (RUN)

Display which of the two modes the HOLD/RUN switch of the operation panel is se to. The switch cannot be manually set to ON or OFF. Using the switch function will display `-1` if the operation panel is set to RUN, `0` if to HOLD.

AS LANGUAGE COMMANDS

4.7.14.28. WS.ZERO SWITCH (OPTIONAL)

Switches to processing mode when welding condition (WS) becomes 0 if the switch is set to ON, processing of welding operations (and pressurization) is carried out whether WS is 0 or not If the switch is set to OFF, pressurization *only* is carried out when WS is 0.

The default setting of the WS.ZERO switch is OFF.

4.7.14.29. REP_CYC SWITCH (OPTIONAL)

Decides whether or not repeat cycles (carried out in repeat mode) are of a designated frequency. If the REP-CYC switch is ON, it becomes possible to set the number of repeat cycles using the REPCYCLE command. It is not possible to set repeat cycle frequency if the REP-CYC switch is OFF.

When REP-CYC becomes ON, REP-ONCE automatically shuts OFF, and vice versa. Shutting one or even both of these will in no way cause one to affect the other.

REP-CYC setting and results		
REP-CYC	REP-ONCE	CONDITION ①
ON	OFF	REPEATABILITY
OFF	OFF	CONTINUOUS
OFF	ON	ONCE
ON	ON	NOTE ②

① When repcycle is OFF, the condition shifts to continuous, thouch REP-CYC remains OFF.

② REP-CYC automatically shuts OFF, when REP-ONCE becomes ON.

When the frequency of repeat cycles is set to using the REPCYCLE command, REP-CYC becomes internally ineffective and will not turn ON. Similarly, repeat information on the Multi-Function Panel will fail to display.

4.7.14.30. ABS.SPEED SWITCH (OPTIONAL)

Switches to a mode where operations are performed at an absolute speed, regardless of monitor speed. When the ABS.SPEED switch is set to ON operations are done at absolute speed. When the following condition is met: max speed X monitor speed > program speed.

4.7.14.31. SLOW_START SWITCH (OPTIONAL)

When the switch is ON, programs will be executed with the first operational step only being performed at a slow speed. Even if a program step is stopped in mid-operation, the step will resume at slow speed once restarted. This has no connection to specialization of the “slow repeat” signal. If this switch is turned OFF, the slow start function will cease to operate.

4.7.15. ZSIGSPEC COMMAND

The ZSIGSPEC command is used to display and set the total number of input/output signals the controller has been physically configured for. To use the ZSIGSPEC command, type ZSIGSPEC at the monitor prompt and press the ENTER key. Type the number of I/O signals that correspond to the board configuration of the controller and press ENTER.

4.7.16. HSETCLAMP COMMAND

The HSETCLAMP command is assigns numbers to the signals that operate material handling clamps. Auxiliary function 114 works in conjunction with the HSETCLAMP command to set the operation of the material needling clamps. Figure 4-48 shows the screen displayed when the HSETCLAMP command is entered.

```
$HSETCLAMP↵
                CLAMP1       CLAMP2      CLAMP3    CLAMP4
                Spot weld    Handling    Not used  Not used
'ON' out. signal  24           24          24        24
'OFF' out. signal  0           0           0         0
                CLAMP5       CLAMP6      CLAMP7    CLAMP8
                Not used     Not used    Not used  Not used
'ON' out. signal  24           24          24        24
'OFF' out. signal  0           0           0         0
Clamp number (1~8, ENTER only: No change, CTRL+C:Exit)
```

Figure 4-48 HSETCLAMP Command

4.7.17. DEFSIG COMMAND

`DEFSIG` is used to display and set dedicated signals to specific conditions. The

AS LANGUAGE COMMANDS

DEFSIG command without an argument will display a list of dedicated signals and their associated conditions. It can be entered alone or with the optional arguments `INPUT` and `OUTPUT`. Enterling `DEFSIG` withiout these arguments will display a list of all dedicated signals and their associated conditions, including one of the arguments will display dedicated signals only of the indicated type. The input and output arguments witch may be shortened to simply `1` and `0` respectively, also allow the user to change dedicated signal numbers or assign remove signals according to specific conditions.

AS LANGUAGE COMMANDS

OUTPUTS	INPUTS
MOTOR ON	EXT.MOTOR ON
ERROR	EXT.ERROR RESET
AUTOMATIC CONDITION RUN IN PANEL SWITCH EXT_IT not set to hold Repeat in panel switch Repeat continuous Step continuous TEACH LOCK OFF CYCLE START ON RGSO on Dryrun mode off	EXT.CYCLE START
	EXT.PROGRAM RESET (EX.RST)
	EXT.program select (Jump) Jump_ON Jump_OFF Jump_ST
	EXT_IT (External Hold)
CYCLE START	EDT.program.select (RPS) RPS_ON RPS_ST Number of RPS Signals First signal number code (0:Binary 1:BCD)
TEACH MODE	
HOME1	
HOME2	
POWER ON	EXT.SLOW REPEAT MODE
RGSO	
Ext.Prog select (RPS) enabled	

Table 4-2 Dedicated Signal Conditions

The following shortcuts may also be used

L: Last (previous) page

N: Next page

Q: Cancel (Does *not* save data)

E: Save data and exit

NOTE

The signals for Jump_ON, Jump_OFF, and Jump_ST are all defined as a group when EXIT. programs select (Jump) is set. The signals for RPS_ON and RPS_ST are all defined as a group when EXIT. program select (RPS) is set. How-ever, Jump_ST and RPS_ST are both output signal but are defined with their associated signals from the DEFSIG

```
$DEFSIG ↵
Dedicated signals set at present
EXT.MOTOR CN=1032
EXT.ERRORRESET=1031
EXT.CYCLE START=1030
MOTOR ON=32
ERROR=31
AUTOMATIC=30
  Condition : Run in panel switch,
  Condition : REPEAT in panel
  switch,
  Condition : Repeat continuous.
  Condition : Step continuous.
CYCLE START=29
TEACH MODE=28
HOME 1=27
```

Figure 4-49 DEFSIG Command

Figure 4-49 shows the screen displayed when the DEFSIG command is entered. Output signal numbers range from 1 to 32, while input numbers range from 1001 to 1032.

Figure 4-50 shows an example of the `DEFSIG output` command. Entering `1` after each menu choice will either set or cancel dedicated signal conditions. The processor will provide a default signal number when dedicated signals are set. This number can be changed by entering the desired signal number and pressing the ENTER. If the number provided by the processor is acceptable, press ENTER again. Attempting to assign the same number to two different, dedicated signals will result in an error.

Dedicated signals cannot be used as general purpose signals within programs. Press `E` followed by ENTER to save data and exit to monitor mode.

```
$DEFSIG OUTPUT↵
MOTOR ON Dedication cancel? (Enter 1 to set)1 ↵
  Signal number 32 Change? (1-32) ↵
ERROR Dedication cancel? (Enter 1 to cancel) ↵
  Signal number 31 Change? (1-32) ↵?
CYCLE START Dedication cancel? (Enter 1 to cancel) 1↵
AUTOMATIC Dedication cancel? (Enter 1 to cancel) e ↵
$
```

Figure 4-50 DEFSIG Output Command

4.7.18. ZZERO COMMANDS

The ZZERO command is used to display and set robot zeroing information. The format for this command is: ZZERO *10_ joint number*. When maintenance is performed that results in a serve motor being changed, or the robot being moved from the position identified by the encoder. (Referenced to a 0° location), the encoders reference count is altered and zeroing is required.

⚠ CAUTION

Changing this data without correctly following the complete zeroing process will move programmed locations to posi-tions that may cause damage to the robot, fixtures, or

The ZZERO command provides the user with the option of selecting either a single joint or *all* joints to reset, the encoder is revolution counter. If a single joint is selected, a set degree angle for that axis may also be entered. Once data from the zeroing process has been entered, encoder revolution count(s) are calibrated by the 1GA board and referenced to a 0° position to provide accurate robot motion. Detailed information regarding the zeroing process is provided in Kawasaki's *C Series Controller, Electrical Maintenance and Troubleshooting Manual*.

Entering ZZERO without an optional argument displays counts and offsets of the current 0° encoder. To set the current encoder count offset to a 0° midpoint reference, enter ZZERO followed by the number of the joint to reset +100. Figure 4-52 shows an example of this with the command ZZERO 102 which will affect joint (axis). Entering

`ZZERO 100` will simultaneously reset all joints to 0°.

Figure 4-51 shows a screen of the ZZERO command in use.

```
$ZZERO ↵
            JT1        JT2         JT3         JT4         JT5         JT6
Set data 268435456  268435456   268435456   268435456  268435456  268435456
Cur data 268435456  268435456   268435456   268435456  268435456  268435456
CHANGE? (if not, hit RETURN only) ↵
            JT1        JT2         JT3         JT4         JT5         JT6
OFFSET       26       65534         63          31       35532       65510
CHANGE? (if not, hit RETURN only) ↵
```

Figure 4-51 ZZERO Command

Zeroing of a single axis is performed by entering `ZZERO` *`axis number`*. The command for zeroing all axes is `ZZERO 0`.
The zeroing procedure is carried out as follows.

1) Jog the robot to the position where scribe marks for the joint(s) to be zeroed are properly aligned.
2) Set current offset of the encoder counter to 0° using the ZZERO command. Enter `ZZERO 100` to zero all joints, or `ZZERO` *`joint number +100`* to zero a single joint.

If, in step one, the robot cannot be moved because its current position is recognized as out of range:

1. Zero the robot where it is (to allow jogging).
2. Move the robot to the position where scribe marks for the joint(s) to be zeroed are properly aligned.
3. Execute zeroing as above.

Figure 4-52 shows two examples: `ZZERO 100` for all joints, and `ZZERO 102` for joint 2.

```
$ZZERO 100 ↵
**Encoder rot. Counter reset (al joints)**
Are you sure (Enter 1 to execute)? 1
Setting completed.
$

$ZZERO 102 ↵
**Encoder rot. Counter reset (2th axis)**
Current angle (deg.mm)? 0
Are you sure? (Enter 1 to execute)? 1
Setting completer.
$
```

Figure 4-52 ZZERO 100 Command

After resetting the revolution counter, use the ZZERO _ command to set joint(s) to 0°. Figure 4-53 shows two examples: `ZZERO 0` for all joints and `ZZERO 2` for joint 2.

```
$ZZERO 0 ↵
            JT1        JT2         JT3         JT4        JT5        JT6
Set data 268435456  268435456   268435456   268435456  268435456  268435456
Cur data 268435456  268435456   268435456   268435456  268435456  268435456
Set furrent values of al joints as zeroing data? (Enter 1 to set) 1 ↵
Setting completed.
```

Figure 4-53 ZZERO 0 Command

4.7.19. EREST COMMAND

Enter `EREST` to reset a current error condition within the controller. This command performs the same function as the ERROR RESET button on the controller cabinet. An error condition that occurs continuously and is not rectified cannot be reset with the EREST command or the ERROR RESET button.

4.7.20. SYSINIT COMMAND

The SYSINIT command is used to clear all information from system memory and reset all default settings. The only information not affected by the initialization process is the error and operations logs.

⚠ CAUTION

Initializing the system will erase all program and variable data from system memory, and reset system data to default settings. If programs are to be reloaded from a diskette or PC card, care must be taken to ensure that system settings are configured correctly or programs will not function as expected (if at all). Pay extra attention when initializing a controller connected to a PC, lest the *PC* be initialized by

Entering SYSINIT command will return the prompt "`Are you sure? (1:yes, 0:no)`". Press 1 to begin the initialization process.

4.7.21. HELP COMMANDS

The HELP group of commands is used to display AS language commands available to the programmer. These commands are displayed in alphabetic order in seven columns. Entering `HELP` will display all available commands. If more specific information is required, the HELP command can be used in conjunction with the following prefixes:

HELP/M : Monitor commands
HELP/P : Program instruction commands
HELP/PPC : PC program commands
HELP/MC : Monitor commands for program control
HELP/DO : Monitor commands used in conjunction with `DO`
Enter *`Letter`* `HELP` to display all commands beginning with that letter, followed by the format for each command.

Figure 4-54 shows an example of the HELP/M and HELP D commands.

```
$HELP/M ↵
ABORT       BASE        BITS      BATCHK    CONTINUE    COPY      DEFSIG
DELETE      DIRECTORY   DLYSIG    DO        EDIT        ERESET    ERRLOG
$HELP D ↵
DECEL           DECEL deceleration ALWAYS
DECOMPOSE             DECOMPOSE array variable [suffix] = location
DEFSIG          DEFSIG INPUT or OUTPUT
DELAY DELAY time
DELETE          DELETE/P/L/R/S program or variable
directory             DIRECTORY
DLYSIG          DLYSIG signal number, time
DO              DO instruction
DO              DO ... UNTIL condition
DRAW            DRAW dx, dy, dz, rx, ry, rz, speed
DRIVE           DRIVE joint number, angle, speed
Press NEXT PG key to continue.
```

Figure 4-54 HELP Commands

4.7.22. ID COMMAND

The **ID** command displays identification information about the robot system, including the current software version . Figure 4-55 shows an example of information available with the ID command.

```
$ID ↵
Robot name         : JS005-E001Num of axes  7 Serial No.1
Software version   : version  000004-04 .....97/01/21 13:11
Servo              : SAOA00-UX120-01
Number of signals : output = 32     Input = 32  Internal = 256
Number of Clamp    : 2        Motor type : 1     Servo type : 1
```

Figure 4-55 ID Command

4.7.23. BATCHK COMMAND

The **BATCHK** command is used to enable or disable the “battery low” check when the controller power is ON. The batteries are located on the 1FX board, and provide power for SRAM memory backup of data stored on the 1GA main CPU board. Entering `BATCHK` will display the prompt `"0:Ineffective, 1:Effective"`. Choosing 0 will disable battery check; 1 will check battery voltage automatically during power up.

4.7.24. ENCCHK_EMG COMMAND

The ENCCHK_EMG command sets the scope of the positional diagnostic performed after rebooting from an emergency stop. The diagnostic checks the robot’s current position (after rebooting) with its position at the time of the emergency stop. If the difference between the robot’s current position and its position at the time of the emergency stop greater then the assigned value, a position offset error will be result The position offset error generated from this function cannot be reset, and motor power cannot be applied. The error range must be reset to a value that will not cause an error. The purpose of this is to prevent interference with fixtures, jigs, or work pieces when the robot is rebooted from an emergency stop.

The acceptable range of data for ENCCHK_EMG 0.1 to 10.0 degrees for axes one to six, and 0.1mm to 100mm for the seventh axis. The default setting for ENCCHK_EMG is 0, which will not perform an error check.

Figure 4-56 shows the ENCCHK_EMG in use.

```
$ENCCHK_EMG ↵
          JT1      JT2      JT3      JT4      JT5      JT6
        0.0deg   0.0deg   0.0deg   0.0deg   0.0deg   0.0deg
Change? (If not, hit return only) ↵
$
```

Figure 4-56 ENCCHK_EMG Command

4.7.25. ENCCHK_PON COMMAND

The ENCCHK_PON command is used to set the range of acceptable encoder deviation when power is supplied to the controller. The encoder value when control power is turned off is compared to the encoder value when control power is turned on. A JT encoder abnormality will result if the difference strays beyond set values. The range of acceptable data for this function is 0.1 to 10.0 degrees for axes one through six, and 0.1mm to 100mm for the seventh axis. The default setting for the ENCCHK_PON function is 2.0 degrees.

Operators should be aware that, if this range is set too low, error messages may be displayed even when the system is performing within performance specifications. Figure 4-57 shows an example of the ENCCHK_PON command in use.

```
$ENCCHK_PON ↵
          JT1      JT2      JT3      JT4      JT5      JT6
        2.0deg   2.0deg   2.0deg   2.0deg   2.0deg   2.0deg
Change? (If not, hit return only) ↵
$
```

Figure 4-57 ENCCHK_PON Command

4.7.26. SLOW_REPEAT COMMAND

The SLOW_REPEAT command allows the user to set the speed of the robot from 1 to 25% of maximum speed. A dedicated input signal is required for this function; When the signal is on, the robot will operate at the speed after receiving the SLOW_REPEAT command.

```
$ SLOW_REPEAT ↵
SLOW REPEAT MODE SPEED (1~25%)
(Enter only : no change, ^C : Exit) : NOW 15 CHANGE? ↵
```

Figure 4-58 SLOW_REPEAT Command

4.7.27. REC_ACCEPT COMMAND

The REC_ACCEPT command is used to set the permission status for entering new data. Entering `REC_ACCEPT` will prompt the user to enable or disable the RECORD or PROGRAM CHANGE functions. RECORD allows the user to prevent the recording of blockstep information by selecting `disable`. PROGRAM CHANGE allows the user to prevent the recording of AS language information by selecting. Attempts to change either once `disable` is selected will result in an error message will be displayed.

Figure 4-59 shows the REC_ACCEPT command being entered. In this example, the RECORD function was changed from being disabled to enabled.

```
$REC_ACCEPT ↵
RECORD (0:Enable, 1:Disable)
  (Enter only : No change) : Now 1 Change? 0 ↵
PROGRAM CHANGE  (0:Enable, 1:Disable)
  (Enter only : No change) : Now 1 Change? ↵
$
```

Figure 4-59 REC_ACCEPT Command

4.7.28. ENV_DATA COMMAND

The **ENV_DATA** command is used to set an auto servo timer and identify if the

controller is operating with a teach pendant installed. Entering ENV_DATA will prompt the user to set information for the auto servo OFF timer and TEACH PENDANT.

The auto servo OFF timer sets a time period that motor power will remain on if no robot movement has occurred. This function is designed to save energy by allowing the brakes to maintain robot position as opposed to using electrical power and servo motors. When the robot has not moved and the auto servo timer has reached its set value, brakes will be applied and power removed from the servo motors. The motor power light will remain on, and the robot will begin motion under the same conditions it would have if the auto servo timer had not removed power from the motors.

The env_data command also allows the user to identify whether or not a teach pendant is installed. The deadman buttons and the emergency stop button are hard-wired, and a jumper (or different user interface) must be installed if the teach pendant is removed.

Figure 4-60 shows the ENV_DATA command in use. The auto servo OFF timer is set to 600 seconds, and a TEACH PENDANT is installed.

```
$ENV_DATA ↵
AUTO SERVO OFF TIMER (0:Not servo off)
  (Enter only : No change) : Now 0 Change? 600 ↵
TEACH PENDANT (0:Connect, 1:Disconnect)
  (Enter only : No change) : Now 0 Change? ↵
$
```

Figure 4-60 ENV_DATA Command

4.7.29. ENV2_DATA COMMAND

The **ENV2_DATA** command allows the user to identify if a Multi Function Panel or terminal is installed. The deadman buttons and the emergency stop buttons of the multi function panel are hard-wired, and a jumper (or different user interface) must be installed if the multi function panel is removed. Figure 4-61 shows an example of the ENV2_DATA command in use.

```
$ENV2_DATA ↵
PANEL (0:Connect, 1:disconnect)
  (Enter only : No change) : Now 0 Change? 600 ↵
TEACH PENDANT (0:Connect, 1:Disconnect)
  (Enter only : No change) : Now 0 Change? ↵
$
```

Figure 4-61 ENV2_DATA Command

4.7.30. CHSUM COMMAND

The **CHSUM** command is used to address a checksum (1019) error. A checksum error occurs when data calculated at controller startup time returns an unexpected value. The CHSUM command is used to change the CLEAR CHECK SUM ERROR setting to `EFFECTIVE`. With this setting, the check sum error can be cleared and the setting returned to `INEFFECTIVE` when control power is cycled. If the error does not clear with the cycling of control power, a second message (shown in figure 4-62) will be displayed identifying additional troubleshooting options.

```
$CHSUM ↵
CLEAR CHECKSUM ERROR. (0:Ineffect, 1:Effect)
(Enter only:No change):Now 0 Change? ↵
If error does not clear, enter CHSUM again.
$CHSUM ↵
Cannot clear sum check error. Check the following command or auxiliary data.
ZZERO
DEFSIG
.
.
.
.
.
```

Figure 4-62 CHSUM Command

4.7.31. WEIGHT COMMAND

The weight command is used to set the load weight, which equals the combined mass of hand and work. The format for the command is `WEIGHT` *`weight load , Center of gravity x direction , Center of gravity y direction , Center of gravity`*

z direction. Load weight set by this instruction is used to optimize addition and subtraction speed of the robot arm. Weight settings range from 0.0kg to maximum payload.

When `weight` is entered as a monitor command and without any parameters, a prompt will appear asking `Change current setting?` There is also an option to disable the "set load center of gravity" function.

⚠ CAUTION

The weight command cannot be used while the robot arm is in operation. When using the type 2 (the Z series is standard: all others are optional). Ensure that mass/center of gravity settings are correctly entered. Faillre to do so increases the risk of damage to structural components,

4.7.32. PLCOUT COMMAND (OPTIONAL)

The PLCAOUT command assigns a real number value to the specified data number of whole number (integer) output. The format for this command is **PLCAOUT** `data number of inter output=real value`. The output number has a range of 1~32. The real value assigned to output data number is in decimal notation.

Specification by variable name is also possible, with a range 1~65535.
Note: The internal sequencer function is effective only when option is ON. If option is not set to ON, the following message appears: It becomes `Error (-896) "Because option is set up, cannot execute."`

```
>PLCAOUT 13=120 ↵    "120(decimal)" is set in the 13th integer output data.
```

Figure 4-63 The PLCAOUT command

4.7.33. TPLIGHT COMMAND (OPTIONAL)

The TPLIGHT command illuminates the backlight of the Multi-Function Panel. Entering `TPLIGHT` when the backlight is already on will extinguish the light ten minutes after receiving the command.

4.7.34. MWLOG COMMAND (OPTIONAL)

Displays the mechanical warning log. Entries up to 20 are displayed.

```
MECHANICAL WARNING LOG SINCE  00/12/14  20:45:37
1-[01/03/01 20:57:23(-60) Peak torque is over.  JT1
  program pg2 step 12  78.7[Arms]  187.2[%]]
2-[01/03/01 20:54:55(-60) Peak torque is over.  JT4
  program pg2 step 20  24.8[Arms]  152.3[%]]
```

Figure 4-64 The mechanical warning log

4.7.35. MWCLR COMMAND (OPTIONAL)

The MWCLR command clears all entries to the mechanical warning log.

4.7.36. IPEAKLOG COMMAND (OPTIONAL)

The IPEAKLOG command displays peak current values of all axis motors. These values are followed by information from the time of the peak current, including: program name, step number, current value (Arms), and its ratio to mechanically or electrically limited current (in percent).

4.7.37. IPEAKCLR COMMAND (OPTIONAL)

Clears the peak current value. This is also cleared by the SYSINI command, and by initialization (when the 1GA board's No.8 dip switch is set to ON).

Entering `IPEAKCLR` will display a prompt for the user to press, to proceed or 0 to abort.

4.7.38. OPEINFO COMMAND (OPTIONAL)

The OPEINFO command provides information ON operation time, servo and motor power ON time, and motor/servo ON frequency, as well as emergency stops. The format for the command is **`OPEINFO`** ***`robot number: axis`***. If no robot or axis information is specified, information is displayed for *all* robot and axes.

4.7.39. OPEINFOCLR COMMAND (OPTIONAL)

Clears all information on operation time, resetting to zero.

4.8. BINARY SIGNAL CONTROL COMMANDS

These commands are used to set various output signal parameters, and include instructions for turning signals ON or OFF, resetting output signals, pulsing/delaying signals, and assigning or evaluating a binary value for a set group of signals.

4.8.1. RESET COMMAND

The RESET command sets external output signals to OFF. Users should be fully aware of the effects upon external equipment and peripheral programming before using the reset command.
Note: Reset does not effect dedicated output signals.

4.8.2. SIGNAL COMMAND

The signal command is used to set external output signals and internal status signals to ON or OFF. The format for the command is **`SIGNAL (-)signal number.`** `(signal number).` A signal is turned ON by entering its number following the SIGNAL command: signals are turned OFF by entering "-" before the signal number. Multiple signals are separated by commas. For example, `SIGNAL 2,4,-5` will turn outputs 2 and 4 ON and output 5 OFF.

The signal command cannot be used to turn specific, dedicated signals OFF or ON, nor with external input signals. The signal command's range of signals includes: 1- the number of external output signals the controller is configured for and 2001 - 2256 for internal status signals.

4.8.3. PULSE COMMAND

The PULSE command is used to turn a specific signal ON for a set period of time. The format for the command is `PULSE signal number, time (seconds).` For example, `PULSE 4,5.6` will turn output signal 4 ON for 5.6 seconds. The range of signals affected by the PULSE command include: 1- the number of external output signals the

controller is configured for, and 2001 - 2256 for internal status signals. Signals that have been assigned as dedicated signals for specific conditions cannot be used with the PULSE command.

Signals can only be turned ON with the PULSE command, they cannot be PULSED off. If a time value is not specified, 0.2 seconds is the default setting and signals will be pulsed on for 0.2 seconds.

4.8.4. DLYSIG COMMAND

The DLYSIG command is used to turn a specific signal ON or OFF after a set period of time has elapsed. The format for the command is `DLYSIG` *`signal number, time (seconds).`* For example, `DLYSIG 2,4.5` will turn output 2 ON after 4.5 seconds have elapsed, and `DLYSIG -2,4.5` will turn output 2 OFF after 4.5 seconds have elapsed.

The range of signals be affected by the DLYSIG command includse: 1- the number of external output signals the controller is configured for, and 2001 - 2256 for internal status signals. Signals that have been assigned as dedicated signals for specific conditions cannot be used with the PULSE command.

4.8.5. BITS COMMAND

The BITS command assigns a set value to an external output or internal signal. The format for the command is `BITS` *`starting signal number, number of signals = value`*. For example, entering `BITS 2001,3` will display the binary value of internal signals 2001, 2002, and 2003. In another example, the instruction BITS 1,-8=100 will set the signal states of signals 1 through 8 to a binary value of 100 (binary 1100100, or signal numbers 8,7, and 3, would be ON). If no value is entered, output status of the designated signal will be displayed. The starting signal number is set as the smallest among all output signals. The number of signals is how many signals are to be evaluated (for a maximum 16), beginning with the starting signal number.

The range of signals for use with the bits command includes: 1- the number of external output signals the controller is configured for and 2001 - 2256 for internal status signals. The bits command cannot be used with specific, dedicated signals.

4.8.6. SCNT COMMAND

The SCNT command output a counter signal when a specified counter value is reached. The format for the command is `SCNT counter signal number = count up signal , count down signal, counter clear signal, counter value`. The *counter signal number* Assigns a number to the counter signal to be output (range: 3097 - 3128). count up signal sets the signal number or logical expression. Countup is performed if this signal is changed from OFF to ON. *count down signal* sets the signal number or logical expression. Countdown is performed if that signal is changed from OFF to ON. *counter clear signal* sets the signal number or logical expression. The internal counter returns to 0 when this signal is turned ON. *count value* sets the counter value as a real number. The counter signal is output when this real number value is reached. Setting this value as 0 will turn the counter signal OFF.

When the scnt command/signal is sent, the value of the internal counter is increased by 1 if the count up signal changes from OFF to ON. Similarly, the value of the internal counter decreases by 1 if the countdown signal goes from OFF to ON. The *counter clear signal* resets the internal count value to 0.
Note that the counter is capable of containing several different values. Each for use with its own counter signal.

4.8.7. SCNTRESET COMMAND

The SCNTRESET command is used to the format for the command is **`SCNTRESET counter signal number`**. Force-clear the internal count value corresponding to a designated counter signal number. The range of setting is from 3097 - 3128.

4.8.8. SFLK COMMAND

The SFLK command is used to designate the ON/OFF cycle of the flicker signal. The format for the command is `SFLK` *`flicker signal number = time`*.
The flicker signal number denotes the signal to be flickered. The setting range is 3065 - 3096. Time assigns a real number value to set cycle time. Entering a negative value will discontinue flickering.

4.8.9. SFLP COMMAND

The SFLP command turns output signals ON or OFF by way of the set and reset signals. The format for the command is `SFLP` *`output signal = set signal expression , reset signal expression`*. `output signal` sets the number of the signal to be output. An on signal is output value is positive, OFF if it is negative.

Only signals capable of output can have their number specified.

Note: the number 1000 is reserved and cannot be used.

`set signal expression` : The signal number or the signal logical expression to set the output signal is specified. **reset_signal_expression** : The signal number or the signal logical expression to reset the output signal is specified. If the set

signal is turning on, the output signal is turned on and if the reset signal is turning on, the output signal is turned off. When both the set and reset signals are turning on, the output signal is turned off. However, be to execute the SFLP instruction that the output signal is turned on and off. It is not a moment when the set signal was turned on.

4.8.10. SOUT COMMAND

The format for the SOUT command is: **SOUT signal_number = signal_expression**. After the condition of the signal type consists, the signal is output. **signal_number** : The signal number which wants to output is specified. Only the signal number which can be output can be specified. (The 1000th cannot be specified) **sinal_expression** : The signal number or the signal logical expression is specified. It is an instruction for the logical operation of the signal. When it describes in the signal type with AND and OR, etc. and the condition consists, the signal is output.

4.8.11. STIM COMMAND

The format for the SOUT command is: **STIM timer_signal = input_signal_number , *time***. The timer signal is turned on time's that the input signal was specified it continuing. in case of turning on. **timer_signal** : The signal number turned on when the input signal continues specified time ON is specified with Numeric value. Range of setting : 3001 ~ 3064. **input_signal_number** : The input signal number or the signal logical expression is specified with the integer. Range of setting: Mounted range of signal. ***time*** : Time to observe the state which is turning on the input signal's it continuing is specified with the real value. After the state of time ON specified after turning ON again continues, the timer signal is turned on if it turns OFF in time that the input signal was specified. When the input signal is turned off, the timer signal is turned OFF at once. However, it is time when ON/OFF of the timer signal executed the STIM command. Even if the input signal is turned off, the timer signal is not turned off if the STIM command is not executed.

```
STIM 3001 = 1,5 ↵

SOUT 2 = 3001 ↵
>PCEXECUT ↵
```

When sig1 is turned on, sig2 is turned on five seconds

4.8.12. SETPICK COMMAND

4.8.13. SETPLACE COMMAND

The format for the SETPICK and SETPLACE commands are **SETPICK** ***time 1, time 2, time3 , time4 , ..., time8***. and **SETPLACE** ***time1 , time2 , time3 , time4 , ... , time8***. Sets beginning time (SETPLACE) of (SETPICK)/the open clamping beginning time of close clamping control to make it execute by ordering CLAMP control respectively of clamping 1-8. ***time 1~8*** : The time of the close open/control of clamping 1-8 is specified. (in second) Range of specification : 0.0 ~ 10.0 sec.
Refer to the CLAMP instruction.

4.8.14. PRINT COMMAND

4.8.15. TYPE COMMAND

The PRINT and TYPE commands display information to the system terminal that regards print data saved as parameters. The format for the these commands are `PRINT` *`device number : print data.`* and `TYPE` *`device number : print data`*. The *device number* can be set as `(1)` for standard PC terminal or `2` for the Multi-Function Panel. The device currently in use will be chosen by default if no selection is made. ***Print data*** may be of the following types:

(1) String expressions, such as `count=`
(2) Real value expressions. (The calculated value) such as `count`
(3) Format specifications. (Multiple types may be entered if divided by commas the format of the output message), such as `/D, /S`

A blank line is output if no parameter is specified.

The following specification is used to control the format of numeric value. This setting remains in effect for subsequent parameters until another format entered. Whichever the format should a numerical value so large that the designated sum cannot be displayed. An asterisk appears in the area of the screen where the value should appear. The keyboard screen comes up automatically when the Multi-Function Panel is entered as the device number (2). The previous screen can be returned to by pressing the next menu key will continue to run even if the keyboard screen does not come up automatically.

AS LANGUAGE COMMANDS

Be sure to enter 2 for the device number only if the Multi-Function Panel is connected at the time of designation. Assigning 2 in the absence of a Multi-Function Panel will result in delays of print/type commands as the nonexistent Panel is seduced for.

Note: *No* output will appear if the messages switch is OFF. A line may contain up to 128 characters. Format code /S can be used to extend this limit, if desired.

AS LANGUAGE COMMANDS

/D	Uses the default format. This is the same as /G15.8, except that zeroes following the numerical value are omitted, as are all spaces but one.
/Em.n	Displays the value scientific notation (for example, 1.234E+02) where “m” is all character in the display, and “n” the digits. (The value of “m” should be larger than “n” by 6 or more and smaller than 32).
/Fm.n	Displays the value in fixed point notation (for example, -1.234) where “m” represents all character in the display, and “n” the digits.
/Gm.n	If values are 0.01 or greater within the m portion, and if display of decimals n or smaller is possible in f format, values will be displayed in f. Otherwise, values are display in Em.n.
/H.n	Displays the value as a hexadecimal number in an “n” digit field.
/I.n	Displays the value as a decimal number in an “n” digit field.

The following formats are used to insert special characters between character strings:

/Cn	Output carriage return (CR) and line feed (LF) “n” numbers of times. If this format is the first or last argument in the PRINT instruction, “n” number of blank lines is output to the terminal, otherwise output is “n” -1.
/S	This format suppresses CR and LF at the end of a message.
/Xn	“n” number of spaces are output.
/Jn (optional)	Though almost the same as Hm, the numerical value of the hexadecimal number is displayed in the area of character n. 0 is embedded in a blank part.
/Kn (optional)	Though almost the same as Jm, the numerical value is displayed in the area of character n as a decimal number. 0 is embedded in a blank part.
/L (optional)	Though output is almost the same as /D/L removes all blanks spaces before the numerical value, whereas /D leaves one remaining.

Example

```
PRINT "Point" i,"= ",/F5.2, point[i]
```

Assuming that the real variable “i” has a value of 5 and, that array element “point [5]” has a value of value 12.66666, entering the following.

```
point 5 = 12.67
```

Will produce this output.

```
point 5 = *****
```

If the value of “point [5]” is 1000, when the above instruction is executed, the following message is displayed : asterisks are displayed because the value is too large to be displayed in the given field.

```
>PRINT "ABC"
>PRINT/S, "DEF"
>PRINT "GHI"
```

If you execute the instruction above, “GHI” does not change line and it is displayed on the screen as follows.

```
| ABC
| DEFGHI
|
```

4.8.16. IFPWPRINT COMMAND

The format for the IFPWPRINT command is **IFPWPRINT Window No. , *Display line , Display digit , Background color , Character color = “Character”, “Display Character” ...*** .

The character string specified for the character displaying window set by “Auxiliary 131 INTERFACE PANEL” is appeared.

Window No. : No. in the window of displaying the character of appeared “Auxiliary 131 INTERFACE PANEL” is specified. Range of setting : 1 - 4. ***Display line*** : The line to which the displaying character in the specified window is specified. Range of setting. : 1 - 4. Assume to the 1 when omitting. ***Display digit*** : The digit to which the displaying character in the specified window is specified. Range of setting. : 1 - 40. Assume to the 1 when omitting. ***Background color*** : The background color in the displaying window is specified. Range of setting. : 0 - 9

The background color becomes white when omitting. ***Character color*** : The color of the displayed character is specified. Range of setting. : 0 - 9. The character color becomes

black when omitting. **Character** : The displayed character string is specified. The character string since piece second is appeared from a specified digit of the next line. The character string since piece second can be omitted.
The IFPWPRINT command can be used only at time in the state that the interface panel can be used. “Character” is one instruction execution and a valid in all in the frame in a specified window. (The parts other than appearing character by one instruction execution are cleared.)
When the omittable parameter is omitted, the executed value is succeeded to window No. last time. When protruding beyond the left end of the frame which the displaying character secures, character strings changes line, and display. (Even the specified displaying digit does the indent.) Moreover, when protruding under the frame, character strings rounds down and display. When the control code is included in the character strings, its replaces with space and display.

AS LANGUAGE COMMANDS


MEMO

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

5.0. PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

AS language program instructions are editor commands that are used within a specified program. AS language program instructions include commands for the control of robot motion, speed, accuracy, clamps, and binary signals. AS language program instructions also include commands that are used to control program execution conditions and provide a flow structure based on the evaluation of conditional parameters.

5.1. MOTION CONTROL COMMANDS

Motion control commands are program instructions that determine how the robot will move to programmed locations. Motion control commands include commands that determine the type of interpolation used for motion, location approach and depart instructions, and commands that move the robot a specified distance.

5.1.1. JMOVE COMMAND

The format for the JMOVE command is **`JMOVE location, clamp number`**. JMOVE means joint interpolated movement, the abbreviation for JMOVE is JM. The command JMOVE specifies the type of path will be taken by the robot to reach a taught position. When the robot performs a JMOVE, each joint moves independently to drive the TCP to the required location. The resultant path of a JMOVE command is an arching path. A JMOVE will allow the robot to move between points at a faster speed than a linear interpolated move.

The *location* specifies the destination of the JMOVE. The location specified must be the name of a location variable and can be a precision point, transformation, or compound transformation location. Examples of the JMOVE command for these types of locations:

JMOVE #pic

JMOVE start
JMOVE start + place

The optional argument *clamp number* specifies the clamp number to be opened or closed when the robot reaches the destination location. To open a clamp, type a "," after the *location* and specify the *clamp number*. To close a clamp, type a "," after the *location* and type a "-" before the *clamp number*. If the *clamp number* is omitted, the clamp signal will not change. Examples of the JMOVE command with clamp instructions:

JMOVE set,2
JMOVE pick,-2.

5.1.2. LMOVE COMMAND

The format for the LMOVE command is **`LMOVE location, clamp number`**. LMOVE means linear interpolated movement, the abbreviation for LMOVE is LM. The command LMOVE specifies the type of path the robot will take to reach a taught position. When the robot performs a LMOVE, each joint moves in a coordinated process to drive the TCP in a straight line. The resultant path of a LMOVE command is that the TCP will move in a straight line from the current position to the specified *location*.

The *location* specifies the destination of the LMOVE. The *location* specified must be the name of a location variable and can be a precision point, transformation, or compound transformation location. Examples of the LMOVE command for these types of locations:

LMOVE #pick
LMOVE start
LMOVE start + place.

The optional argument *clamp number* specifies the clamp number to be opened or closed when the robot reaches the destination location. To open a clamp, type a "," after *the location* and specify the *clamp number*. To close a clamp, type a "," after the *location* and type a "-" before the *clamp number*. If the *clamp number* is omitted, the clamp signal will not change. Examples of the LMOVE command with clamp instructions:

LMOVE set,2
LMOVE pick,-2

5.1.3. JAPPRO COMMAND

The format for the JAPPRO command is **`JAPPRO`** ***`location, distance`***. JAPPRO means joint approach, the abbreviation for the JAPPRO command is JA. The command JAPPRO instructs the robot to perform a joint interpolated move that places the TCP at a specified *distance* from the *location*. The *distance* is specified in mm and is in the direction of the Z axis of the tool coordinate system. A positive *distance* will place the TCP at a position that is approaching (back from) the *location* in the direction of the Z tool axis. A negative *distance* will place the TCP at a position that is beyond (forward of) the *location* in the direction of the Z tool axis. Figure 5-1 shows the orientation of the Z tool axis with no tool defined.

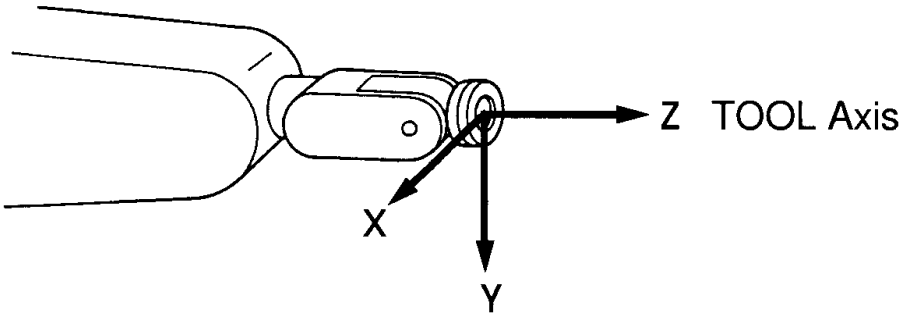


Figure 5-1 Z Tool Axis

Figure 5-2 shows two examples of the JAPPRO command. In the first example the TCP is defined as being the tip of the tool, in the second example the TCP is at the tool mounting flange (tool null). In both examples, the JAPPRO *spot,200* program instruction will cause the TCP to be moved to a position *200 mm* from the location *spot* in the Z Tool direction.

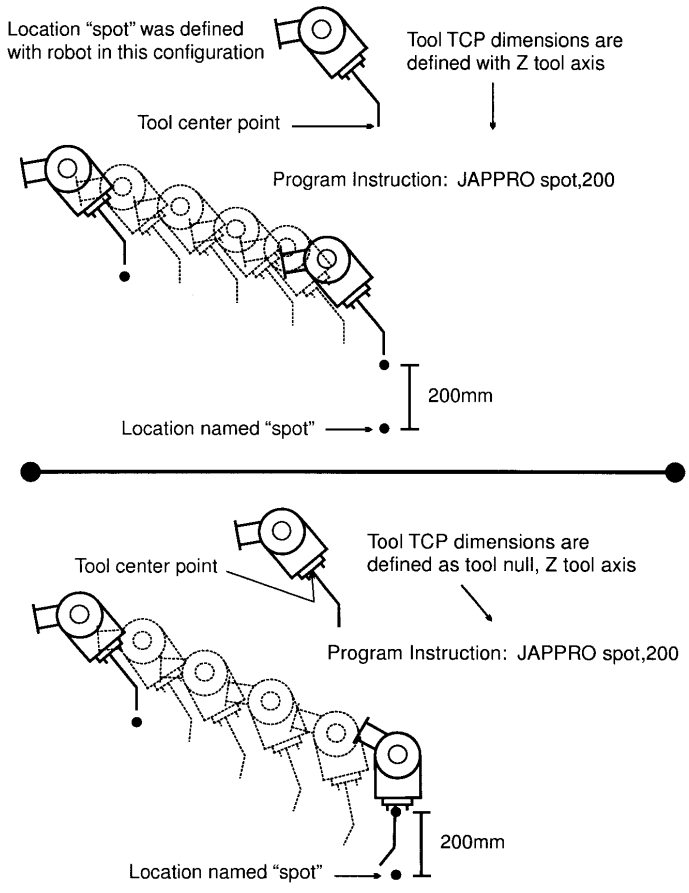


Figure 5-2 JAPPRO Command

5.1.4. LAPPRO COMMAND

The format for the LAPPRO command is **`LAPPRO`** ***`location, distance`***. LAPPRO means linear approach, the abbreviation for the LAPPRO command is LA. The command LAPPRO instructs the robot to perform a linear interpolated move that places the TCP at a specified *distance* from the *location*. The *distance* is specified in mm and is in the direction of the Z axis of the tool coordinate system. A positive distance will place the TCP at a position that is approaching (back form) the location in the direction of the Z tool axis. A negative *distance* will place the TCP at a position that is beyond (forward of) the *location* in the direction of the Z tool axis. The LAPPRO command is similar to the JAPPRO command, the only difference is the type of motion the robot will perform to reach the desired location.

5.1.5. JDEPART COMMAND

The format for the JDEPART command is **`JDEPART`** ***`distance`***. JDEPART means joint depart, the abbreviation for the JDEPART command is JD. The command JDEPART instructs the robot to perform a joint interpolated motion that moves the TCP a specified *distance* from the last taught location. The location is not specified with the JDEPART command because the robot is already at the depart location. The *distance* is specified in mm and is in the direction of the Z axis of the tool coordinate system. A positive *distance* will place the TCP at a position that is pulled back from the last taught location in the direction of the Z tool axis. A negative *distance* will place the TCP at a position that is beyond (pushed forward of) the last taught location in the direction of the Z tool axis. In the example shown in figure 5-3, the program instruction JDEPART 200 moves the TCP *200 mm* in the direction of the Z tool axis.

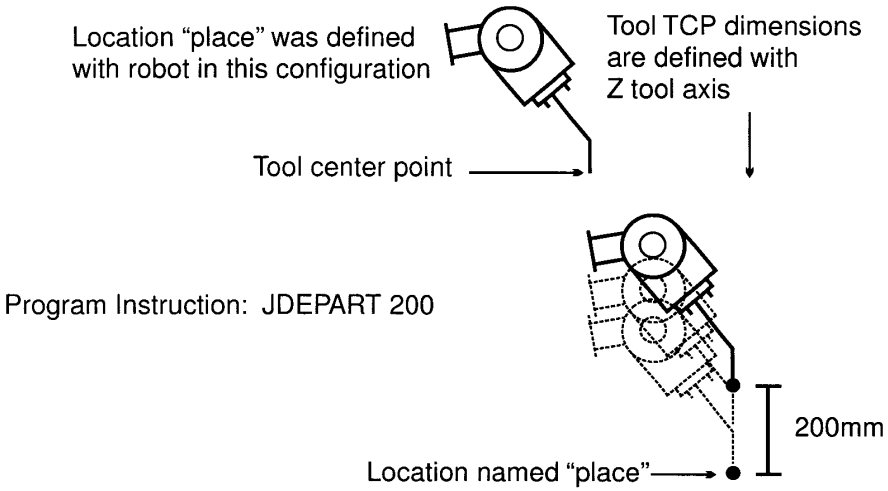


Figure 5-3 JDEPART Command

5.1.6. LDEPART COMMAND

The format for the LDEPART command is **LDEPART *distance***. LDEPART means linear depart, the abbreviation for the LDEPART command is LD. The command LDEPART instructs the robot to perform a linear interpolated motion that moves the TCP a specified *distance* from the last taught location. The location is not specified with the JDEPART command because the robot is already at the depart location. The *distance* is specified in mm and is in the direction of the Z axis of the tool coordinate system. A positive *distance* will place the TCP at a position that is pulled back from the last taught location in the direction of the Z tool axis. A negative *distance* will place the TCP at a position that is beyond (pushed forward of) the last taught location in the direction of the Z tool axis.

5.1.7. HOME AND HOME2 COMMANDS

The program instruction **HOME** or **HOME2** will cause the robot to move to the previously defined HOME or HOME2 position with a joint interpolated move. The HOME and HOME2 positions are defined with the SETHOME and SET2HOME commands. The SETHOME and SET2HOME commands are covered in unit 4 of this manual. To send the robot to the HOME or HOME2 position with a monitor command, the DO HOME or DO HOME2 commands are entered at the $ prompt.

5.1.8. DRIVE COMMAND

The format for the DRIVE command is **DRIVE *joint number, change amount, speed***.

The DRIVE command is used to move a single *joint* of the robot a specified number of degrees at a desired speed. The DRIVE command will cause the robot to move from its current position, not a zero reference. The *joint number* (1 to 6) is the joint of the robot that is to be moved. The *change amount* is the number of degrees that the specified joint is to be moved. The argument *speed* specifies the percent of the repeat condition speed the command will be performed at. If the optional argument *speed* is omitted, the *speed* will default to 100%. In the example, the DRIVE command will cause *joint 3* to move *-31*° at *75%* of the repeat condition speed.

DRIVE 3,31,75

5.1.9. DRAW COMMAND

The format for the DRAW command is ***DRAW X distance, Y distance, Z distance, X rotation, Y rotation, and Z rotation, speed***. The DRAW command is used to move the robot in a linear interpolated motion a specified *distance* (in mm) in each direction of the base coordinate system. The DRAW command will cause the robot to move the specified distance from the current position, not a zero reference. If a *distance* is omitted from the argument, a zero will be assumed. The *X, Y*, and *Z rotation* is the number of degrees the tool rotates about the specified axis. The range of rotation for these arguments is + or - 180°. The argument *speed* can be specified as a percentage of the program repeat speed or to an absolute value specified in mm/sec. If the optional argument *speed* is omitted, the speed will default to 100%. In the example below, the DRAW command will cause the robot TCP to move in a straight line from its current position to a location 324 mm in the X direction and 277 mm in the Z direction of the base coordinate system. The second "," is entered as a delimiter to indicate that there is no Y *distance* in the instruction.

DRAW 324,,277

5.1.10. TDRAW COMMAND

The format for the TDRAW command is ***TDRAW X distance, Y distance, Z distance, X rotation, Y rotation, and Z rotation, speed***. The TDRAW command is used to move the robot in a linear interpolated motion a specified *distance* (in mm) in each direction of the tool coordinate system. The TDRAW command will cause the robot to move the specified distance from the current position, not a zero reference. If a *distance* is omitted from the argument, a zero will be assumed. The *X, Y*, and *Z rotation* is the number of degrees the tool rotation about the specified axis. The range of rotation for these arguments is + or - 180°. The argument *speed* can be specified as a percentage of the program repeat speed or to an absolute value specified in mm/sec. If the optional argument speed is omitted, the speed will default to 100%. In the example below, the TDRAW command will cause the robot TCP to move in a straight line from its current position to a location *714* mm in the X direction, *-57* mm in the Z direction, and rotated *45*° around the Z axis of the tool coordinate system. Commas are entered as delimiters to indicate no data is to be evaluated for that section of the command format.

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

TDRAW 714,,-57,,,45

5.1.11. ALIGN COMMAND

The **ALIGN** command is used to move the robot so that the Z tool axis is parallel to the closest axis of the base coordinate system. The ALIGN command is useful in setting the tool direction so that it is parallel and perpendicular to the base coordinate system before teaching a series of locations.

5.1.12. XMOVE COMMAND

The XMOVE command is used to begin a linear interpolated motion toward a specified location. The format for the command is `XMOVE` *`mode location name`* `TILL` *`signal number`*. If at any time during the robot's move to this location, the specified signal number being monitored is turned on, the robot will change course and proceed to the next location. A negative value entered with *signal number* indicates that an off state of the signal will satisfy the TILL condition.

If parameter mode is omitted, the designated signal is observed by rising (of falling) detection.
Designating /ERR (optional) to parameter will display an error message if , when signal observation begins. There are pre-existing signal conditions.
Designating /LVL (optional) to parameter will when signal observation begins, execute immediate, procedures for interrupt.

NOTE : Interrupt occurs when attempting to use rising or falling detection not because of the signal itself, but because of the change in status. Accordingly, in order to observe OFF-ON signal changes if the signal is already on when an XMOVE instruction is executed, interrupt will not occur until that signal turns off and then on again (assuming that signal number information is positive). Note that a signal must remain constant for at least so mill seconds to accurately detect signal changes.

Figure 5-4 shows an example of the XMOVE command. The robot moves towards location *bottom* until signal *1001* turns ON, at which time the robot changes direction and starts toward location *stack*.

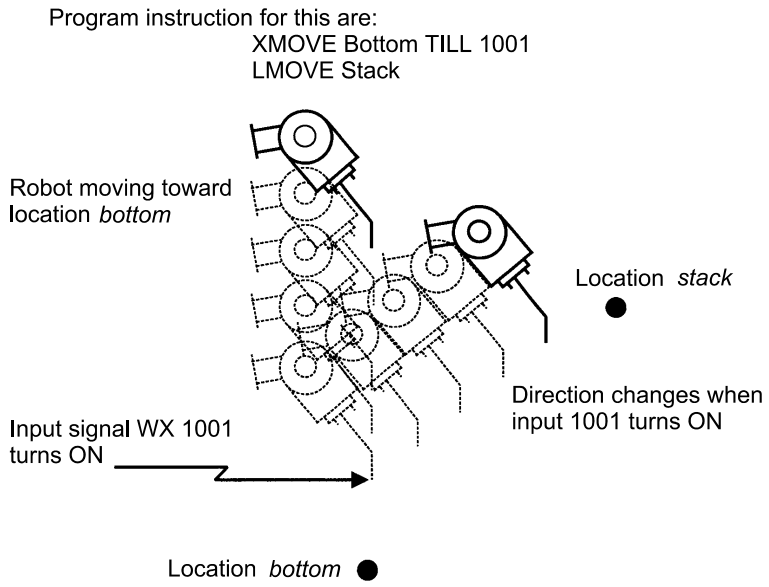


Figure 5-4 XMOVE Command

5.1.13. HMOVE COMMAND

The format for the HMOVE command is **HMOVE *location, clamp number***. HMOVE is an abbreviation for hybrid interpolated move. The command HMOVE is used when the robot configuration is such that straight line moves cause ambiguity errors. This condition is referred to as “singularity”. Singularity is most likely to occur when the arm is “lined up” and the rotation of joint 4 or joint 6 will both result in the same movement of the TCP. The HMOVE command will produce a linear path that will configure the robot in such a way that singularity is avoided.

The *location* specifies the destination of the HMOVE. The *location* specified must be the name of a location variable and can be a precision point, transformation, or compound transformation location. Examples of the HMOVE command for these types of locations are:

HMOVE #pic
HMOVE start
HMOVE start + place

The optional argument *clamp number* specifies the clamp number to be opened or closed when the robot reaches the destination location. To open a clamp, type a “,” after the location and specify the *clamp number*. To close a clamp, type a “,” after the

location and type a "-" before the *clamp number*. If the *clamp number* is omitted, the clamp signal will not change. Examples of the HMOVE command with clamp instructions are:

HMOVE set,2
HMOVE pick,-2

5.1.14. DELAY COMMAND

The format for the DELAY command is **DELAY *time***. The DELAY command causes the robot to stop its motion for a specified period of *time* when the TCP reaches the accuracy range of the last motion step. When the program instruction DELAY is encountered, the robot will not begin movement to the next location until the specified *time* has elapsed. If the program contains steps that do not cause robot motion between the DELAY command and the next motion instruction, those instructions will be processed even though the DELAY has not elapsed.

An example of the DELAY command is shown in figure 5-5, robot motion will stop for 5.5 seconds when the TCP reaches the 100 mm accuracy range of location "aa", program instructions 4, 5, and 6 will be executed while the robot is in the DELAY period. After the 5.5 seconds have elapsed, the robot will move to location "bb".

```
$LIST/P test
.PROGRAM test()
1  ACCURACY 100 ALWAYS
2  JMOVE aa
3  DELAY 5.5
4  SIGNAL 2,4,6
5  SIGNAL -3,-5,-7
6  stack = 126
```

Figure 5-5 DELAY Command

5.1.15. STABLE COMMAND

The format for the STABLE command is **STABLE *time***. The STABLE command causes the robot to stop its motion for a specified period of *time* when the TCP reaches the exact position of the previous location, regardless of the accuracy range. When the program instruction STABLE is encountered, the robot will not begin movement to the next location until the specified *time* has elapsed. While the STABLE time period is in effect, no program instructions will be processed.

An example of the STABLE command is shown in figure 5-6, robot motion will stop for 5.5 seconds when the TCP reaches the exact location aa, program instructions 4, 5, and 6 will not be executed while the robot is in the STABLE period. After the 5.5 seconds have elapsed, program execution will resume at step 4.

```
$LIST/P test ↵
.PROGRAM test()
1  ACCURACY 100 ALWAYS
2  JMOVE aa
3  STABLE 5.5
4  SIGNAL 2,4,6
5  SIGNAL -3,-5,-7
6  stack=126
7  LMOVE bb
```

Figure 5-6 STABLE Command

5.1.16. C1MOVE COMMAND

5.1.17. C2MOVE COMMAND

The format for the STABLE commands are **C1MOVE location, *clamp_number*** and **C2MOVE location, *clamp_number***.
Start the circular interpolated motion to the specified location.
location : Location to which the robot moves, in the form of transformation/precision point variable, location function or compound transformation. ***clamp_number*** : Number of clamp (1 to 8 maximum) to be opened (or closed) when the robot reaches the destination. A positive number causes the clamp to open, and a negative number causes the clamp to close. The maximum clamp number which can be specified is that set by the SETCLAMP command. If omitted, no clamp signal changes.
The C1MOVE instruction is used with the location of the arc, while The C2MOVE instruction is used with the location at the arc end. Three locations are needed to determine the circular trajectory. These key locations are selected as follows:

C1MOVE: (1) location specified in the previous motion instruction
(2) location given in the C1MOVE instruction
(3) location specified in the next C1MOVE or C2MOVE instruction
C2MOVE: (1) location specified in the previous C1MOVE instruction
(2) location specified in the motion instruction before C1MOVE
(3) location given in the C2MOVE instruction

Warning : Before C1MOVE instruction, on of the following motion instructions is needed:
ALIGN, C1MOVE, C2MOVE, DELAY, DRAW, TDRAW,
DRIVE, HOME, JMOVE, JAPPRO, JDEPART, LMOVE,
LAPPRO, LDEPART, STABLE, XMOVE

Next motion to the C1MOVE must be either C1MOVE or C2MOVE motion.
Previous motion to the C2MOVE must be C1MOVE motion.

5.1.18. UVLMOVE COMMAND

The UVLMOVE command moves the robot to a designated location, at a constant speed and via linear interpolation. The format for this optional is UVLMOVE *location* **name,**

clamp number. The robot's target position is assigned in *location name.* This involves either precision point, Transformation and compound transformation values, or the location function.
Clamp number (1~8) designates the hand to open and close.
A positive value executes closing, a negative value opening. The number of assigned clamps may only be as many as those designated using the HSETCLAMP command. Omitting the parameter does not change the clamping signal. This type of motion moves the tool point of the robot along linear locus.

5.1.19. UVC1MOVE COMMAND

5.1.20. UVC2MOVE COMMAND

The UVC1(2)MOVE commands move the robot to a designated location, maintaining a constant speed while drawing a circular locus. The format for these optional command are `UVC1MOVE` *location*, ***clamp number***. `VC2MOVE` *location*, ***clamp number***.
Location designates target position. (involves either precision point, transformation, and compound transformation values or the location function). *Clamp number* designates (1-8) the hand to open and close. A positive value executes closing, and a negative value opening. The number of assigned clamps may only be as many as those designated using the HSPTCLAMP command. Omitting the parameter does not change the clamping signal.
THE UVC1MOVE instruction is for use when a circle is not yet complete, and UVC2MOVE for a circular end point. Care should be taken when having three teaching point to carry out circularly-interpolated operations at a constant speed, as these points will differ according to whether UVC1MOVE or UVC2MOVE is entered.
The three points for UVC1MOVE deal with the locations of:

(1) The movement instruction immediately previous
(2) The UVC1MOVE instruction
(3) The movement instruction immediately following (commands UVC1MOVE or UVC2MOVE)

While the three points for UVC2MOVE deal with the location of:

(1) The UVC1MOVE instruction immediately previous
(2) The movement instruction before the UVC1MOVE command
(3) The UVC2MOVE instruction.

Caution : Motion commands such as the following are required immediately prior to

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

UVC1MOVE: ALIGN, UVC1MOVE, UVC2MOVE, DELAY, DRAW, TDRAW, DRIVE, HOME, JMOVE, JAPPRO, JDEPART, LMOVE, LAPPRO, LDEPART, STBLE, XMOVE, UVLAPPRO, and UVLDEPART.

Also, use of the UVC1MOVE command requires a second UVC1MOVE command (or UVC2MOVE) to immediately follow. Requires that a UVC1MOVE be immediately previous use of UVC2MOVE.

5.1.21. UVLAPPRO COMMAND

The UVLAPPRO command moves the robot by specified distance from a designated location via linear interpolation at a constant speed. The format for this optional command is `UVLAPPRO` *`location name, distance`*. *Location name* defines the final position, involving transformation value and precision point. *Distance* sets the distance in millimeters, between the designated final position and the actual target position along the tools Z axis. Designating a positive distance, puts the location behind the final position (negative direction along the tool's Z axis), and designating a negative distance puts the location ahead of the final position (positive direction along the tool's Z axis).

In this instruction the tool posture is designated by *location name* and position can be changed by designated distance along the direction of the tools Z axis.

5.1.22. UVLDEPART COMMAND

The UVLDEPART commands moves the robot, at a constant speed and via linear interpolation, by an assigned distance from its' current location. The format for this optional command is `UVLDEPART distance`. *Distance* sets in millimeters, the distance between the robots current position and the target location along the Z axis of the tool coordinate system. Setting a positive distance has tool motion become *pullback* (in a negative direction along the Z axis); A negative distance has movement mefore *pushout* (a positive direction along the Z axis).

5.2. SPEED AND ACCURACY COMMANDS

5.2.1. SPEED COMMAND

The SPEED command is used to set robot velocity; the format for this command is `SPEED` ***`value, rotation speed`*** **`ALWAYS`**. The actual operational speed of the robot is the product of monitor (repeat condition speed) speed and program instruction SPEED. The maximum speed identified in the specifications for each robot model cannot be exceeded.

Parameters for the program instruction `SPEED` *`value`* can be specified as a percentage of maximum speed, an absolute speed, or a timed movement as follows:

- **Percentage**. Enter `SPEED` followed by the percentage *value*. The acceptable range for *value* is from 0.01 to 100.
- **Absolute speed**. Enter `SPEED` followed by the desired velocity in MM/second or MM/minute.
- **Time movement**. Enter `SPEED` plus the desired time value, and followed by `s`.

Note: Absolute speed parameters control tool tip speed, and only affect linearly-interpolated movements. Joint interpolated movements are covertness by the timing of *joint movement*, an dare not affected by absolute speed instructions. When a joint move is executed, the proportion of absolute speed to maximum speed is calculated and used.

Rotation speed (option) designates revolution speed of the tool posture. This is normally recorded as a percentage (range: 0.01 - 100%), but entering the desired DEG/S or DEG/MIN following *value* will allow it to be defined as an absolute speed. If no denomination is specified for *rotation speed*, the percentage scare is automatically adopted; If *rotation speed* is omitted, its value is set at 100%.
Note: If rotation speed is set higher than 100% the value will default to 100%. However, attempting to set rotation speed when the control of orientation angle option is not enabled will result in error.
The argument ALWAYS is optional, if it is omitted, only the program step following the SPEED command is effected by the instruction. If the arguments ALWAYS is specified

after the SPEED command, all program steps will be executed at the specified SPEED until another speed command is encountered in the program. The following are examples of the SPEED command:

SPEED *50*	Sets next motion speed to 50%.
SPEED *75* ALWAYS	Sets the motion speed to 75% (until changed by another SPEED command).
SPEED *20 mm/s*	Sets the tool tip speed to 20 mm per second for the next linear motion command.
SPEED *6000 mm/min*	Sets the tool tip speed to 6000 mm per minute for linear interpolated motion commands (until changed by another SPEED command).
SPEED *8 s*	Sets a target time of eight seconds to complete the next motion step.

5.2.2. ACCURACY COMMAND

The format for the ACCURACY command is **`ACCURACY value ALWAYS`**. The ACCURACY command is used to set the positioning accuracy of robot motion. The *value* is specified in mm, the acceptable ACCURACY range is from *0.5 mm* to *5000 mm*. The acceptable accuracy range is from *0.5 mm* to *5000 mm*. When the robot is in the repeat mode of operation, the playback accuracy is affected by a number of things, including: the distance between the taught points, the settings of the CP switch, and if the step contains timers or other wait conditions.
The argument ALWAYS is optional, if it is omitted, only the program step following the ACCURACY command is effected by the instruction. If the argument ALWAYS is specified after the ACCURACY command, all program steps will executed at the specified ACCURACY until another accuracy command is encountered in the program.

5.2.3. ACCEL COMMAND

The format for the ACCEL command is **`ACCEL value ALWAYS`**. The program instruction ACCEL is used to set the robot motion acceleration velocity. The default setting for the acceleration rate is 100%. The ACCEL command can be used to lengthen the time the robot takes to reach its commanded speed. The value is specified in a percentage, the acceptable *value* range is from *0.01%* to *100%*.

The argument ALWAYS is optional, if it is omitted, only the program step following the ACCEL command is effected by the instruction. If the argument ALWAYS is specified after the ACCEL command, all program steps will executed at the specified ACCEL velocity until another acceleration command is encountered in the program.

5.2.4. DECEL COMMAND

The format for the DECEL command is **DECEL *value* ALWAYS**. The program instruction DECEL is used to set the robot motion deceleration velocity. The default setting for the deceleration rate is 100%. The DECEL command can be used to lengthen the time the robot takes to slow down from its commanded speed. The value is specified in a percentage, the acceptable *value* range is from *0.01%* to *100%.*
The argument ALWAYS is optional, if it is omitted, only the program step following the DECEL command is effected by the instruction. If the argument ALWAYS is specified after the DECEL command, all program steps will executed at the specified DECEL velocity until another deceleration command is encountered in the program.

5.2.5. BREAK COMMAND

The **BREAK** command is used to suspend the execution of the next program step until the current motion step is completed. The BREAK command will cause an interruption of a continuous path motion, and the robot will stop momentarily at the transition point. In the example shown in figure 5-7, the break command is used to ensure the robot reaches the location 500 mm above “abc” before the location “xyz” is defined. If the program did not contain the BREAK instruction, location “xyz” would be defined at some location while the robot was performing the DRAW command.

```
$LIST/P test ↵
.PROGRAM test()
1  ACCURACY 100 ALWAYS
2  JMOVE ab
3  DRAW ,,500
4  BREAK
5  HERE xyz
```

Figure 5-7 BREAK Command

5.2.6. BRAKE COMMAND

The `BRAKE` command is used to stop the current robot motion and transition to the next program step. The BRAKE instruction stops robot motion immediately and program execution skips to the next step.

5.2.7. BSPEED COMMAND

The format for the BSPEED command is **`BSPEED peed`**.
Move speed of the robot (block speed) is set. The operation speed of the robot becomes as follow:

repeat speed * program speed * block speed

speed : Set the robot moving speed. Range of setting : 1~1000 Unit : %. Present BSPEED becomes a valid until the following BSPEED is executed.

The operation speed of the robot is obtained as follow:

repeat speed * program speed * block speed

However, 100% cannot be exceeded as a whole.
It becomes as follow:

100% * 50% * block speed

When assuming the for example, the repeat speed 100%, and the program SPEED 50%. Even if the price which rises 200% is set, it becomes same speed as 200% though SPEED changes according to the value if block speed is 200% or less.

Warning :

(1) If the program is newly executed, block speed reaches initial value (100%). Execution is execution, Multi-Function Panel of the EXECUTE instruction or newly execution of the program after selecting it programming from Small Teach Pendant, etc.
Though it becomes new execution about External Program Selection in the selection with External Program Reset; The selection by RPS and the JUMP signal does not become new execution.

(2) Because it does not operate in program speed of the desire when time when the

program was continued on the way and the program fly and are continued occasionally.
It stops the robot in a for example and an undermentioned program the step three robot motions. When the robot is operated after changing the step to step 25, block speed of step 25 becomes speed of block1.

```
step1   BSPEED   block1      ;An initial value of speed of block 1 has been set.
step2   Joint    speed9   .....
step3   linear   speed9   .....

step12  BSPEED   block2      ;An initial value of speed of block 2 has been set.
step13  Joint    speed9   .....
step14  linear   speed9   .....

step24  BSPEED   block3      ;An initial value of speed of block 3 has been set.
step25  joint    speed9   .....
step26  linear   speed9   .....
```

(3) Though the speed input can be input up to 1000; Move speed repeat speed' program speed' block speed of the robot does not become 100% or more.
For example : When you change speed in the External signal four bits. SPEED can be changed by doing the programming as follows according to the External signal.
a=BITS(The first signal number of External speed selection,4)

```
BSPEED block1[a]                    The program example when speed is
                                    selected from External is shown.
BSPEED block1                       ; Set initial value of block speed.
IF SIG(External speed enable)THEN   ; Judge whether external speed selection
                                    disable or enable.
a=BITS(First signal number of external speed selection,4);
                                    Get the external selection number
IF(a<11)THEN                        ; If number is set 11 or more, set a disable.
                                    And if number is set block 11 [ ] till 15, not
                                    use a number.
BSPEEDblock11[a]                    ; Set the selected black speed.
END
```

```
    END
    JOINT SPEED9.....          ; Robot move the specified black speed.
    JOINT SPEED9.....
```

Real value variable block1 should be set beforehand.

```
    block1=50
    block11[0]=10
    block11[1]=20
    block11[2]=30
    block11[3]=40
```

When the first signal number of the for example and the External SPEED selection is assumed to be 1010, the signal input such as follow:

```
    1010.....OFF
    1011.....ON
    1012.....OFF
    1013.....OFF
```

Real variable a is a=2. Then, block 11[2] is selected and block speed becomes 30%.

5.3. CLAMP CONTROL COMMANDS

Clamp control commands are program instructions that are used to provide various clamp operational tasks. The C series controller can support the control of up to eight different clamps. Auxiliary function 114, Clamp Specifications, is used to set the clamp control parameters.

5.3.1. OPEN COMMAND

The format for the OPEN command is **`OPEN`** ***`clamp_number`***. The OPEN command is used to set the output signal that opens the clamp hand. The open command sets the output signal to open the clamp hand at the beginning of the next robot motion. The *clamp_number* specifies the clamp to be opened, up to eight clamps can be used. If the *clamp_number* is omitted, clamp one is specified.

5.3.2. OPENI COMMAND

The format for the OPENI command is **OPENI *clamp_number***. The OPENI command is used to set the output signal that opens the clamp hand. The OPENI command causes a BREAK to occur if a continuous path is in progress. The output signal to open the clamp hand is set at the completion of the current motion. The *clamp_number* specifies the clamp to be opened, up to eight clamps can be used. If the *clamp_number* is omitted, clamp one is specified. Figure 5-8 shows the differences between the OPEN and OPENI commands.

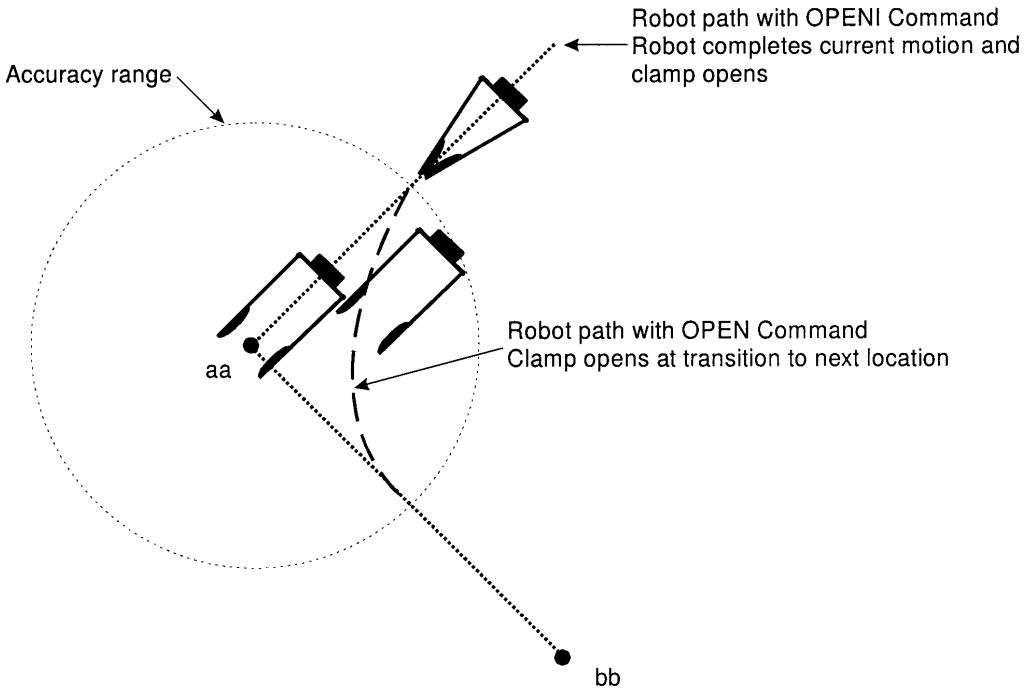


Figure 5-8 OPEN and OPENI Commands

5.3.3. CLOSE COMMAND

The format for the CLOSE command is **CLOSE *clamp number***. The CLOSE command is used to set the output signal that closes the clamp hand. The close command sets the output signal to close the clamp hand at the beginning of the next robot motion. The *clamp number* specifies the clamp to be closed, up to eight clamps can be used. If the *clamp number* is omitted, clamp one is specified.

5.3.4. CLOSEI COMMAND

The format for the CLOSEI command is **`CLOSEI`** ***`clamp number`***. The CLOSEI command is used to set the output signal that closes the clamp hand. The CLOSEI command causes a BREAK to occur if a continuous path is in progress. The output signal to close the clamp hand is set a the completion of the current motion. The *clamp number* specifies the clamp to be closed, up to eight clamps can be used. If the *clamp number* is omitted, clamp one is specified.

5.3.5. RELAX COMMAND

The format for the RELAX command is **RELAX *clamp number***. The RELAX command is used to turn off both sides of a pneumatic valve. The RELAX command sets the output signals to release pneumatic pressure at the clamp hand at the beginning of the next robot motion. The *clamp number* specifies the clamp to be relaxed, up to eight clamps can be used. If the *clamp number* is omitted, clamp one is specified.

5.3.6. RELAXI COMMAND

The format for the RELAXI command is **RELAXI *clamp number***. The RELAXI command is used to turn off both sides of a pneumatic valve. The RELAXI command causes a BREAK to occur if a continuous path is in progress. The output signals to release pneumatic pressure to the clamp hand are set at the completion of the current motion. The *clamp number* specifies the clamp to be relaxed, up to eight clamps can be used. If the *clamp number* is omitted, clamp one is specified.

5.3.7. GUNONTIMER COMMAND

5.3.8. GUNOFFTIMER COMMAND

The format for the GUNONTIMER and GUNOFFTIMER commands are **GUNONTIMER gun_number, time**. and **GUNOFFTIMER gun_number, time**.
Time compensates the gun spray timing (ON/OFF).
gun_number : Set to 1 (gun 1) or 2 (gun 2). **time** : The time of the correction is specified (in seconds).
When quickening, it is negative and when retarding, specifies a positive value.
It is considered 0 seconds when omitting.
Because the time correction is decided by environment data of the cancer system (distance the cancer ahead and kind and temperature etc. from the valve of paints); Set it at the head of the program. Moreover, make time Parameter a variable when you change time outside the program according to paint color and the viscosity, etc.
Only this instruction sets the time of the correction of the exhalation timing and the ON/OFF processing of the cancer is not executed.

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

```
GUNONTIMER 1,-0.5
```

The turning on timing of cancer 1 is made 0.5 seconds earlier.

5.3.9. GUNON COMMAND

5.3.10. GUNOFF COMMAND

The format for the GUNON and GUNOFF commands are **GUNON gun_number, *time***. and **GUNOFF gun_number, *time***. The cancer signal is turned on and off. Moreover, cancer exhalation timing (ON/OFF) is corrected in the distance. **gun number** : Set to 1 (gun 1) or 2 (gun 2). **distance** : The distance of the correction is specified. (The unit is mm)
When quickening, it is negative and when retarding, specifies a positive value.
It is considered 0mm when omitting.
When the following robot motion instruction of the GUNON/GUNOFF instruction is executed, the cancer is turned on and off. As for the cancer exhalation timing, the time set in the distance specified by ordering GUNON/GUNOFF with GUNONTIMER/GUNOFFTIMER set before is added.

```
GUNON 2, 100
```

The turning on timing of cancer 2 is retarded by 100mm and it turns on.

5.3.11. OPENS COMMAND

5.3.12. CLOSES COMMAND

5.3.13. RELAXS COMMAND

The format for the OPENS, CLOSES and RELAXS commands are **OPENSNS *clamp _ number*. , CLOSES *clamp_number***. and **RELAXS *clamp_number***.
The opening and shutting signal of the empty pressure control electromagnetic valve is ON/OFF. **clamp_number** : Clamp_number is specified. It becomes one when omitting.
These instructions are different in the OPEN/CLOSE/RELAX instruction, the OPENI/CLOSEI/RELAXI instruction, and the following point.

OPEN/CLOSE/RELAX instruction

When the following robot motion starts, the signal is output.

OPENI/CLOSEI/RELAXI instruction

When the CP robot motion interrupts in the robot motion the robot (BREAK) and the robot motion is completed, the signal is output.

OPENS/CLOSES/RELAXS instruction

The signal is output at once at the time of execution of this instruction.

These instructions are not influenced by the reading prohibition (PREFETCH.SIGINS switch) the signal instruction ahead.

5.4. CONFIGURATION INSTRUCTIONS

5.4.1. RIGHTY AND LEFTY COMMANDS

The **RIGHTY** and **LEFTY** instructions will force a robot configuration (posture) change during the next motion so that the first three joints of the robot arm are configured to resemble a person's right or left arm. The RIGHTY and LEFTY commands are not effective during linear interpolated motion or if the destination is a precision location.

5.4.2. ABOVE AND BELOW COMMANDS

The **ABOVE** and **BELOW** instructions will force a robot configuration (posture) change during the next motion so that the "elbow joint" (joint 3) is configured in an above or below position relative to the wrist. The ABOVE and BELOW commands are not effective during linear interpolated motion or if the destination is a precision location.

5.4.3. UWRIST AND DWRIST COMMANDS

The **UWRIST** and **DWRIST** commands are instructions that will cause a configuration (posture) change during the next motion so that the angle of Joint 5 will have a positive value for an UWRIST command and a negative value for a DWRIST command. The UWRIST and DWRIST commands are not effective during linear interpolated motion or if the destination is a precision location.

5.5. PROGRAM CONTROL COMMANDS

5.5.1. GOTO COMMAND

The format for the GOTO command is **GOTO *label IF condition***. The GOTO command causes program execution to move to (branch) the specified *label*.
A *label* is assigned to a program step for the purpose of identifying that spot in the program for execution to branch. *Labels* will maintain their relative locations within a program when steps are inserted or deleted, unlike program step numbers which will be reassigned with the insertion or deletion of steps. *Labels* can be a combination of alphanumeric characters up to 15 characters long. To place a label in a program, type

the desired *label* before the keyword in a program instruction. If the *label* begins with a alpha character, a ":" will automatically be added after the *label* name.
IF is an optional argument that can be used in conjunction with the GOTO command. If the *IF* argument is not included with the GOTO command, the GOTO instruction is unconditional and the program will branch to the specified *label* each time the program step is processed. If the *IF* argument is included with the GOTO command, the GOTO instruction is conditional and the program will branch to the specified *label* only when the *condition* statement is true. If the conditional statement is not true, the program will proceed to process the next step.

In the example shown in figure 5-9, step 9 of the program has an unconditional GOTO instruction that will cause the program to branch to the *label* "line". Step 3 of the program has a conditional GOTO instruction that will cause the program to branch to the *label* "200" when x is greater than 5, if x is less than 5, steps 4 through 8 will be processed.

```
$LIST/P test ↵
.PROGRAM test()
1  line: JMOVE pounce
2  x=x+1
3  GOTO 200 IF x>5
4  BREAK
5  JMOVE aa
6  LMOVE bb
7  LDEPART 100
8  GOTO line
9  200 Home
```

Figure 5-9 GOTO Command

5.5.2. IF COMMAND

The format for the IF command is **`IF condition GOTO label`**. The IF command causes program execution to move to (branch) the specified label when the conditional expression is true.
The IF commands includes a GOTO command in its format. The IF command is always conditional. When the IF command is processed and the conditional expression is true, the program will branch to the specified *label*. If the conditional statement is not true, the program will proceed to process the next step.

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

A *label* is assigned to a program step for the purpose of identifying that spot in the program for execution to branch. *Labels* will maintain their relative locations within a program when steps are inserted or deleted, unlike program step numbers which will be reassigned with the insertion or deletion of steps. Labels can be a combination of alphanumeric characters up to 15 characters long. To place a *label* in a program, type the desired *label* before the keyword in a program instruction. If the *label* begins with a alpha character, a ":" will automatically be added after the *label* name.
In the following examples, the IF command is used to evaluate weather the real variable *n* is greater than *20*, if *n* is greater than 20 the program will branch to the *label 100*.
The IF command is also used with a the logical *and condition* of signals 1001 and 1002, the IF *condition* is true if signal 1001 is ON and signal 1002 is OFF, the program will branch to the *label 250*.

IF n>20 GOTO 100
IF SIG(1001,-1002) GOTO 250

5.5.3. CALL COMMAND

The format for the CALL command is **CALL *program name***. The CALL command causes program execution to jump to the first step of the specified *program* (subroutine). After completion of the subroutine, execution returns to the next step after the CALL instruction.

A subroutine cannot be called by both the robot control program and the PC program at the same time. Multiple calls from subroutines (calling another subroutine from the one being executed) are possible up to twenty times. A subroutine cannot call itself. Figure 5-10 shows examples of the CALL command.

```
> EDIT pg00
 1? HOME
 2? If SIG(1001) THEN
 3? CALL PG02
 4? END
 5? HOME 2
```

```
> EDIT pg02
 1?     JMOVE aa
 2?     LMOVE bb
 3? 150 CALL pg 07
 4?     DELAY 2
 5?     JMOVE cc
 6?     GOTO 150 IF SIG(1020)
```

```
> EDIT pg07
 1? JMOVE a1
 2? C1MOVE a2
 3? C2MOVE a3
 4? C1MOVE a4
 5? C2MOVE a1
```

Figure 5-10 CALL Command

In pg00 at step 2, if input signal 1001 is high, step 3 will be processed and call pg02. If input signal 1001 is low, step 5 will be executed. If pg02 is called by pg00, pg02 will be executed. In pg02, step 3 will cause the program to execute pg07. After pg07 is

executed, the execution will return to the caller program. The execution will continue from step 4, and if input signal 1020 is high (step 6), the program will branch to step label 150 at which point pg07 will be executed again. If input signal 1020 is low, then execution will return to pg00 (the caller program).

5.5.4. RETURN COMMAND

The **RETURN** command terminates execution of the subroutine and returns program execution to the original program, at the step following the instruction which called the subroutine.

If no caller program exists, as in the case of a subroutine program that was executed by the EXECUTE instruction, the RETURN instruction will be processed the same as a STOP instruction. If there are remaining cycles to be executed, execution will continue with the first step.

If no RETURN instruction exists in the called subroutine, program execution returns to the caller program in the same way that it would had the RETURN instruction been at the end of the subroutine.

5.5.5. WAIT COMMAND

The format for the WAIT command is **WAIT *condition***. The WAIT command causes program execution to wait at the current step until the specified *condition* (real value expression) becomes TRUE. The WAIT instruction is used to suspend program execution until the specified *condition* is satisfied. A programmed WAIT instruction can be bypassed with the CONTINUE NEXT command or the WAIT OVERRIDE function. For example, the instruction WAIT SIG (1001, -1003) will cause program execution to wait until external input signal 1001 is turned ON and signal 1003 is turned OFF. The instruction WAIT TIMER (1)>10 will cause program execution to wait until the value of timer 1 exceeds ten seconds. The instruction WAIT n>100 will cause program execution to wait until the real variable n has a value greater than 100.

5.5.6. SWAIT COMMAND

The format for the SWAIT command is **SWAIT *signal number***. The SWAIT command causes program execution to wait until the specified external I/O signals or internal I/O signals are set to the specified states. The signal number represents the number of an external or internal I/O signal. A “-“ (minus) sign is used to specify the desired signal state is OFF. If all of the specified SWAIT signal states are satisfied, program

execution proceeds to the next step. If all of the specified SWAIT signal states are not satisfied, program execution waits at the current step. A programmed SWAIT instruction can be bypassed with the CONTINUE NEXT command or the WAIT OVERRIDE function.

5.5.7. TWAIT COMMAND

The format for the TWAIT command is **`TWAIT`** ***`time`***. The TWAIT command causes program execution to wait at the current step for the specified period of *time* (in seconds). For example, the command **TWAIT *0.5*** causes execution to wait for 0.5 seconds while the command **TWAIT *fix*** causes execution to wait for the time set by the real variable "fix".

5.5.8. PAUSE COMMAND

The **`PAUSE`** command stops program execution and sends a message to the display screen. The CONTINUE command can be used to resume program execution after a PAUSE instruction. The PAUSE command can be useful when troubleshooting programs, insert a PAUSE command at any point in the program to stop execution for a moment and check the current values of variables.

5.5.9. HALT COMMAND

The **`HALT`** command stops program execution. When a HALT command is encountered, program execution cannot be resumed regardless of remaining execution cycles. When the HALT command is processed, a message will be displayed. Program execution that has been stopped by the HALT instruction cannot be resumed by the CONTINUE or CONTINUE NEXT command.

5.5.10. STOP COMMAND

The **`STOP`** command terminates the current execution cycle. If there are remaining cycles to be completed, execution returns to the first step, otherwise, execution ends. If more execution cycles are remaining, execution continues with the first step of the main program (even if STOP was executed in a subroutine or an interrupt-handling program, execution returns to the beginning of the main program). A RETURN instruction in a

main or caller program performs the same function as the STOP instruction. After a program has been ceased by a STOP instruction, the CONTINUE command cannot be used to resume execution.

5.5.11. SCALL COMMAND

The **SCALL** command is similar to the CALL command. The SCALL command is used with a program name that is assigned to a string variable.

5.5.12. LOCK COMMAND

The format for the LOCK command is **LOCK** ***priority***. The LOCK command is used in process control (PC) programs and sets the execution *priority* of subroutines. The range or priority setting available is from 0 to 127. If subroutines that are called by the PC program do not have a LOCK priority assigned, a 0 is assumed.

5.5.13. ONE COMMAND

The format for the ONE command is **ONE *program name***. The ONE command is used in process control (PC) programs and is used to call the specified *program name* when an error occurs. The ONE command causes the program to return to the program step where the error was generated. The *program name* specified with the ONE command cannot have any robot motion instructions.

NOTE
The program set once while the program is executed is a valid even if extending as for the program. When terminate the execution of the program, it become an invalid.

5.5.14. RETURNE COMMAND (RETURN ERROR)

The **RETURNE** command is a PC program command and is used to return program execution to the step following the step an error was generated on. The RETURNE command is used in conjunction with the ONE command to execute the next step of a program after an error has occurred.

5.5.15. MVWAIT COMMAND

The format for the MVWAIT command is **MVWAIT numeric**.
The robot is waiting the execution of the program until the distance (time) of remainder of current robot motion shortens more than the specified distance (time).
numeric : The distance which wants to stand by is specified (in mm).
Distance of remainder of robot motion. <= Distance. (Parameter)
or The time which wants to stand by is specified. (in seconds)
Time of remainder of robot motion. <= Time. (Parameter)
When the input of the unit is omitted, the unit of treatment mm is added as “mm”.

The execution of the program can be advanced by using this instruction synchronizing with the robot motion of the robot arm. However, because the distance of the remainder of the robot motion (Or, time) is obtained by using the command value; Note that only the amount which corresponds at the delay time becomes inaccurate.
The distance specified when operating by the joint interpolation might shift largely with an actual distance.

Execution moves to the following instruction without standing by in no robot motion. The waiting can be released by using “CONTINUE NEXT” instruction at the waiting.

Warning : This instruction cannot be used by PC program. You cannot use it by instructing in DO command.

When it faces location lc1, signal 21 is turned on at the position of lc1 at 100mm in this side as shown in the figure below. (However, it is time when it is in the valid and the range of the precision “Read the signal instruction ahead”.)

```
LMOVE lc1
MVWAIT 100mm
SIGNAL 21
```

Signal 21 is turned on 0.2 seconds when the robot motion is completed ago as shown in the figure below in the place where it faces location lc1. (However, it is time when it is in the valid and the range of the precision “Read the signal instruction ahead”.)

```
LMOVE lc1
MVWAIT 0.2S
SIGNAL 21
```

5.5.16. CALLAUX COMMAND

The format for the CALLAUX command is **`CALLAUX function_number`**.
Display the designated auxiliary function screen from the program. Designate an auxiliary function number screen which want to be displayed to the function_number of the parameter. (1-899 integer (Number which exist in auxiliary function)) You can setting and displaying the data of an auxiliary function in the program. The auxiliary function screen disappear when holding in the CALLAUX instruction execution. When the CALLAUX instruction is executed from the plural program, do the waiting or holding of the next CALLAUX instruction until the executed CALLAUX instruction ends. The error occurs if you specify the auxiliary function number which does not exist by this instruction. Specify function_number (1 - 899) which exists in the auxiliary function.

5.5.17. REPCYCLE COMMAND

The format for the REPCYCLE command is **`REPCYCLE repeat_cycle_number`**.
Only the frequency specified by the repeat condition does the repeat motion. Specify for repeat_cycle_number of parameter when the repeat condition is a specified frequency. Range of setting: 0 - 99
This instruction is a valid only at the time of SYSTEM SWITCH “REP_CYC=ON” and “REP_ONCE=OFF”. Even if SYSTEM SWITCH is set in “REP_CYC=ON” when repeat cycle number is 0, it does not become specification of the frequency. (it become the continuous state) It become the continuous state if SYSTEM SWITCH is “REP_CYC=OFF”.

5.6. CONTROL FLOW STRUCTURE INSTRUCTIONS

The following section covers the five control flow structures available with the AS language. Control flow structure instructions are special types of program commands that consist of more than one line of code to form a group or block of steps. Depending on the structure used, these blocks of instructions evaluate variables and based on the results of those evaluations provide sequence control, decision making, looping, and the ability to select a set of instructions to be processed from many possible sets. With the exception of the DO UNTIL command structure, all control flow structures are defined by a format that starts with a structure command and has an END instruction to identify the last command in the structure.

5.6.1. IF...THEN...ELSE...END COMMAND

The format for this structure is:

IF *logical expression* THEN
.....
program instructions to be executed when *the logical expression* is true
.....
ELSE (optional)
.....
program instructions to be executed when the *logical expression* is false
.....
END

The IF...THEN...ELSE...END control flow structure is an extension to the Logical IF instruction. This structure permits more than one program instruction to be executed if the logical expression being evaluated is true. The logical IF statement permits only a single statement to be given after the logical expression. The IF...THEN...ELSE...END control flow structure also permits an optional ELSE instruction to be included. If the *logical expression* is not true and the structure includes an ELSE instruction, the program instructions that follow the ELSE will be processed. It includes statements to be executed whenever the logical condition is false.

When the *logical expression* is true, the program instructions following the THEN statement are processed and executed. The program then transfers out of the control flow structure to the program instruction following the END statement. When the *logical expression* is false and the optional ELSE command is not used, no steps in the control flow structure are processed and to the program instruction following the END statement is processed. If an ELSE command is used and the logical expression is false, the program instructions following the ELSE instruction are processed and executed. The program then transfers out of the control flow structure to the program instruction following the END statement. The IF...THEN...ELSE...END control flow structure can have additional structure(s) “nested” within the original structure. When a control flow structure has nested structure, the inner most structure is processed first.

```
$LIST/P test ↵
.PROGRAM test()
10 x=x+1
11 IF x<20 THEN
12 JMOVE pounce
13 LMOVE picup
14 JMOVE dropspt
15 ELSE
16 HOME
17 SWAIT 1003
18 END
19 GOTO 250
```

```
$LIST/P test ↵
.PROGRAM test()
21 IF SIG (1011) THEN
22 LMOVE tipclean
23 TWAIT 12
24 END
25 GOTO line
```

Figure 5-11 IF...THEN...ELSE..END Command

In the first example shown in figure 5-11, the IF...THEN...ELSE..END structure is used to evaluate the real variable x, if the value of x is less than 20, the three motion commands in steps 12, 13, and 14 will be executed and then the program will process step 19 and GOTO the program instruction associated with the label 250. If the value of x is greater than 20, the program instructions of steps 16 and 17 will be processed and the robot will go to the HOME position and wait for signal 1003 to turn ON, if signal 1003 is on then after the robot reaches HOME the program will process step 19 and GOTO the program

instruction associated with the label 250. In the example on the right of figure 5-11, when signal 1011 is ON, the program instructions of steps 22 and 23 will be processed. When signal 1011 is OFF, the first program instruction after the control flow structure (step 25) will be processed.

5.6.2. WHILE...DO...END COMMAND

The format for this control flow structure is:

WHILE ***logical expression*** **DO**

.....

program instructions to be executed when the *logical expression* is true

.....

END

The WHILE...DO...END control flow structure permits the repeated execution of a group of statements while a specified *logical expression* is found to be true. If the *logical expression* is true, then the program instructions following the WHILE...DO command and preceding the END instruction are executed. The program processing then returns to the WHILE..DO statement and retests the *logical expression*. This flow of processing continues repeatedly until the logical expression becomes false, or until another conditional statement inside the control flow structure causes a transfer out of the original structure. When the logical expression is false (which includes the first time it is evaluated), program processing resumes at the instruction following the END.

```
$LIST/P test ↵
.PROGRAM test()
21 WHILE SIG (1021,1022) DO
22 CALL bigpg
23 CALL smallpg
24 END
25 GOTO line
```

Figure 5-12 WHILE...DO...END Command

In the example shown in figure 5-12, as long as both signals 1021 and 1022 are ON, the subroutines bigpg and smallpg will be called. When the subroutine is completed, the program execution will return to the caller program and reevaluate signals 1021 and 1022, if either of them is not ON, the program will process step 25.

Typically the WHILE...DO...END structure is with a loop type variable. The loop variable is a real variable with a value that is continually tested in the logical expression.

This variable must be correctly initialized prior to entering the WHILE...DO...END structure, if the loop is to be executed the correct number of times. The value of the loop variable should be updated within each loop structure. The update of the loop variable is usually the last executable statement in the WHILE...DO...END structure. Figure 5-13 is an example of this type of loop variable being used with a WHILE...DO...END structure.

```
$LIST/P test ↵
.PROGRAM test()
31 welds=0
32 WHILE welds<50 DO
33 jm spot1
34 jm spot2
35 jm spot3
36 welds=welds+1
37 END
```

Figure 5-13 WHILE...DO...END Command

In the example shown in figure 5-13, the real (loop) variable “welds” is initialized with a value of zero in step 31. The first time step 32 is processed the value of “welds” will be 0. Since 0 is less then 50, the robot will execute the three motion steps 33, 34, and 35. Step 36 will increment the value of “welds” by 1, the program will not process out of the structure at this time but will reevaluate the value of “welds” in step 32. This looping process will repeat itself 51 times until the value of “welds” is not less than 50,

The END instruction is a nonexecutable command. If program execution is transferred to a statement inside a control flow structure, the program instructions are executed sequential order, and the END command is ignored. Program execution may be transferred to a statement outside the control flow structure at any time. The WHILE...DO statement cannot be the range statement in a DO loop, nor the given statement in a logical IF.

5.6.3. DO...UNTIL COMMAND

The format for this control flow structure is:

DO

.....
Program instructions
.....

UNTIL *logical expression*

The DO...UNTIL control flow structure provides a method to control the execution of a group of program instructions based on the evaluation of a *logical expression*. Typically, when the DO...UNTIL structure is used, some action or process occurs within the structure that will change the result of the logical expression from TRUE to FALSE causing the structure to be exited.
The DO...UNTIL control flow structure provides a similar type of logic structure as the previous WHILE...DO...END instructions. However, with the DO...UNTIL structure the program instructions within the structure will always be processed at least once.

No program instructions are needed between the DO and UNTIL commands. When there are no such instructions, the UNTIL criterion is continuously evaluated until it is satisfied, at which time program execution continues with the instructions following the UNTIL instruction.

```
$LIST/P test ↵
.PROGRAM test()
51 DO
52 CALL lftfender
53 CALL rgtfender
54 UNTIL SIG(-1019)
```

Figure 5-14 DO...UNTIL Command

In the example shown in figure 5-14, the CALL instructions of steps 52 and 53 will be processed at least one time regardless of the status of signal 1019. The CALL instructions of steps 52 and 53 will be processed as long as signal 1019 is ON. When signal 1019 changes state to OFF, the program will advance to step 55 and continue execution from that point.

5.6.4. FOR...TO...STEP...END COMMAND

The format for this control flow structure is:

```
FOR loop variable = start value TO end value
  STEP step value
    .....
    program instructions to be processed
    .....
END
```

The FOR...TO...STEP...END control flow structure is used to repeat the program instructions within the structure a specified number of times. The FOR...TO...STEP...END structure utilizes a *loop variable* (a real variable used as a counter), that will process the structure a specified number of times. The number of times the structure will be processed is determined by the range of the *start value* TO the *end value*. Each time the structure is processed, the loop variable receives a new value. If the loop variable is in the range of values specified by the *start value* TO *end value*, the program instructions within the structure are executed once. The looping process is completed when the loop variable has exceeded the *end value*.

The FOR...TO section of the structure identifies the *loop variable*, and specifies the beginning and ending range for the loop counter. If the value of the *loop variable* is not within the range identified by the start value and the *end value*, the program will not process any other instructions within the structure. The *start value* and *end value* can be expressed as constants or arithmetic expressions. The optional STEP instruction specifies the amount (*step value*) by which the *loop variable* will be increased or decreased each time the structure is processed. If the STEP instruction is omitted, the default increment amount is set at one. If the STEP value is a positive number, the value of the counter will increase by that amount each time the structure is processed. If the STEP value is a negative, the counter will decrease.

```
$LIST/P test ↵
.PROGRAM test()
01 spots = 0
.......
61 FOR spots = 0 TO 25
62 LMOVE upspot
63 JMOVE downspot
64 LMOVE overspot
65 END
```

Figure 5-15 FOR...TO...STEP...END Command

In the example shown in figure 5-15, the real variable spots is being used as a *loop variable*, step 1 sets the value of spots to 0, at step 61 the range for the structure is set from 0 (*start value*) to 25 (*end value*), because 0 is within the acceptable range specified in step 61, the program instructions in steps 62, 63, and 64 will be processed 26 times before the program step after the END (step 66) is executed. These steps will be executed 26 times because the initial value was 0. In this example the STEP instruction was omitted causing the loop variable spots to be incremented by a value of one each time it was evaluated.

5.6.5. CASE...OF...VALUE...ANY...END COMMAND

The format for this control flow structure is:

CASE ***index variable*** **OF**
VALUE (1st)
.....
program instructions to be processed
.....
VALUE (2nd)
.....
program instructions to be processed
.....
ANY
.....
program instructions to be processed
.....
END

The CASE block structure implements a selected VALUE depending on the index variable. This block structure provides capabilities similar to those of a GO TO statement. The general form of the CASE block is shown above.

The *index variable* is a real valued expression whose value is used to select the proper VALUE and its related statements. The CASE structure evaluates the *index variable*, and then examines the VALUE statements sequentially to find the same VALUE as the *index variable*. After the statements belonging to the selected VALUE statement are executed, control of execution resumes at the statement immediately following the END statement (unless a GO TO statement was encountered).

If no VALUE step is found that contains the same value as the CASE index variable, and there is an ANY step in the structure, then the group of instructions following the ANY step will be executed.

If no VALUE match is found in the structure, and there is no ANY step, none of the

instructions in the entire CASE structure will be executed.

```
$LIST/P test ↵
.PROGRAM test()
61 pg = BITS (1021,4)
62 CASE pg OF
63 VALUE 1
64 CALL pg01
65 VALUE 2
65 CALL pg02
66 VALUE 3
67 CALL pg03
68 ANY
69 HOME
70 PRINT "An unexpected value was received."
71 PRINT "Wait override required to continue."
72 SWAIT 1013
73 END
```

Figure 5-16 CASE...OF...VALUE... ANY... END Command

In the example shown in figure 5-16, step 61 assigns the binary value signals 1021 through 1024 to the real variable *pg*. Step 62 uses the CASE *pg* OF command to establish pg as an index variable, the value of pg will be evaluated by the VALUE instructions. When the value of *pg* is 1 (signal 1021 ON, signals 1022, 1023, and 1024 OFF), the VALUE command at step 64 is satisfied and pg01 is called. Steps 65 and 66 follow the same evaluation process. As long as the value of *pg* remains 1, 2, or 3, the subroutines pg01, pg02, and pg03 will be executed. If the value of pg is not 1, 2, or 3, the ANY command in step 68 will be processed and the robot will go to HOME, print the character strings specified in steps 70 and 71, and then wait for signal 1013. After the SWAIT is satisfied, the program will process the next step after the structure.

5.7. BINARY SIGNAL CONTROL COMMANDS

5.7.1. RESET COMMAND

The **RESET** command is used to reset all external output signals to OFF. The reset command does not turn OFF dedicated output signals. Before the RESET command is entered, users should be aware of the affect the command will have on external equipment and peripheral programming.

5.7.2. SIGNAL COMMAND

The format for the SIGNAL command is **SIGNAL** ***(-)signal number, ...*** . The signal command is used to set external output signals and internal status signals to ON or OFF. To turn the desired signal(s) ON the *signal number(s)* are entered after the SIGNAL command and are separated by a ",". If the signal(s) are to be turned OFF, a "-" symbol is placed before the signal number. Signals that have been assigned as dedicated signals for specific conditions cannot be turned ON and OFF with the SIGNAL command. Input signals cannot be designated with the SIGNAL command. The range of signals that can be affected by the SIGNAL command include: 1 to the number of external output signals the controller is configured for and 2001 to 2256 for internal status signals. For example, SIGNAL 2,4,-5 will turn outputs 2 and 4 ON and output 5 OFF.

5.7.3. PULSE COMMAND

The format for the PULSE command is **PULSE** ***signal number, time (seconds)***. The PULSE command is used to turn ON a specified signal for a set period of the time. The range of signals that can be affected by the PULSE command include: 1 to the number of external output signals the controller is configured for and 2001 to 2256 for internal status signals. Signals that have been assigned as dedicated signals for specific conditions cannot be used with the PULSE command. Signals can only be turned ON with the PULSE command, they cannot be PULSED off. If a time value is not specified, 0.2 seconds is the default setting and signals will be pulsed on for 0.2 seconds. For example, PULSE 4,5.6 will turn output 4 ON for 5.6 seconds.

5.7.4. DLYSIG COMMAND

The format for the DLYSIG command is **DLYSIG** ***signal number, time (seconds)***.

The DLYSIG command is used to turn a specified signal ON or OFF for a set period of time. The range of signals that can be affected by the DLYSIG command include: 1 to the number of external output signals the controller is configured for and 2001 to 2256 for internal status signals. Signals that have been assigned as dedicated signals for specific conditions cannot be used with the PULSE command. Signals can be turned ON or OFF with the DLYSIG command, a "-" is used before the signal number if the signal is to be turned OFF. For example, DLYSIG 2,4.5 will turn output 2 ON after 4.5 seconds have elapsed, DLYSIG -2,4.5 will turn output 2 OFF after 4.5 seconds have elapsed.

5.7.5. RUNMASK COMMAND

The format for the RUNMASK command is **`RUNMASK starting signal, number of signals`**. The RUNMASK command allows signals that were turned on within a program, to remain ON only while the program is being executed. If a signal is turned ON within a program by a SIGNAL, PULSE, or DYLDSIG instruction, the RUNMASK command will cause the signal to be turned OFF when program execution stops.

The *starting signal* is the first signal number to be affected by the RUNMASK command. The *number of signals* specifies how many signals will be affected by the RUNMASK command. If the *number of signals* is omitted, only one signal will be affected. A negative value before the *signal number* indicates that the mask function is to be canceled; that is, these unmasked signals will not be turned OFF when program execution stops.

5.7.6. BITS COMMAND

The format for the BITS command is **`BITS starting signal number, number of signals = value`**. The BITS command is used to set a specified number of output signals to a binary value or to display the current status of the signals specified. The *starting signal number* is the smallest signal number to be set (least significant bit). The *number of signals* (bits) is how many signals are to be evaluated (maximum 16), starting from the *starting signal number*. The range of signals that can be used with the bits command include: 1 to the number of external output signals the controller is configured for and 2001 to 2256 for internal status signals. Signals that have been assigned as dedicated signals for specific conditions cannot be used with the BITS

command. For example, the instruction **BITS *2001,3*** will display the binary value of internal signals 2001, 2002, and 2003. In another example, the instruction **BITS *1-8=100*** will set the signal states of signals 1 through 8 to a binary value of 100 (binary 1100100 or signal numbers 8, 7, and 3 would be ON).

5.7.7. EXTCALL COMMAND

The **EXTCALL** command (external call) is used to select an external program number (subroutine) by utilizing the RPS function and designated external input signals.

To utilize the EXTCALL command, the following programming sequence is required.

1. RPS-ST signal must be turned ON.

2. The program then scans for the RPS-ON signal to be turned ON.

3. When the RPS-ON signal is detected, a binary number is calculated from the designated input signals and a external subroutine program is called. The inputs used for subroutines program numbers include binary signals RPS1 through RPS64. The EXTCALL instruction selects programs with a (Pg) prefix and the number of the calculated binary signal status.

The EXTCALL instruction is effective only when external input signals have been designated for RPS-ST, RPS-ON, RPS1 - RPS64, and the RPS system switch is ON.

5.7.8. ON COMMAND

The format for the ON command is `ON` *`mode signal number CALL program name, priority`* or `ON` *`mode signal number`* `GOTO` *`label, priority`*. The ON command is used to monitor the specified external input signal, and when the signal is turned to the desired state, CALLs the specified *program name* (subroutine), or, in the case of a GOTO instruction, proceeds to the specified *label*. After the ON command has been processed the first time, the program continuously monitors the specified signal for the desired state. The CALL or GOTO action of the ON command is not dependant on when the individual program step containing the ON instruction is processed.
If no *mode* parameter is specified, signal change is monitored rising (falling) detection.
When the /ERR parameter is specified, an error will occur if the ON command is entered when the specified signal state is already present. The /LVL parameter will immediately execute interrupt processing if the ON command is entered when the specified signal state is already present.

When the *signal number* specified changes state to the specified condition, the program will complete the current motion step it is processing and then CALL the *program name* or process the GOTO statement and proceed to the *label* identified.

The range of *signal number* that can be monitored by the ON command are from 1001 to the number of external signal number that are configured or 2001 to 2256 for internal signals. Signal monitoring initiated with an ON command is canceled in the following cases:

- An IGNORE instruction is executed for the signal for which the ON or ONI instruction was executed.

- A corresponding interruption occurs.

- An ON (or ONI) instruction is executed for the same signal.

5.7.9. ONI COMMAND

The format for the ONI command is `ONI mode signal number CALL program name, priority` or `ONI mode signal number GOTO label, priority`. The ONI command is used to monitor the specified external input signal and when the signal is turned to the desired state, CALLs the specified *program name* (subroutine), or, in the case of a GOTO instruction, proceeds to the specified *label*. After the ONI command has been processed the first time, the program continuously monitors the specified *signal number* for the desired state. The CALL or GOTO action of the ONI command is not dependant on when the individual program step containing the ONI instruction is processed.

If no *mode* parameter is specified, signal change is monitored by rising (falling) detection. When the /ERR parameter is specified, an error will occur if the ON command is entered when the specified signal state is already present. The /LVL parameter will immediately execute interrupt processing if the ON command is entered when the specified signal state is already present.

When the *signal_number* specified changes state to the specified condition, the program will terminate the current motion step it is processing and immediately CALL the *program name* or process the GOTO statement and proceed to the *label* identified.

The range of *signal_number* that can be monitored by the ONI command are from 1001 to the number of external signals that are configured, or 2001 to 2256 for internal signals. Signal monitoring initiated with an ONI command is canceled in the following cases:

- An IGNORE instruction is executed for the signal for which the ON or ONI instruction was executed.
- A corresponding interruption occurs.
- An ON (or ONI) instruction is executed for the same signal.

The argument *priority* is optional. The acceptable range for the priority argument is from 1 to 127. If the *priority* is omitted, it is defaulted to 1. The greater the *priority* number indicated, the higher the priority.

The signal monitoring does not process according to the state of the signal itself. The interruption caused by a satisfied ONI command occurs when the signal state changes. When an ONI instruction is in a program, and the *signal number* being monitored is ON, the interruption does not occur until the signal is turned OFF then ON again. To be detected, signal changes must remain stable for at least 50 msec. Signal changes are ignored while program execution is stopped.

ONI -1001 CALL alarm,10

In this example, the program starts monitoring the external input signal 1001(WX1) the first time this line of code is processed. When signal 1001 changes from ON to OFF (designated by the minus sign before the signal number), the current robot motion is stopped at once, and execution branches to the program named “alarm”. If the priority of the current program is 10 or higher, the branch does not take place until the priority of the current program is below 10.

5.7.10. IGNORE COMMAND

The IGNORE command cancels signal monitoring initiated by the ON or ONI instruction. This command has the effect of clearing the last ON or ONI instruction.

5.7.11. SCNT COMMAND

The format for the SCNT command is **`SCNT counter signal number = count up signal, count down signal, counter clear signal, count value`**. The counter signal outputs it when becoming a specified counter value. **`counter signal number`** : The signal number to want to output it is specified. Range of setting : 3097~3128. **`count up signal`** : Specifies it with the signal number or the signal logical expression. If this signal had changed from turning off into turning on when the instruction was executed, the count improvement is done. **`count_down_signal`** : Specifies it with the signal number or the signal logical expression. If this signal had changed from turning off into turning on when the instruction was executed, the down of the count is done. **`counter clear signal`** : Specifies it with the signal number or the signal logical expression. If this signal is turned on when the instruction is executed, an internal counter is adjusted to 0. **`count value`** : The count value to output the counter signal is

specified with the real value. When the internal count value reaches this value when the instruction is executed, outputs it.
When 0 is set in the counter value, the counter signal is turned off.
If the count improvement signal had changed from turning off into turning on when the SCNT instruction (instruction) was executed, the internal count value is made and if the down of the count signal has changed from turning off into turning on, it adjusts the internal counter value to -1 by +1 doing. And, the count signal is output when becoming a count value for which the internal counter value is specified. If a counter clear signal is input, the internal count value becomes 0. It individually has the internal count value according to the counter signal number.
Use the SCNTRESET instruction when 0 clearing the internal count value compulsorily.

5.7.12. SCNTRESET COMMAND

The format for the SCNTRESET command is **`SCNTRESET counter signal number`**. 0 clearing of the internal count value which corresponds to the counter signal number. **`counter_signal_number`** : The counter signal number which wants to clear 0 is specified. Range of setting : 3097~3128.

5.7.13. SFLK COMMAND

The format for the SFLK command is **`SFLK flicker signal number = time`**. It turns on and off at the cycle that the flicker signal was specified of time. **`flicker signal number`** : The signal number to want to do flicker is specified. Range of setting : 3065~3096. **`time`** : Time of the ON/OFF cycle is specified with the real value. When a negative value is specified, flicker is discontinued.
ON/OFF is assumed to be a round period and ON/OFF is repeated at the specified cycle of time.

5.7.14. SFLP COMMAND

The format for the SFLP command is **`SFLP output signal = set signal expression, reset signal expression`**. The output signal is turned on and off by the set signal and the reset signal. **`output signal`** : The signal number which wants to output is specified. Turning on is output and the turning off output is done when the value is negative when the value is positive. Only the signal number which can be output can be specified (The 1000th cannot be specified). **`set signal expression`** : The signal number or the signal logical expression to set the output signal is specified. **`reset signal expression`** : The signal number or the signal logical expression to reset the output signal is specified.
If the set signal is turning on, the output signal is turned on and if the reset signal is turning on, the output signal is turned off. When both the set and reset signals are turning on, the output signal is turned off. However, be to execute the SFLP instruction that the output signal is turned on and off. It is not a moment when the set signal was turned on.

5.7.15. SOUT COMMAND

The format for the SOUT command is **SOUT signal number = signal expression**.
After the condition of the signal type consists, the signal is output. **Signal number** :
The signal number which wants to output is specified. Only the signal number which can be output can be specified (The 1000th cannot be specified). **signal_expression** :
The signal number or the signal logical expression is specified.
It is an instruction for the logical operation of the signal. When it describes in the signal type with AND and OR, etc. and the condition consists, the signal is output.

5.7.16. STIM COMMAND

The format for the STIM command is **STIM timer signal = input signal number , time**. The TIMER signal is turned on continuing during time that the input signal was specified. in case of turning on. **timer signal** : The signal number turned on when the input signal continues specified time ON is specified with Numeric value. Range of setting :3001~3064. **input signal number** : The input signal number or the signal logical expression is specified with the integer. Range of setting: Mounted range of signal. **time** : Time to observe the state which is turning on the input signal痴 it continuing is specified with the real value.
After the state of time ON specified after turning it on again continues, the TIMER signal is turned on if it turns off in time that the input signal was specified. When the input signal is turned off, the TIMER signal is turned off at once. However, it is time when ON/OFF of the TIMER signal executed the STIM instruction. Even if the input signal is turned off, the TIMER signal is not turned off if the STIM instruction is not executed.

```
STIM 3001 = 1, 5
SOUT 2 = 3001
```

When sig1 is turned on when PCEXECUTE is done, sig2 turns on program five seconds later.

5.7.17. SETPICK COMMAND

5.7.18. SETPLACE COMMAND

The format for the SETPICK and SETPLACE commands are **SETPICK *time1 , time2 , time3 , time4 , ... , time8*.** and **SETPLACE *time1 , time2 , time3 , time4 , ... , time8*.** Sets beginning time(SETPLACE) of beginning time(SETPICK)/the open clamping control of the close clamping control to make it execute by ordering CLAMP about each of clamping 1-8. ***time 1~8*** : Close open/control time of clamping 1-8 is specified (in seconds). Range of specification : 0.0~10.0 sec. Refer to the CLAMP instruction.

5.7.19. CLAMP COMMAND

The format for the CLAMP command is **CLAMP *clamp number1 , clamp number2 , clamp number3 , clamp number4 ,..., clamp number8*.** It does the opening and shutting control of the clamping signal in front of time in which it instructs from TIME which ends the robot motion of a present robot arm by ordering SETPICK/SETPLACE according to the clamping number of Parameter specification.

clamp number 1~8 : The clamping number is specified. The clamping number controlled by the absolute value of Numeric value is shown and clamping control the opening is shown by positive/negative of the sign of Numeric value.

This instruction outputs the signal to open and shut the hand to the control valve of the empty pressure hand. When shorting than time which remainder time of the robot motion executing it now in no robot motion the robot arm after ordering executing it set by ordering SETPICK/SETPLACE, the signal is output at once. Moreover, when the axis agrees, the signal is output when overlapping with the robot motion of the following step starts as the axis agrees before reaching time which remainder time of movement S executing it now set. When the conflicting specification like - CLAMP1,1 is done, the specification of the side in the back becomes a valid.

`12 SETPICK 4, 3, 2, 1`	The close of clamping 2 in ago of three seconds which get to position a.
`13 SETPLACE 0.2, 0.4, 0.6, 0.8`	The opening of clamping 1 in ago of 0.2 seconds which get to position a.
`14 LMOVE a`	The opening of clamping 4 in ago of 0.8 seconds which get to position a.
`15 CLAMP -1, 2, 3, -4`	The close of clamping 3 in ago of two seconds which get to position a.

5.7.20. SWAIT COMMAND

The format for the SWAIT command is **SWAIT *signal number***. The SWAIT command cause program execution to wait until the specified external I/O signals or internal I/O signals are set to the specified states. The signal number represents the number of an external or internal I/O signal. A “-” (minus) sign is used to specify the desired signal state is OFF. If all of the specified SWAIT signal states are satisfied, program execution proceeds to the next step. If all of the specified SWAIT signal states are not satisfied, program execution waits at the current step. A programmed SWAIT

instruction can be bypassed with the CONTINUE NEXT command. A similar function can be achieved even by the WAIT instruction.

5.8 INPUT AND OUTPUT INSTRUCTIONS

5.8.1. TYPE AND PRINT COMMANDS

The format for the TYPE and PRINT commands is **`TYPE (or PRINT)`** ***`device number: print data.`*** The TYPE and PRINT commands result in the same output format on the C-series controller multi function panel. The *device number* is to identify if the output will be displayed on a personal computer or multi function panel. If the output is to go to a personal computer, a *1*: is entered. If the output is to go to a multi function panel, a 2: is entered. If no device number is entered, the default setting is *1*: . If a multi function panel is being used and the device number is set to *1*:, the output from the PRINT and TYPE commands will only be displayed if the user places the multi function panel in the keyboard mode. When a 2: is entered for the *device number* and a TYPE or PRINT message is to be output, the multi function panel will automatically change from the current display to the keyboard display mode. The system switch MESSAGES must be ON for PRINT and TYPE commands to be displayed.

When you designate the Multi Function Panel(Type 2) to the device number, be careful about slowing of the execution of the PRINT/TYPE command if the Multi Function Panel is not connected for connected confirmation. Do not designate “2” to the device number when the Multi Function Panel is not connected.

The type and print commands can be used in conjunction with string expressions by placing the characters to be displayed between quotation marks. For example, the program instruction TYPE “*Kawasaki Robotics Inc*.” will output Kawasaki Robotics Inc. to the display screen. The value of real variables and calculations can also be output to the display screen with the TYPE and PRINT commands by placing the data to be evaluated between ( ) marks. A comma is required between arguments used with the PRINT and TYPE commands. For example, if the real variable total_parts has a value of 50 assigned to it and the program instruction

TYPE “The number of total parts is ”, *(total_parts)* , “.”

is entered, the output to the display screen will be: The number of total parts is 50. An example of the TYPE and PRINT commands used with a calculation of variables follows: the real variable total_parts has a value of 50 assigned to it, the real variable parts_comp has a value of 20 assigned to it and the program instruction

TYPE “he number of parts processed is ”, *(total_parts - parts_comp)* , “.”
is entered, the output to the display screen will be: The number of parts processed is 30.

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

The output from TYPE and PRINT commands can be modified to display the information that is formatted to the programmers specifications. The following format specification are used to control the format of numeric values, blank spaces, and carriage returns. The format setting becomes effective for subsequent parameters until another format instruction is given.

/D	Use the default format. This is the same as /G15.8 except that following zeros, and all spaces, except one between the values, are removed.
/Em.n	The value is displayed in scientific notation (for example, 1.234E+02), whole numbers in “m” digits and the fractional part in “n” digits. (The value of “m” should be larger than “n” by six [6] or more, and smaller than thirty-two [32].)
/Fm.n	The value is displayed in fixed point notation (for example, -1.234), whole in “m” digits field with the fractional part in “n” digits.
/Gm.n	If the value is 0.01 or more and can be displayed in the format “F” in the “m” digit field, the value is displayed in “F” format. Otherwise, the value is displayed in the “Em.n” format.
/Hm	The value is displayed as a hexadecimal number in the “n” digit field.
/In	The value is displayed as a decimal number in the “n” digit field.

The following formats are used to insert special characters between character strings.

/Cn	A set of carriage returns (CR) and line feeds (LF) is output for “n” times. If this format is the first or the last argument in the PRINT instruction, there will be “n” blank lines on the terminal.
/S	This is effective only when this format is the first argument. This format suppresses the output of (CR) and (LF) at the beginning of a message.
/Xn	“n” spaces are output.
/Jn (option)	Though it is almost the same as Hm, the numerical value of the hexadecimal number is displayed in the area of n character. 0 is embedded in a blank part.
/Kn (option)	Though it is almost the same as Jm, the numerical value is displayed in the area of n character as a decimal number. 0 is embedded in a blank part.
/L (option)	Though it is almost the same as /D, all blank previous characters of the numerical value are removed about the numerical value in /L in /D while one blank character adheres previous.

If the value of “point [5]” is 1000, when the above instruction is executed, the following message is displayed : asterisks are displayed because the value is too large to be displayed in the given field.

```
PRINT "ABC"
PRINT/S, "DEF"
PRINT "GHI"
```

If you execute the instruction above, “GHI” does not change line and it is displayed on the screen as follows.

```
| ABC
| DEFGHI
|
```

5.8.2. PROMPT COMMAND

The format for the PROMPT command is ***PROMPT device number, "...character string message...", variable value to be processed***. The PROMPT command is used to allow the operator to interact with the program being processed by providing a means to input variable data.

The *device number* is used to specify the type of interface being used. A 1 is entered if a PC is used while a 2 is entered if the interface device is a multifunction panel. If the device number is omitted, the multi function panel will be set as the default interface device.

The *character string message* is entered by the programmer and is the message that the operator will read and respond to. The response of typing a value and pressing the enter key, assigns a value to the specified real variable. When a program is executing and a PROMPT command is processed, the program will not process any additional steps until the PROMPT command has been responded to. If no value is entered at the PROMPT message, a value of zero will be assigned.

```
$LIST/P test ↵
.PROGRAM test()
71 HOME
72 PROMPT "Please enter the number of parts to be processed", parts
73 JMOVE pickup
74 FOR totpt = 1 TO parts
```

```
75 LMOVE pt1
76 CLOSE
77 LMOVE pt2
.....
81 END
```

Figure 5-17 PROMPT Command

The example in figure 5-17 shows the PROMPT command in step 72 being used to send a message to the operator (multi function panel in operation). When the operator enters a number as a response to the PROMPT message, the value is assigned to the real variable parts. In step 74 of the program a loop variable is being set up with the FOR...TO...END control flow structure, the end value of the counter will be assigned the value entered from the PROMPT command, *parts*.

5.8.3. IFPWPRINT COMMAND

The format for the IFPWPRINT command is **IFPWPRINT Window No. , *Display line , Display digit , Background color , Character color* = "Character" , *"Display Character", ...*** . The character string specified for the character displaying window set by "Auxiliary 131 INTERFACE PANEL" is displayed. **Window No**. : No. in the window of displaying the character of displayed "Auxiliary 131 INTERFACE PANEL" is specified. Range of setting. : 1 - 4. ***Display line*** : The line to which the displaying character in the specified window is specified. Range of setting. : 1 - 4. Assume to the 1 when omitting. ***Display digit*** : The digit to which the displaying character in the specified window is specified. Range of setting. : 1 - 40. Assume to the 1 when omitting. ***Background color*** : The background color in the displaying window is specified. Range of setting. : 0 - 9. The background color becomes white when omitting. ***Character color*** : The color of the displayed character is specified. Range of setting. : 0 - 9. The character color becomes black when omitting. ***Character*** : The displayed character string is specified. The character string since piece second is displayed from a specified digit of the next line. The character string since piece second can be omitted.

The IFPWPRINT instruction can be used only at time in the state that the interface panel can be used. "Character" is one instruction execution and a valid in all in the frame in a specified window. (The parts other than displaying character by one instruction execution are cleared.)

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

When the omittable parameter is omitted, the executed value is succeeded to window No. last time. When protruding beyond the left end of the frame which the displaying character secures, character strings changes line, and display (Even the specified displaying digit does the indent.). Moreover, when protruding under the frame, character strings rounds down and display. When the control code is included in the character strings, its replaces with space and display.

5.9. VARIABLE DEFINITION INSTRUCTIONS

5.9.1. HERE COMMAND

The HERE command is used to define or store in memory the robot's current locational value as a specified transformation or precision location. The format for the command is `HERE location name.` *Location name* can be used to designate transformation, precision or compound transformation locations.

CAUTION:
If the location variable is a compound transformation, only the transformation on the farthest right is defined. An error will occur if any transformations except one on the furthest right have been left undefined.

5.9.2. POINT COMMAND

The POINT command is used to assign a location name of the left of the equals sign to a location value on the right. The format for the command is `POINT location name = location value, precision point value`. The location name variable specifies a single locational info variable (transformation or precision point) or compound transformation variable.

The precision point parameter is only effective when (1) location name is a precision point, and (2) location value is a transformation. Location cannot be assigned if the location name is anything but a precision point.

When the *precision point* value is specified, the transformation value is calculated in the arm configuration provided by the precision point. Omitting *precision point* value will calculate the transformation as a value of current configuration.

Note:
(1) An error will result if no *location value* of the right part is defined.
(2) If a compound transformation is defined in *location name*, the POINT command will define only the transformation on the farthest right.
(3) An error will result if, within the compound transformation mentioned in (2), even one transformation beyond the right edge remains undefined.

PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE

5.9.3. VARIATIONS OF THE POINT COMMAND

The format for the HERE command are `POINT/X transformation 1 = transformation 2, POINT/Y transformation 1 = transformation 2, POINT/Z transformation 1 = transformation 2, POINT/OAT transformation 1 = transformation 2, POINT/O transformation 1 = transformation 2, POINT/A transformation 1 = transformation 2, POINT/T transformation 1 = transformation 2 and POINT/7 transformation 1 = transformation 2.`
Substitute the value of the element that a right transformation corresponds to a specific element of a left transformation. Specify the compound transformation whose single transformation variable or last paragraph is a transformation variable for transformation 1 of the parameter. Specify the transformation variable, the compound transformation or the transformation function which has been defined for transformation 2 of the parameter.
A right location is not defined, and it become an error at the case where the value was not obtained. Only the transformation on the part of the rightmost of that is defined by this instruction at the case where the compound transformation is specified left. Moreover, if either of the transformation other than a right edge in the compound transformation are undefined in this case, it become an error.

5.9.4. DECOMPOSE COMMAND

The DECOMPOSE command extracts the value of each component of locational information to the array variables. The format this command is `DECOMPOSE` *`array variable name [element No.] = location name.`* The locational “components”, “above refers to :

- If transformation X, Y, Z, O, A, T.
- If precision point JT1 through JT6.

Array variable name designates the name of the deal value array that stores a value for each component. *Element No.* sets the number of the first array element to store a value. *Location name* specifies the name of each location to be divided into its individual components (Transformation and Precision Point).

This instruction substitutes the value of each element of a specified location for each element of a specified array. The six elements corresponding to XYZOAT are

substituted in the case of transformation, and angular values of each joint substituted in the case of Precision Point.

5.9.5. BASE COMMAND

The BASE command resets the transformation values of the robot's base coordinate system. The format for this command is `BASE` *`transformation value`*. The *transformation value* set new base coordinates. If `NULL`, BASE is entered. Base is initialized to a null Base.

When this instruction is executed, the robot will break its continuous locus motion, and set the specified base coordinates as a Base transformation.

5.9.6. TOOL COMMAND

The format for the TOOL command is `TOOL transformation value`. Sets the transformation (tool transformationed) in the system which shows the position and the direction of the tool point relatively viewed from the tool installation flange of the robot. The *transformation* set a new tool transformation (or compound transformation). If `NULL` is entered, the tool is initialized to a null tool of 0 value.

When this instruction is entered with the robot will break its continuous locus motion, and set the specified transformation as a Tool transformation.

5.9.7. LLIMIT COMMAND

5.9.8. ULIMIT COMMAND

These commands set the upper and lower limit of the robot's work envelope via a joint angle value. The format for the LLIMIT and ULIMIT commands are `LLIMIT` *`precision`* `point` and `ULIMIT` *`precision point`*.

5.9.9. TIMER COMMAND

The TIMER command sets the time of a designated timer. The format for this command is `TIMER` *`timer number = time`*. Timer number designates the number of the timer to be set (from 1-10). The *time* parameter sets the timer in seconds.

Times set using the TIMER command become instantly valid when executed. You can know the value of the timer with the TIMER function.

5.9.10. ON/OFF COMMAND OF SYSTEM SWITCH

This command renders a designated system switch effective or ineffective. The format for the command is `switch name, ON/OFF`. *Switch name* designates the system switch to be turned ON or OFF. Multiple switch names can be entered by separating fasten name by comma. The present status of a system switch can also be displayed by entering the switch command.

5.9.11. NCHON COMMAND

5.9.12. NCHOFF COMMAND

The NCHON/NCHOFF commands affect the movement of the step following a designated step, and turn the notch filter ON and OFF respectively. The initial value of the notch filter is *valid* (on). Initialization is carried out via the EXECUTE command, or primary execution of PRIME, STEP and MSTEP commands.

CAUTION :
Outcome of the NCHON/NCHOFF commands differs according to servo type.

- Type1 Turns the notch filter ON/OFF
- Type2 Toggles between valid and invalid states of observer feedback.

The servo type can be checked by using the ID command.

NOTE :
For operations carried out at very low speed, it is recommended to use NCHOFF command if a faint vibration occurs.

5.9.13. WEIGHT COMMAND

The WEIGHT command is used to set load mass (mass of work plus robot hand) in kilograms. The format for the WEIGHT command is `WEIGHT` *`load mass , x-direction of center of gravity , y-direction of center of gravity , z-direction of center of gravity`*. The load mass set by this command is used to optimize acceleration/deceleration of the robot arm. The load mass parameter sets the mass of work + robot hand with a range of 0 to the maximum payload.

The *direction* of *center of gravity* specifies, in millimeters. The center of gravity for load

mass to be in the direction of the X, Y, or Z axis. If these parameters are omitted, the default value becomes the maximum payload.

CAUTION :
The option to forbid setting of the load's is center of gravity is also available. In the case of motion type 2 (standard with the 2 series of robots, optional in others) mass and center of gravity *must* be correctly entered: incorrect values can overload and reduce the work life of component parts, as well as causing deviation errors.

5.9.14 TRHERE COMMAND

The TPHERE command defines the current location of the robot with a X traverse unit as a transformation value. The format for the command is `TRHERE` *`location name`*. The location name parameter specifies the location name (transformation or compound transformation) be memorized.

In the absence of compound transformations, `TRHERE` is the same as the HERE command. When compound transformations are present, the traverse and X axis elements are combined, a relative instruction operation is carried out, and then the traverse element is subtracted.
The value of this traverse element then becomes the value of the traverse axis of the current position.

The TRHERE instruction should be used when there is a relative instruction that has referenced traverse axis elements.

CAUTION :
If the location variable is a compound transformation, only the transformation farthest to the right will be defined. Also, an error will occur if transformations of the compound transformation has not yet been defined.

5.9.15. MC COMMAND

The MC command executes a monitor command from the program. The format for the command is `MC` *`monitor command`*. monitor commands available for use with MC are ABORT, CONTINUE, ERESET, EXECUTE, HOLD and SPEED. The example below shows the MC command used within program autostart.pc.

Example the MC command (line2 of autostart.pc) executes start up of robot program pg1. Note that Motor Power should be turned ON when executing the robot program. Therefore, the program should contain a step such as line, below, that confirms whether or not Motor Power is indeed ON.

```
autostart.pc()
```

```
1 10 IF SWITCH(POWER)==FALSE GO TO 10
2 MC EXECUTE pg1
.END
```

Note: The MC command can be executed only by PC and not by robot program.

5.9.16. PLCAOUT COMMAND (OPTIONAL)

The PLCAOUT command sets real number values for the numbers of specific data output as integers. The format for the command is `PLCAOUT` *`Integer output data No. = real number value`*. The integer output can be from 1-32, and the real number value from 0-65535.

CAUTION :
This command is only effective when the optional internal sequencer function is ON. If the functions is OFF. An -856 error will result.

5.9.17. TPLIGHT COMMAND (OPTIONAL)

The TPLIGHT command illuminates two backlight the Multi-Function-Panel. Entering the TPLIGHT command when the backlight is already ON will extinguish the Light. (Off state will last for 60 consecutive minutes.)

5.9.18. UTIMER COMMAND (OPTIONAL)

The UTIMER command sets the initial timer value settings. The format for the command is **UTIMER @*timer variable* = *timer value***. Any number timers can utilize user variables, and be assigned arbitrary names. The @*timer variable* parameter specifies the name of the array and user variable. The @ denotes an integer variable. The *timer value* parameter sets the initial timer value, which ranger 0 - 2147483647.

5.9.19. BSHIFT COMMAND (OPTIONAL)

The BSHIFT command shift a specified transformation to the base coordinate by the input X, Y, Z, O, A, T components. The format for the command is `BSHIFT` *`transformation , X , Y , Z , O , A , T`*. The transformation parameter specifies the transformation variable to be shifted to base coordinates. Each transformation component can be individually specified by the X, Y, Z, O, A, and T parameters.

5.9.20. TSHIFT COMMAND (OPTIONAL)

The TSHIFT command shifts a specified transformation to the tool coordinate by the input X, Y, Z, O, A, T components. The format for the command is `TSHIFT` *`transformation , X , Y , Z , O , A , T`*. The transformation parameter specifies the transformation variable to be shifted to the tool direction. Each component can be individually specified by the X, Y, Z, O, A, T parameters.

The Tool direction shows coordinates on the specified transformation, and is not a value of the current of the robot tool.

5.10. PROGRAM AND DATA MANAGEMENT

5.10.1. DELETE COMMAND

The DELETE commands are used to remove specified data from system memory. The format for each command is as follows:

- DELETE program name
- DELETE/P program name
- DELETE/L location name
- DELETE/R real variable name
- DELETE/S character string variable name

Multiple items can be listed for deletion if separates by commas.
The DELETE command completely removes all subroutines, location information, real variables, and character strings used by the named program or subroutine.

NOTE :
The DELETE/P instruction will remove neither program subroutine nor variables used by the subroutine. It will delete only the specified program. If either variable name or only variable name[ ] is specified for an array variable name, all array variables are deleted. If the variable name [Element No.] is specified, only the array element is deleted.

5.10.2. NLOAD COMMAND (OPTIONAL)

The NLOAD command reads a file to system memory. The format for the command is `NLOAD/IF/ARC` *`device number`* = `file name +` *`file name, status variable`*.
The device number parameter designates the source of the data to be uploaded.
Uploaded 1: Standard terminal (personal computer)
Uploaded 2: Multi Function Panel

When the device number parameter is omitted, uploading is from the device used for program execution.
File name is used to enter a name for the five to be read to memory. Multiple five names are separated by + *status variable* is used to save error information for later reference, so that a program does not have to stop for error processing. 0 is returned for normal processing. If status variable is omitted, the robot will stop in the event of an error. Error will also occur if there is a step in the uploaded data that cannot be understood, at which time the following message is output: 0: Comment out and continue reading, 1: delete the program and exit. An error will occur if the name of the file to be read already exists in system memory, and a new file cannot be uploaded.

NLOAD/IF is used to read data from the interface panel, NLOAD/ARC for arc welding data. If 0 is selected and uploading continued please use the editor to correct the error once uploading has concluded editor after finished reading when continuing by “0”.

CAUTION :
If location, real or character string variables in memory have the same name as information loaded from a disk file, the pre-existing data will be deleted and replaced with data read from the disk. *Without* displaying an alert message.

5.10.3. SLOAD COMMAND

The SLOAD command reads to memory a file whose name is expressed by a character string. The format for the command is **SLOAD/IF/ARC *device number =*** *string expression , status variable*. The device number parameter designates the source of the data to be uploaded.

Uploaded 1 : Standard terminal (personal computer)
Uploaded 2 : Multi Function Panel

When the device number parameter is omitted, uploading is from the device used for program execution. The *string expression* parameter is used to designate the file to be read to memory. Multiple files *cannot* be designated.

The status variable parameter is used to save error code information for later reference, so that a program does not have to stop for error processing. 0 is returned for processing. If the status variable parameter is omitted the robot will stop in the event of an error. Error will also occur if there is a step in the uploaded data that cannot be understood, at witch time the following message is output: 0 : Comment out and continue reading, 1: Delete.
SLOAD/IF is used to read data from the interface panel, SLOAD/ARC for arc welding data. If 0 is selected and uploading continued, please use the editor to correct the error once uploading has concluded.

CAUTION :
If location, real or character string variables in memory have the same names as information loaded from a disk file, the pre-exciting data will be deleted and replaced with data read from the disk *without* displaying an alery message. .

5.10.4. TRACE COMMAND

The TRACE command turns the TRACE (for logging purposes) for a robot or PC program ON/OFF. The format for the command is `TRACE stepper number : ON/OFF`. The stepper number parameter designates the real value number of the program to be traced.

1 : Robot program
1001: Program 1
1002: Program 2
1003: Program 3

If stepper number is omitted, the program currently running becomes the object of the trace. The ON parameter begins the trace, OFF finishes it.

Executing TRACE ON without first securing user memory by the SETTRACE command will result in error, in which case the SETTRACE command should be entered.

AS LANGUAGE FUNCTIONS

6.0. AS LANGUAGE FUNCTIONS

AS language functions are operators that create a value. Functions are most often program instructions combined as a part of the evaluation and determination process within a program. However, some functions can stand alone in a program. AS language functions are divided into four categories: real value functions, location functions, mathematical functions, and character string functions.

6.1. REAL VALUE FUNCTIONS

6.1.1. SIG FUNCTION

The format for the SIG command is **SIG (*signal number, signal number, ...*).** The SIG command is used to return a logical (true -1 or 0 false) value based on the status of the specified signals. If more than one signal is included in the SIG command, the evaluation performed is a logical AND, all signal states specified in the command must be satisfied for the response to be true.

```
$LIST/P sigstate ↵          LIST/R ↵
.PROGRAM sigstate()         x=-1  y=0
z=-1
11 x= SIG (1001)
12 y= SIG (1004)
13 z= SIG (1020, -1004)
```

Figure 6-1 SIG Function

In the example shown in figure 6-1, the SIG command is used to assign logical true/false values to the real variables x, y, and z. For this example, when the LIST/R command was entered, signal 1001 is ON, signal 1004 is OFF, and signal 1020 is on. The value stored for real variables x and z is -1 (true), the value stored for real variable y is 0 (false).

6.1.2. BITS FUNCTION

The format for the BITS command is **BITS (*starting signal number, number of signals*)**. The BITS command is used to read consecutive binary signals and return the decimal value which corresponds to the binary bit pattern of the specified signals. The *starting signal number* is the first signal number to be read. This signal number is also referred to as the least significant bit in the function. The *number of the signals* to be read identifies how many signals will be evaluated beginning with *starting signal number*. When *number of signals* is not specified, one (1) is assumed. The maximum number of signals that can specified is sixteen (16).

```
$LIST/P bitstate ↵      LIST/R ↵
.PROGRAM bitstate()     x=1 (1008 ON, 1009, 1010, and 1011 OFF)
11   x= BITS (1008,4)   LIST/R
                        x=5 (1008 and 1010 ON, 1009 and 1011 OFF)
                        LIST/R
                        x=10 (1009 and 1011 ON, 1008 and 1010 OFF)
```

Figure 6-2 BITS Function

In the example shown in figure 6-2 the BITS command is used to assign a numeric value to the real variable x. The numeric value of x will change according to the signal state of the signals 1008 through 1011. The LIST/R command is used to show what the numeric return will be based on three different signal state combinations.

6.1.3. TIMER FUNCTION

The format for the TIMER function is **TIMER (timer number)**. The TIMER function returns the current value (in seconds) of the indicated timer. The *timer number* represents the number of the timer to be read. The *timer number* must range from 1 to 10.

Using the TIMER function, the current value of a timer can be read, used in conjunction with a prompt or type command, or used with conditional program instructions. The value of the TIMER indicates the sum of the value last set by the TIMER instruction, and the elapsed time (in seconds) since the last execution of the TIMER instruction.

```
$EX timstate,-1 ↵        Print Output:
.PROGRAM timstate()
...                      The time to complete steps 42 to 46 was 14.58 seconds.
41 TIMER 1 = 0
42 JMOVE aa
43 LMOVE bb
44 JAPPRO pp,200
45 LMOVE pp
46 LDEPART,150
47 PIRNT "The time to complete steps 42 to 46 was", /F5.2,TIMER (1)," seconds."
```

Figure 6-3 TIMER Function

In the example shown in figure 6-3 the TIMER function is used to display the time it takes to complete 5 steps of the program “timestate”.

6.1.4. DISTANCE FUNCTION

The format for the DISTANCE function is **DISTANCE (*transformation value, transformation value*)**. The DISTANCE function calculates the straight line distance between the two transformation locations and returns a value in mm. The order of the two points listed in the command does not affect the value of the result. For example,

x=DISTANCE (Start, HOME)

will assign the value of the straight line distance in mm between the location start and the location *HOME* to the real variable "X".

6.1.5. DX, DY, AND DZ, FUNCTIONS

The format for the DX, DY, and DZ function is **DX (*transformation value*)** or **DY (*transformation value*)** or **DZ (*transformation value*)**. The DX, DY, and DZ functions calculate the X, Y, or Z component base coordinate value in mm. The result can be assigned to a real variable. For example,

y=DX (Start)

will assign the X component base coordinate value in mm to the real variable "y".

6.1.6. ASC FUNCTION

The format for the ASC function is **ASC (*character string, character number*)**. The ASC function is used to return the ASCII value of the specified character (*character number*) in the given *character string* as a real value. If the *character number* is omitted, the ASCII value of the first character will be returned. For example, the command

w = ASC ("sample", 2)

will assign the ASCII value of character "a" (97) to real variable "w". ASCII values can be found in the appendix of this manual.

6.1.7. LEN FUNCTION

The format for the LEN command is **LEN (*string*)**. The LEN function is used to return the number of characters which are contained in the specified *string*. For example, the command

r = LEN (“sample”)

will assign a value of 6 to real variable “r”.

6.1.8. TRUE FUNCTION

The **TRUE** function is used to return a value of -1.0 which represents the logical value of a TRUE or ON condition. For example,

r = TRUE

will assign a value of -1.0 to the real variable “r”.

6.1.9. FALSE FUNCTION

The **FALSE** function is used to return a value of 0.0 which represents the logical value of a FALSE or OFF condition. For example,

w = FALSE

will assign a value of -1.0 to the real variable “w”.

6.1.10. INT FUNCTION

The format for the INT command is **INT (*expression*)**. The INT function is used to return nearest integer value of the *expression*, rounded down from the left of the decimal point. For example:

x = INT(0.123) **0 is returned for x**

y = INT(10.8)	**10 is returned for x**
w = INT(-5.76)	**-5 is returned for x**
t = INT(3.125E+2)	**131 is returned**
INT(cost+0.5)	**The value of cost, rounded to the nearest integer, is returned.**

6.1.11. TASK FUNCTION

The format for the TASK command is **`TASK(task number)`**. The TASK function is used to return the execution status of a PC program. A 1 or 2 is entered to represent the robot and 1001, 1002, 1003 are used to represent the PC program.

No execution returns 0
Program executing returns 1
Program in hold returns 2
Program waiting for stepper 3

6.1.12. ERROR FUNCTION

The **`ERROR`** function is used to return the error code number and message being generated. For example, the command

TYPE $ERROR(ERROR)

will type the error code number and message currently being generated.

6.1.13. SWITCH FUNCTION

The format for the SWITCH command is **`SWITCH(switch name)`**. The SWITCH function is used to return a -1.0 (ON) or 0.0 (OFF) value of the identified switch. For example:

d = SWITCH(CP)	**If the CP switch is ON, a value of -1.0 is returned as the value of "d". If the CP switch is OFF, a value of 0.0 is returned as the value of**

"d".

6.1.14. MAXVAL FUNCTION

The format for the MAXVAL command is **MAXVAL (*real variable 1, real variable 2, ... etc.*)**. The MAXVAL function is used to return the highest value of the identified real variables. For example, if *a=10*, *b=15*, and *c=20*, the command

m = MAXVAL (a,b,c)

will assign a value of 20 to the real variable "m".

6.1.15. MINVAL FUNCTION

The format for the MINVAL command is **MINVAL (*real variable 1, real variable 2, ... etc.*)**. The MINVAL function is used to return lowest value of the identified real variables. For example, if *a=10*, *b=15*, and *c=20*, the command

m = MINVAL (a,b,c)

will assign a value of 10 to the real variable "m".

6.1.16. DEXT FUNCTION

The format for the DEXT command is **DEXT(location_name , element number)**. A specified element of the specified location is returned. **location_name** : The location by which tries to extract the element is specified (Precision Point and transformation). **element number** : The element which wants to be extracted is specified with the real value.

Element. number	location Transformation	Precision Point
1	X Element	JT1
2	Y Element	JT2
3	X Element	JT3
4	O Element	JT4
5	A Element	JT5
6	T Element	JT6
7	JT7 Element	JT7

For example, assume that the transformation variable “aa” are;0,0,0,-160,0,0,300.
The type (aa, 7) displays 300 of JT7 component.

6.1.17. VAL FUNCTION

The format for the VAL command is **VAL (character string , *code*)**. The real value shown by the character which the specified character string includes is returned. **character string** : Character string constant, character string variable or string expression. ***code*** : Real value expression. Which base is used when the character string is converted is specified by the following figures. (It is time when this is not included for the code by which the base is clearly specified for the character string)
It is considered that 0 was specified excluding the omission and 0, 1, and 2.

Set value.	Numeration system.
0	10 Base.
1	2 Base.
2	16 Base.

Warning : The character string can be specified by expressing the exponent.
The code by which the specified character string shows the numeration system first can

be included. The character which cannot be interpreted as a code by which the numeration system is shown partially of Numeric value perhaps is treated as a sign by which the end of the entire target character string for the conversion is shown.

VAL("123 Elm Street")	Real value 123 is returned.
VAL("1.2E-5")	Real value 0.00001 is returned.
VAL("^OHFF")	Real value 255 is returned.

6.1.18. INSTR FUNCTION

The format for the INSTR command is **`INSTR (`*`search-start_position,`*** **`character_string1, character_string2)`**. The position in which the first character of a partial character string exists in the specified character string is returned. ***`search_start_position`*** : The position of the character by which the search for string expression 2 is started in string expression 1 is specified. The search starts from the head of string expression 1 when omitting it. **`character_string1`** : The character string constant, the character string variable, and the string expression are specified. It is examined whether string expression 2 is included in this character string. **`character_string2`** : The character string constant, the character string variable or the string expression is specified. It is examined where in string expression 1 this character string is. When an empty character string is specified, the value of the search beginning position (1 when omitting it) is returned.
When the character string of string expression 2 is included in the character string of string expression 1, the position with the first character is returned. When the character string of string expression 2 is not included in the character string of string expression 1, 0 is returned as a function value. The search starts from the first character of string expression 1 when the value of the search start position is 1 or is smaller than that of one. Moreover, when the value of the search start position is larger than the length of string expression 1, 0 is returned as a function value. When the agreement of the character string is judged, the capital letter and the small letter are considered to be the same character.

INSTR("file.ext",".")	Real value 5 is returned.
INSTR("file",".")	Real value 0 is returned.
INSTR("abcdefgh","DE")	Real value 4 is returned.
INSTR(5,"1-2-3-4","-")	Real value 6 is returned.

6.1.19. PRIORITY FUNCTION

The format for the PRIORITY command is: **`PRIORITY`**. The priority of the robot program of present is returned. The priority of the robot program being registered now is returned by the real value. There is no priority of the PC program. The priority of the robot program is 0 usually. Priority can be changed by ordering LOCK instruction.

6.1.20. INRENGE FUNCTION

The format for the INRANGE command is **INRANGE (location_name , *precision point*)**. Searches the location is in the work envelope and a fixed value is returned. **location_name** : The location variable name is specified (Precision Point, transformation, and compound precision point). ***precision point*** : When the location name is only a transformation or a compound transformation, it is valid. When this Parameter is specified, the transformation is judged in the form given by this Precision Point.
The transformation is judged in the form of the current position when omitting.

The value which this function returns is as follows.

0	In work envelope.
1	JT1:Outside the work envelope.
2	JT2:Outside the work envelope.
4	JT3:Outside the work envelope.
8	JT4:Outside the work envelope.
16	JT5:Outside the work envelope.
32	JT6:Outside the work envelope.
16384	The interference check is exaggerated.
32768	When the link is composed the robot arm and it does not reach.

Warning : It is the one to search the specified location is in the work envelope in this function. A path to the location on the way is all in the work envelope, it does not understand.
3000

```
IF INRANGE(lc1, #p) GOTO ERR STOP
:
ERR STOP:
TYPE "Location lc1 is outside the work envelope."
PAUSE
```

6.1.21. PLCAIN FUNCTION

The format for the PLCAIN command is **PLCAIN (integer_input_data_number)**. The value of the specified integer input data number is returned.

`integer_input_data_number` : The number of input data is specified with the integer. Range of specification : 1~32.

Warning : "Built-in sequencer function". It is a valid only to turn on the option.
It becomes error (-896) "Because the option is not set up, can't execute" when Option is off.

aa=PLCAIN(AS) The value set in the 12th integer input data is returned.

6.1.22. UTIMER FUNCTION

After the @*timer* variable has been set, the UTIMER command is used to return the current value of the timer variable. The format for the UTIMER command is `UTIMER (@timer variable name)`. The @timer variable name parameter is used to designate timer variables assigned by the UTIMER command. Denotes an integer variable name.

6.1.23. SYSDATA FUNCTION (OPTIONAL)

The SYSDATA command (pus keyboard) returns internal as parameters. The format for the command is `SYSDATA (keyword , opt1 , opt2 )`. We describe the key word, opt1, and opt2 of the parameter as follows.

M.SPEED : The SYSDATA function returns monitor speed in a percentage. If executed when *not* in a motion step, however this command will return a value of - 1. *opt1* designates time robot number. If omitted, this number defaults to robot 1. *opt2* is not yet in use, and can be omitted.

MSTEP : The SYSDATA function returns the motion step number of a robot in mid motion step or whose step has already been completed. If executed when *not* in motion step, however, this command will return a value of -1. *opt1* designates the robot number. If omitted, this number defaults to robot 1. *opt2* is not yet in use, and can be omitted.

STEP : The SYSDATA function returns the step number of a robot in mid run step, or whose step has already been completed. If executed when not in a motion step, however, this command will return a value of -1 (PRIME is not executed). opt1 sets robot number or PC task number (from 1001 to the PC program number) If omitted

ROBOT 1 is selected by default. opt2 is not yet in use, and may be omitted.

P. SPEED : The SYSDATA function returns the motion speed (in percent) of an operation in progress, or already completed. When operation speed (in seconds) is set, a value of -1 is returnded. opt1 sets robot number, which will default to ROBT1 if no selection is name. opt2 is not yet in use, and may be omitted.

P. SPEED.M : The SYSDATA function returns motion speed in millimeters per second. When operation speed is set in seconds, a value of -1 is returned. opt1 is used to set robot number. If omitted ROBOT1 is chosen by default. opt2 is not yet in use, and may be omitted.

6.1.24. MSPEED FUNCTION (OPTIONAL)

6.1.25. MSPEED2 FUNCTION (OPTIONAL)

These commands return current monitor speed, a value of 0 - 100%. `MSPEED` is for ROBOT 1, `MSPEED2` for ROBOT 2.

6.2. LOCATION FUNCTIONS

6.2.1. DEST FUNCTION

The **DEST** command returns the transformation location which represents the planned destination location of the current robot motion.

The DEST function can be used to indicate the destination to which the robot was moving after the robot motion was interrupted for some reason. This location value can be used for all motion instructions. The location where the robot actually stops and the location that the DEST function returns to are not always identical. If the robot motion has been interrupted for some reason (for example, when the HOLD/RUN switch on the panel is turned to the HOLD position), the robot will stop immediately and the DEST function will return the robot to the original destination location.

6.2.2. DEST FUNCTION

The **#DEST** command returns the precision location which represents the planned destination location of the current robot motion.

The #DEST function can be used to indicate the destination to which the robot was moving, after the robot motion was interrupted for some reason. This location value can be used for all motion instructions. The location where the robot actually stops and the location that the #DEST function returns are not always identical. If the robot motion has been interrupted for some reason, (for example, when the HOLD/RUN switch on the panel is turned to the HOLD position) the robot will stop immediately and the #DEST function will return the robot to the original destination location.

The DEST function is also used to continue the motion that was interrupted because of an ONI CALL program instruction. By inserting the following steps in an interrupt subroutine, the interrupted motion can be resumed. For example:

```
POINT save = HERE     ; Store current location in the variable “save”
POINT old = DEST      ; Stores the destination location in  the variable “old”
JDEPART               ; Backs the tool away by 50 mm
```

AS LANGUAGE FUNCTIONS

JMOVE old ; Starts motion to the planned destination

6.2.3. FRAME FUNCTION

The format for the FRAME command is **`FRAME(location1, location2, location3, location4)`**. The FRAME function is used to return transformation values that represent a unique relative coordinate system. In the FRAME function, *location1* and *location2* are the transformation location values which determine the direction of the X-axis of the new coordinate system. The direction of the X-axis will be from *location1* to *location2*. The *location3* transformation value determines the direction of the Y-axis of the new coordinate system. The direction of the Y-axis is determined such that the X-Y plane contains *location1, location2, and location3*. The *location4* transformation value will be the origin of the new coordinate system. Figure 6-4 shows how a relative coordinate system can be defined with the FRAME function.

POINT F1 = FRAME(O1,X1,Y1,O1)

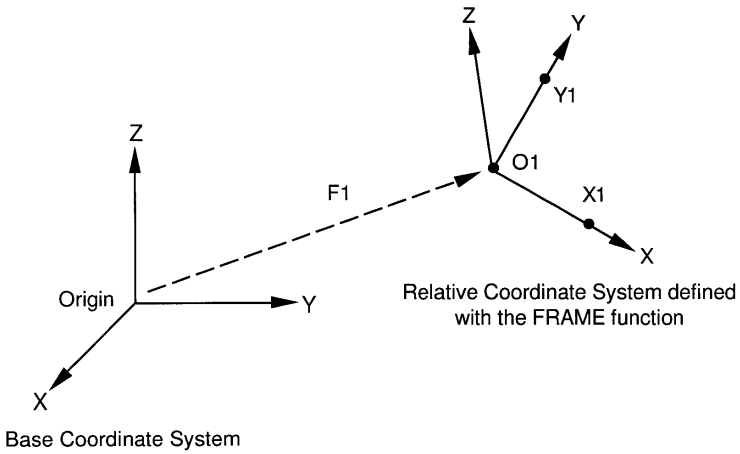


Figure 6-4 FRAME Function

The following section of a program is an example of a palletizing program that uses the FRAME function. The FRAME function must be used with a compound transformation, precision points will not be affected by the FRAME command.

```
;POINT pal = FRAME (org,x,y,org)
POINT put = start
FOR lay = 0 TO lay.max
    FOR col = 0 TO col.max
        FOR row = 0 TO row.max
            POINT put = SHIFT(start BY100*row, 75*col, 200*lay)
            POINT put_pt = pal+put
            JAPPRO #a, 100
            LMOVE #a
            LDEPART 100
```

AS LANGUAGE FUNCTIONS

```
            JAPPRO put_pt, 100
            LMOVE put_pt
            LDEPART 100
        END
    END
END
```

The FRAME function is used when the orientation of the tasks being performed are not parallel and perpendicular to the base coordinate system. To improve the accuracy of the new coordinate system, the four *locations* should be defined at positions as far away from one another as possible. The tool definition and orientation must also be considered when defining locations that will be used by the FRAME function to define a relative coordinate system.

6.2.4. NULL FUNCTION

The **NULL** function is used to return a transformation value with all component values being 0. The NULL function can be used as a convenient method for generating a transformation value with no rotational component when using the shift function. For example the command

Point new = SHIFT (NULL BY x.shift, y.shift, z.shift) + old

will define the transformation variable “new” having only the specified X, Y, and Z components shifted from the transformation location “old”. In the example below, the NULL function is used to return a distance from the origin of the base coordinate system to the transformation location “test.loc”.

dist = DISTANCE (NULL, test.loc)

6.2.5. HERE FUNCTION

The **HERE** function is used to return the current transformation location of the tool center point. In the following example, the HERE function is used to calculate the distance from the robot’s current position to a previously defined transformation location named “spot”.

dist = DISTANCE (HERE, spot)

The value (distance in mm) assigned to the real variable *dist* can be utilized by the programmer in a number different ways, including: printing the calculation, using the information in a mathematical expression, or the calculation could be utilized in a logical evaluation.

6.2.6. #HERE FUNCTION

The **#HERE** function is used to return the current precision location in joint angles of the robot. The #HERE function can be used to assign precision locations to previously undefined precision locations.

6.2.7. TRANS FUNCTION

The format for the TRANS command is **`TRANS (X component, Y component, Z component, O component, A component, T component)`**. The TRANS function returns a value of a transformation location which consists of the coordinate system components (X, Y, Z) and rotation (O, A, T). The TRANS function can be used to assign location components or in conjunction with the DECOMPOSE command to assign elements of an ARRAY to specified locations. In the following example,

POINT wing = TRANS (s[0],s[1]+200, s[2]+300, s[3], s[4],s[5])

the transformation components of the ARRAY variable “s” are assigned to the transformation location named “wing”. In this example, 200 mm are added to the Y component and 300 mm are added to the Z direction component.

6.2.8. #PPOINT FUNCTION

The format for the #PPOINT command is **`#PPOINT (jt1, jt2, jt3, jt4, jt5, jt6)`**. The #PPOINT function returns the precision location which consists of the given components. The precision component of each joint of the robot is represented as *jt1, jt2, jt3, jt4, jt5*, and *jt6*. If a value for a specified joint is omitted, it is set to zero (0). The #PPOINT command can be used to position the robot to specific joint angles or to

manipulate the joint angles of previously defined precision points.

6.2.9. RX, RY, RZ FUNCTION

The format for the RX, RY, and RZ function is **RX (*angle*), RY (*angle*), and RZ (*angle*)**. The RX, RY, and RZ functions return transformation values that represent rotation around the specified axis of the base coordinate system. For example, the command

POINT crnr = RX(45)

will assign a transformation value which is 30 degrees of rotation about the X axis of the base coordinate system to the transformation location named *crnr*.

6.2.10. SHIFT FUNCTION

The format for the SHIFT command is **SHIFT (*transformation_location BY X_shift, Y_shift, Z_shift*)**. The SHIFT function returns a transformation value which represents the shifted position from the specified *transformation location* by the given shift amounts. The *X_shift, Y_shift*, and *Z_shift* elements of the command represent the shift value to be added to the corresponding transformation components of the specified *transformation location*. The following program section is an example of a palletizing program that uses the SHIFT function.

```
POINT put = start
    FOR col = 0 TO 5
        FOR row = 0 TO 10
            POINT put = SHIFT(start BY100*row, 75*col, 0)
            JAPPRO #a, 100
            LMOVE #a
            LDEPART 100
            JAPPRO put, 100
            LMOVE put
            LDEPART 100
        END
    END
```

In this example, the location “start” is assigned to the location “put” with the point command. Two FOR TO control flow structures are established, the structure to fill up the rows is nested so that a complete row will be filled before the program shifts to fill the next column. Each time the instructions are processed the location start is moved 100 mm in the row direction and 75 mm in the column direction. Six rows and eleven columns will be filled before the program will process instructions outside of the two loops defined with the FOR TO structure.

6.2.11. AVE_TRANS FUNCTION

The format for the AVE_TRANS command is **AVE_TRANS (*transformation 1, transformation 2*)**. The AVE_TRANS function returns the average X, Y, and Z

component location of the two specified locations.

6.2.12. BASE FUNCTION

The **BASE** function returns the location components of the defined BASE.

6.2.13. TOOL FUNCTION

The **TOOL** function returns the location components of the defined TOOL.

6.2.14. TRPOINT FUNCTION

The format for the TRPOINT command is: **`TRPOINT (transformation_value 1 , transformation_value 2)`**. Returns the compound transformation of transformation 1 and transformation 2. **`transformation_value 1, transformation_value 2`** : Transformation which tries to do compound transformation.
When the compound conversion which considers the traverse joint is done, you uses it.
The value of the traverse joint reaches the value of the traverse joint of transformation 2.

p operates on doing the teaching and the f1 frame on frame f0.

TRHERE f0+p HERE f0+p
LMOVE TRPOINT(f1,p) LMOVE f1+p

6.2.15. TRADD FUNCTION

The format for the TRADD command is **`TRADD(transformation_value)`**. The traverse joint element is added to X element of the transformation.
`transformation_value` : Transformation which wants to add running element.

6.2.16. TRSUB FUNCTION

The format for the TRSUB command is **`TRSUB (transformation_value)`**. The traverse joint element is subtracted from X element of the transformation.
`transformation_value` : Transformation which wants to subtract running element.

6.2.17. #HOME FUNCTION

The format for the #HOME command is **#HOME (*home position*)**. The location of Home Position being set now is returned. **home position** : Home Position is set. It becomes one when omitting .

1: The value of Home Position 1 set by instructing in SETHOME is returned.

2: The value of Home Position 2 set by instructing in SET2HOME is returned.

The location of Home Position being set now is returned as Precision Point.

Warning : The function of # HOME is treated as a location of Precision Point. Therefore, when this function is used, # should be applied to the head.

For example, It does not operate directly but it rises from a present position to equal height to the location of Home Position to the location of Home Position when moving to the location of Home Position. It assumes to the location of Home Position that it wants to operate afterwards. An undermentioned program achieves such a robot motion.

```
LCCONV homepos = #HOME(1)
IF DZ(homepos) > DZ(HERE) THEN
HERE tmppos
POINT/Z tmp = homepos
LMOVE tmp
END
HOME
```

6.2.18. CCENTER FUNCTION (OPTIONAL)

The CCENTER command returns the center point of an circle defined by three points. The format for the command is `CCENTER( transformation1 , transformation2 , transformation3 , posture transformation)`. The three points of the arc are plotted via transformations 1,2 and 3. The posture is specified by the *posture transformation parameter*.

6.2.19. CSHIFT FUNCTION (OPTIONAL)

The CSHIFT command returns a point shifted from an object point towards the center point of an circle defined by three points. The format for the command is `CSHIFT (transformation1 , transformation2 , transformation3 , object transformation BY amount of shift)`. The three points of the arc are plotted by transformations 1,2 and 3. The object point is specified by the *objet transformation* parameter. Shift amount is specified in the *amount of shift* parameter.

6.3. MATHEMATICAL FUNCTIONS

The following table identifies the mathematical functions, provides a brief description of each function and shows the format for mathematical functions for programming purposes.

Table 6-1 Mathematical Functions

Keyword	Function	Format
ABS	Returns absolute value of specified numerical expression.	ABS(value)
SQRT	Returns square root of numerical expression.	SQRT(value)
PI	Returns constant p (3.1415...).	PI
SIN	Returns sine value of specified angle.	SIN(angle)
COS	Returns cosine value of specified angle.	COS(angle)
ATAN2	Returns the arctangent value of an angle whose tangent equals v1/v2.	ATAN2(v1,v2)
RANDOM	Returns random number between 0.0 and 1.0.	RANDOM

6.4. CHARACTER STRING FUNCTIONS

6.4.1. $CHR FUNCTION

The format for the $CHR command is **`$CHR (real_value_expression)`** . The ASCII character string of one character with the given ASCII value is returned.

real_value_expression : Numeric value which wants to be converted into the ASCII character is specified. The value which can be specified is a range from 0 to 255.

When the specified ASCII value is a range from 0 to 255, the corresponding ASCII character string of one character is returned.

$CHR(65) The ASCII value returns the character “A” of 65.
$CHR(^H61) The ASCII value returns the character “a” of 97.

6.4.2. $LEFT FUNCTION

The format for the $LEFT command is **`$LEFT (string_expression , extracted_character_number)`**. The character string of specified character’s worth is returned from the left of the specified character string. **`string_expression`** : The character string constant, the character string variable or the string expression is specified. **`extracted_character_number`** : The number of characters which wants to be extracted from the left of the character string is specified. When 0 or smaller value than 0 is specified, dead letter is returned.

The character string for a few minutes of the extraction character is returned from the left of the specified character string. When the values of the number of extraction characters are larger than the length of the string expression, the entire string expression is returned.

$LEFT(“abcdefgh”, 3) The character string “abc” is returned.
$LEFT(“*1*2*3*4*5”, 15) “*1*2*3*4*5” of the entire character string is returned.

6.4.3. $RIGHT FUNCTION

The format for the $RIGHT command is **`$RIGHT (string_expression, extracted_character_number)`**. The character string of specified character’s worth is returned from the right of the specified character string. **`string_expression`** : The character string constant, the character string variable or the string expression is

specified. **extracted_character_number** : The number of characters which wants to be extracted from the right of the character string is specified. When 0 or smaller value than 0 is specified, dead letter is returned.
The character string for a few minutes of the extraction character is returned from the right of the specified character string. When the values of the number of extraction characters are larger than the length of the string expression, the entire string expression is returned.

$RIGHT("abcdefgh" 3) The character string "fgh" is returned.

6.4.4. $SPACE FUNCTION

The format for the $SPACE command is **$SPACE (blank_character_number)**.
The specified blank character for a few minutes of the character is returned.
blank_character_number : A blank number of characters is specified.
0 or positive value.
For example, "a dog" is appeared in type "a" + $SPACE(1) + "Dog".

6.4.5. $MID FUNCTION

The format for the $MID command is **$MID (string_expression , *real_value_expression , extracted_character_number*)**.
A partial character string of the specified character string is returned.
string_expression : The character string variable, the character string constant or the string expression is specified. A partial character string is taken out of this character string. ***real_value_expression*** : Real value expression which can be omitted. Characters in the character string how many whether is assumed to be the first character of a partial character string is specified. ***extracted_character_number*** : Real value expression which can be omitted. The number of characters taken out as a partial character string is specified.
A partial character string starts from the first character of the specified character string when the real value expression is omitted or the value of one or less. The number of extraction characters returns and this function returns dead letter when the character string for which the value of the real value expression is specified is larger than length (number of characters) at 0 or negative time. When the number of characters from the first character which omits the number of extraction characters or is specified to the character end is fewer than the values of the number of extraction characters, the

character string obtained as a result becomes the one which contains the end of the specified character string. That is, the error might not be generated and be delayed to the length which the result specified as a result.

For example, the shown instruction substitutes the character string “cd” for character string variable $substring. Because, the character string of two characters which starts from the third character of the character string “abcdef” is “cd”.

$substring= $MID(“abcdef” 3,2)

6.4.6. $DECODE FUNCTION

The format for the $DECODE command is **$DECODE (Character string variable , Delimiter character , *mode*)**. Part of character string delimited by "Delimiter character" is taken out from among the character string. **Character string variable** : Character string variable which tries to be examined. The parts of the remainder except the part returned as a function value enter in this variable among original character strings after this function is processed. **Delimiter character** : A necessary character string is defined and the character used to delimit it is defined from among the input character string (That is, each character included in this character string becomes "Delimiter character".)
mode : Real value by which operation done by this function is decided. When the value of the mode is negative or is 0 or is omitted, the part to the first "Delimiter character" in the input character string is returned as a function value and the part is removed from the input character string. If the value of the mode is positive, the part to the first "Characters other than the delimiter character" in the input character string is returned as a function value and the part is removed from the input character string. In other words, the "Delimiter" provided at the head of input character string (If plural numbers of the "Delimiter" are being arrayed; all) is returned as a function value.
This function is used to take out character strings up to dissected by any of the characters designated as "Delimiter" after examining the input character string. When this function is returned as a portion of character string, simultaneously the partial character string is deleted from the input character string. The character string which is returned as a function value (simultaneously deleted) does not contain "Delimiter" or can set up a character string which consists of only "Delimiter" This means that, this function takes out all characters_; normally this portion is necessary_, or all "Delimiter" existed at the head of character string_, normally this portion is discarded,_ is taken out.

Warning : This function returns the character string and changes the content of the input character string. It does not relate whether the character in the character string which provides for the character in the input character string or the delimiter character is a capital character or small character when searching for the delimiter character.
For example, the instruction shown below sequentially takes out the figure delimited by the blank or the comma of character string $input. The first figure in $input is taken out and it substitutes the first instruction in the DO structure for the variable named $temp

(At this time, the part is deleted from $input). Next, it converts the character string of the figure obtained now into the real value corresponding to it by using the function named VAL. it substitutes it for the following element of the array “Value” of the real value. And, it looks for the character to which Numeric value is delimited by using the $DECODE function again.
(This character is thrown away.)

```
i=0; Counter clear.
DO
$temp=$DECODE($input,”,”, 0)    ; Takes out character string up to
value[i]=VAL($temp)             ; Converts into the real value.
if $input ==”” GOTO 100
$temp = $DECODE($input,”,”, 1)  ; Takes out,.
i=i+1                           ; Increment of counter
UNTIL $input==””                ; Continues process until characters are consumed.
100 TYPE “END”
```

Some Numeric values (As a character string) say to $input. Those Numeric values assume the delimitation by combining the blank and the comma. Let’s assume that a for example and following Numeric values (character string) entered.

1234, 93465.2, .4358, 3458103,

If an instruction on is executed about this character string, the value of four elements of the start of the array “Value” is as follows.

value[0] 1234.0
value[1] 93465.2
value[2] 0.4358
value[3] 3458103.0

Moreover, the content of character string variable $input becomes dead letter (“”) as a result.

6.4.7 $ENCODE FUNCTION

The format for the $ENCODE command whose formats **`$ENCODE (output data)`**. Returns the character string generated according to the *output data* parameter. This character string is the same as that output by the `TYPE` command. **`output data`** can be of the following types:

- String expression
- Real value expression (Calculated value)
- Format specification code (for determining the format of output messages)

Note: Multiple output data parameters must be separated by commas.

AS LANGUAGE FUNCTIONS

The $ENCODE function allows the type of character string usually obtained by the type instruction to be generated within a program. In other words, it creates an `internal` rather than external string. The format designation codes below stipulate the form in witch numerical values are expressed. This setting remains in effect until another format is specified. No matter witch format is specified. If numeric output is too large to be displayed in the assigned space, an asterisk (*) will appear in the general area of the value in question.

/D	Specifies output to be in the default format. This output, while resembling that of /G15.8 differs in that (1) numeric-final zeroes are omitted, and (2) only 1 space is left to separate numeric values all others are omitted.
/Em.n	Numeric values are expressed using exponents. (for example `-123.4E+2`) `m` represents the total number of all numeric letters in the viewing area, and n the number of digits after decimal point. The value of in should be fives times (or more) larger than n.
/Fm.n	Numeric values are expressed in fixed decimal point. (for example -123.45) `m` represents the total number of all numeric letter in the viewing area, and `n` the number of decimal point.
/Gm.n	Values of figure n within character field m and located after the decimal point will if possible, be displayed in F format. If this format is not posture output will be in E m.n format.
/Hn	Numeric figures of n are shown as hexadecimal integers.
/In	Numeric figures of n are shown as decimal integers.
	The following format specifications are used to insert a special character into a character string.
/Cn	The carriage return (CR) and line feed (LF) characters are inserted n times. *n* number of blank spaces will also be displayed if the character string received by the &ENCODE function has this format setting at either the head or tail of the function parameters when output to the terminal. In all other cases, *n*-1 blank spaces are displayed.
/Xn,n	*n* number of blank spaces is inserted.
/Jn (optional)	Though resembling Hn, /Jn differs in that numerical values within the *n* character area are displayed as hexadecimal numbers. Zeroes are embedded within blank spaces.
/Kn (optional)	Though resembling in /Kn differs in that numerical values within the *n*

character area are displayed as decimal numbers. Zeroes are embedded within blank spaces.

/L (optional) /L differs from /D in that the latter removes all blank spaces save one directly before in a numerical value. /L removes *all* blanks spaces.

$output = $output + $ENCODE(/F6.2,count)

This expression attaches the character string displayed in the format that set the value of real number variable `count` to the tail end of the character string inserted into variable `$output`.

6.4.8. $ERROR FUNCTION

The format for the $ERROR command is **`$ERROR (error code),`** which returns the error message of the specified error code. *Error code*, a negative real number, specifies the error message to be returned.

6.4.9. $DATE FUNCTION

The format for the $DATE command is **`$DATE (date format),`** which returns the system date in a specified format. Format is assigned by entering a real value of 1 through 3 as follows.

$DATE(1) :
outputs date in *mm/dd/yyy* format (For example, March 10, 2001 is output as 03/10/2001)

$DATE(2) :
output date in *dd/mmm/yyyy* format (For example, March 10, 2001 is output as 10/MAR/2001) Abbreviations for the months of January to December are as follows: JAN / FEB / MAR / APR / MAY / JUN / JUL / AUG / SEP / OCT / NOV / DEC.

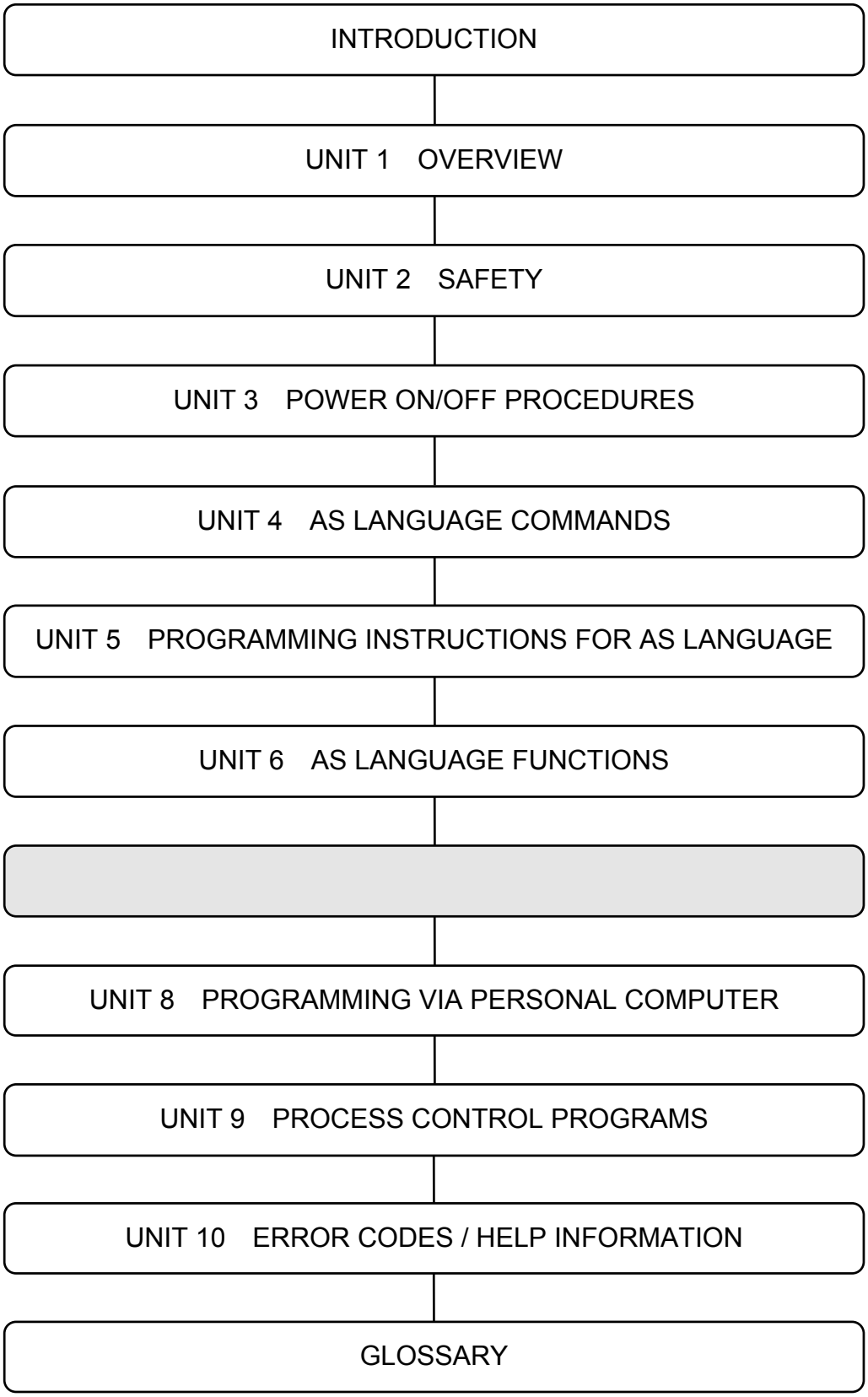
$DATE(3) :
output date in *yyyy/mm/dd* format (For example, March 10, 2001 is output as 2001/03/10)

6.4.10. $TIME FUNCTION

Return system time as a character string in *hh:mm:ss* format (For example, PM 6:27 and so seconds is output as 18:27:50, the 24 hours are represented by 23 (npm)

AS LANGUAGE FUNCTIONS

MEMO


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION
GLOSSARY

CREATING AND EXECUTING PROGRAMS

AS LANGUAGE FUNCTIONS


MEMO

7.0. CREATING AND EXECUTING PROGRAMS

An AS language program is a collection of instructions which directs the system to move the robot, interface with other automation devices, evaluate variables, perform mathematical calculations, execute logic instructions, and set external signals. AS language provides programmers the options and flexibility to program for unique applications and requirements.

7.1. TYPES OF AS LANGUAGE PROGRAMS

7.1.1. ROBOT CONTROL PROGRAMS

A robot control program is a program which results in the direct movement of the robot. In the robot control program, all program instructions including robot motion instructions can be used.

7.1.2. PC PROGRAM (PROCESS CONTROL)

A PC program is a program which can be executed simultaneously with a robot control program. A PC program is used to monitor or to control external devices through external binary signals. PC programs can evaluate variables, perform mathematical calculations, execute logic instructions, and set external or internal signals.

Unlike robot control programs, PC program instructions cannot cause robot motion. The only exception to this requirement for PC programs is the BRAKE instruction. In addition, the BASE and TOOL instructions are not available for PC programs. PC programs can be used to display messages on the terminal by means of the PRINT instruction. All internal and external binary signals can be used in PC programs.

7.1.3. SLOGIC PROGRAMS

SLOGIC programs are used to provide logic instructions for remote IO or ControlNet

communication. SLOGIC programs execute from the optional 1FS board and provide instructions for signal mapping, timers and counters similar to ladder logic programs. SLOGIC programs are created on the 1GA board and downloaded to the 1FS board.

7.2. PROGRAM STEP FORMAT

Each step of an AS language program has the following format:

step number label program instruction ; comment

1. Step number
   A step number is assigned to each line of a program. Steps are numbered consecutively beginning with 1. Step numbers are automatically adjusted whenever lines are inserted or deleted. The only step number that can be blank (contain no information), is the step after the last step of the program.

2. Label
   Step numbers cannot be used as branch destinations with the program instruction GOTO. Therefore, labels are used for identifying specific locations within a program for the purpose of branching to other parts of a program. A label can be either an integer from 1 to 9999 or a character string up to 15 characters. Labels are inserted at the beginning of a program line.

3. Comment
   A semicolon (;) indicates that all of the information to the right of the semicolon is a comment. Comments are not processed as program instructions when the program is executed. Comments are used by programmers to help explain the contents of the program. Comment lines with no label and no instruction can be created for improved program readability.

7.3. PROGRAM EXECUTION

NOTE

Many operators utilize a “mainline” program to select all robot control programs. While not a necessity, mainline programs are routinely named pg00. If you robot is integrated into a system that is designed to operate with a mainline program, it is important that the mainline program is selected to start production. If an individual program is selected and run independent of the mainline program, the required operations of the mainline program will not be processed.

7.3.1. EXECUTING ROBOT CONTROL PROGRAMS

In order to execute a robot control program, first select the program to be run. If the program to be executed is already selected the monitor command EXECUTE can be used. This will run the program one time and a message will be displayed indicating the program was completed. The EXECUTE command can also be used with the optional argument of a program name to run a program that is not on the stack. The number of times a program is to be executed can be entered after the program name, to specify the program is to be run continuously a -1 is entered.

The PRIME command, in conjunction with the CYCLE START button, can also be used when executing programs The PRIME command is used to specify the step of the program where execution will start, if no number is specified with the PRIME command, program execution will start with the first step of the program. When a program is run for the first time or for debugging a program, it is a good idea to confirm each program line by line using the STEP command or the CHECK key on the multi function panel.

7.3.1.1. STOPPING ROBOT CONTROL PROGRAMS

An EMERGENCY STOP button (located on the multi function panel and controller cabinet) should be pressed anytime an operator needs to stop robot motion immediately. However, it is recommended that the emergency stop buttons not be used as a routine method of stopping robot motion. When an emergency stop button is pressed, power to the motors is immediately turned off and the brakes applied. Because normal deceleration of the robot does not occur in an emergency stop, the mechanical unit may be subjected to severe dynamic shock loads.

The AS language command ABORT can be used to stop program execution. With this command, the robot will complete the current instruction and stop after a normal deceleration to the taught location. Motor power will remain on when the ABORT command is issued. Motor power can then be turned off by pressing an EMERGENCY STOP button.

If the running program is to be stopped using the available switches, the following procedure can be followed:

1. Switch the HOLD RUN switch to the HOLD position, deceleration of robot motion will begin at the point that the switch is turned to hold, the robot will come to a complete stop after deceleration. The MOTOR POWER and CYCLE START lamps will remain on.

2. Pressing the EMERGENCY STOP button will turn off the cycle start lamp, remove motor power from the servo motors and apply the mechanical brakes.

3. Control power can now be turned off.

To restart a program from the point it was stopped, move the HOLD RUN switch from the HOLD to the RUN position while the motor power and CYCLE START lamps are on. This will resume the program execution from the point it was stopped when the HOLD RUN switch is turned to RUN. The CONTINUE command can also be used to resume program execution.

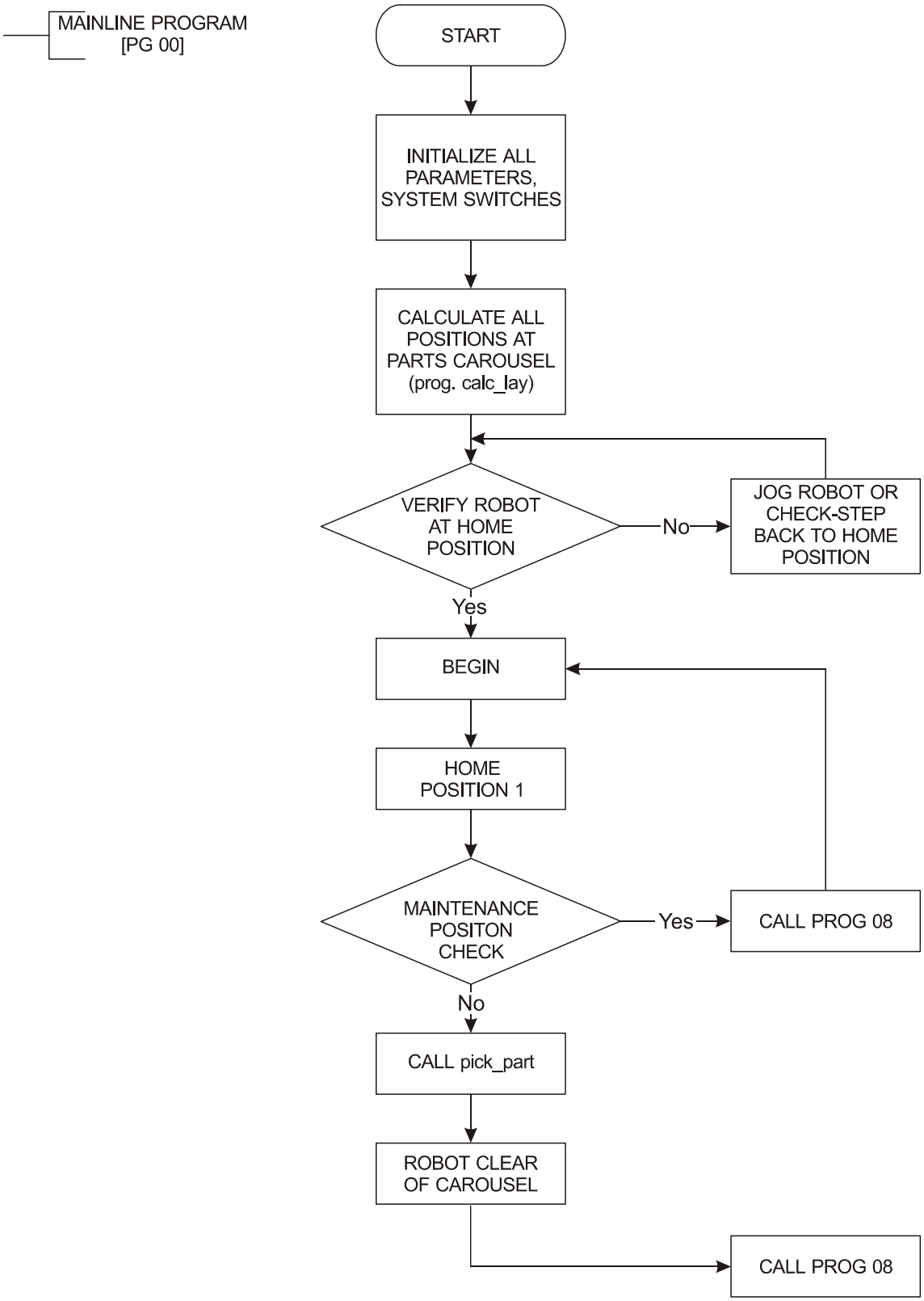
7.4. EXECUTING PC PROGRAMS

PC programs are executed by the PCEXECUTE monitor command, or when a PCEXECUTE instruction is processed in a robot control program. The PCABORT command stops execution of the PC program immediately. The PCEND command stops execution of the specified PC program at the end of the current cycle. The PCCONTINUE monitor command resumes execution suspended by the PCABORT command or because of an error.

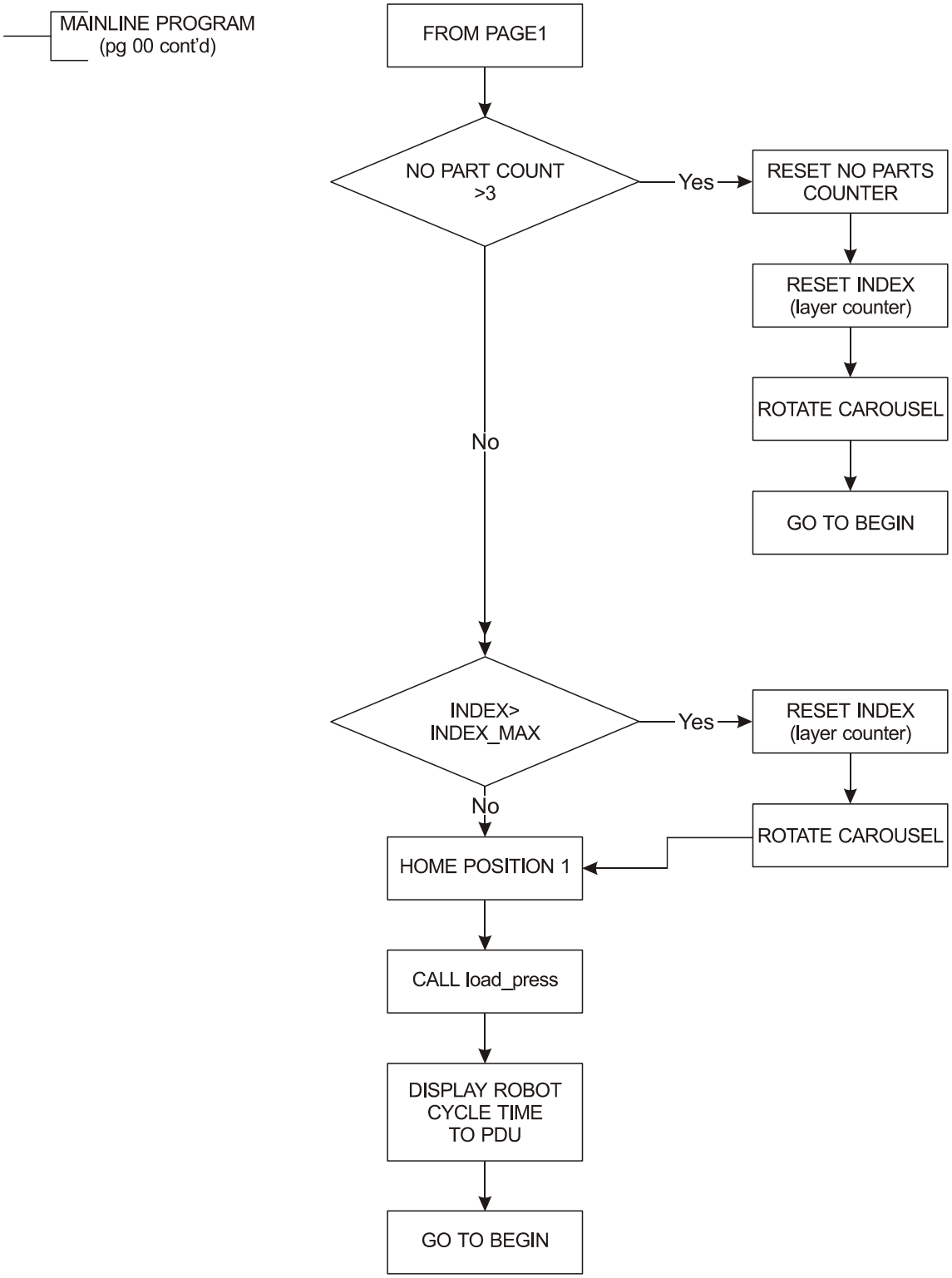
7.5. FLOWCHARTING

The procedure for writing complicated AS language programs often begins with creating a flowchart to identify the major elements of the process. AS language program(s) are then written to meet the objective identified in the flow chart. Figure 7-1 shows an example of a working flowchart.

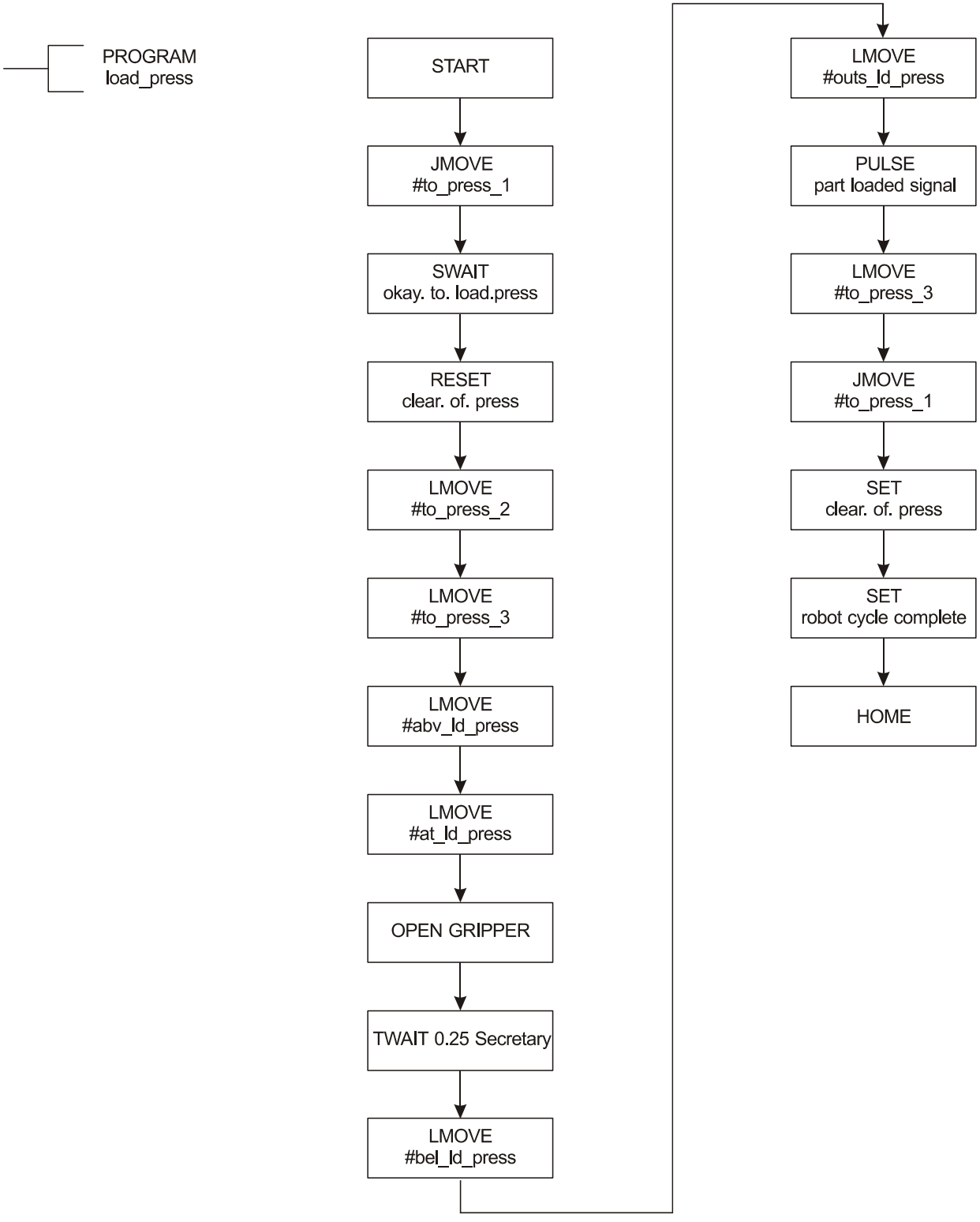
CREATING AND EXECUTING PROGRAMS



CREATING AND EXECUTING PROGRAMS



CREATING AND EXECUTING PROGRAMS



7.6. SAMPLE PROGRAMS

```
.PROGRAM bar.code()
;Robot position is set relative to the bar code reader focal length.
;
      code.fault = FALSE
      SPEED 5
      LDEPART 30
      BREAK
      TYPE /C6, “Ceading bar code”
      PULSE 8,0.05
      ONI 1019 GOTO 100
      WAIT SIG(1008) ;Wait for plc to confirm a good read.
      RETURN
100   CALL code.fault
      RETURN
.END


.PROGRAM code.fault()
;Bar code fault motion recovery program.
;
      WEIGHT load
      SPEED 100 ALWAYS
      ACCURACY 100 ALWAYS
      TOOL reader
;
      TYPE /C6, ”A bar code read fault has occurred, look to the left to correct.”
      BREAK
      SPEED 5
      LDEPART 250
      SPEED 10
      LMOVE #top.stack
      SPEED 25
```

```
    JMOVE #safe.sta.1
    SPEED 30
    HOME
    BREAK
    code.fault = TRUE
.END


.PROGRAM cut.01()
;Lite cutout program #01
    WEIGHT load
    SPEED 100 ALWAYS
    ACCURACY 500 ALWAYS
    TOOL router1
;
    SIGNAL -21 ;Not clear of sta.#2
    SPEED 75
    JAPPRO door.or.x.y+cut1[1]+TRANS(0,0,0,o,a,t),50
    CLOSEI 1
    SPEED 25 mm/s
    FOR x = 1 TO 5
    IF right THEN
    y = (6-x)
    ELSE
    y = x
    END
    ACCURACY 1 ALWAYS
    LMOVE door.or.x.y+cut1[y]+TRANS(0,0,zoff,o,a,t)
    SPEED ctspd mm/s ALWAYS
    BREAK
    TWAIT delay
    END
    BREAK
    SPEED 10
    LDEPART 75
```

CREATING AND EXECUTING PROGRAMS

```
        SPEED 20
        ACCURACY 500 ALWAYS
        JDEPART 25
        BREAK
        CALL poke.waste
        OPENI 1
        SPEED 100 ALWAYS
;
        SPEED 75
        JMOVE #router.safe
        BREAK
.END
```

```
.PROGRAM main()
;Door routing cell
;
100   WEIGHT load
      SPEED 100 ALWAYS
      ACCURACY 100 ALWAYS
      ACCEL 60 ALWAYS
      DECEL 60 ALWAYS
;
      SIGNAL -32,-16
      OPENI 1
      IF SIG(2020) THEN
      CALL at.maint
      END
      SIGNAL 17,21,25,29
      IF SIG(1017) THEN ;Station #1 ready
      SIGNAL 2001
      END
      IF SIG(1021) THEN ;Station #2 ready to receive a door
      SIGNAL 2002
      END
      IF SIG(1022) THEN ;Station #2 ready to remove a door
      SIGNAL 2003
      END
      IF SIG(1025) THEN ;Station #3 ready to receive a door
      SIGNAL 2004
      END
      IF SIG(1026) THEN ;Station #3 ready to remove a door
      SIGNAL 2005
      END
      IF SIG(1029) THEN ;Station #4 ready to receive a door
      SIGNAL 2006
      END
;
      IF (SIG(1019) OR SIG(1006)) THEN
```

```
    CALL maint
    END
;
    status = BITS(2001,6)
;
    SIGNAL -2001,-2002,-2003,-2004,-2005,-2006
;
    CASE status OF;
    VALUE 3,11,19,35,43,51 :
    SIGNAL 2010
    CALL sta.1.control
    IF size THEN
    IF SIG(-1019) THEN
    CALL router.control
    ELSE
    WAIT SIG(1019)
    END
    END
    size = TRUE
    VALUE 12,13,44,45 :
    SIGNAL 2011
    CALL sta.2.pick
    VALUE 48,50,52,53 :
    SIGNAL 2012
    CALL sta.3.pick
    VALUE 18,20,21 :
    TYPE /C6,"STATION #4 IS NOT READY"
    TYPE /C2,"Make sure the robot PC task is running, if not type PCEX,-1
    and hit enter."
    TWAIT 3
    VALUE 10,42 :
    TYPE /C6,"STATION #1 IS NOT READY "
    TWAIT 3
    ANY   :
    PRINT /C1, "Waiting on system input"
```

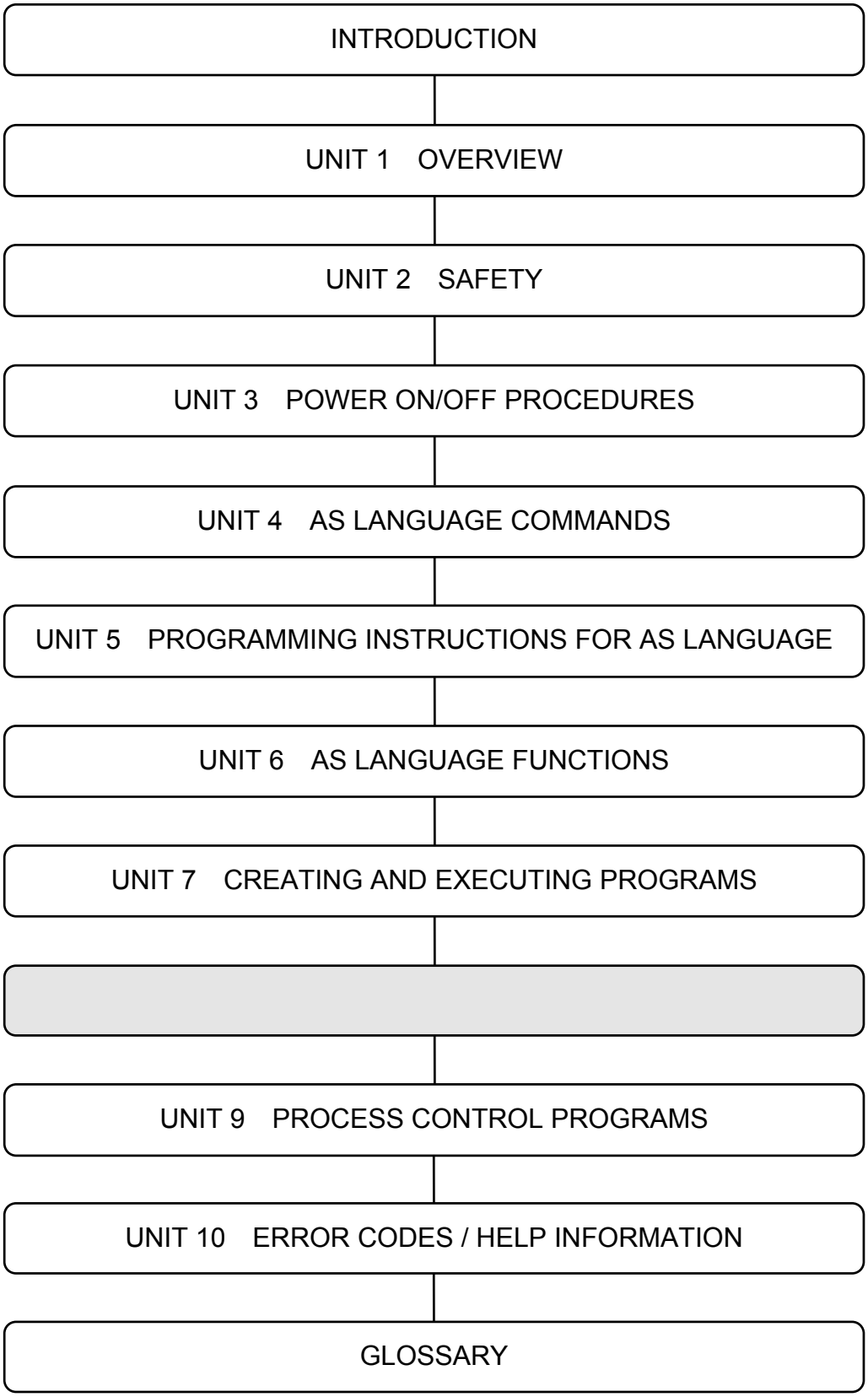
```
      BREAK
      TWAIT 1
      END
      GOTO 100
.END

;Lite cutout program #08

      WEIGHT load
      ACCURACY 500 ALWAYS
      SPEED 100 ALWAYS
      TOOL router1
;
      SIGNAL -21 ;Not clear of sta. #2
      SPEED 75
      JAPPRO door.or.x.y+cut8[1]+TRANS(0,0,0,o,a,t),50
      CLOSEI 1
      SPEED 25 mm/s
      FOR x = 1 TO 5
      IF right THEN
      y = (6-x)
      ELSE
      y = x
      END
      BREAK
      ACCURACY 1 ALWAYS
      LMOVE door.or.x.y+cut8[y]+TRANS(0,0,zoff,o,a,t)
      SPEED ctspd mm/s ALWAYS
      BREAK
      TWAIT delay
      END
      BREAK
      SPEED 10
      LDEPART 75
      ACCURACY 500 ALWAYS
```

CREATING AND EXECUTING PROGRAMS

```
    SPEED 20
    JDEPART 25
    BREAK
    CALL poke.waste
    OPENI 1
    SPEED 100 ALWAYS
    SPEED 75
    JMOVE #router.safe
    BREAK
.END
```


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION
GLOSSARY

PROGRAMMING VIA PERSONAL COMPUTER


MEMO

8.0. PC INTERFACE

This unit covers operation of the C-series controller from a personal computer. To properly utilize the information in this unit, the user will need a basic understanding of the MS-DOS and Microsoft Windows operating environments as well as general computer procedures.

8.1. PC INTERFACE OVERVIEW

The Kawasaki C-series controller and robot system is designed to communicate serially over a standard RS232C interface with a personal computer (PC). The RS232C 25 to 9 pin D connector cable can be purchased as an option from Kawasaki.- The PC interface provides the user a programming method utilizing a larger keyboard than is available with the multi function panel. With a PC the user can do the following:

- View system data
- Type AS language commands from the monitor prompt
- Edit AS language programs
- Edit SLOGIC programs
- Edit block style programs
- Import, manipulate, and save data in a DOS or Windows environment
- Test program data
- Change system data: software limits, dedicated I/O, repeat speed, system switches
- Save/load all system data to/from diskette or hard drive
- Print data

8.2. PC INTERFACE CONFIGURATION

To interface a PC with the C-series controller robot system the following is needed:

- Personal computer running DOS 3.x, or higher, with a 3.5 inch 1.44MB disk drive (user supplied)
- 25 pin plug to 9 pin socket RS232C cable/connector - option, or user supplied
- Kawasaki diskette containing the AS monitor software, KCMON for DOS based operation or KCWIN for windows based operation.

8.3. INSTALLING KAWASAKI AS MONITOR SOFTWARE

The user will need to install the AS monitor software files from the Kawasaki diskette to the hard drive of the PC. The AS monitor software gives the user the ability to interface with the robot system through the PC. The KCMON and KCWIN software contains the source code needed to execute AS language commands.

The following process describes installation of KCMON files onto the C: drive for use in the DOS mode. To load files onto another drive designation the same process can be followed.

The following procedure is used to install the KCMON software:

1. Ensure that the robot system and the PC are properly connected through the RS232C connection.
2. Power up the controller and make sure it is in the TEACH and HOLD modes.
3. Make certain that the write protect tab on the KCMON diskette is set to write protected.

4. Turn on the PC and allow it to go through its power-up sequence. If the PC automatically boots into Windows the user will need to exit Windows and enter DOS.

5. Make a backup copy of all the files on the KCMON disk in case the first copy becomes corrupted.

6. Confirm that the diskettes both contain the following files:
   - KCMON_.EXE
   - KCMON.BAT
   - EDT.BAT
   - FUTL.BAT
   - KCMONENV.DAT
   - FORMAT.EXE

7. Make a new directory on your C: drive for the KCMON files to be copied into.

8. Change to the drive containing the KCMON software diskette.

9. Type: INST C: \ KCMON then press the ENTER key. This copies six files from the diskette to the KCMON directory on the C: drive.

10. Change back to drive C: and display the contents of the new directory, using the DIR command, to make sure that the designated files were successfully copied.

11. Edit the config.sys file of the C: drive to include the following: DEVICE = C: \ DOS \ANSI.SYS

12. Type KCMON from the newly created directory to display the KCMON opening screen.

The following process describes installation of KCWIN files onto the C: drive for use with the windows OS.

1. Ensure that the robot system and the PC are properly connected through the RS232C connection.

2. Power up the controller and make sure it is in the TEACH and HOLD modes.

3. Make certain that the write protect tab on the KCWIN diskette is set to write protected.

PROGRAMMING VIA PERSONAL COMPUTER

4. Turn on the PC and allow it to go through its power-up sequence.

5. Make a backup copy of all the files on the KCWIN disk in case the first copy becomes corrupted.

6. Confirm that the diskettes both contain the following files:
   - KCWIN.EXE
   - KCWINE.EXE
   - KCWIN.INI

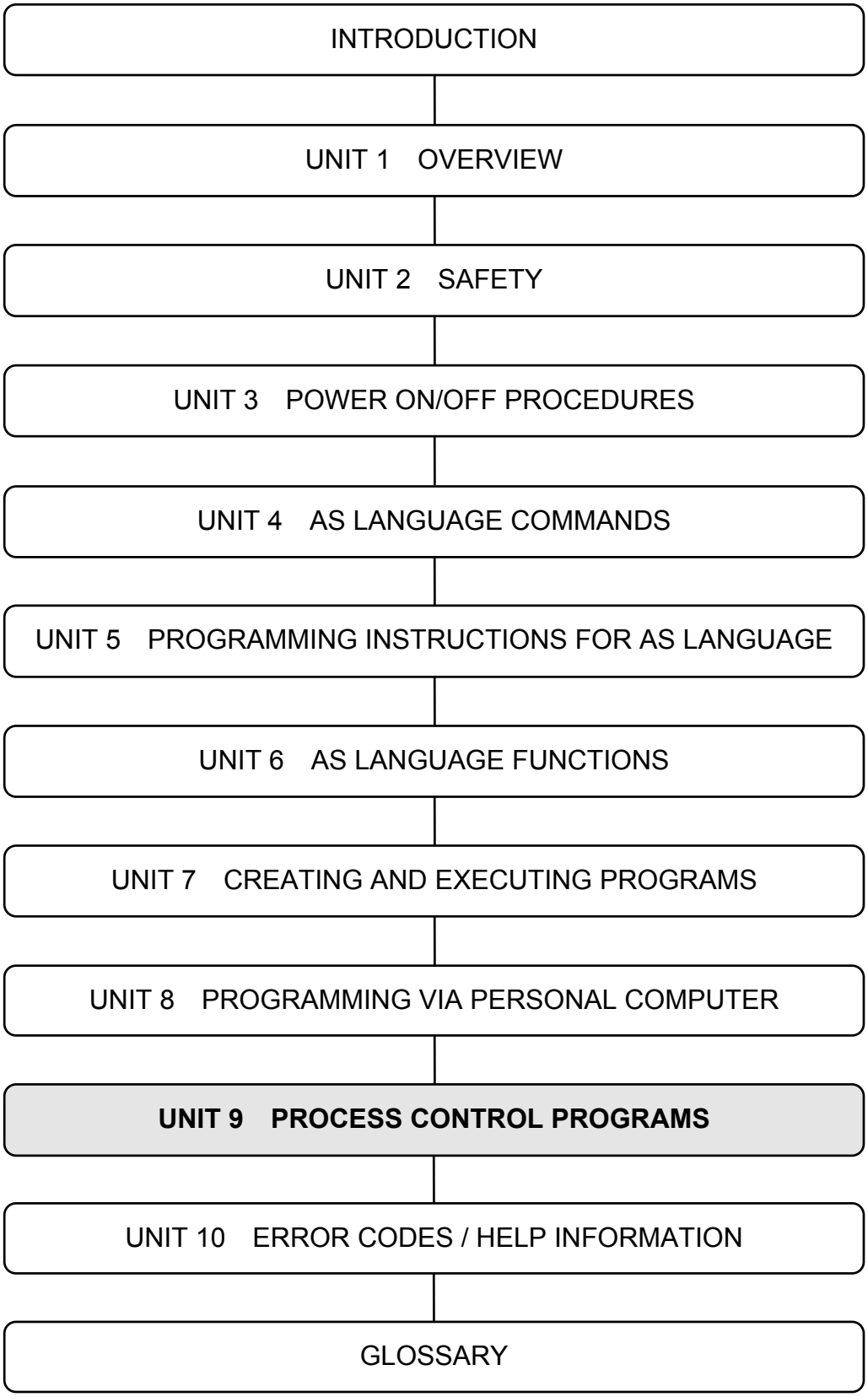
7. From the start menu, select run and enter A: install.

8. The files KCWIN.EXE and KCWINE.EXE are copied to the root directory. The file KCWIN.INI is copied to the WINDOWS directory.

9. If desired, create a short cut icon to be displayed with the start menu for easy access to the AS monitor program.

8.4. PC OPERATION

Auxiliary function 95, Environmental DATA2, must be accessed and the setting for operation from a PC terminal set to connected. When the PC is properly connected and the AS monitor software program is running, all of the commands and functions that were available from the multi function panel key pad can be accessed with the PC.


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION
GLOSSARY

PROCESS CONTROL PROGRAMS


MEMO

9.0. PROCESS CONTROL PROGRAMS

A process control (PC) program is an AS language program which can be executed simultaneously with a robot control program or up to two other PC programs. A PC program is used to monitor or to control external devices through external binary signals. PC programs can evaluate variables, perform mathematical calculations, execute logic instructions, and set external or internal signals.

Unlike robot control programs, PC program instructions cannot cause robot motion. The only exception to this requirement for PC programs is the BRAKE instruction. In addition, the BASE and TOOL instructions are not available for PC programs. PC programs can be used to display messages on the terminal by means of the PRINT instruction. All internal and external binary signals can be used in PC programs.

9.1. PROCESS CONTROL COMMANDS

9.1.1. PCSTATUS COMMAND

The format for the PCSTATUS command is: **PCSTATUS PC *program number***. The PCSTATUS command is used to display the status of a specified PC program. The PC *program number* specifies which PC program will be displayed. The acceptable range for the PC *program number* setting is from 1 to 3. When the PC *program number* is omitted, 1 is assumed.

The PCSTATUS command will display the status of the specified PC program in the format shown in figure 9-1.

```
PC status:    Program is not running
Execution cycles
Completed cycles:   11
Remaining cycles:   Infinite
Program name   Prio    Step No.
pc_test        0    1    PRINT "step1"
```

Figure 9-1 PCSTATUS Command

9.1.2. PCEXECUTE COMMAND

The format for the PCEXECUTE command is: **PCEXECUTE *program name, execution cycles, starting step***. The PC EXECUTE command is used to execute PC programs. This is command is identical to the EXECUTE monitor command except that this command executes a PC program instead of a robot control program.

9.1.3. PCABORT COMMAND

The format for the PCABORT command is: **PCABORT *PC number***. The PCABOT command is used to stop the execution of PC programs. The PC number specifies the number of the PC program to be aborted, 1 to 3. The PCABORT command is identical to the ABORT monitor command except that this command aborts the current PC program instead of the current robot control program.

9.1.4. PCCONTINUE COMMAND

The format for the PCCONTINUE command is: **PCCONTINUE** ***PC number, NEXT***. The function of the PCCONTINUE command is to resume execution of a PC program that has been interrupted by a WAIT condition. If the PCCONTINUE command is issued with the NEXT argument, the WAIT instruction currently being executed by the PC program is skipped. The PCCONTINUE command is identical to the CONTINUE monitor command, except that it handles process control programs instead of robot control programs.

9.1.5. PCKILL COMMAND

The **PCKILL** command is used to remove the current PC program from active status. A PC program cannot be deleted if it is on the active program stack. The PCKILL command is identical to the KILL monitor command, except that it handles process control programs instead of robot control programs.

9.1.6. PCSTEP COMMAND

The format for the PCSTEP command is: **PCSTEP** ***PC program name , execution cycles, starting step***. The PCSTEP command is used to execute a single step of a PC program. The PC *program name* is the name of the PC program that is to have a step executed. *Execution cycles* is the number of cycles for the PC program to be executed with the PCSTEP command. The *starting* step is the step number of the program which will be executed first.

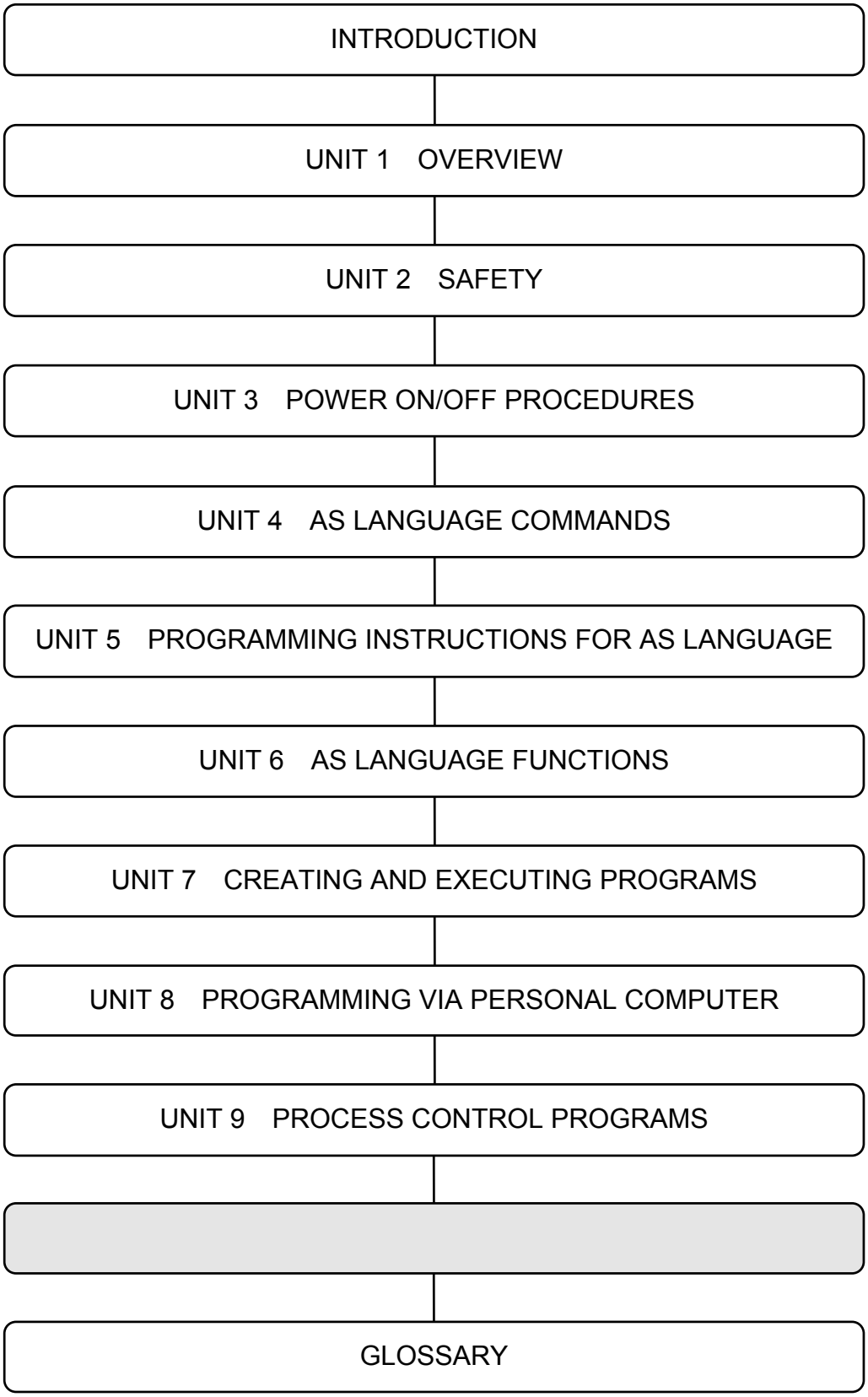
9.1.7. PCSCAN COMMAND

The format for the PCSCAN command is: **PCSCAN** ***time***. The PCSCAN command is used to specify the length of *time* it will take to process the PC program one time. The *time* is specified in increments of one second. If the PCSCAN command is not used, the PC program will be processed at the CPU processor speed.

9.1.8. PCEND COMMAND

The format for the PCEND command is: **PCEND** ***PC number : task_number***.
Terminates the execution of the PC program when the current PC program executes a STOP instruction. The PC *program number* is specified. The range of setting is 1-3. it becomes one when omitting. The task_number is specified -1 or1. It is considered that one was specified when omitting.
When the current PC program executes a STOP instruction (or an equivalent instruction such as a RETURN in the main program), the execution of the current PC program will be stopped immediately, regardless of the remaining cycles to be executed.
If there are still remaining cycles when execution stops, the PC program can be resumed with a PCCONTINUE command.
This instruction (or command) waits until the program actually stops at a STOP instruction. If the PC program loops internally and the current execution cycle never ends, the PCEND command will wait forever. In order to cancel this PCEND command, give a negative number as an argument. In order to terminate a PC program that loops continuously, use the PCABORT command.


MEMO


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
GLOSSARY

ERROR CODES/HELP INFORMATION

ERROR CODES/HELP INFORMATION

 MEMO

10.0. ERROR CODES/HELP INFORMATION

This unit provides information about the error codes that are displayed on the multi function panel or other user interfaces that provide display screen information. The error codes are listed by code number and the message that is displayed with the associated code. In the additional information column (explains in the next paragraph of the error code message.
), an expanded explanation of the message is provided along with possible methods to clear or prevent the specific error. This troubleshooting information is preceded by a ⇨ symbol.

Table 10-1 Error Codes / Help Information

-50 Warning! Cannot move along straight line in this configuration.
Joint speed may exceed maximum, joints4 and 6 aligned.
⇨Change angle of joint5, slow the speed, change to joint mode.

-57 Set low speed because of exceeding joint max. speed in check.
When joint speed is checked with commanded speed, the difference exceeds acceptable range.
⇨Slow the speed.

-58 Operation information were cleared.

-100 Matrix Calculation Error.
The vector element of the matrix cannot be operated because of 0.
⇨Change and calculate the value again.

-101 Turn off motor power.
The Motor Power cannot be executed in the state of turning on according to the command and the instruction.
⇨Turn OFF Motor Power and execute the command and the instruction again.

-102 Application is changed. Turn control power OFF & ON.

ERROR CODES/HELP INFORMATION

-103 There is no external axis.

-104 Illegal positioner type.

-105 Cannot change data, user data exist.

-106 Graphic area error.

-107 Option is OFF.

-200 Cannot execute a program because motor power is OFF.
Program will not start because motor power is not on.
⇨Turn motor power on.

-201 Cannot execute a program in TEACH mode.
Programs cannot run when in the teach mode of operation.
⇨Ensure that the controller is in the repeat mode of operation.

-202 Cannot execute a program because teach lock is ON.
Programs cannot be run with the teach lock in the on position.
⇨Turn the TEACH LOCK switch to OFF and execute the program again in repeat mode.

-207 Turn to HOLD at HOLD/RUN sw.
Occurs when attempt to perform DO, STEP, MSTEP, CONTINUE, or EXECUTE commands is made with the RUN/HOLD switch in the RUN position. Only applies if the CHECK.HOLD system switch is ON.
⇨Turn the RUN/HOLD switch to HOLD position.

-208 Teach pendant is not connected.
Hard wired switches for teach pendant and multi function panel must be jumpered and equipment configuration identified in environmental data functions.
⇨Install teach pendant or configure system accordingly.

ERROR CODES/HELP INFORMATION

-211 Cannot edit a program because teach lock is ON.
Programs cannot be edited if the teach lock is on.
⇨Turn teach lock off.

-212 Cannot execute because executed by other device.

-213 Cannot execute a program because of EXT-IT.

-300 Program is already running.
Occurs when an attempt is made to edit or execute a program that is currently running.
⇨Stop the program prior to editing or checking.

-301 Robot control program is already running.
The DO command executed while the robot motion program is already running. The Tool and the Base command, etc. were executed while running the robot program.
⇨Wait for program in the robot motion to end. End execution program compulsorily.

-302 Can't continue. Use EXEC.
The CONTINUE command is not permitted because of program selection status.
⇨Use the EXECUTE command to start program.

-303 Robot is moving now.
Displayed if any of the following commands are entered while a program is running: EXECUTE, CONTINUE, TOOL, BASE, DO. SYSINIT OR CYCLE START.
⇨Stop the program or confirm the operation to be performed.

-304 Cannot execute because in error now. Reset error.
Occurs when attempt is made to start robot motion if an error has not been cleared.
⇨Clear any errors and re-enter the command.

-306 Cannot execute with DO command.

Displayed when the DO command is entered with an instruction that is not of acceptable format.
⇨Execute the instruction from within a program or use acceptable instruction format for DO command.

-308 PC program is running.
Occurs when a PC program is running and instructions are entered that are not allowed.
⇨Stop the PC program and re-enter the command.

-314 Cannot execute because the program is already used.
Occurs when a program being edited is selected to run by a CALL, ON, ONI or PC program instruction.
⇨Stop editing the program or stop the program that is calling the program being edited.

-316 Waiting weld completion.
Displayed when a command to change the step is entered while a welding sequence is in progress.
⇨Wait until after the weld sequence is completed or force a weld complete condition.

-317 Position offset error at last E-stop JTxx.
The error message is generated when an E-stop is applied and the position of the robot is not within a range of the commanded position. The error deviation range is specified in auxiliary function 42.
⇨Before the error is reset, operators must be aware of the robot's position within the work envelope.

-318 Waiting retract or extend pos. input signal.
While the spot weld sequence is processing the following operations to which the current step was changed was done until the Retract/Extend detection signal was input to the robot after inputting the weld completed : Cycle start (Include EXECUTE, CONTINUE Command), Program selection, Step change, Record.
⇨Input the Retract/Extend detection signal to the robot. Or press the "WX" key

and “Wait override” key on Multi function Panel.

-319 Spot sequence is running.

While the spot welding sequence is executing, the step change, program registration, and program execution excluding the state of “waiting weld completed” or “waiting Retract/Extend position input signal”.

⇨The step change, program registration, and program execution after the Spot welding sequence ends.

-320 Cannot operate because teach pendant on operation.

When you operate the program selection/teaching etc. by Multi-Function Panel, the operation of the EDIT, EXECUTE, and TEACH command is prohibited on the personal computer side.

⇨Operate the personal computer side after ending the operation of Multi Function Panel once.

-324 Cannot execute with MC instruction.

The instruction which cannot be executed by MC instruction was executed.

⇨Use the instruction which can be executed by MC instruction.

-325 Cannot execute the instruction in robot program.

The command and the instruction which cannot be used with robot program were executed.

⇨Input the command and the instruction which can be used with robot program again.

-326 Cannot delete because used by another command.

Program is using by another. (When the step is being made in the editor, and deleting. Or COPY, DELETE, and XFER command.)

⇨Delete them after the processing under execution ends.

-327 Used in programs.

The variable used with program was deleted.

⇨Confirm the deleted variable.

ERROR CODES/HELP INFORMATION

-328 Used in editor.
When program was used in the editor, the program was deleted.
⇨Delete the program after the editor ends.

-329 KILL or PCKILL to delete program.
Occurs when an attempt to delete a program is made and that program is still on the stack (selected).
⇨Select another program or KILL/PCKILL the program, then delete.

-330 Cannot past.

-350 Illegal input data.
Input data from AS language monitor command is improper for the instruction.
⇨Enter data that is within acceptable range.

-351 Too many arguments.
Input data from AS language editor commands exceeds the number of user specified items allowed by the format.
⇨Verify input data and format of command.

-353 Input data is too big.
Data entered for the POINT or HERE commands exceeds the allowable ange.
⇨Enter data that is within acceptable range.

-360 Illegal WHERE parameter.
Occurs if data entered with the WHERE command is not an integer between 1 and 6.
⇨Ensure that is within acceptable range.

-361 Illegal PC number.
PC program number is available 1-3.

-365 Illegal Robot number.
Robot number is available 1 at the standard specification.

-366 Illegal program.

-367 Illegal priority.
The designate of the priority level is wrong. The priority level is up to 0-127.
⇨Input a correct priority level.

-368 Invalid coordinate value.
In the WORD SPACE output, the coordinates value of the upper limited value is smaller than the coordinates value of the lower limited value.
⇨Input the coordinates value of a correct upper and lower limit.

-390 Loging is running.

-391 Undefinded memory.

-400 Syntax error.
Occurs when an AS language command is entered that does not follow the correct format or contains typing or spelling errors.
⇨Correct format or spelling of command/instruction.

-401 Invalid statement.
Occurs when an AS language command is entered that has typing errors, incorrect spelling or is in the wrong format.
⇨Correct the input data spelling or format.

-402 Ambiguous statement.
Displayed when an abbreviation is entered incorrectly or has missing letters.
⇨Enter the correct abbreviation or entire command.

-403 Cannot use this command or instruction here.
Displayed because a program or monitor command was entered that could not be executed while a program is running.
⇨Stop program execution or wait for completion.

-404 Cannot execute with DO command.
A program instruction that is not acceptable to use with the DO command was entered.

⇨Place the desired instruction within a program or choose an acceptable instruction for use with a DO command at the monitor prompt.

-405 Statement cannot be executed.
Occurs when the AS language instruction entered was not acceptable for the mode of operation.
⇨Use instructions and commands that are compatible with the input mode.

-406 Not a program instruction.
Specific information on this error code was not available at the time of publication.

-407 Too many arguments.
The argument of the command and the instruction is not correctly set.
⇨Input the argument correctly.

-408 Missing argument.
Displayed when a DO command is not followed by an acceptable program instruction.
⇨Correct the input and re-enter.

ERROR CODES/HELP INFORMATION

-410 Illegal expression.

A real number expression must be present for processing DECOMPOSE command. Also displayed when incorrect numerical information is entered with arguments.

⇨Ensure correct format and numerical expressions are entered.

-411 Illegal function.

Occurs when functions are used to assign values to variables but the data is incompatible. For example: assigning XYZ coordinate data to precision points.

⇨Ensure function is compatible with variables.

-412 Illegal argument of function.

Occurs when function and argument are not in correct format.

⇨Use correct format for functions and arguments.

-413 Invalid variable (or program) name.

Displayed when illegal variable or program name is entered from the editor or monitor modes.

⇨Define program names and variables correctly.

-414 Illegal variable type.

Displayed when illegal variable is entered from editor or monitor modes.

⇨Use compatible variable type for commands or instructions.

-415 Illegal array index.

Displayed when an attempt is made to use a variable that has previously been defined as an array. May also occur if the order of an array is reversed when editing or entering monitor commands.

⇨Enter the correct array variable information.

-416 Missing parenthesis.

Occurs when parenthesis are not entered as a pair, containing both a left and right parenthesis.

⇨Enter parenthesis in left and right pairs.

ERROR CODES/HELP INFORMATION

-417 Expected to be a binary operator.
Specific information on this error code was not available at the time of publication.

-418 Illegal constant.
Displayed when a variable or non constant is used in a format that requires a constant or integer.
⇨Use correct input variable type for argument.

-419 Illegal qualifier.
Displayed when monitor commands are followed by unexpected qualifiers.
⇨Use only acceptable qualifier.

-420 Invalid label.
Occurs in the editor mode when a GOTO instruction is combined with are served character.
⇨Do not use reserved characters in label identification.

-421 Invalid name.
Displayed when an unidentified program, file, variable etc., is used in a command.
⇨Use only names that have been defined.

-422 Missing expected character.
Occurs when commands or instructions are entered with an incorrect format.
⇨Use correct format for commands and instructions.

-423 Illegal switch name.
Displayed when a system switch is incorrectly identified.
⇨Use only available switch manes.

-424 Ambiguous switch name.
Displayed when a switch name has been entered that is not available for the software version that is operating in the controller.
⇨Use only system switches that are compatible with the software version that is operating.

ERROR CODES/HELP INFORMATION

-425 Illegal format specifier.
Occurs when the TYPE or PRINT command is not used with an acceptable format portion of the instruction.
⇨Use only specified format instructions with TYPE and PRINT commands.

-426 Duplicate statement label.
A specific program label name can only be used once per program. Error is displayed if the same label name is entered a second time in the same program.
⇨Use label names only once per program.

-430 Cannot define as array.
Program name is being defined with the array.
⇨Do not define program name with the array.

-431 Dimension exceeds 3.
The dimension of the array was defined more 3.
⇨Define the dimension of the array 3.

-432 Different dimensional array exist.
The array with a different dimension has already been defined.
⇨Change the variable name.

-433 Array variable exist.
The array variable which has already been defined was defined.
⇨Confirm the array variable.

-434 None array variable exist.
None array variable which has already been defined was defined.
⇨Confirm none array variable.

-435 Array variable expected.
The things except the real value array were specified for the parameter of the DECOMPOSE instruction.
⇨Specify the real value array for the parameter of the DECOMPOSE instruction.

-440 Local variable expected.

When the subroutine argument was defined, the global variable was used.

⇨Specify all local variables when you put up the argument.

-441 Unexpected suffix.

When the subroutine argument was defined, the array with the suffix was used.

⇨Correct not to define the suffix in the array argument.

-442 Mismatch of arguments at subroutine call.

The number of arguments which call the subroutine is different.

⇨Correct the argument correctly.

-443 Mismatch of argument type at subroutine call.

The type of the argument when the subroutine call is executed is different from the type of the defined argument.

⇨Correct the type of the argument correctly.

-450 Control structure error.

Displayed when an illegal program control flow structure is evaluated.

⇨Use correct syntax and components in control flow structures.

-451 Step:xxx Wrong END statement.

Occurs when an illegal END statement is entered during editing.

⇨Use correct syntax and components in control flow structures.

ERROR CODES/HELP INFORMATION

-452 Step:xxx Extra END statement.

Occurs when an extra END statement is present and there is no corresponding structure.

⇨Use correct syntax and components in control flow structures, check structure of END statements.

-453 Step:xxx Cannot terminate DO with END.

When the control flow structure DO....UNTIL is entered with an END statement this error is displayed.

⇨Use correct syntax and components in control flow structures.

-454 Step:xxx No VALUE statement after CASE.

Occurs when the control flow structure CASE OF....END is entered without a value to evaluate.

⇨Use correct syntax and components in control flow structures.

-455 Step:xxx Preceding IF missing.

Displayed when control flow structure does not contain the correct structure for IF....THEN....ELSE....END commands.

⇨Use correct syntax and components in control flow structures.

-456 Step:xxx Preceding CASE missing.

Displayed when control flow structure does not contain the correct structure for CASE....of....VALUE...ANY....END commands.

⇨Use correct syntax and components in control flow structures.

-457 Step:xxx Preceding DO missing.

Displayed when control flow structure does not contain the correct structure for DO....UNTIL.

⇨Use correct syntax and components in control flow structures.

-458 Step:xxx Can't find END of xxx.

Occurs when control flow structure that requires an END statement does not contain the necessary END.

⇨Use correct syntax and components in control flow structures.

-459 Step:xxx Too many control structures.
Occurs when 11 layers of control flow structure are exceeded.
⇨Limit control flow structure to11 layers.

-460 Variable (or program) already exists.
Displayed when a variable is entered that is already part of the system memory.
⇨Do not use variable names for more than one item.

-461 Variable of different type already exists.
Displayed when a variable is entered that is already part of the system emory.
⇨Do not use variable names for more than one item.

-464 Internal buffer over.
Interpreter's internal buffer was exhausted as follows : The input type is complex. There are a lot of arguments of command argument (DIR....etc.) of the list type. There are a lot of arguments of the subroutine (25 or more).
⇨Make to a simple structure and input them.

-465 Undefined Variable (or program).
Specific information on this error code was not available at the time of publication.

-466 Illegal clock value.
Specific information on this error code was not available at the time of publication.

-470 Expect \'=\'.
There is no "= " necessary for the argument.
⇨Check the argument and correct it.

-471 Expect \') \'.
There is no ") " necessary for the argument.
⇨Check the argument and correct it.

ERROR CODES/HELP INFORMATION

-472 Expect \'] \'.

There is not "] " necessary for the argument.

⇨Check the argument and correct it.

-473 Expect \"TO\"!.

There is no " TO ".

⇨Correct it.

-474 Expect \"BY\"!.

There is no "BY" necessary for the argument.

⇨Check the argument and correct it.

-475 Expect \':\:'!.

There is no " : " necessary for the argument.

⇨Check the argument and correct it.

-476 Expect \"ON/OFF\"!.

A different instruction was input when the ON/OFF designation of SYSTEM SWITCH was necessary.

⇨Check ON/OFF of SYSTEM SWITCH and input it correctly.

-490 Program name not specified.

Displayed when no program is on the stack and the EDIT command is entered without specifying a program name.

⇨Identify the name of the program to be edited.

-494 Program is interlocked by another procedure.

Specific information on this error code was not available at the time of publication.

-499 Invalid statement.

Occurs when the program is executed and instructions are encountered that can not be processed as AS language commands.

⇨Use correct syntax and components for AS language commands.

-507 Communication error.

When a vision system is incorporated and the transmission of data is interrupted (transmission line problem or stoppage of the program) this error will be displayed.

-514 Device is not ready.

Specific information on this error code was not available at the time of publication.

-523 Illegal file name.

Specific information on this error code was not available at the time of publication.

-543 Data read error.

Specific information on this error code was not available at the time of publication.

ERROR CODES/HELP INFORMATION

-544 Floppy disk data transfer error.
Specific information on this error code was not available at the time of publication.

-545 Record inhibited.
Set "record accept" and operate again. Displayed when data was entered to edit data but the RECORD INHIBIT system switch was set to inhibit.
⇨Change the setting of the RECORD INHIBIT system switch.

-548 Program-change inhibited. Set “ACCEPT” and operate again.
When “Program-change” of two items of REC_ACCEPT monitor command or ”AUX.71 RECORD (PROGRAM-CHANGE) ACCEPT/INHIBIT” was set in ”INHIBIT”, you executed as following monitor command: EDIT, TEACH, XFER, COPY, LOAD.
⇨Execute the above-mentioned monitor command after setting “PROGRAM change” in “ACCEPT”.

-551 Cannot open the file.
The file was not able to be opened with monitor command SAVE, LOAD, and ZBSAVE.
⇨Check the state of the file and open again.

-565 RI/O board is not installed.

-580 Retry error.
Occurs when there is a problem with the communication link between the controller and a host communication PC.
⇨Check the integrity of the controller and host PC link.

-581 Stop of process.
The SEND, RECEIVE instruction of the host communication was ended on the halfway.

-583 Receive not data after receive request.
“ACK” signal is transmitted and the data reception begins when data reception

demand code “ENQ” signal is received on the reception side. This error occurs when the data is not received even if the time set after data receiving start.

-584 Too long receive data (MAX=255 character).

The length of the character string of the reception data exceeds 255 characters. The character data after that is abandoned though the first 255 characters of the text character string remain as internal character-string data.

-585 Abnormal data (EOT) received in communicating.

The “EOT” signal which shows an abnormal code in the transmitting and receiving processing is received, it is judged that Peripheral equipment is abnormal termination, and interrupts the transmitting and receiving processing. Moreover, when the “EOT” signal is received while processing all except the last EOT end of the reception processing, the processing is interrupted at once.

ERROR CODES/HELP INFORMATION

-586 Time out.

The reception data waiting state has passed specified time while communicating with the reception data waited. However, when the time-out time is not specified or specified time is specified by 0, the time-out function becomes an invalid.

-591 Illegal device number.

A careless port was selected by the communication port. Two kinds of selections of the sensor port and the personal computer terminal port are possible in the serial port. This error occurs when port No.s other than the sensor port (the 1st) in the port which can be selected and the personal computer terminal port (the 2nd) are specified.

-596 Cannot attach terminal.

The prompt instruction is executed from two or more programs at the same time.

⇨Do not execute the prompt instruction from two or more programs at the same time.

-597 Cannot attach communication port.

The RECEIVE instruction and the SEND instruction are executed from two or more programs at the same time.

⇨Do not execute the above instruction from two or more programs at the same time.

-598 Cannot execute on this terminal.

The command which cannot be executed with the terminal was executed from the terminal. It is time when it tried to execute the “O” command of the editor excluding a standard terminal.

⇨Use the above command in a standard terminal.

-599 Waiting input data for PROMPT. Connect input device.

Specific information on this error code was not available at the time of publication.

ERROR CODES/HELP INFORMATION

-600 Motor power OFF.
Displayed whenever an emergency stop is encountered.
⇨Reset emergency stop button and reapply motor power.

-605 PLC communication error.

-606 Weld cont. #xx not connected.

-607 Weld cont. #xx no response.

-608 Weld cont. #xx response error.

-610 Weld completion time over.
Occurs when a weld complete signal is not received in a specified time period.
⇨Override the wait condition, ensure that weld complete signal specifications are correctly identified, check the operation of the weld gun or controller.

-611 Illegal extend (retract) output signal.
Occurs when the output signals for the extend and retract operation of a spot welding application are not properly set.
⇨Check the setting in auxiliary functions 114-10 and12.

-612 Weld fault input.
Displayed when the controller receives a weld fault signal.
⇨Check operation of welding equipment, ensure signal numbers are correctly set.

-613 Retract pos. monitor error.
Occurs when the input signal for the retract operation of a spot welding application is not received.
⇨Check operation of welding equipment, check the signal numbers setting in auxiliary function 114-10.

-614 Extend pos. monitor error.
Occurs when the input signal for the extend operation of a spot welding application is not received.

⇨Check operation of welding equipment, check the signal numbers setting in auxiliary function 114-10.

-615 Weld completion signal is already inputted.

Displayed when the weld complete signal has been received before the weld initiate output has not been issued.

⇨Check operation of welding equipment, check the signal numbers setting in auxiliary function 114-11.

-616 Gun retract position mismatch.

In the check mode, retractable gun output signals are monitored and compared to open/close data for a specific step, if the data does not compare an error is displayed.

⇨Check settings in clamp conditions, check operation of gun with clamp key.

ERROR CODES/HELP INFORMATION

-653 Illegal DOUBLE OX output.
For example, when OX1, and OX2 are set in the mutually exclusive dichotomy, this is inhibited. Both are being turned ON (OX=1,2) or Both are being turned OFF (OX= -1,-2)
⇨Modify by the teaching.

-654 Cannot use DOUBLE OX.
The signal set in mutually exclusive dichotomy OX was operated with BITS, PULSE, and DLYSIG, etc.
⇨Modify by the teaching.

-656 Not dedicated work sensing signal.

-657 Work sensing signal ON.

-658 Work sensing signal not dedicated.

-659 Distance too short.

-662 Start point position error for circle.
The circular motion start point shifts from calculated circular tracks by 4mm or more. There is a possibility to cause when the circular is overlapped, is held on the road, and restart, too.
⇨Check the start point of the circular interpolation and correct them.

-663 MASTER robot already exists.

-664 This robot is not MASTER.

-665 SLAVE robot already exists.

-666 Interrupt in cooperative control.

-667 Already in MASTER mode.

ERROR CODES/HELP INFORMATION

-668 Already in SLAVE mode.

-669 Foece to quit cooperative control.

-670 In cooperative mode.

-671 Cannot execute in check back mode.
The step reached the instruction which cannot be executed with check back mode.
⇨Execute after select the step which check back selectable.

-672 Cannot execute in ONE program.
The robot motion instruction was executed with program called by ONE instruction.
⇨Do not use the robot motion instruction for program called by ONE instruction.

-673 Angle between JT2 and JT3 is out of range at start location.
Specific information on this error code was not available at the time of publication.

-674 Angle between JT2 and JT3 is out of range at end location.
Specific information on this error code was not available at the time of publication.

-675 Terminal is not connected.
Specific information on this error code was not available at the time of publication.

-676 Cannot input /output to multi function panel.
Specific information on this error code was not available at the time of publication.

-677 SOFT ABSORBER error. Turn control power OFF & ON.

-678 Invalid coordinat value X.

ERROR CODES/HELP INFORMATION

-679 Invalid coordinat value Y.

-680 Invalid coordinat value Z.

-681 Illegal pallet number.
Pallet number in the PALUP instruction or the PALLET function is the values other than 1-9.
⇨Correct program so that pallet number may become between 1-9.

-682 Illegal goods number.
Work number in the block step type instruction or the PALLET function is the values other than 1-999.
⇨Correct program so that work number may become between 1-999.

-683 Illegal pattern number.
Number which cannot process pattern number of work specified with the block step type instruction or the PALLET function or number not registered.
⇨Set the pattern again by “Auxiliary 8-1 PALLETIZE WORK”.

-684 Illegal pattern type.
Pattern types of work specified with the block step type instruction or the PALLET function were the one other than one (column)-5(teaching) or is not registered.
⇨Set “Auxiliary 9-1 PALLETIZE WORK”.

-685 Illegal goods data.
The mistake is found in the work data of work specified with the PALLET function or the block step type instruction or is not registered.
⇨Set “Auxiliary 9-1 PALLETIZE WORK”.

-686 Illegal pallet data.
The mistake is found in the pallet data of pallet specified by the PALLET function, the block step type instruction, and ordering PALUP or is not registered.
⇨Set “Auxiliary 8-2 PALLETIZE PALLET”.

ERROR CODES/HELP INFORMATION

-687 CHANGE GAIN error. Turn control power OFF & ON.

-690 Cannot use I/F panel signal.

-700 No free memory.
No free memory is available to teach or edit programs.
⇨Delete unused programs and variables, or, expand system memory to maintain the required capacity.

-800 Program does not exist.
No program is on the stack at the time of cycle start or execution command (without a program being specified).
⇨Identify program to be executed.

-801 No program step.
The step specified for execution does not exist.
⇨Select valid step numbers for execution.

-802 Nonexistent label.
Occurs when executing the GOTO command and the destination label is not defined.
⇨Ensure valid labels are used within the program.

-803 Undefined variable.
Variable data for a specific argument in a command is not defined.
⇨Ensure variables are properly defined.

-804 Undefined location data.
The location variable for the BASE, TOOL or POINT command is not specified. Also, a named position in a program is not defined in system memory.
⇨Define all locations identified in programs.

-805 Undefined string variable.
String variables that are evaluated by ASC and LEN functions are not defined.
⇨Define the string variables to be evaluated or correct the name of the string

variable used for evaluation.

-807 Undefined program or label.
The program name or label associated with an ON or ONI command does not exist.
⇨Define the program or label used with the ON or ONI command.

-808 Illegal value.
The numeric value entered exceeds the upper or lower limits of the acceptable range.
⇨Enter data that is within acceptable range.

-809 Undefined array suffix.
Specific information on this error code was not available at the time of publication.

-810 Divided by zero.
Occurs when the system encounters a mathematical evaluation that involves division by "0". Typically associated with the FRAME function and circular interpolation.
⇨Check data source for calculation.

-811 Floating point overflow.
Numeric value overflows with arithmetic operation (+,-,*,/) logical operation etc. When the argument is converted into the integer value, the floating point is overflowed.
⇨Correct the program.

-812 Too long string.
Character strings associated with arithmetic or comparative operators or the LEN function are too long.
⇨Correct the program.

-813 Illegal exponential operation.
Numeric values that have exponents must be positive in value.

⇨Correct equations in program.

-814 Too complicated expression.

A numeric calculation too complex to be evaluated was encountered.

⇨Simplify mathematical equations.

-815 No expressions to evaluate.

The type of data in an argument is incompatible with the operation being performed.

⇨Correct the program to evaluate compatible expressions.

-816 Unexpected error while evaluating expression.

Occurs when the system is evaluating the argument in an expression and the data of the argument is found to be incompatible or missing.

⇨Correct the program so that data is compatible with arguments and expressions.

-817 SQRT parameter is negative.

In the argument of a SQRT function a negative number was entered for evaluation.

⇨Do not enter negative numbers for evaluation by the SQRT function.

-820 Illegal array index.

Occurs when the array subscript number exceeds the acceptable range from 0 to 9999.

⇨Ensure the range of array subscripts is acceptable.

-821 Illegal argument value.

Displayed when the parameter specified for an command or instruction is illegal.

⇨Use correct argument value.

-822 Illegal joint number.

Specific information on this error code was not available at the time of publication.

-823 Illegal signal number.

When the SIG or BITS command is used and the specified signal number is beyond the range permitted by the system configuration, this error will be displayed.

⇨Use acceptable signal numbers for system configuration.

-824 Illegal timer number.

Displayed when a timer was specified that was not within the acceptable range of between 1 and 10.

⇨Specify timers in the range between 1 and 10.

-825 Illegal signal number.

When the RUNMASK, SIGNAL, BITS, PULSE, or SWAIT commands are used and a signal number that exceeds the range permitted by system configuration, this error will be displayed.

⇨Check signal number specified in instruction and ensure it is within system configuration.

-826 Illegal clamp number.

Displayed when the clamp number entered exceeds the maximum permitted by system configuration.

⇨Use only clamp numbers that are supported by system configuration.

-827 Illegal time value.

Displayed when a negative number is entered as part of a DELAY or TIMER command.

⇨Correct the time setting to a positive number.

-828 No value set.

Occurs when an instruction like the BITS command is evaluated and there is no corresponding value set.

⇨Correct program code to evaluate existing values.

-829 Illegal signal number.

This error will be displayed when the RUNMASK, SIGNAL, BITS, PULSE, or

SWAIT commands are used with a signal number that exceeds the range permitted by system configuration.
⇨Check signal number specified in instruction and ensure it is within system configuration.

-832 Illegal time input data.
Occurs when erroneous data has been entered in the setting of the TIME and DATE function. For example: a date of Feb. 30.
⇨Input time and date information correctly, mm/dd/yy.

-834 Program name already exists.
When using the RENAME command a new program name must be specified. If an existing name is used to rename a program this error will be displayed.
⇨Specify unused program names when renaming programs.

-835 Can't KILL because the program is running.
Displayed when an attempt was made to KILL a program that was in the process of executing.
⇨Stop program execution with HOLD or ABORT commands before program is removed from the stack with a KILL command.

-837 Cannot use dedicated signal.
Occurs when a previously dedicated signal was used as a general purpose signal.
⇨Use signals that have not been dedicated for general functions.

-838 Not RPS mode.
Occurs when the required input signals are not dedicated at the time an attempt to run an externally selected program is made.
⇨If RPS is to be used the necessary signals must be dedicated.

-839 Cannot use negative number.
Displayed when a negative number has been used in conjunction with the PULSE or ACCURACY commands.
⇨Use only positive numbers in the acceptable range for the PULSE and ACCURACY commands.

-840 Too many subroutines.
Occurs when more than 20 subroutines are nested with EXTCALL or CALL instructions.
⇨Do not exceed 20 nested subroutines.

-842 Nonexistent subroutine.
Displayed when the program identified by a CALL, ON, or ONI does not exist.
⇨Select only existing programs to be run as subroutines.

-846 No program exist.
Specific information on this error code was not available at the time of publication.

-850 Out of absolute lower limit.
Displayed when an attempt has been made to set the software lower limits of robot travel to a value that is to low.
⇨Set lower software travel limits to an acceptable range.

-851 Out of absolute upper limit.
Displayed when an attempt has been made to set the software upper limits of robot travel to a value that is to high.
⇨Set upper software travel limits to an acceptable range.

-852 Out of user lower limit.
Displayed when an attempt has been made to set the software lower limits of robot travel to a value that is to low.
⇨Set lower software travel limits to an acceptable range.

-853 Out of user upper limit.
Displayed when an attempt has been made to set the software upper limits of robot travel to a value that is too high.
⇨Set upper software travel limits to an acceptable range.

-855 Motion start location of jt-x is out of range.

Prior to beginning a program or motion to a step, the software has calculated the location to be outside of the allowable upper or lower software limits.
⇨Correct location to within working envelope or expand software limits to an acceptable location.

-856 Motion and location of jt-x is out of range.
While executing a motion to a step, the software has calculated the location destination of the specific joint number to be outside of the allowable upper or lower software limits.
⇨Correct location to within working envelope or expand software limits to an acceptable location.

-857 Destination is out of range.
While executing a motion to a step, the software has calculated the location destination of all joints to be outside of the allowable upper or lower software limits.
⇨Correct location to within working envelope or expand software limits to an acceptable location.

-858 Illegal configuration for linear motion.
System software has determined that the start and end points of a linear move will cause the robot to exceed the acceptable motion parameters.
⇨Change motion interpolation to a joint move; move location to avoid configuration.

-871 Illegal joint number.
Occurs when the DRIVE command is used and specifies a joint number that is not part of the robot configuration.
⇨Confirm robot configuration before using the DRIVE command.

-872 Cannot execute motion instruction in PC program.
A PC program can not contain instructions that initiate robot motion. If a motion instruction is encountered in a PC program this error will be displayed.
⇨Correct the PC program by removing motion instructions.

ERROR CODES/HELP INFORMATION

-873 Illegal auxiliary data number.
The value selected for auxiliary data (speed, timer, tool etc.) exceeds allowable range.
⇨Correct the value of auxiliary data.

-874 No circular location.
Program circular motion instructions must have C1 moves followed by either a C1 or C2 move.
⇨Correct program instructions.

-875 No C1MOVE(CIR1) ins.
Program circular motion instructions must have C2 moves preceded by a C1 move.
⇨Correct program instructions.

-876 Cannot create circle.
Circular interpolation moves can not be processed because the points identified are too narrow or are on a straight path.
⇨Correct program instructions.

-877 Cannot execute, because of sealing type.
Occurs when a command for a sealing application is evaluated by a controller that is not configured for sealing applications.
⇨Correct program instruction to much software configuration.

-879 Cannot execute, because of not sealing type.
Occurs when a GUNON, GUNOFF,GUNONTIME or GUNOFFTIME command for a sealing application is evaluated by a controller that is not configured for sealing applications.
⇨Correct program instruction to much software configuration.

-896 Option is not set up.
Specific information on this error code was not available at the time of publication.

-909 Watch-Dog Error RS485 Special Communication board.

-919 No RS485 Special Communication board.

-967 No AVC board.

-968 Too many taught points for AVC.

-969 Out of AVC tracking value.

-971 Out of AVC tracking capacity.

-972 AVC voltage deviation error.

-1003 Data base error.
The program storage area of the system memory has been damaged and is not linking data correctly.
⇨Initialize once to try to clear program, troubleshoot CPU board.

-1011 Clock service time over.

-1012 Command position of jt-x has suddenly changed.
In the repeat mode, the commanded position of the joint identified has exceeded 1.3 times the maximum arm speed. In the check mode, the commanded position of the joint identified has exceeded 250 mm/sec.
⇨Troubleshoot servo amp system.

-1014 Commanded position of jt-x is out range.
The commanded position for the joint identified has exceeded a specific motion range during a fixed time period. Condition is monitored in both the check and repeat modes.
⇨Correct taught position to avoid out of range location.

-1017 Angle between JT2 and JT3 is out of range.
The command value of Joint interferes mutually when 2, 3 axes of the UX, UT, UZ series etc. are robots of the link mechanism. The teaching posture (include

posture in move) is bad.
⇨Change the teaching posture.

-1019 Check sum error of system data.

The check sum data of the system data (machine model, number of axes, and option setting, etc.) of the AS software was rewritten for some reasons. When the AS software is exchanged, doing LOAD are normal the data file which the system/the robot data enters when this error is occurred when setting the system/the robot data (machine model and option) is changed. When this error is occurred in situations other than the above-mentioned : Defect of memory backup. Defect etc. of 1GA board. Memory error by noise etc.
⇨Set the error detection invalid function in the valid by “Auxiliary 78 CLEAR CHECK SUM ERROR”, CHSUM command and do error reset. When the error cannot do the change by executing the CHSUM command, the command finding abnormality in the data is displayed. Again, release the error again by CHSUM command after setting each command. Check the battery and exchange the 1GA board when the error cannot do the change and the error is occurred again excluding the above-mentioned content.
Caution : There is a form which describes set of each value when each model and each title machine are shipped with in the controller or each manual. Confirm this set table. Keep this set table importantly so as not to lose. Keep from a set content in this set table or another form when you change a set content.

-1022 RAM battery low voltage <board name>.

When control power is applied, a voltage check is performed on the batteries that maintain RAM memory when power is off. This message is displayed if low voltage is detected.
⇨Back up system and program data and replace batteries.

-1025 As Flash memory sum check error.

For abnormality the check sum of the AS system area in the flash memory on 1GA board when the control power supply is turned on. When the AS system was down-loaded, the fchk command was not executed. The addressing of the fchk command was wrong. The flash memory and 1GA board Defective. The system in the flash memory damaged.
⇨Confirms the content of the description the command file “as_load.cmd” file

in the IC card if immediately after down-loading the AS system. Execute down- again. Down-load the AS system again and confirm whether to occur this error. Exchange 1GA board when this error relapses.

-1026 Servo Flash memory sum check error.

For abnormality the check sum of the servo system area in the flash memory on 1GA board when the control power supply is turned on. When the servo system was down-loaded, the fchk command was not executed. The addressing of the fchk command was wrong. The flash memory and 1GA board Defective. The system in the flash memory damaged.
⇨Confirms the content of the description the command file “as_load.cmd” file in the IC card if immediately after down-loading the servo system. Execute down-loading again. Down-load the servo system again and confirm whether to occur this error. Exchange 1GA board when this error relapses.

-1028 IP board memory error. (xx)

-1029 Touch panel switch is short circuited.

The touch panel short-circuit check is a valid. The touch panel is pressed when control power supply turn ON or Teach→Repeat is changed.
⇨Do not touch the touch panel at the control power supply turn ON or Teach→Repeat change. Exchange the touch panel.

-1031 Other robot is in the interference area.

-1051 Cannot execute in this robot arm.

Specific information on this error code was not available at the time of publication.

ERROR CODES/HELP INFORMATION

-1100 CPU Error.

This error is caused by several types of interruptions detected by the CPU on the 1GA board. May be associated with memory anomalies.
⇨Replace defective components or reload software.

-1101 Main CPU BUS error.

In 1GA board, the bus error (In the VME bus line, data processing was not able to be done normally) was occurred. Defect of 1GA board. Defect of AS software. Malfunction by noise etc.
⇨Confirm whether to occur error with which board referring to the message displayed in the terminal software by the personal computer. Turn ON the control power supply from turning OFF. At this time, LOAD again in ･gNo･h if the message of the initialization confirmation is displayed. Do the teach data in LOAD again after initializing when the error does not return normally even if this operation is done. The defect of the AS system is thought if this error occurs in a certain specific part or a certain specific operation was done when the teaching program is executed. Therefore, report details (content of all messages, models, control models, title machines, AS software/servo soft are versions, when the above-mentioned error is occurred, situations such as program lists, and content of the option etc.) to KHI. Exchange 1GA board etc. when reproducibility cannot be confirmed and error cannot return normally and look at the appearance.

-1102 VME BUS error.

This error occurs when the CPU does not receive a response from one of the I/O bus devices within a specific time.
⇨Replace the 1FR or 1FS board.

-1200 Encoder board is not installed.

Specific information on this error code was not available at the time of publication.

-1201 Power sequence board is not installed.

When the control power supply is turned on, the first address of the power sequence board (1FP board etc.) of the first piece cannot be read (Not

mounted). However, the control power supply does not enter when not mounted actually. The power sequence board (1FP board etc.) is not mounted. Set mistake in address of power sequence board. Defect of power sequence board.
⇨Mount on the position where setting the address of the power sequence board is set to the first piece and the card lack is correct. Exchange the power sequence board (1FP board etc.).

-1202 No2 power sequence board is not installed.

The first address of the power sequence board of the second piece (1FP board etc.) cannot read for the twin arm specification etc. to turn on the control power supply (Not mounted). The power sequence board (1FP board etc.) is not mounted. Set mistake in address of power sequence board. Defect of power sequence board.
⇨Mount on the position where setting the address of the power sequence board is set to the second piece and the card lack is correct. Exchange the power sequence board (1FP board etc.).

-1203 No x I/O board is not installed.

Cannot read the first addresses of I/O board (1FR board etc.) of the number of signals of having set with DO (output point), DI (input point) of the ZSIGSPEC command when the control power supply is turned on(Not mounted).
ZSIGSPEC DO, DI・”The number of sheets of mounting I/O board*32 points.
Set mistake I/O board (1FR board etc.) in the ZSIGSPEC command is detached. Set mistake in board address of I/O board (1FR board etc.). Defect of I/O board (1FR board etc.).
⇨Set a correct number of signals with the ZSIGSPEC command. Set the board address of the I/O board (1FR board etc.) correctly and mount on the card lack.
Caution : This error is in the state of the I/O board's not being mounted after error reset is done and an automatic operation can be executed. However, the input not mounted is fixation while turned off. And, the output is possible in the AS software ON/OFF as usual. However, the number of maximum signals in the software which can be set with the ZSIGSPEC command is DO, DI=256, INT=512 but DO and DI are restricted by restricting hardware and other option.

-1204 Option SIO port is not installed.

It is time when is not mounted IC(SIO) for serial communications for command line 3 and 4CH for the communication with the servo board on 1GA board and it sets more than seven axes at control power supply ON. Set more than seven axes with an experimental board with SIO not mounted. Defect of 1GA board.
⇨Exchange 1GA board.

ERROR CODES/HELP INFORMATION

-1205 Power sequence board any error.

The error is not classified though the error summary signal comes from the power sequence board (1FP board etc.). Or, the error message which corresponds to the AS software when the error detection is newly added on the power sequence board side does not exist. As for the error summary signal and the error classification, etc. the AS software checks all register information in EPLD on the power sequence board by way of VME bus. This error is not detected because error is displayed in each error code usually. The error detection function of the power sequence board does not correspond to the error processing function of the AS software. Defect of power sequence board. Defect of AS software.

⇨Exchange the power sequence board. Improve the version of the AS software.

Caution : This error is shown to occur some errors excluding the defect of the power sequence board. Investigate the situation such as the occurring situation and reset of the error possible.

-1206 Built-in sequence board is not installed.

The presence of the installation of the built-in sequencer board is checked when the control power supply turn ON. This error occur when the built-in sequencer board is not installed.

⇨Install the built-in sequencer board.

-1208 RI/O board is not installed.

-1209 RI/O board initialize error.

-1210 INTER-BUS board is not installed.

-1211 Dual port memory to communicate is not installed.

-1247 Dual port memory to communicate is not installed.

-1247 Axis setting data incorrect.

When the set data was changed by “Auxiliary 901 EXTERNAL AXIS SET”,

tried to register the wrong data.
⇨Input a set value correctly.
Caution : There is a form which describes set of each value when each model and each title machine are shipped with in the controller or each manual. Confirm this set table. Keep this set table importantly so as not to lose. Keep from a set content in this set table or another form when you change a set content.

ERROR CODES/HELP INFORMATION

-1248 Number of axes changed! SYSINI.

Not initializing though the axes number of robots was changed.
⇨Initialize when you change the number of axes.

-1249 Servo parameter changed! Control power turn OFF & ON.

The servo software and the servo parameter, etc. are usually written on the servo board (1GB board etc.) only at control power supply ON. When the data of the servo parameter of the external axis, model setting and number etc. of correspondence axes are changed with LOAD etc. of the robot/the system data, the new data should write in the servo board at 1GA board is exchanged and option axis since seven axes is used. Therefore, try to turn on the control power supply.
⇨Make the control power supply from turning off to turning on.
Caution : When the model was changed, the robot motion stripes had been normally corked if the servo software was not exchanged for the corresponding model software.

-1250 Servo board (X) Initialize error.

When the servo software was written from the flash memory on 1GA board to the servo board (1GB board etc.) in the command line, it did not end normally at control power supply ON. Defect of servo board (one axis amplifier). Defect of the harness between power sequence board and servo board. Motherboard JP set mistake. The servo software is not in the flash memory on 1GA board. Malfunction by noise etc. Version suitable combination of servo software and AS software. Version suitable combination of monitor ROM in servo software and servo board.
⇨Exchange of 1GA, servo board, and each harness. Confirm the servo software and the servo board monitor ROM version and install the corresponding servo software again. The confirmation of the version can be confirmed by “Auxiliary 90 SOFTWARE VERSION DISPLAY”, ID command.

-1251 Servo board (X) communication error.

Among communications in the command line of each CPU of the servo board (1GB board etc.) and 1GA board, the data reception from the servo board failed twice continuously. When each CPU of the servo board stops, system

might be by this error. Defect of 1GA board. Defect of servo board. Defect of each communication harness. Malfunction by noise etc.
⇨Exchange of 1GA, servo board, and each harness.

-1252 Servo board (A) hardware error code = xxxx.

Error not basically occurred. Though the error on hardware was detected from the servo board (1GB board etc.), system is processed by this error when there is no error processing corresponding to the AS software side. The internal error code number which the servo software detected is displayed in xxxx. Defect of servo software. Malfunction by noise etc. Defect of harness during servo board and power sequence board (1FP board etc.). Defect of servo board. Suitable combination of version of servo software and AS software. Something wrong servo software and AS software.

⇨The version up of the servo software and the AS software. Exchanges such as 1GA board, servo boards, and each harness. When this error is occurred, report details (error code, code number displayed in xxxx, models, control models, title, AS software/servo software versions, operation when occurring making an error and situation such as program list, and content of the option etc.) to KHI.

-1253 Servo board (B) hardware error code = xxxx.

This is the same as the above error code (-1252).

-1254 Servo board (C) hardware error code = xxxx.

This is the same as the above error code (-1252).

-1255 Servo board (D) hardware error code = xxxx.

This is the same as the above error code (-1252).

-1256 Servo board (A) software error code = xxxx.

Error not basically occurred. Though the error on hardware was detected from the servo board (1GB board etc.), system is processed by this error when there is no error processing corresponding to the AS software side. The internal error code number which the servo software detected is displayed in xxxx. Defect of servo software. Malfunction by noise etc. Defect of harness during servo board and power sequence board (1FP board etc.). Defect of servo board. Suitable combination of version of servo software and AS software. Something wrong with servo software and AS software.

⇨The version up of the servo software and the AS software. Exchanges such

as 1GA board, servo boards, and each harness. When this error is occurred, report details (error code, code number displayed in xxxx, models, control models, title machines, AS software/servo software versions, operation when occurring making an error and situation such as program list, and content of the option etc.) to KHI.

-1257 Servo board (B) software error code = xxxx.
This is the same as the above error code (-1256).

-1258 Servo board (C) software error code = xxxx.
This is the same as the above error code (-1256).

-1259 Servo board (D) software error code = xxxx.
This is the same as the above error code (-1256).

-1260 Option changed! SYSINI
Not initializing though the option was changed.
⇨Resetting the error by SYSINIT command or “Auxiliary 100 system initializations”.

-1261 Servo board (x) parameter setting error.
When setting was changed by “Auxiliary 976 SERVO PARAMETER”, a case different from the content of the transmission was consecutive and occurred answer-back from the servo software of each unit two times or more for the transmission of the data to the servo board (1GB board etc.) through the command line. Defect of 1GA board. Defect of servo board. Defect of each communication harness. The AS software or the servo software does not correspond. Malfunction by noise etc.
⇨Exchange of 1GA, servo board, and each harness. The version up of the AS software or the servo software. Resetting the error becomes possible when normally ending by setting again though error reset is basically impossible. The control power supply should be OFF→ON when not normally ending.

-1262 EXT AXIS cutting error (code xx).

ERROR CODES/HELP INFORMATION

-1263 EXT AXIS connecting error (code xx).

-1300 Servo CPU-(X) watch-dog error.
Watch dog timer built into in each CPU of 1GB board operated. Defect of servo board. Something wrong with servo software.
⇨Exchange of servo board.

-1306 Servo board command error.
The servo software receives a command of the uncorrespondence and a command not adequate when each command is transmitted from the AS software to the servo software and servo software has returned the error code (Each error code of the servo system is another) to the AS software. Defect of 1GA board and servo board. Something wrong with AS software and servo software. Suitable combination of version of AS software and servo software (uncorrespondence etc. of option). Malfunction by noise etc. Defect of the harness between servo board and power sequence board.
⇨Exchange of 1GA board, servo board, and harness. Version up of AS software and servo software.

-1308 Motor power off.
Occurs when software recognizes motor power on even though there is no signal feedback indicating motor power is on.

-1333 Monitor ID of servo board mismatch!
The monitor ROM version (Basic software printed out to CPU. It is not a main body of the servo software.) of each unit on 1GB board is different. Defect of 1GB board.
⇨Exchange 1GB board.

-1334 Servo control line error.
Information of having done the demand of the brake open (It is not brake opening confirmation) does not return within the fixed time (2 seconds) according to the command line communication from the servo board even if servo control ON (SVCN) signal is sent from the AS software to the servo board (1GB board etc.) through the register of EPLD on the power sequence board (1FP board etc.). Defect of power sequence board. Defect of servo

board. Defect of the harness between power sequence board and servo board.
⇨Exchange the power sequence board and the servo board, etc.

-1336 Safety circuit was cut off.

-1337 Two MC lines are not consistent.

-1338 K1 and/or K2 works wrong.

-1401 Amp over current jt-x.

At the savoring, the feedback current from a current sensor in the power block exceeded 144% of the maximum current of the instantaneous of the motor. A set value of Joint is set with the hardware of 1GM board installed on the servo board (1GB board) and is detected by HIC. Short in U, V, W from power block to motor, and earthing wire line. Defect of motor. Defect in power block. Defect of servo board (1GB board etc.).
⇨Check whether U, V, W to the motor, and the earthing wire line are disconnected from the power block. And, exchange the separation harness/the inside machine harness. Exchange the motor. Exchange Servo Unit.

ERROR CODES/HELP INFORMATION

-1407 AMP power unit error.

When the servo system error is occurred, this error is displayed. Refer to the content of each error because each servo error is displayed at the same time. Turning OFF becomes becoming of Motor Power when servo error signal (SVER) is output from the servo board (1GB board etc.) with the hardware of the power sequence board (1FP board etc.). Moreover, this signal is transmitted to the AS software via the register of EPLD. Afterwards, after this error, the error code which corresponds to each error is displayed because error information from the servo board is sent through the command line. The servo system error was occurred. Disconnection or connect defect of the harness between power sequence board and servo board.
⇨Furnace when error of servo system is occurred referring to each error code. Check and exchange the power sequence board for the disconnection of the harness between servo boards.

-1413 Regenerative resistor overheat or disconnect.

Specific information on this error code was not available at the time of publication.

-1420 Current detector type (X) mismatch!

When the control power supply is turning ON, the ID code data that the ID code and the AS software of 1GM board installed on 1GB board correspond is not corresponding. 1GM is fixation with 6 axes machine and KHI standard 7 axes machine for the traverse joint against the model. However, because the model of the motor since the 7th axis is different, you should specify the ID code on the AS software excluding KHI standard traverse and more than 7 axes machine. 1GM board is not suitable for the AS software (robot model). The data of the ID code in the AS software is different from 1GM actual board things except more than 7 axes and a standard traverse joint. The AS software does not correspond to the 1GM board (model).
⇨Install 1GB board equipped with 1GM correct board.

-1500 Motor overload jt-x.

The current feedback from the power block was observed and the current which exceeded the continuous ratings current of the motor flowed more than permissible time. The servo software detects this error by internal data (OL curve). Mechanical factor : The robot arm comes in contact with the treatment device etc. The harness etc. are caught to the robot arm. The decelerator, the gear, and the bearing, etc. are damaged. In the gear deceleration part, the backlash is narrow. The weight of the load exceeds the ratings pay load of the robot. It is a robot motion (counter teach etc.) by which the robot motion pattern exceeds ratings of the motor. The motor brake is not released. Electric factor :Motor power line U, V, and W phase are disconnected. The brake line is disconnected or the brake drive circuit (Build in the servo board (1GB board etc.)) is damaged. The power block is damaged. Defect of servo board. Defect of motor.

⇨Check the decelerator etc. for a mechanical factor. And, exchange decelerator etc. if it is necessary. Save the robot arm with Teach mode when the robot arm is interfering. Afterwards, confirm the dislocation by driving part and the arm transformation such as decelerators. Check the harness and Servo Unit for an electric factor. And, exchange the harness and Servo Unit if it is necessary.

-1501 Motor hot.

The thermal contact connected with the inside machine harness worked. However, there is a model short-circuited without building a thermal contact nto according to the model in connector BOX of the robot arm or model from which plural series are connected with thermal. Disconnection of thermal line. The is defective of the separation harness. Defect of servo board. When thermal is built-in : Immediately after occurring error such as overloads. The robot rated weight capacity is exceeded. The ambient temperature of the robot exceeds the temperature of the robot of use environment. The robot motion which the load such as the counter teaches rested upon was executed continuously. Hold of cooling fan which places to robot and defect of air purge for cooling. The servo ring was continued for a long time by posture hanging the gravity load on the arm. Defect of thermal switch.

⇨When a thermal motor works, error reset cannot be done until cooling ends.

ERROR CODES/HELP INFORMATION

Confirm the use condition of the robot and use the robot in ratings. Do the disconnection check of a thermal line and exchange the harness and the servo board. Put from interlock etc. into the state of HOLD when robot continue the servo ring during a long time (line holding etc.) by the posture hanging the gravity load on the arm. Or, set “Automatic Servo Off” function with “Auxiliary 91 ENVIRONMENT DATA” and ENV_DATA command.

-1503 Speed error jt-x.

Joint speed (motor speed) calculated with encoder value exceeded a regulated value. At the repeat : Joint ratings speed of robot*1.2. At teach/check : At the equivalent radius position (The slide joint is actual command value speed), 250mm/sec *1.5. Disconnect the motor power line U, V, and W phase, and fall of arm by power defective block etc. Malfunction due to wiring mistake in motor power line and encoder line. Disconnection of encoder signal line, short-circuit and defect of main body of encoder. The robot depends on the singularity motion. Defect in servo board and power block. The moment of inertia exceeded the motor torque by installing the load more than ratings at the highest velocity robot motion.

⇨Check the disconnection and the short-circuit etc. of the wiring mistake and wiring and exchange the harness and the encoder, etc. Exchange Servo Unit etc. Correct the teach data such as the reductions of the modification and setting of the speed at the position (posture) in case of the singularity motion.

-1504 Position envelope error jt-x.

The difference between the current value from the encoder and the command value on the AS software exceeded a regulated value. Mechanical factor : The robot arm comes in contact with the treatment device etc. The harness etc. are caught to the robot arm. The decelerator, the gear, and the bearing, etc. are damaged. In the gear deceleration part, the backlash is narrow. The weight of the load exceeds the ratings pay load of the robot. It is a robot motion (counter teach etc.) by which the robot motion pattern exceeds ratings of the motor. The motor brake is not released. Electric factor : Motor power line U, V, and W phase are disconnected. The brake line is disconnected or the brake drive circuit (Build in the servo board (1GB board etc.)) is damaged. The power block is damaged. Defect of servo board. Defect of motor. When robot do the singularity motion.

⇨Check the decelerator etc. for a mechanical factor. And, exchange decelerator etc. if it is necessary. Save the robot arm with Teach mode when the robot arm is interfering. Afterwards, confirm the dislocation by driving part and the arm transformation such as decelerators. Check the harness and Servo Unit for an electric factor. And, exchange the harness and Servo Unit if it is necessary. Correct the teach data in case of the singularity motion.

-1505 Velocity envelope error jt-x.

Specific information on this error code was not available at the time of publication.

-1506 Commanded speed error jt-x.

Specific information on this error code was not available at the time of publication.

-1507 Commanded acceleration error jt-x.

Specific information on this error code was not available at the time of publication.

-1510 Encoder harness disconnect jt-x.

Specific information on this error code was not available at the time of publication.

-1511 Encoder battery low voltage.

The voltage of the battery connected with the encoder battery backup board (1FG board etc.) which built-in the robot arm has decreased to 3.2V or less (ratings 3.6V). Or, encoder battery alarm signal (BAT_AL) signal from the encoder battery backup board was disconnected. However, this error is detected only at control power supply ON time and Motor Power ON time. Longevity of battery for encoder backup. Defect of encoder battery backup board. The defect of the encoder or, short of the harness from the encoder battery backup board to the encoder are a cause, the amount of the consumption of the battery increases, and the battery is discharged. Battery alarm signal (BAT_AL) from the encoder battery backup board was disconnected. Defect of servo board (1GB board etc.).

Caution : There is a possibility that the internal data of the encoder is not maintained when the control power supply is turned OFF at the error occur. ⇨There is a possibility that the internal data of the encoder is not maintained when this error is occurred. Therefore, confirm whether the degree is 0 degrees after putting all axes together on the position where the robot is marked (mechanical origin) with Teach mode without fail. Execute zeroing

when they are not 0 degrees. After error reset is done, this error can be automatic operation. However, note that there is a possibility to cause the dislocation when the robot is operated without confirming zeroing like the above-mentioned. Note that there is a possibility that the internal data of the encoder is not maintained when the control power supply is turned off when this error is occurred. Exchange the battery promptly when the battery for the encoder backup decreases.(Exchange the battery for the conveyer, or for multi axis at the same time). Check each harness in disconnecting and the short-circuit and exchange the harness. Exchange the encoder, the servo board, and the encoder backup board, etc.

-1513 Encoder rotation data abnormal jt-x.

Occurs when there is a difference between the rotation data in the serial encoder data and the calculated rotation data by incremental technique.
⇨Replace encoder, replace 1GB board. Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Ring out the machine harness and separation harness. Replace 1FG board.

-1516 Encoder data abnormal jt-x.

When difference between current value when control power supply is turned off and current value when control power supply is turned on is more than value set by “Auxiliary 43 ENCODER ERROR RANGE”, ENCCHK_DATA command. The ENCCHK_DATA command is usually set in 2.0 degrees with the rotation axis. This error is not detected when setting as 0.0 degrees. When a usual control power supply is turned on, this error is occasionally occurred when a too small value is set. The internal data of the encoder is not maintained with a disconnection of the decrease of the battery for the encoder backup and the encoder harness and defective encoders, etc. The control power supply was turned OFF by an abnormal power supply etc. in the robot motion (Make to turning OFF). The reason for the arm is that the gap might be caused in the final current value which the AS software memorized and the current value which stops actually (Turns ON the control power supply) because hold does not occur rapidly. Initializing was executed. The motor (encoder) was exchanged. The arm (motor) operated by force when the control power supply was turned OFF.

⇨When this error is occurred immediately after initializing is executed, it is not abnormal. Do zeroing promptly when you exchange the motor (encoder). There is a possibility that the internal data of the encoder is not maintained when this error is occurred. Therefore, confirm whether the degree is 0 degrees after putting all axes together on the position where the robot is marked (mechanical origin) with Teach mode without fail. Execute zeroing when they are not 0 degrees. After error reset is done, this error can be automatic operation. However, note that there is a possibility to cause the dislocation when the robot is operated without confirming zeroing like the above-mentioned. Exchange the battery promptly when the battery for the encoder backup decreases. Check each harness in disconnecting and the short-circuit and exchange the harness. Exchange the encoder, the servo board, and the encoder backup board, etc.

-1517 Can't read initial data encoder jt-x.

When the encoder data was read immediately after the control power supply was turned ON, the steady data was not able to be read. Disconnection of encoder signal line and short-circuit and defect of main body of encoder. Defect of servo board (1GB board etc.).
⇨Disconnection and short-circuit of encoder signal line and exchange the main body of the encoder.

-1518 Miscount of encoder data jt-x.

Specific information on this error code was not available at the time of publication.

-1521 Mismatch ABS and INC encoder of jt-x.

Occurs when there is a difference between encoder data in the serial encoder data and the calculated data by incremental technique.

⇨Replace encoder, replace 1GB board. Check connections at encoder , 1FG board, and separation harness. Check connections, jumper, and switches on 1GB board. Ring out the machine harness and separation harness. Replace 1FG board.

-1524 Encoder line error of jt-x.

Specific information on this error code was not available at the time of publication.

-1550 Encoder initialize error jt-x.

Occurs when encoder continuously outputs Busy signal during controller reads of encoder data at initial power-up.

⇨In the case of a conveyor encoder, ensure power-up speed is below 300 RPM. Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Replace encoder, replace 1GB board, replace 1FG board. Ring out the machine harness and separation harness.

-1553 Encoder response error jt-x.

Occurs when the encoder does not respond to the data request signal from the 1GB/1GE board.

⇨Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Replace encoder, replace 1GB board. Ring out the machine harness and separation harness. Replace 1FG board.

-1554 Encoder communication error jt-x.

Occurs when the encoder serial data is not correctly transmitted according to communication protocol.

⇨Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Replace encoder, replace 1GB board. Ring out the machine harness and separation harness. Replace

1FG board.

-1555 Encoder data conversion error jt-x.

Occurs when the M-code data from the encoder has error pattern.
⇨Replace encoder, replace 1GB board. Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Ring out the machine harness and separation harness. Replace 1FG board.

-1556 Encoder ABS-track error jt-x.

Occurs when the encoder outputs ABSLAM in the serial encoder data.
⇨Replace encoder, replace 1GB board. Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Ring out the machine harness and separation harness. Replace 1FG board.

-1557 Encoder INC-pulse error jt-x.

Occurs when the encoder outputs AINPALM in the serial encoder data.
⇨Replace encoder, replace 1GB board. Check connections at encoder, 1FG board, and separation harness. Check connections, jumpers, and switches on 1GB board. Ring out the machine harness and separation harness. Replace 1FG board.

-1558 Encoder MR-sensor error jt-x.

The state of the MR sensor in the encoder does not match with JT ENCODER in one rotation.
⇨Exchange the encoder harness.

-1559 Power module error jt-x.

The error signal from the IPM module in the power block was detected. Defect in power block. Defect of servo board (1GB board etc.). Short-circuit of motor power line U, V, and W phase. Cooling by Hold etc. of the cooling fan in Servo Unit is defective. Defect the harness between the servo board and the power blocks and defect of connect.
⇨Exchange Servo Unit. Check the short-circuit of the motor harness and

exchange it.

-1561 Current sensor disconnect [Servo (X)].

The current sensor cable between the servo board (1GB board etc.) and the power blocks is not connected. The channel of the servo board is displayed in X. 1GB board is not connected with the current sensor cable between the power blocks. Defect of 1GB board.

⇨Connect a current sensor cable of 1GB board and the power block.

-1563 Servo unit 12VDC error [Servo (X)].

When ±12V supplied to 1GB board reaches the below value : +12V : +10.75V or less, -12V : -10.4V or more. The channel of the servo board is displayed in X. ±12V supplied to 1GB board decreases. Defect such as the harness between 1GB board ←→ motherboards ←→ AVR and the connect is defective. Defect of 1GB board.

⇨Check the voltage and exchange 1GB board and AVR for the control power supply, etc.

-1567 Regenerative resistor error [Servo (X)].

The regenerative time (time to throw the current to the regenerative resistance) was consecutive from the power block and 6sec or more continued. The motor enters the state of power generation by the inertia of the arm when the robot decelerates. The voltage flows backward to Motor Power (Voltage between P-N) and the voltage between P-N enters the state of the over-voltage. The regenerative processing is a processing by which the current is thrown to the regenerative resistance when the voltage rises more than a constant value and the elevation of the voltage is prevented. In the condition to throw the current to the regenerative resistance, the voltage between P-N is the below condition. The unit name of the servo board is displayed in X. The counter teach is high-speed in the robot. Fail the scorch of the regenerative resistance etc., and defect in power block. Defect of servo board (1GB board etc.). The connect is defective of the harness during servo board (1GB-CN13) ←→ power block (CN9). Defect of regenerative resistance unit for increase when regenerative resistance unit for built more is connected.

⇨Do not do the counter teach to the robot at the teaching modification. Or,

reduce execution speed. Install the built more regenerative unit option. Exchange the servo board, the power block, and the regenerative resistance unit for the built more, etc.

-1568 Servo unit P-N low voltage [Servo (X)].

The voltage between P-N supplied to the power block is DC60V or less at the servo ring. The channel of the servo board is displayed in X. Operation Unit side : Defect of MS. Defect of relay board (1FY board etc.). Defect of power sequence board (1FP board etc.). Servo Unit side : Defect of defective power unit. Defect of Motor Power circuit (diode bridge and MS, etc.). Defect of servo board (1GB board etc.). Defect of each harness between Operation Unit ←→ Servo Units and defect of connect.

Caution : The arm falls between until detecting of the error because Motor Power is not supplied at the servo ring when this error occurs. Other errors of abnormal speed and abnormal deflection, etc. might be detected.

⇨Check and exchange the Motor Power circuit and each equipment. Exchange the power sequence board, the relay board, and Servo Unit.

-1569 Servo unit P-N high voltage [Servo (X)].

The voltage between P-N supplied to the power block exceeded DC410V at the savoring. The channel of the servo board is displayed in X. Defect of regenerative resistance control circuit, fail scorch of regenerative resistance, defect of voltage watch circuit, and defect in power block. Defect of servo board (1GB board etc.). Defect of built more regenerative resistance unit. The counter teach etc. which hung excessively the regenerative were executed.

⇨Exchange Servo Unit and the built more regenerative unit, etc. Add the regenerative resistance unit option when the regenerative is excessive.

-1570 Regenerative resistor over-heat [Servo (X)].

Thermal (140℃) for resisting regenerative in power block, thermal (140℃) for resisting regenerative in built more regenerative unit option, and a heat sink thermal (90℃) are worked. The channel of the servo board is displayed in X. Defect of cooling by hold of cooling fan in Servo Unit and regenerative resistance unit. The regenerative ability is low at the counter teach etc. The temperature of the controller of use environment is high. Defect in power block. Defect of regenerative resistance unit. Defect of servo board (1GB board

etc.). The connect is defective of the harness during servo board (1GB-CN13) ←→ power block (CN9). Defect of the harness between power block (1GC-CN21) ←→ built more regenerative units, and defect of connect.
⇨Exchange Servo Unit and the built more regenerative unit, etc. Reduce the ambient temperature. Check each harness.

-1600 Uncoincidence error jt x.

Even if the fixed time has passed since the command value arrived at the robot motion terminal, the current value does not become an axis coincidence. Mechanical factor : The axis is heavy because of the damage of the decelerator and the bearing, etc. In the gear deceleration part, the backlash is narrow. The arm is interfering in the treatment device etc., or the harness etc. are caught etc. The motor brake is not released. Electric factor : Defect of servo board (1GB board etc.). Defect in power block etc. Disconnection of motor power line and brake line. Defect of encoder and encoder harness. Defect of the teaching data. The command value is converted into joint command after being calculated with the wrist posture maintained in the XYZ coordinates system and operates usually, at the interpolation motion. However, this error might occur when the angle of the wrist at the operation terminal position is different even if the wrist posture (opposite direction such as JT6 etc.) at the robot motion beginning point and the terminal position where the teaching was done is the same on XYZ coordinates.
⇨Adjust the exchange and the backlash such as decelerators for a mechanical factor. Exchange Servo Unit and the harness, etc. for an electric factor. Do the teach modification when the error occurs by a specific program step.

-1601 Limit switch of jt x is ON.

The axis restriction limit switch operated with the current value exceeded the work envelope of the user setting. The work envelope of each Joint of the user setting has been set with “Auxiliary 51 SOFTWARE LIMIT” /ULIMIT, LLIMIT command. Set the work envelope of the user setting as becoming to the follow condition. Maximum work envelope > Mechanical stopper > Axis restriction limit switch > User set work envelope. So as not to operate, the AS software restricts robot more than the angle set by the user set work envelope usually. The signal line in two systems is connected from CN4 of 1FP board to 1FG board in the robot. However, this is connected in parallel in 1FP board. If either is short-circuits when the limit switch in two systems is connected with 1FG board to detect only 1CH, it can actually detect the error. Moreover, CN11, 12, 13 of 1FG board, and a part of model are connected and when the limit switch is not connected with the LS connector of the inside machine harness, it should connect the short-circuit connector. When the axis restriction limit switch is turning ON, the user set work envelope was set in smaller value than the current value. The axis was moved by using the manual brake release etc. and the limit switch was turned ON. The signal of the axis restriction limit was disconnected when the robot was outside the user set work envelope. The axis restriction limit was turned ON by wrong of setting the zeroing and reckless driving, etc. The limit was turned ON by the over shot by the inertia when the robot worked when the user set work envelope is a value close to the robot motion angle of the axis restriction limit switch. Disconnection of axis restriction limit signal and defect of harness.

⇨When this error occurs, robot releases the axis restriction limit switch signal only while the over ride switch of the power sequence board (1FP board etc.) is being pressed. You can turn ON Motor Power. And, move the arm with Teach mode in the work envelope. Or, You connects manual brake release BOX of the option with CN5 of 1GB board. Release the brake with the manual and move the arm in the work envelope. Turn OFF Motor Power (MC OFF) and put into the state to release EMERGENCY STOP Switch (contains External EMERGENCY STOP Switch) when you use the manual brake release. Set the user set work envelope in an appropriate value. Change the installation angle of the axis restriction limit switch to an appropriate value. Check and exchange the harness and the limit switch. Exchange the power sequence board.

ERROR CODES/HELP INFORMATION

-1602 Limit switch signal line is broken.

The axis restriction limit switch operated with the current value exceeded the work envelope of the user setting. The work envelope of each Joint of the user setting has been set with “Auxiliary 51 SOFTWARE LIMIT” ULIMIT/LLIMIT command. Set the work envelope of the user setting as becoming to the follow condition. Maximum work envelope > Mechanical stopper > Axis restriction limit switch > User set work envelope. So as not to operate, the AS software restricts robot more than the angle set by the user set work envelope usually. The signal line in two systems is connected from CN4 of 1FP board to 1FG board in the robot. However, this is connected in parallel in 1FP board. If either is short-circuits when the limit switch in two systems is connected with 1FG board to detect only 1CH, it can actually detect the error. Moreover, CN11, 12, 13 of 1FG board, and a part of model are connected and when the limit switch is not connected with the LS connector of the inside machine harness, it should connect the short-circuit connector. The signal of the axis restriction limit was disconnected (ON) while had set in the value whose user set work envelope is larger than the limit switch. The axis restriction limit was turned ON by wrong of setting the zeroing and reckless driving, etc. Disconnection of axis restriction limit signal, defect of harness. Defect of limit switch. Defect of power sequence board.

⇨When this error occurs, robot releases the axis restriction limit switch signal only while the over ride switch of the power sequence board (1FP board etc.) is being pressed. You can turn ON Motor Power. And, move the arm with Teach mode in the work envelope. Or, You connects manual brake release BOX of the option with CN5 of 1GB board. Release the brake with the manual and move the arm in the work envelope. Turn OFF Motor Power (MC OFF) and put into the state to release EMERGENCY STOP Switch (contains External EMERGENCY STOP Switch) when you use the manual brake release. Set the user set work envelope in an appropriate value. Change the installation angle of the axis restriction limit switch to an appropriate value. Check and exchange the harness and the limit switch. Exchange the power sequence board.

-1610 Torch is interfered.

ERROR CODES/HELP INFORMATION

-1800 AC primary power off.

The instantaneous decrease in the primary power was detected in AVR for the control power supply. However, because the control power supply is turned OFF when this error occurs; Confirm displaying the error remains when you turn ON the control power supply to next time. NFB for the control power supply was turned OFF. AC200/220V supplied to AVR for the control power supply caused the instantaneous decrease. Defect of AVR for control power supply. Defect of NFB for control power supply. The voltage of the power supply circuit and Motor has decreased short. Or, NFB tripped. The limit switch was turned off because the plate (door) of Operation Unit and Servo Unit was opened and NFB tripped. Defect etc. of primary power. Defect of power sequence board (1FP board etc.). Defect of relay board (1FY board etc.).
⇨It is normal that this error is displayed when NFB for the control power supply is turned OFF. Confirm the limit switch in plate (door) part of Operation Unit and Servo Unit when NFB trips. Check the power supply circuit such as AVR for the control power supply and NFB. Confirm whether the primary power is supplied according to ratings.

-1801 24VDC power source is low.

DC+24V supplied to the power sequence board (1FP board etc.) by AVR for the control power supply detected by about 21.6V or less. However, because the control power supply is turned OFF when this error occurs; Confirm displaying the error remains when you turn ON the control power supply to next time. Defect of AVR for control power supply. Defect of power sequence board. Defect of relay board (1FY board etc.). When Motor Power ON circuit (EMERGENCY STOP Switch line and axis restriction limit switch line, etc.) is short-circuited to other circuits (power supply) etc. Short of inside machine valve and sensor line.
⇨Check the power supply circuit, the inside machine valve, and the short-circuit of the sensor line etc., and check the disconnection. Exchange AVR for the control power supply, the power sequence board, and the relay board.

-1802 Primary power source is high.

The voltage watch level height (when AC267-277V or more continues 1-2sec)

of AVR for the control power supply was detected. However, because the control power supply is turned OFF when this error occurs; Confirm displaying the error remains when you turn ON the control power supply to next time.
The primary power exceeds ratings. Defect of AVR for control power supply. Defect of power sequence board. Defect of relay board and power supply circuit.
⇨Confirm whether the primary power is supplied according to ratings. Check the power supply circuit such as AVR for the control power supply.

ERROR CODES/HELP INFORMATION

-1803 Primary power source is low.

The voltage watch level low (when AC150-158V or less continues 1-2sec) of AVR for the control power supply was detected. However, because the control power supply is turned OFF when this error occurs; Confirm displaying the error remains when you turn ON the control power supply to next time. Defect etc. of primary power. AC200/220V supplied to AVR for the control power supply became an instantaneous decrease. Defect of AVR for control power supply. Defect of NFB for control power supply. Defect of power sequence board (1FP board etc.). Defect of relay board (1FY board etc.).
⇨Confirm whether the primary power is supplied according to ratings. Check the power supply circuit such as AVR for the control power supply and NFB.

-1804 5VDC or ±12DVC is abnormal.

DC+5V ,DC±12V supplied to 1GA board are not full of the ratings voltage. DC+5V : Excluding +4.85-+5.45V DC+12V : +10.75V or less. DC-12V : -10.4V or more. However, because the control power supply is turned OFF when this error occurs; Confirm displaying the error remains when you turn ON the control power supply to next time. Defect of AVR for control power supply. Defect of each board such as 1GA boards. The contact of the motherboard is defective. Short and defect of MFP and Small T/P harness. Short in servo board and power block, etc. Defect of encoder backup board (1FG board) such as in robots. Short of harness from Servo Unit to encoder backup board. Shorts of operation switch and lamps (error reset switch, cycle start switch, and lamp, etc.). Short in option board and the connect equipment of built-in vision etc. The power capacity of AVR for the control power supply is insufficient because of the use such as the option boards.
⇨Exchange AVR and each board for the control power supply. Check short-circuits of each harness such as MFP, the operation panel equipment's, and the separation harness and exchange it.

-1805 Memory is locked because of AC_FAIL.

After a power supply abnormality (ACFAIL) (the control power supply includes the time of turning OFF) had been detected, memory was operated.
⇨If the control power supply is turned ON again, the error is released though it might usually be generated when the control power supply is turned OFF.

-2008 Clock service timer over.

10.1. ERROR RECOVERY

Figure 10-1 shows troubleshooting procedures for when the controller becomes unresponsive to commands, or an error is encountered that cannot be cleared. Troubleshooting should begin with confirmation of basic system integrity.
Ensure that:

- The power supply is on and meeting supply requirements.
- All cables are correctly attached.
- All peripheral equipment is wired correctly.

System Initialization

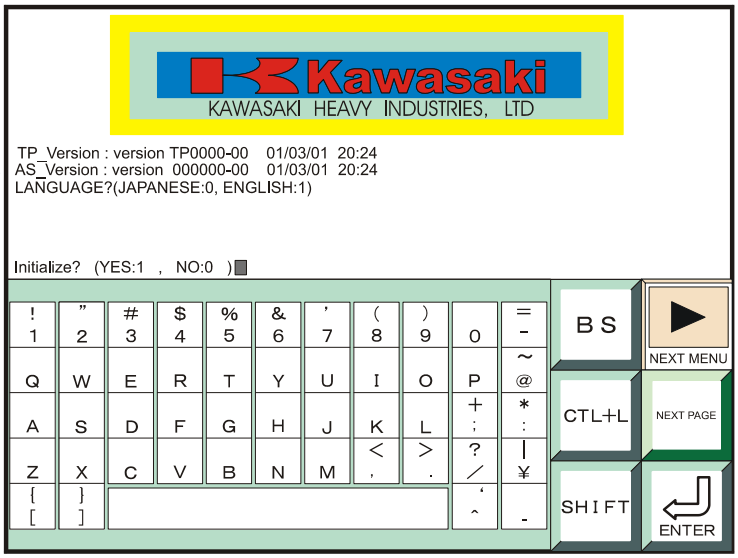
⚠ WARNING

Never execute the system initialization unless (1) upgrading system software, or (2) system-wide error leaves no other alternative.

Following are screenshots of the initialization screen for Multi Function Panel and personal computer.

Multi Functional : This screen is displayed when the control power switch is turned to ON.

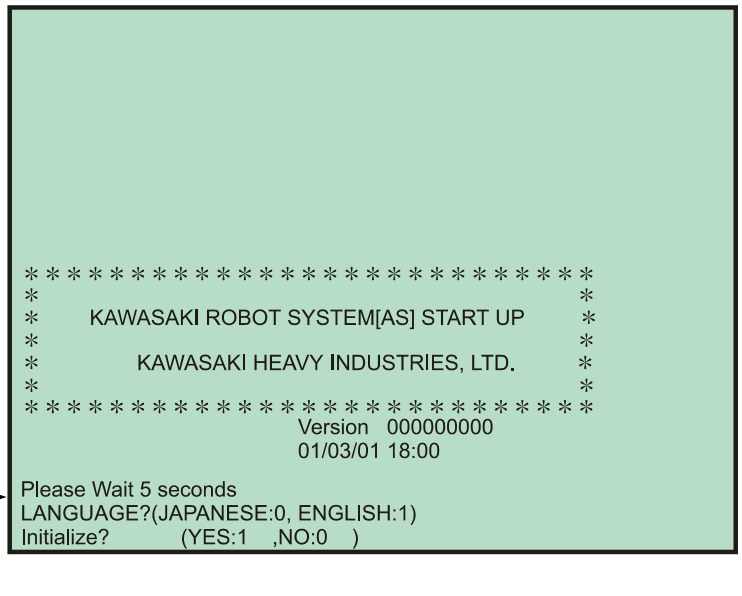
ERROR CODES/HELP INFORMATION



Note the option to choose English or Japanese as the default language.

ERROR CODES/HELP INFORMATION

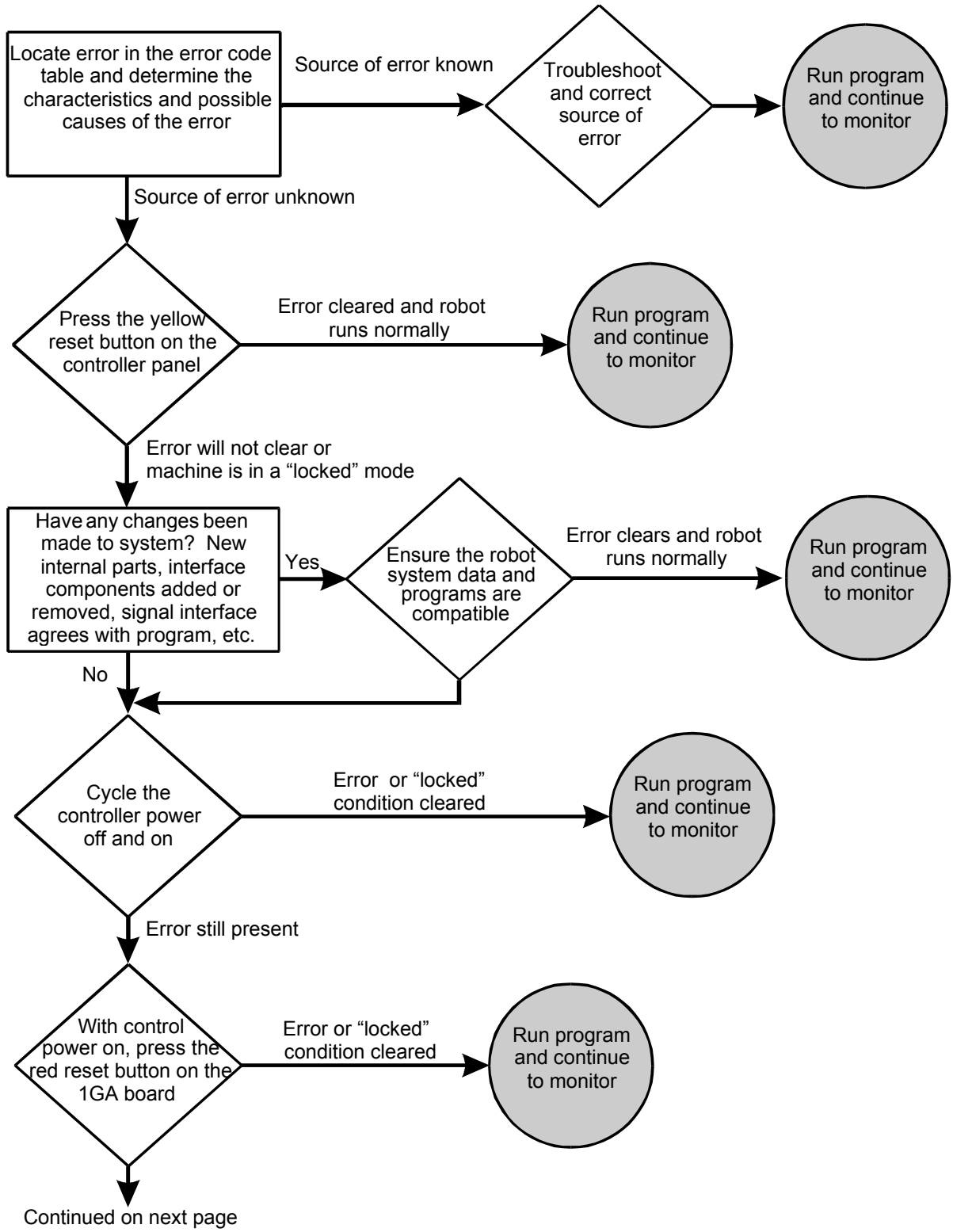
Personal Computer: This screen is displayed when the AS monitor software loads.



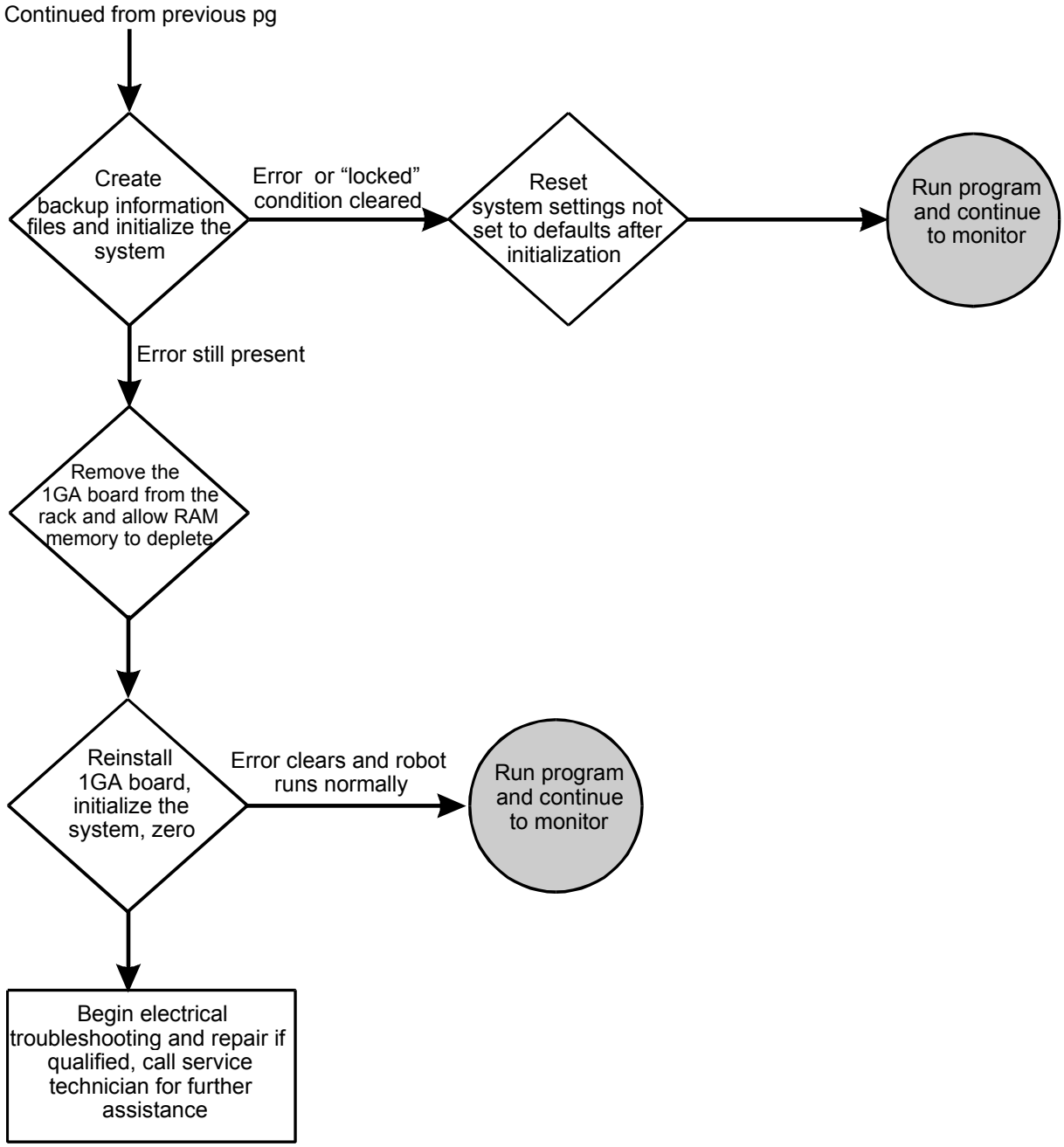
Note the option to choose English or Japanese as the default language.

These screens are accessed via the *call initialization screen* procedure.

ERROR CODES/HELP INFORMATION



ERROR CODES/HELP INFORMATION



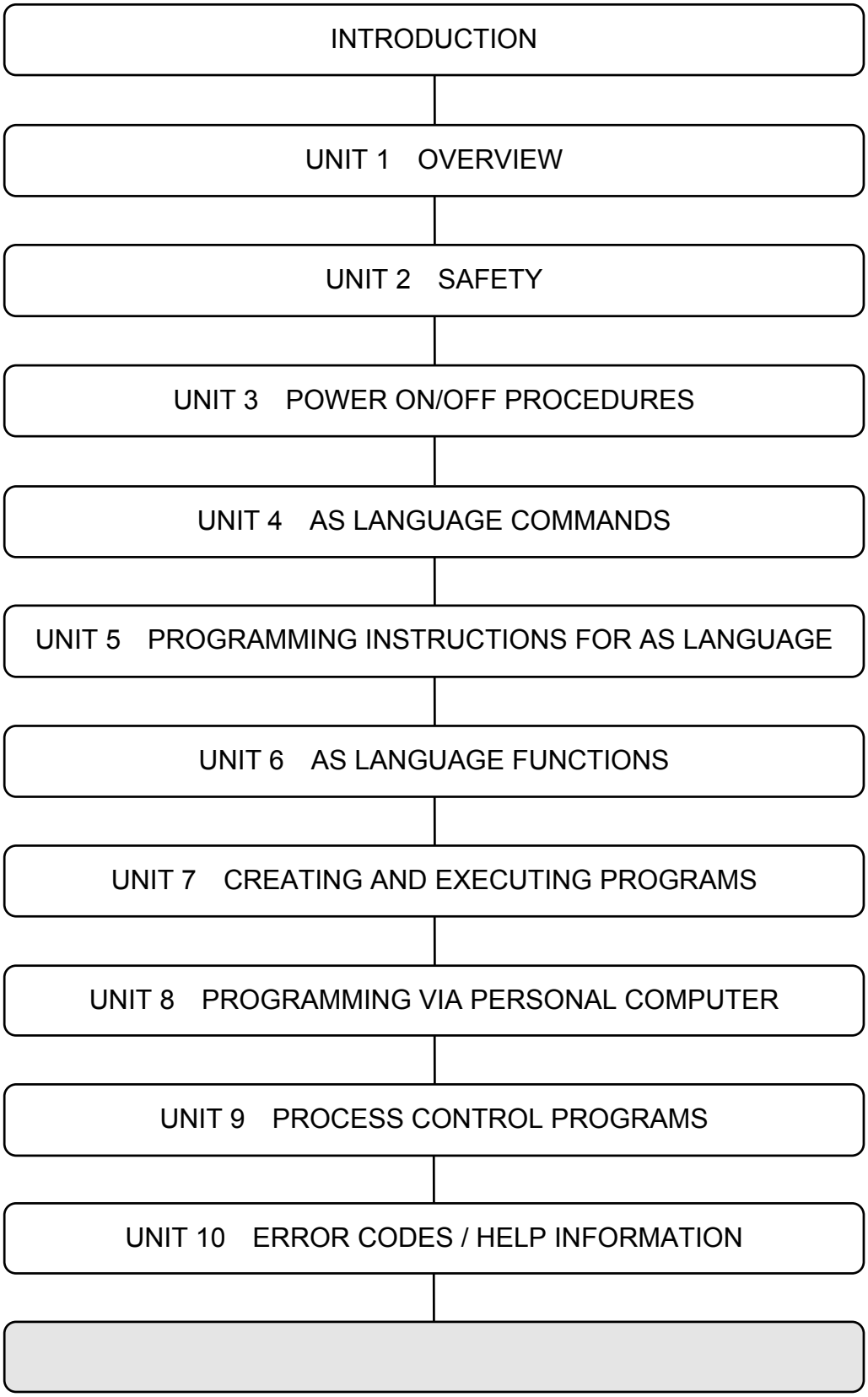
10.2. HELP COMMANDS

The **HELP** commands are used to display the AS language commands that are available to the programmer. The commands are displayed in alphabetic order in seven columns. The HELP command will display all of the AS language commands available. If more specific information is required, the HELP command can be used in conjunction with the following prefixes: HELP/M to display monitor commands, HELP/P to display program instruction commands, HELP/F to display function commands, HELP/PPC to display PC program commands, HELP/MC to display monitor commands for program control, HELP/DO to display commands that can be used from the monitor mode with the DO command. When the HELP command preceded by a space and a single letter, all of the AS language commands that begin with that letter will be displayed to the far right of the screen followed by the format for each command. Figure 4-54 shows an example of the HELP/M and HELP D commands.

```
$HELP/M symbol 191 ¥f "Symbol" ¥s 10↵
ABORT       BASE        BITS        BATCHK      CONTINUE    COPY        DEFSIG
DELETE      DIRECTORY   DLYSIG      DO          EDIT        ERESET      ERRLOG
$HELP D symbol 191 ¥f "Symbol" ¥s 10↵
DECEL       DECEL deceleration ALWAYS
DECOMPOSE   DECOMPOSE array variable [suffix] = location
DEFSIG      DEFSIG INPUT or OUTPUT
DELAY       DELAY time
DELETE      DELETE/P/L/R/S program or variable
DIRECTORY   DIRECTORY
DLYSIG      DLYSIG signal number, time
DO          DO instruction
DO          DO ... UNTIL condition
DRAW        DRAW dx, dy, dz, rx, ry, rz, speed
DRIVE       DRIVE joint number, angle, speed
Press NEXT PG key to continue.
```

Figure 10-2 HELP Commands


MEMO


INTRODUCTION
UNIT 1 OVERVIEW
UNIT 2 SAFETY
UNIT 3 POWER ON/OFF PROCEDURES
UNIT 4 AS LANGUAGE COMMANDS
UNIT 5 PROGRAMMING INSTRUCTIONS FOR AS LANGUAGE
UNIT 6 AS LANGUAGE FUNCTIONS
UNIT 7 CREATING AND EXECUTING PROGRAMS
UNIT 8 PROGRAMMING VIA PERSONAL COMPUTER
UNIT 9 PROCESS CONTROL PROGRAMS
UNIT 10 ERROR CODES / HELP INFORMATION

GLOSSARY

This glossary contains definitions of terms used by operators, programmers, and maintenance personnel who work with Kawasaki Robots. The definitions are listed in alphabetical order.

A

- ACCELERATE
  To speed up a process.

- ACCURACY
  A measure of the difference between the commanded robot arm position and the actual position. Also identifies how well an indicated value conforms to a true value (i.e., an actual or accepted standard value).

- ACRONYM
  A term made up of the initial letters of words in a set phrase. For example, LED is an acronym for light emitting diode.

- ADDRESS
  A number that identifies a specific location in the computer's or processor's memory. Means of identifying a location or data in a control system.

- ADDRESSING
  Computer operations store data in specific memory locations or addresses. The largest memory location determines the amount of data that can be stored. The larger the number, the larger the possible program.

- AIR CUT
  Moving a weld gun into position but without generating an arc.

- ALGORITHM
  A finite set of well-defined rules or procedures for solving a problem step-by-step.

- ALPHANUMERIC

GLOSSARY

Pertaining to a set of symbols that contain both letters and numbers, either individually or in combination.

- AMBIENT TEMPERATURE
  The temperature of air or liquid that surrounds a device.

- AMPERE (AMP)
  A unit of electrical current flow that is equivalent to one (1) coulomb per second. One (1) volt across one (1) ohm of resistance causes a current flow that is equivalent to one (1) amp.
- ANALOG
  A continuously changing electrical voltage signal. In robot systems, the magnitude or value of the signal represents commanded robot axis motion.

- ANALOG DATA
  Information that is represented by a characteristic of the value or magnitude of an electrical signal, such as the amplitude, phase, or frequency of the voltage, the amplitude or duration of a pulse, the angular position of a shaft, or the pressure of a fluid number.

- ANTI-FRICTION BEARING
  A rolling element which is used to support a rotating shaft.

- ARC SENSOR
  A sensor that detects weld lines utilizing arc characteristics.

- ARGUMENT
  A value applied to a procedure; data used by a function or other command. For instance, in the AS command JMOVE flange, 2. The variable, flange, and the clamp number 2 are the arguments of the function JMOVE.

- ARRAY
  An ordered set of addresses or their values. Elements of an array can be referenced individually or collectively. Array elements all have the same type of data, for instance, integer or character, and are usually presented in rows and columns.

GLOSSARY

・ARTICULATED
To join together permanently or semipermanently by means of a pivot connection for operating separate segments as a unit.

・ARTICULATED ROBOT
A robot arm which contains at least two consecutive revolute joints, acting around parallel axes, resembling human arm motion. The work envelope is formed by partial cylinders or spheres. The two basic types of articulated robots, vertical and horizontal, are sometimes called anthropomorphic because of the resemblance to the motions of the human arm.

・ASCII
An acronym for American Standard Code for Information Interchange. This standard 8-bit code is used by many devices, such as keyboards and printers.

・AS LANGUAGE
Kawasaki robot language used to communicate commands and instructions from a keyboard to the CPU. (AS : Advanced Superior, pronounced AZU)

・ASSIGNMENT
An instruction used to express a sequence of operations, or used to assign operands to specified variables, or symbols, or both.

・ASYNCHRONOUS
A means of data communication where the data is sent a character at a time preceded by a start bit and followed by a stop bit. No direct timing signal links the transmitter and receiver.

・AUXILIARY DATA
Information about a point, other than the positional data, such as speed, accuracy, weld schedule and clamp condition.

・AXIS
A straight line about which sections of the mechanical unit rotate (e.g., joints JT1, JT2, JT3 etc.).

GLOSSARY

B

- BACKLASH
  The clearance, slack, or play between adjacent gears, or the jar or reaction often caused by such clearance when the parts are suddenly put in action or are in irregular action.

- BASE COORDINATE
  A fixed coordinate system having an origin at the intersection of the X, Y, and Z axes.

- BAUD RATE
  Determines the number of bits per second (bps) or characters transmitted between devices.

- BCD
  An acronym for binary coded decimal. The BCD 8-4-2-1 code expresses each decimal digit by its own 4-bit binary equivalent. The 8-4-2-1 code is identical to binary through the decimal number 9. Above the decimal number 9 each decimal digit is represented by its own 4-bit binary equivalent. For example, using the 8-4-2-1 binary-coded system, the number 10 is interpreted as 0001 0000.

- BINARY CODE
  A system in which characters are represented by a group of binary digits, that have the value of either 0 or 1, true or false, on or off.

- BIT
  Acronym for binary digit, having one of two values: 0 or 1.
- BOOT
  The method by which computers are brought from a non-operating to an operating state. During this sequence, the computer memory is usually reset. This is often performed to restart the computer after a crash, to bring it on-line.

- BUFFER
  A temporary memory storage area in a computer or electronic device.

GLOSSARY

・BUG
A problem in a software or hardware element of a system.

・BUS
The primary communication path in the controller along which internal signals are sent among processors and memories.

C

・CABLE CARRIER
A device which carries cables and hoses (including power sources) from a stationary location to a linear moving device.

・CARTESIAN COORDINATE
A location in space defined by three axes at right angles to each other, commonly labeled X, Y, Z.

・cc
cubic centimeter

・CELL
A manufacturing unit consisting of two or more work stations or machines, and the material transport mechanisms and storage buffers that interconnect them.

・CENTER OF GRAVITY
The point at which the entire weight of a body may be considered as concentrated, so that if supported at this point the body would remain in equilibrium in any position.

・CHARACTER
A term that describes all numbers, letters, and other symbols typically found on a computer keyboard.

・CHECK MODE
A procedure that allows the user to check positional data and auxiliary data while in the teach mode with the Kawasaki robot. This procedure is in many ways analogous to reverse point and forward point operations in other robot models.

GLOSSARY

- CHECKSUM
  A method by which the contents of data or a transmission are verified to be accurate. This method “ums” all the characters and translates them into a number which is appended to the data.

- CHEMICAL ANCHOR
  A threaded rod installed in a structure (e.g., a concrete floor) and secured by epoxy, for the purpose of securing hardware.

- CIRCULAR INTERPOLATION
  A path taken by the robot that connects at least three points with an arching motion. The CPU will calculate a path that places the taught points on a section of a circle.

- CLOSED-LOOP SYSTEM
  A system in which a command value is output and a feedback value is returned. The resulting error, the difference between the command and the feedback, is used to correct the signal. In a robot system, the command signal is output by the controller, causing the robot arm to move, and the feedback signal is produced by the encoder, which reads the current position of the arm.

- CODE
  A set of rules for expressing information in a language that is understood and processed by a control system.
  Also, a term for instructions in a computer program. Code performs a process, and data is the information that is processed.

- COMMAND
  An analog signal, or group of signals or pulses, which cause a specified function to be performed. An instruction or request in a computer program that performs a particular action. Commands that are needed to run the operating system are called a command language.

- COMMENT
  Optional, non-executing remarks added to a program to explain various aspects of the program.

GLOSSARY

- COMPILER
  A system task that translates a program written in source code, into binary code that can be understood by the processor.

- COMPOUND TRANSFORMATION
  A location in the Cartesian coordinate system that is defined relative to another Cartesian coordinate location.
- CONTIGUOUS FILE
  A file that is stored in continuously adjacent areas of memory, in contrast to a file which is scattered to make more efficient use of disk space.

- CONTINUOUS PATH CONTROL
  A type of robot control in which the robot moves according to a replay of closely spaced points programmed on a constant time base during teaching.

- CONTROL ERROR CODE
  A code which identifies system problems whenever an alarm condition occurs.

- CONTROLLED AXIS
  A robot axis that is operated by electrical or hydraulic power.

- CONTROLLER
  An electronic device, with processing capabilities and software, which controls the robot actions and functions.

- CONVEYER TRACKING
  Used to make the robot follow a part on a conveyor, without the use of a traverse axis.

- COORDINATE
  A set of numbers that locate a point in space.

- CPU
  Acronym for central processing unit. A collection of hardware in a computer which

performs all calculations, handles I/O, and executes programmed tasks.

- CRASH
  A situation where the computer fails to operate, due to a software or hardware problem.

- CRT
  An acronym for cathode ray tube. A CRT is a charge storage tube in which the information is written by means of the cathode ray beam.

- CURRENT LOOP
  A circuit in which a portion of the output is returned to modify the control circuit output. This circuit may be used as a limiting device, for safety protection.

- CURSOR
  A pointer or indicator on a computer screen, that identifies the current position on the screen.
- CYCLE
  A complete path of projectory performed by the robot for a specific application.

- CYCLOIDAL DRIVE
  A mechanical gear reduction unit that reduces the speed of the input and increases the torque capacity. The cycloidal unit consists of an internal arrangement of discs and pins that are driven by an eccentric drive cam. This type of gear reduction offers low gear train backlash and the capability to achieve high reduction ratios from a single contained unit.

D

- DATA
  A term given to information, instructions, words or symbols that are usually transmitted, processed, or stored as a group.

- DETENT

GLOSSARY

A part of a mechanism that locks or unlocks a movement.

- DISCONNECT
  A switch that isolates a circuit or one or more pieces of electrical apparatus after the current has been interrupted by other means.

- DEVIATION ERROR
  In all mechanical devices, the actual position of the mechanical unit will lag behind the electrical command of the controller. An allowable limit is assigned for this difference. However, if the controller detects a condition where the difference between this mechanical value and the desired electrical position is larger than the established value limit, the robot controller will generate a deviation error. This error is sometimes referred to as a following error in the robot industry.

- DEBUG
  The process by which an operator's program is checked for mistakes and then corrected.

- DECIMAL NUMBER
  Numbers in the base-10 numbering system, which uses the numerals 0 - 9.

- DEDICATED
  A term used to describe a system resource, such as an I/O device or terminal, which is used for only one purpose, or assigned a single function.

GLOSSARY

- DEDICATED SIGNAL
  A term used to describe a signal which is used for only one purpose, or assigned a single function. Both inputs and outputs can be dedicated.

- DEFAULT
  A value or operation that is automatically entered by the system, if the operator does not specify one. Typically, the default is the standard or expected response.

- DELETE
  A command which will eliminate unwanted data.

- DELIMITER
  A character which separates a group of items or a character string, from other groups, or which terminates a task.

- DEVICE
  Any peripheral hardware connected to the processor and capable of receiving, sorting, or transmitting data.

- DIAGNOSTICS
  Function performed by the processor to identify and check for error conditions in the robot arm and peripheral devices.

- DIP SWITCH
  DIP is an acronym for dual in-line package. A set of small switches on circuit boards that can be set for different configurations.

- DIRECTORY
  A logical structure that organizes a group of similar files.

- DISK
  A high-speed, random-access memory device.

- DISK-BASED SYSTEM
  System in which programs and files are stored on the hard disk and are read into

memory when requested by the user.

・DISK PACK
A device which is used to store additional data in a computer system, and is usually removable.

GLOSSARY

E

- ECHO
  Process in which characters that are typed on a keyboard are also displayed on the screen or are sent to the printer.

- EDITOR
  An aid for entering information into the computer system and modifying existing text.

- EMERGENCY STOP (E-Stop)
  An immediate stop of robot motion, selected by the operator with a switch.

- ENCODER
  An electromechanical device that is connected to a shaft to produce a series of pulses that indicate the position of the shaft.

- EPROM
  Acronym for erasable programmable read-only memory. The contents of this memory (computer chip) are retained, even when power to the system is turned off. Usually stores executive programs and critical system variables.

- ERROR LOG
  A report which contains a sequential list of system error messages.

- ERROR MESSAGE
  Messages displayed on the plasma screen of the robot controller, when the action requested by the operator could not be completed. Error messages can occur when components malfunction or if an incorrect command is typed by the operator.

- EXPRESSION
  A combination of real-valued variables and functions, and mathematical and logical operators. When evaluated, this combination yields a numeric value.

GLOSSARY

F

- FEEDBACK
  The transmission of a signal from a measuring device (e.g., encoder, transducer) to the device which issued the command signal within a closed-loop system. See CLOSED-LOOP SYSTEM.

- FIELD SIGNALS
  All electrical signals that exit or enter an electrical panel.

- FILE
  A set of related records or data elements, which are stored using one name and are arranged in a structure that can be used by a program.

- FILESPEC
  Includes the name, creation date and size of the specified file.

- FIXED DISK
  An electromagnetic mass storage device which is not removable. Hard disks have much higher storage capacity than floppy disks.

- FLOPPY DISK
  An electromagnetic mass storage device which can be removed and exchanged.

- FORM FEED
  Process which causes a printer to advance the paper to the top of the next page.

- FUNCTION
  A formula or routine for evaluating an expression.

G

- GAIN
  A proportional increase in power or signal value relative to a control signal. The ratios of voltage, power, or current as related to a reference or control signal input.

GLOSSARY

- GLOBAL
  Refers to a function or process that affects the entire system or file.

- GRAY CODE
  A positional binary number notation in which any two numbers whose difference is one are represented by expressions that are the same except in one place or column and differ by only one unit in that place or column.

H

- HALF-DUPLEX COMMUNICATION
  Data transmission between two devices, where the signal is sent in only one direction at a time.

- HANDSHAKING PROTOCOL
  Communication rules used for data transmissions between devices. Each device must recognize the same protocol in order to communicate.
- HANG
  A term which refers to the state of a computer system that seems to be inoperative when processing should be taking place.

- HARDSTOP
  A mechanical constraint or limit on motion.

- HARDWARE
  Physical equipment and devices such as computer hard disk, cables, printer, etc.

- HAZARDOUS SIDE
  The unsafe side of a component or panel, such as the inside of the control panel when power is applied and functions are being performed.

- HOLD
  When an external or an internal input is available for a hold condition, the robot will stop its motion and servo drive power will be removed from the robot. When an external hold reset is performed, the servo drive power will be energized.

GLOSSARY

・HOME POSITION
Refers to the starting or resting position of the robot.

・HYBRID ENCODER
On the Kawasaki robot a hybrid encoder is used to generate positional data, and is composed of an incremental encoder that generates incremental pulses, and an absolute encoder that generates gray code binary data.

I

・ID
Abbreviation for Identification.

・INCHING
A value that is used during the jogging process that allows the user to position the robot in small minute increments.

・INCREMENTAL CODE
A digital closed loop feedback code that provides digital feedback pulses to the robot controller for the purpose of providing positional information. These incremental pulses are generated by an encoder through the use of an optical disk with alternating opaque and transparent bars or lines around the periphery of the disk. On one side of the disk a light source is mounted, and on the opposite side a phototransistor. When the disk rotates, the phototransistor is alternately forced into saturation and cutoff, producing the digital signal.

・INPUT
Transmission of an external signal into a control system.

・INTEGRATED CIRCUIT (IC)
A combination of interconnected circuit elements which are within a continuous substrate.

・INTERACTIVE SYSTEM
System where the user and the operating system communicate directly; the user through the keyboard, and the operating system via the display screen.

GLOSSARY

- INTERLOCK
  An arrangement whereby the operation of one part or mechanism automatically brings about or prevents the operation of another.

- INTERPRETER
  A program that changes English-like commands into machine language. An interpreter translates and executes one command at a time.

- INSTRUCTIONS
  Discrete steps in a computer program that are commands or statements that tell a computer to do something or identify data.

- INTEGER
  A whole number, a number without a fractional part such as 7, -318, or 19.

- INTERFACE
  The circuitry that fits between a system and a peripheral device to provide compatible coupling between the two pieces of equipment.

- INTERPOLATION
  The mathematical process that the CPU utilizes to plot a path for the robot to travel from one position to another. A mathematical process that evaluates a number of dependent and independent variables for the purpose of comparison and prediction.

- INTERRUPT
  An external signal that halts program execution so that the computer can service the needs of some peripheral device or subsystem.

- INTRINSIC SAFETY BARRIER (ISB)
  An electronic device used in robot controllers to restrict current and voltage to a safe level.

- INVERTER
  A circuit which switches a positive signal to a negative signal, and vice versa.

GLOSSARY

・I/O
Acronym for Input/Output. The interconnections through which the computer and its peripheral devices communicate.

・IPM
Acronym for Intelligent Power Module

J

・JOG
A term used to describe the process in which the user moves the mechanical unit through interaction with the robot controller and the teach pendant. Sometimes referred to as slewing.

・JOINT
1. A term used to describe the individual axes of a robot.
2. A term used to describe the jogging process in which the robot is jogged one axis at a time.

・JOINT MOVE
A mode of operation in which the robot moves from one point to the next with an arching path. All axes motors (required for the move) begin and end their rotation at the same time. The tool center point does not follow a linear path to reach the taught position.

L

・LABEL
An identifier for a program command line. To identify an instruction, memory location, or part of a program.

・LAN
An acronym for local area network. A group of computer terminals interconnected by

cables, allowing communication of information via the network.

・LCD

An acronym for liquid crystal display. This type of display is made of material whose reflectance or transmittance of light changes when an electric field is applied.

GLOSSARY

- LIMIT SWITCH
  An electrical switch positioned to be switched when a motion limit occurs, thereby deactivating the actuator that caused the motion.

- LINEAR MOVE
  An operation where the rate and direction of relative movement of the robot arm are continuously under computer control.

- LINE PRINTER
  A high-speed output device that prints a line at a time.

- LINE TURN-AROUND
  Changing the source of transmission in half-duplex communications.

- LOGICAL OPERATION
  Any of several operations that manipulate information according to the rules of logic (e.g., AND, OR, NOT, and exclusive OR).

- LM
  Abbreviation for linear motion.

- LOAD
  The weight applied to the end of the robot arm.

- LOCKOUT
  Serving to prevent operation of a device or part of it.

- LSB
  Abbreviation for least significant bit.

M

- MANIPULATOR
  Another term for the mechanical portion of the robot system.

- MACHINE LANGUAGE
  A low-level computer language, usually written in binary code.

GLOSSARY

- MASS-STORAGE DEVICE
  An input/output device that retains data input to it. Examples include: hard disk drives, magnetic tapes, floppy diskettes, and disk packs.

- MECHANICAL UNIT
  robot (excluding controller)
- MEMORY
  An area of the computer which stores data, either permanently or temporarily. When a program is requested, it is first loaded into memory so it can be accessed quickly by the processor.

- MHz
  Abbreviation for megahertz. One million cycles per second.

- MIRROR IMAGE
  A process which converts the positive and negative values of a taught path from a right-handed robot to a left-handed robot, or vice versa. The actions of the opposing robots are then coordinated and synchronized.

- mm
  Abbreviation for millimeter.

- MNEMONIC
  A term used to help the operator remember a large string of words or commands.

- MODEM
  A signal conversion device that modulates and demodulates data into an audio signal for transmission.

- MOMENT OF INERTIA
  Used to calculate end of arm tooling and handling weights. The sum of the products formed by multiplying the mass of the load by the square of the distance from the tool mounting flange.

- MONITOR PROGRAM
  An administrative computer program that oversees operation of a system. The AS monitor accepts user input and initiates the appropriate response, follows instructions

GLOSSARY

from user programs to direct the robot, and performs the computations necessary to control the robot.

- MSB
  Abbreviation for most significant bit.

- msec
  Abbreviation for millisecond (0.001 seconds).

N

- NOISE
  Any unwanted disturbance within a dynamic, mechanical, or electrical system.
- NULLED
  An electrical zero state.

O

- OCTAL NUMBER
  A numeral in the base-8 numbering system, which uses the numerals 0 - 7.

- OFF LINE
  A state in which communications between two devices cannot occur (e.g., between a printer and a computer, if the printer is off line).

- ON LINE
  A state in which communication between two devices can occur.

- OPERATING SYSTEM
  A set of system tasks and commands that are entered by the operator and interpreted and performed by the system.

- OPEN LOOP
  A control which does not use feedback to determine its output.

GLOSSARY

- OPERATOR
  Any mathematical action or function. The arithmetic operators are: add, subtract, multiply, divide, modulo, and exponentiate. The relational operators are: greater than, less than, equal to, and their combinations. The logical operators are: AND, OR, exclusive OR, negate. The binary logical operators are AND, OR, exclusive OR, ones complement.

- OPTO
  An optical isolator or level converter.

- OVERFLOW
  When a value or buffer exceeds a predefined limit.

- OVERTRAVEL
  An error condition that exists when the robot exceeds its normal software limit values, and then actuates an overtravel limit switch.

- OVERVELOCITY
  When an axis exceeds a preset value for velocity.

GLOSSARY

・OX (OUTPUT EXTERNAL)
Information transferred from the robot controller through output modules to control output devices.

P

・PARITY
Method by which errors are detected. In this method the combined binary values of a byte are flagged as 1 or 0.

・PARSE
To break a command string into individual elements, so it can be interpreted.

・PASSWORD
A code, entered by the user, to permit access to protected information.

・PAYLOAD
The maximum weight that a robot can handle satisfactorily during its normal operations and extensions.

・PC PROGRAM
PC is an acronym for process control. A PC program cannot contain any step that causes robot motion. PC programs are used to evaluate logic and variables and execute program CALL and GOTO commands

・PERIPHERAL DEVICE
Hardware equipment which is external to the CPU, but that transmits and/or receives I/O from the processor. Examples include: printer, CRT screen, or disk.

・PHASE
The angular relationship between current and voltage in alternating current circuits. In a waveform or period function, the fraction of the period that has elapsed, as measured from a reference point. Phase angle is determined by multiplying the phase by 360 degrees.

GLOSSARY

- PINCH POINT
  Any point where it is possible for a part of the body to be injured between the moving or stationary parts of a robot and the moving or stationary parts of associated equipment, or between the material and moving parts of the robot or associated equipment.

- PLA
  Acronym for programmable logic array. Used in many servo drive circuits.
- PLAYBACK
  An operation where a taught path is run for evaluation purposes.

- PLC
  Acronym for programmable logic controller. Usually referred to as the cell module controller.

- POINT-TO-POINT
  A robot motion control in which the robot can be programmed by a user to move from one position to the next. The intermediate paths between these points cannot be specified.

- POLARITY KEYS
  These teach pendant or multi function panel keys allow the user to jog or slew the robot in the Joint, Base (XYZ), or Tool coordinates system.

- PORT
  The connection point of an opening or passage that is usually located outside the housing of a device.

- POSITIONAL DATA
  The location in space of the robot manipulator.

- POUNCE POSITION
  A positional location at a point near the workpiece, clear of the transfer mechanism and part, from which the robot is ready to begin production.

GLOSSARY

・PRINTED CIRCUIT
An assembly of electronic elements that provide a complete path of electrical current through conductive material deposited between terminals on an insulated surface.

・PRECISION POINT
The play back of robot location based on the angular position of the six axes, joint angles.

・PRINTING
A process in which characters are stamped on a surface, usually paper.

・PROGRAM
A predefined, step-by-step set of instructions that are entered into a computer so a specific process can be performed repeatedly without re-entering all the steps. Robot paths are stored and run as part of programs.

・PROGRAM EDIT
Modification of an existing program.

・PROCESSOR
Generally, any hardware or software system for carrying out programs and acting on data.

・PULSE WIDTH MODULATION (PWM)
A modulation process in which the instantaneous sampling of the modulating wave is caused to modulate the duration of the pulse. This type of modulation is also referred to as pulse duration modulation (PDM), or pulse length modulation (PLM).

R

・RAM
Acronym for random access memory. An area used by the CPU for processing and temporarily loading programs so they can be accessed quickly. The contents of

GLOSSARY

RAM are lost when the computer is powered down, unless battery backup is provided.

- REAL NUMBER
  A number with a fractional part, such as 1.75, -31.89, .5, -4.00, etc.

- REAL-TIME
  The actual time during which the computer analyzes and processes data: information is usually processed as it is received.

- REAL VARIABLE
  AS language term for a variable that has had a real value assigned to it.

- REPEAT MODE
  A mode of operation that allows the user to check positional and auxiliary information at a selected speed value, in a continuous or a step by step type of movement of the mechanical unit.

- REPEATABILITY
  The measurement of the closeness of agreement among repeated measurements of the same variable under the same conditions.

- REWRITE MODE
  A mode of operation which allows the user to rewrite positional, or auxiliary data, and to insert or delete step address locations.

- RISC
  Acronym for reduced instruction set computer.

- ROM (read-only memory)
  A memory device which is programmed at the factory and whose contents thereafter cannot be altered.

- RS-232C
  An ASCII specification for connections and communication between serial devices.

GLOSSARY

・RUN
A mode of operation that allows the user to select servo motor power to provide drive energy to the robot, allowing it to perform such modes of operation as teach, check, repeat, and rewrite.

S

・SAFETY PLUG
A device used with safety fencing to interlock the opening of the fence with the removal of power to the robot.

・SCROLL
When more information exists than can be displayed on one screen, the operator can move up and down through the data to view it. When data is scrolled to the screen, the information previously viewed moves up off the screen, and new information enters the screen from the bottom.

・SENSOR
A device used to detect various conditions: proximity, heat, pressure, etc. An electrical signal from the sensor can be used to communicate information to a robot program.

・SERIAL
A method of transmitting data by which only one bit is sent or received at any one point in time.

・SEQUENTIAL ACCESS
A method used by many computers whereby data is read in the order in which it is physically stored.

GLOSSARY

- SINGULARITY POSITION
  When the robot is processing a linear or circular move and two or more joints are in alignment. The CPU can not process the ambiguity of a singularity configuration and an error is generated.

- SOFTWARE LIMITS
  Programmed values that are included in a program at the point before a mechanical device hits an overtravel limit switch or a hard limit.

- SOFTWARE
  A set of written programs and instructions that are executed by a computer system.

- SOURCE CODE
  A program that contains the actual software instructions entered by the user, in contrast to object code which is source code that has been translated into a language which can be interpreted by the computer.

- STRING
  A series of characters that have been entered in a distinct sequence that can be interpreted as a valid statement or command.

- STROKE
  The movement in either direction of a mechanical part having a reciprocating motion. The entire distance passed through in such a movement.

- SUBSCRIPT
  A set of numbers that identifies an element of an array.

- SUBROUTINE
  A set of instructions that is run by another routine.

- SYMBOL
  A character or design that has a distinct meaning and/or is associated with something.

GLOSSARY

・SYNTAX
The proper way in which commands and phrases should be typed in order to be understood by the control system. If the operator incorrectly types a command (i.e., misspelled or invalid characters), a syntax error message will be displayed.

・SYSTEM DATA
Data that is specific to a individual robot. Zeroing data, upper and lower software limits, and software switch settings are all examples of system data.

・SYSTEM SWITCH
Software switches that are set to determine various configurations and characteristics of the robot system performance.

T

・TEACH MODE
A mode selected on the operator panel, during which robot arm axes positions can be taught by the operator and are recorded by the robot.

・TEACH PENDANT
A hand-held, portable device used by the operator during teach and jogging operations.

・TOOL COORDINATES
A Cartesian coordinate system in which the origin point is at the face plate of the robot and the orientation of the tool can be expressed in terms of a 3-dimensional space representation of X, Y, and Z projections.

・TOOL MODE
A mode of operation in which all motions are calculated to maintain the orientation of the tool in space.

・TORQUE
Something which produces or tends to produce rotation or torsion and whose

effectiveness is measured by the product of the force and the perpendicular distance from the line of action of the force to the axis of rotation.

- TRANSFORMER
  A device to convert the current of a primary circuit into variations of voltage and current used in secondary circuits.

- TRANSFORMATION
  A mathematical description of a location that defines the position and orientation of the location without regard for the configuration of the robot when it is at that location.

- TRAP POINT
  see pinch point

- TTL
  Acronym for transistor-transistor logic.

GLOSSARY

・TWO'S COMPLEMENT
A means of representing a negative number as one more than the binary complement of the absolute value of the number.

・TRIANGULAR WAVEFORM
A waveform that has the shape of a triangle and is used in determining sampling values for servo drive circuits that utilize pulse width modulation.

U

・UHF
Acronym for ultra-high frequency.

V

・VARIABLE
The name of a memory location or stored value. A variable can refer to a scalar or an array.

・VELOCITY COMMAND
This analog signal is directly proportional to motor speed, and provides the initial signal that is processed by the servo drive system to drive a servo motor.

・VELOCITY ERROR
When the robot controller detects an axis that has exceeded a preset value for velocity, the robot controller will E-stop the robot.

・VOLT
A unit of electrical potential difference and electromotive force. One volt is equivalent to the force required to produce one amp of current through one ohm of resistance.

W

GLOSSARY

- WORK ENVELOPE
  The effective range or reach of a robot's axes.

- WORLD COORDINATES
  A Cartesian coordinate system in which the origin point is near the base of the robot, and robot movement can be expressed in terms of a 3-dimensional space representation of X, Y, and Z projections.

- WRITE
  In computer systems, a process in which information is output to and stored by a device or area in memory.

- WS (WELD SCHEDULE)
  Data that is stored in the weld controller and provides the specific current, clamp pressures, etc. for spot welding applications.

- WX (WAIT EXTERNAL)
  The wait external signal is one of many inputs that are processed by the robot controller. When the robot encounters a wait external condition, the robot will cease motion and the servo power will be removed.

Z

- ZEROING
  This procedure provides the robot controller with encoder data that is referenced from a known mechanical position (zeroing witness marks, in simple zeroing, or inclinometer values in precision zeroing) and then establishes an encoder value for this known position. Two methods can be used, simple and precision zeroing. Some robot manufacturers call this procedure mastering or calibration.

KAWASAKI ROBOT CONTROLLER C Series

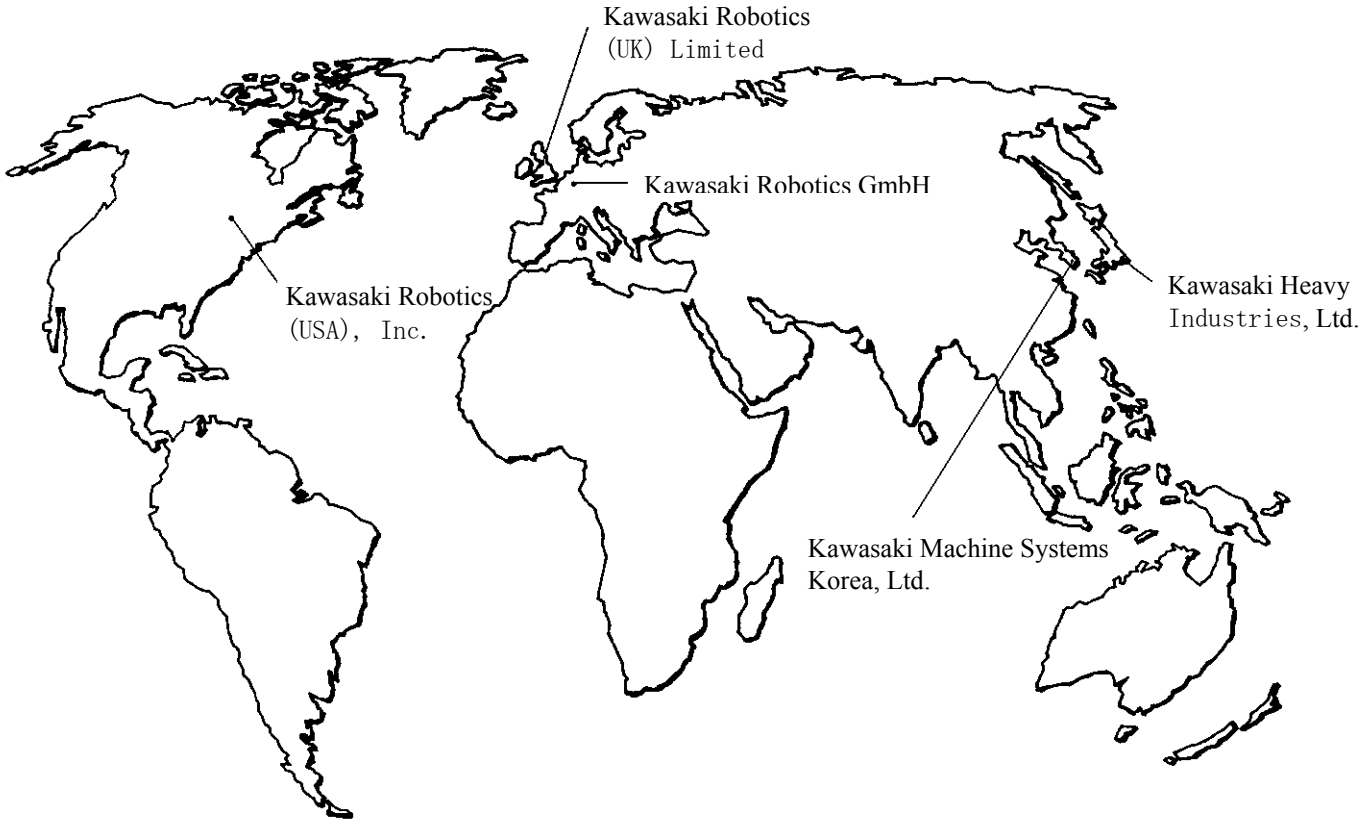
AS LANGUAGE PROGRAMMING MANUAL

October 1998 : 1st Edition
October 2001 : 3rd Edition

Publication : KAWASAKI HEAVY INDUSTRIES, LTD.
ROBOT DIVISION

90209-1006DEC

Kawasaki Robot



KAWASAKI HEAVY INDUSTRIES, LTD.

Kawasaki Heavy Industries, Ltd.	4-1, Hamamatsu-cho 2-Chome, Minato-ku Tokyo 105-6116 JAPAN	TEL +81-3-3435-6908 FAX +81-3-3437-9880
Kawasaki Robotics (USA), Inc.	28059 Center Oaks Court Wixom, Michigan, 48393 U.S.A.	TEL +1-248-305-7610 FAX +1-248-305-7618
Kawasaki Robotics (UK) Limited	Greengate, Middleton Manchester, M24 1SA UNITED KINGDOM	TEL +44-1925-71-3000 FAX +44-1925-71-3001
Kawasaki Robotics GmbH	Sperberweg29, 41468 Neuss, GERMANY	TEL +49-2131-3426-0 FAX +49-2131-342622
Kawasaki Machine Systems Korea, Ltd.	3FI(307), Industrial Complex Support Bldg., 637 Kojan-Dong, Namdong-Gu, Inchon, 405-310 Korea	TEL +82-32-821-6941 FAX +82-32-821-6947
Akashi Works	1-1 Kawasaki-cho, Akashi 673-8666 JAPAN	TEL +81-78-921-2946 FAX +81-78-923-6548