



**C SERIES CONTROLLER  
AS LANGUAGE REFERENCE MANUAL  
MPPCCONTO11E-2**

This publication contains proprietary information of Kawasaki Robotics (USA), Inc. and is furnished solely for customer use only. No other uses are authorized or permitted without the express written permission of Kawasaki Robotics (USA), Inc. The contents of this manual cannot be reproduced, nor transmitted by any means, e.g., mechanical, electrical, photocopy, facsimile, or electronic data media, without the express written permission of Kawasaki Robotics (USA), Inc.

All Rights Reserved.

Copyright © 2001, Kawasaki Robotics (USA), Inc.  
Wixom, Michigan 48393

The descriptions and specifications in this manual were in effect when it was submitted for publishing. Kawasaki Robotics (USA), Inc. reserves the right to change or discontinue specific robot models and associated hardware and software, designs, descriptions, specifications, or performance parameters at any time and without notice, without incurring any obligation whatsoever.

This manual presents information specific to the robot model listed on the title page of this document. Before performing maintenance, operation, or programming procedures, all personnel are recommended to attend an approved Kawasaki Robotics (USA), Inc. training course.

### **KAWASAKI ROBOTICS (USA), INC. TRAINING**

Training courses covering operation, programming, electrical maintenance, and mechanical maintenance are available from Kawasaki Robotics (USA), Inc. These courses are conducted at our training facility in Wixom, Michigan, or on-site at the customer's location.

For additional information contact:

Kawasaki Robotics (USA), Inc.  
Training and Documentation Dept.  
28059 Center Oaks Court  
Wixom, Michigan 48393

**REVISION HISTORY**

<b>Revision Number</b>	<b>Release Date</b>	<b>Description of Change</b>	<b>Initials</b>
-0	6/7/99	Initial PDF release	BF
-1	9/22/00	Revision 1, based on revision 1 of print copy	CB
-2	1/15/01	Revision 2, based on revision 2 of print copy	CB

---

## INTRODUCTION

<b>I.0 INTRODUCTION .....</b>	<b>I-2</b>
<b>I.1 Robot Controller Design Specifications .....</b>	<b>I-3</b>

---

## INTRODUCTION

### I.0 INTRODUCTION

The *AS Language Reference 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 programs, running programs, and editing programs using AS Language commands. AS Language is relatively easy to learn with many keywords, syntax sequences, and interface commands being intuitive.

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 provides the ability to integrate peripheral components into the program. Typical component interfacing with AS Language programs includes: programmable logic controllers (PLCs), lasers, weld controllers, gray scale vision, and remote sensing systems.

AS Language programs 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 and moved based on parameters and variables identified in the AS Language program.

---

## INTRODUCTION

### I.1 ROBOT CONTROLLER DESIGN SPECIFICATIONS

<b>Control System:</b>	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 using pulse width modulation (PWM) circuitry
<b>Number of Axes:</b>	6 standard; 7th optional
<b>Motion Control:</b>	Teach mode - Joint Base Tool  Repeat mode - Joint move Linear move Circular move (optional) FLIN move (optional)
<b>Memory:</b>	CMOS RAM
<b>Memory Capacity:</b>	Standard - 1024 KB (approximately 4,000 steps) Optional - 4096 KB (approximately 34,000 steps)
<b>Accuracy:</b>	Adjustable in increments of 0.0001 mm within the ranges below:  F-series Adjustable between 0.1 mm - 5,000 mm  UX/UT-series Adjustable between 0.5 mm - 5,000 mm  UZ-series Adjustable between 0.3 mm - 5,000 mm  Z-series Adjustable between 0.3 mm - 5,000 mm

---

## INTRODUCTION

<b>Speed:</b>	Proportional speed - percentage of maximum joint or TCP speed. Adjustable in increments of 0.0001 up to 100% (rounding occurs as necessary).	
	Absolute speed - speed of TCP in mm/s. Adjustable in increments of 0.0001 mm/s up to maximum robot TCP speed (rounding occurs as necessary).	
<b>Data Editing:</b>	Step insertion and deletion, and rewriting of auxiliary and positional data.	
<b>Software Features:</b>	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	
<b>I/O Signals:</b>	1GW I/O board	32 inputs/32 outputs (256 maximum) (including dedicated signals)
	1FS RI/O board	(optional)
	Robot I/O	256 I/O (including dedicated signals)
	Robot internal	256
	Relay circuit	32 I/O
	A-B PLC	64 I/O
	Weld controller	32 I/O
	Non-retentive	128 I/O
	Retentive	16 I/O
	Timers	16 I/O
	Counters	16 I/O
	Message display	64 I/O
	Slogic status	16 I/O
	Control Net (option)	

---

## INTRODUCTION

<b>Dedicated Signals:</b>	Outputs -	Motor power ON Error occurrence Automatic CYCLE_START Teach mode HOME1 HOME2 Power ON RGSO Ext. program select (RPS) enabled
<b>Dedicated Signals:</b>	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
<b>Error Messages:</b>		Error code messages, self-diagnosis, error logging, operation logging
<b>Special Features:</b>		Program check mode Adjustable restriction of JT1 Terminal box on robot arm (optional) Robot application interface panel (optional) Overtravel limit switch - JT1 (JT2, JT3 optional) Power lockout Ethernet (optional)



---

## INTRODUCTION

<b>Multi Function Panel: (optional)</b>	Deadman safety switches 7.2 inch color LCD Touch panel Teach-lock function Emergency stop switch Pen for touch panel PC card insertion section
<b>Teach Pendant: (optional)</b>	Deadman safety switches Teach-lock function Emergency stop switch Membrane switch keypad Alphanumeric LCD
<b>Supplemental Data Storage:</b>	PC flash RAM memory card 8 MB, PCMCIA 2.1 slot Floppy disk drive (optional) Personal computer (optional)
<b>Power Requirements:</b>	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  Tolerance: +/- 10%  Frequency: 50/60 Hz  Rated Load: 10.5 kVA  Ground: less than 100 ohm ground line separated from welder power ground
<b>Dimensions:</b>	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.: W x D x H, 550mm x 500mm x 1150mm (inches) (21.7 x 19.7 x 45.3)  European Spec.: W x D x H, 550mm x 500mm x 1150mm (inches) (21.7 x 19.7 x 45.3)
<b>Weight:</b>	Standard Spec.: approx. 80 kg (176 lbs)  North Am. Spec.: 250 kg (550 lbs)  European Spec.: 250 kg (550 lbs)

---

## SYSTEM OVERVIEW

<b>1.0</b>	<b>SYSTEM OVERVIEW</b>	1-2
1.1	AS System Status	1-2
1.2	Notations and Conventions	1-6
1.3	Displaying with the Terminal	1-7
1.4	Location Information	1-8
1.4.1	Precision Location	1-8
1.4.2	Transformation Location	1-8
1.4.3	Compound Transformation Location (Relative Transformation)	1-10
1.5	Numeric Information	1-12
1.5.1	Integers	1-12
1.5.2	Real Numbers	1-12
1.5.3	Logical Values	1-12
1.5.4	ASCII Values	1-13
1.6	Variable Names	1-13
1.6.1	Location Variables	1-14
1.6.2	Real Variables	1-14
1.6.3	Character String Variables	1-15
1.7	Numerical Expression	1-17
1.7.1	Operators	1-17
1.8	Monitor Commands	1-20
1.8.1	Program Instructions	1-21

---

## SYSTEM OVERVIEW

### 1.0 SYSTEM OVERVIEW

AS Language for the C controller is a software based control system and high-level language used to interface with the robot controller and control robot motion. The AS software is permanently stored in the robot controller's memory and is activated as soon as controller power is turned on. It continuously generates robot control commands and can simultaneously interact with a programmer, permitting on-line program generation and modification. The multi function panel and/or a personal computer is used to access AS Language.

### 1.1 AS SYSTEM STATUS

The AS system consists of the monitor mode, the editor mode, and the playback mode.

- **Monitor Mode:** This is the basic mode in the AS system. Monitor commands are executed in this mode. The editor or playback modes are accessed from this mode.
- **Editor Mode:** This mode enables the user to create a new program or modify an existing program. Only editor commands are accepted by the system in this mode.
- **Playback Mode:** The system is in the playback mode during program execution. Computations for robot motion control are performed and commands entered from the terminal are processed during unoccupied CPU time. Some monitor commands cannot be executed in playback mode. Refer to unit 4, Monitor and Editor Commands for details.

The AS system is controlled by the following system switches:

- **CHECK.HOLD**

This switch is used with the AS Language commands EXECUTE, DO, STEP, MSTEP and CONTINUE. When the CHECK.HOLD switch is ON these commands are available only if the HOLD/RUN switch is in the HOLD position. The controller accepts these commands with the HOLD/RUN switch in the HOLD position but robot motion is not initiated until the switch is manually placed in the RUN position. Default setting is OFF.

## SYSTEM OVERVIEW

- CP

The CP switch is used to enable or disable the continuous path function. When the switch is ON, the robot makes smooth transitions between motion segments within the accuracy ranges set. When the switch is OFF, the robot decelerates and stops at the end of each motion segment regardless of accuracy. Default setting is ON

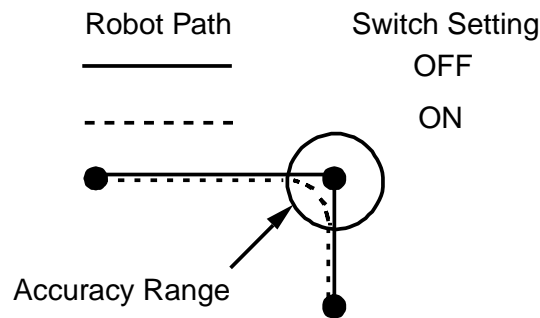


Figure 1-1 CP Switch

- CYCLE.STOP

This switch is used in conjunction with an external input signal to stop the motion of the robot. With the switch ON, when the input signal is received the robot stops and the cycle start light turns OFF. When the program is started again it starts at the beginning. If the program is called from another program, the program restarts at the beginning of the main program. With the switch OFF, when the input signal is received the robot stops and the cycle start light remains ON. The robot is in a hold condition and when the program is started again, it continues at the point in the cycle where it was stopped. Default setting is OFF.

- OX.PREOUT

This switch affects the timing of output signal generation in block step programs. When the switch is ON, an output programmed for a given point is turned ON when the robot begins motion to the point. With the OX.PREOUT switch OFF, an output programmed for a given point is not turned ON until the robot reaches the accuracy range of the point. Figure 1-2 shows the effects the OX.PREOUT switch on signal timing. Default setting is ON.

## SYSTEM OVERVIEW

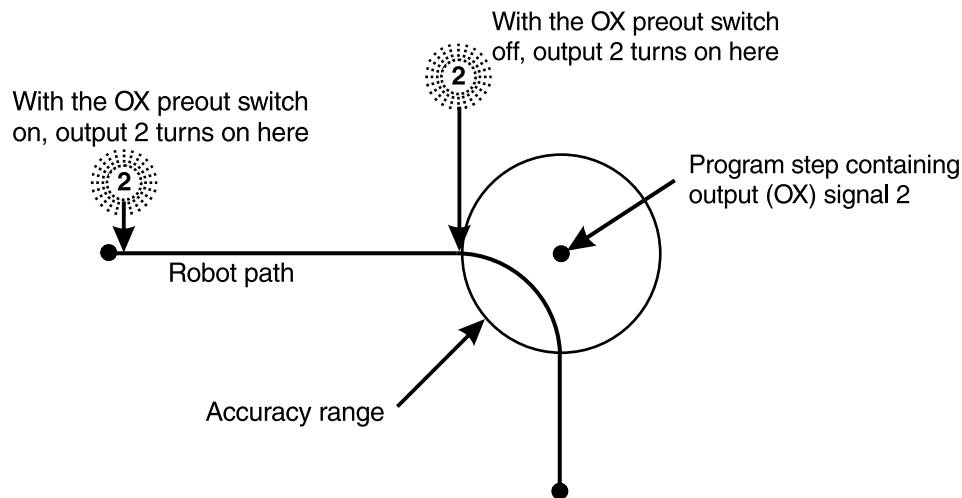


Figure 1-2 OX.PREOUT Switch

- PREFETCH.SIGINS

This switch is used in conjunction with AS Language instructions and has the same effect on signal timing as the OX.PREOUT switch has with blockstep instructions. Default setting ON. The AS Language instructions affected are; SWAIT, TWAIT, SIGNAL, PULSE, DLYSIG, RUNMASK, RESET and BITS.

- QTOOL

This switch allows the user to identify tools to use in block step or AS Language programming. When the QTOOL switch is ON, nine tools are available for programming and jogging. The tool dimensions are recorded and assigned a tool number using auxiliary function 48. When the QTOOL switch is ON, the selected tool dimensions are in effect for jogging and linear playback of block step programs. When the QTOOL switch is OFF, the tool identified with AS Language instructions is used. Default setting is ON.

- REP\_ONCE (Repeat Once)

When this switch is ON, programs run one time. With the switch OFF, the program runs continuously. Default setting is OFF.

---

## SYSTEM OVERVIEW

- STP\_ONCE (Step Once)

When this switch is ON, the repeat condition function of progressing through a program one step at a time is active. The step forward key is used to step through a program. When the switch is OFF, programs run continuously.  
Default setting is OFF.

- AFTER.WAIT.TIMER

When this switch is ON, timers begin timing for a specified step when all wait conditions are satisfied. With the switch OFF, timers begin timing when the robot reaches coincidence of the taught point.  
Default setting is OFF.

- AUTOSTART.PC

The AUTOSTART.PC, AUTOSTART2.PC, and AUTOSTART3.PC switches automatically start the associated PC program when controller power is turned on.  
Default setting is OFF.

- ERRSTART.PC

When this switch is ON and specified errors (assigned dedicated signals) occur, a PC program is run as soon as the error is detected.  
Default setting is OFF.

- MESSAGES

Enables or disables message output (PRINT or TYPE) to the keyboard screen.  
Default setting is ON

- RPS (Random Program Selection)

This switch enables or disables the random selection of programs based on binary status of dedicated inputs.  
Default setting is OFF

- SCREEN

This switch enables or disables scrolling of the screen when information is too large to fit on one screen.  
Default setting is ON

---

## SYSTEM OVERVIEW

- DISPIO\_01

This switch allows the user to select the type of display for viewing the status of inputs and outputs. If the switch is ON, 1s and 0s are displayed to identify the signal state of individual signals. A 1 represents an ON signal, while a 0 represents an OFF signal. If the switch is off, an ON signal is represented by an O, while an X represents a OFF signal. Dedicated signals are represented by uppercase Xs and Os.

Default setting is OFF.

### 1.2 NOTATIONS AND CONVENTIONS

A mixture of uppercase and lowercase words is used throughout this manual, all key words are shown in uppercase and all elements freely specified by the user are shown in lowercase.

Abbreviated notations are used as well. For example, the EXECUTE command can be abbreviated as EX.

At least one space (blank) or tab is necessary as a delimiter between the instruction (or command) name and its arguments. The excess spaces or tabs are ignored by the system.

Monitor commands and program instructions are processed by pressing the ENTER key.

Many instructions or commands have arguments which can be omitted. If there is a comma following the optional argument, the comma should be retained even if the argument is omitted. If all successive arguments are omitted, commas may also be omitted.

In this manual, values are expressed in decimal notations, unless noted otherwise. Some instructions and commands require several types of arguments. Mathematical expressions can be used to designate the value as arguments. The acceptable value may be restricted. The following rules show how the values are interpreted in various cases.

- The AS Language follows the conventions established by the American Standard Code for Information Interchange (ASCII). An ASCII character is specified by prefixing a character with an apostrophe ('). For example, ddd = 'A assigns 65 to ddd.
- All numerical expressions evaluated by the system result in a real value.

## SYSTEM OVERVIEW

- DISTANCE is used to define the position to which the robot moves. The unit for distance is millimeters, although units are not required to be entered with values. Values entered for distances can be positive or negative, with their magnitudes limited by a number representing the maximum reach of the robot. For example,  
  
> DO DRAW 50,100,-50 moves the robot 50 mm in X, 100 mm in Y, and 50 mm in the Z Cartesian direction.
- ANGLES in degrees are entered to define and modify orientations the robot assumes at named locations, and to describe angular positions of robot joints. The values can be positive or negative, with their magnitudes limited by 180 degrees or 360 degrees depending on the context. For example,  
  
> DO DRIVE 2,45,75 moves joint 2 of the robot 45° at 75% of the repeat speed.
- JOINT NUMBER is an integer value from one to the number of joints available on the robot, including a servo-controlled external axes.
- SIGNAL NUMBER is used to identify binary (on/off) signals. The value is an integer in the range of 1-256 (output signals), 1001-1256 (input signals) depending on the number of I/O signals available in the controller. Negative signal numbers indicate an OFF state.
- Whenever an existing program is saved, or renamed, the new name is entered first, followed by the old name. The above also holds true for the POINT command. For example:

<u>Command</u>	<u>New Name</u>	=	<u>Old Name</u>
SAVE	Right_Side	=	Fender3
RENAME	test	=	test.tmp

### 1.3 DISPLAYING WITH THE TERMINAL

The operator can display various types of information in the monitor mode or playback mode. Directories and listings of programs, locations, variable data, and weld conditions are displayed by entering specific monitor commands. For additional information, refer to section 4.2, Program and Data Control Commands.



---

## SYSTEM OVERVIEW

### 1.4 LOCATION INFORMATION

Locations recorded in the controller's memory are comprised of values which designate destinations for robot motion. The values recorded in memory are either Cartesian coordinates or robot joint angles. A Cartesian coordinate represents a point in the robot workspace with a tool center point orientation at that point. A location recorded with joint angles specifies a robot arm configuration at that point. When the robot is directed to move to a Cartesian location, two actions occur simultaneously: the robot is moved so the tool center point moves to the specified point, and the tool is rotated to the prescribed orientation. When the robot is directed to move to a location recorded with joint angles, the processor calculates a motion path based on the encoder values of the recorded point, then moves the arm until all encoder values match those of the recorded point. There are two types of location information, transformation locations (Cartesian coordinates) and precision locations (joint angles).

#### 1.4.1 PRECISION LOCATION

A precision location's value is represented by the exact position of the individual robot joints in degrees. There are several characteristics of precision locations that should be considered. These characteristics result from joint angles being recorded.

Advantages of precision locations: Playback precision is achieved and there is no ambiguity about robot configuration at a location.

Disadvantages of precision locations: The values recorded can be used by any model of robot, however the tool center point location is different when used by a robot of different physical size. Precision locations cannot be easily modified to compensate for location changes in the robot workspace, because a change requires complete knowledge of the relationship between the positions of all robot joints and the locations in the robot workspace.

#### 1.4.2 TRANSFORMATION LOCATION

A transformation location is represented by defining the location in terms of a Cartesian (XYZ) reference frame fixed to the base of the robot. The position of the tool center point is defined with X, Y, and Z coordinates, and the tool orientation is defined by three angles measured from the coordinate axes.

---

## SYSTEM OVERVIEW

Advantages of transformation locations: A value defined for use with one robot can be used with a different robot having a similar work envelope because the value is defined in terms of workspace coordinates. Transformations are easily modified to change a location within the robot workspace. A powerful feature of transformation locations is the ability to define locations as combinations of values. This is called compound or relative transformation. Such values are used to define the location of a part relative to its fixturing.

Disadvantages of transformation locations: Since a transformation location defines the location of the tool center point in terms of coordinates in the workspace, no information is provided about the specific robot configuration at the location. Whenever a transformation is used to define the destination of a robot motion, the AS system must convert the transformation location into an equivalent precision location so it knows how to move the individual joints. This conversion can introduce small location errors. Despite these disadvantages, transformation locations are generally much more convenient than precision locations.

## SYSTEM OVERVIEW

### 1.4.3 COMPOUND TRANSFORMATION LOCATION (RELATIVE TRANSFORMATION)

Compound transformations are defined by a combination of transformation locations, used to create a location or locations that are relative to the first transformation value, in the compound transformation (Figure 1-3).

#### **transformation\_value+transformation\_value....**

The last component of the compound transformation value, defines the actual location.

If the transformations are subtracted an inverse value results.

#### **transformation - transformation**

This is useful when several locations are defined relative to a reference location.

To change the location points defined relative to a reference location, only the transformation location of the reference must be updated. All locations defined relative to the reference point are automatically changed to reflect the change.

Unlike usual addition or subtraction, the commutative law does not hold true for the transformation operation. The compound expression "loc.a + loc.b" does not necessarily equal "loc.b + loc.a" because the turning angles O,A,T are taken into consideration. An example of this is shown below.

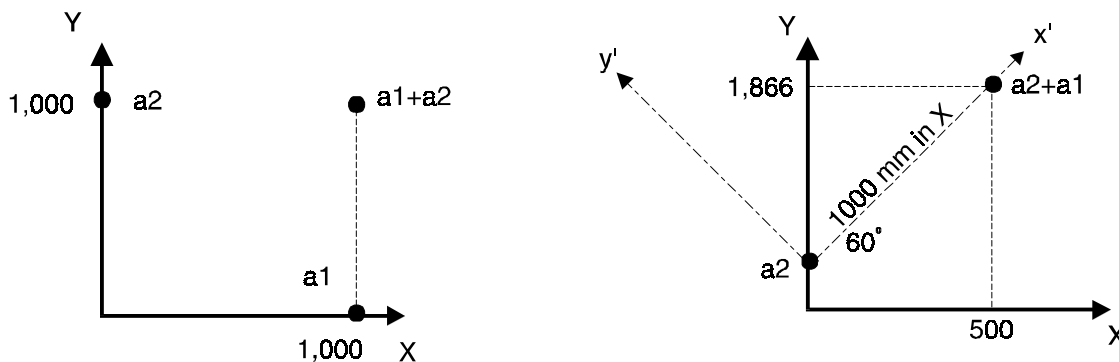
Assuming:

$$a1 = (1000, 0, 0, 0, 0, 0)$$

$$a2 = (0, 1000, 0, 60, 0, 0)$$

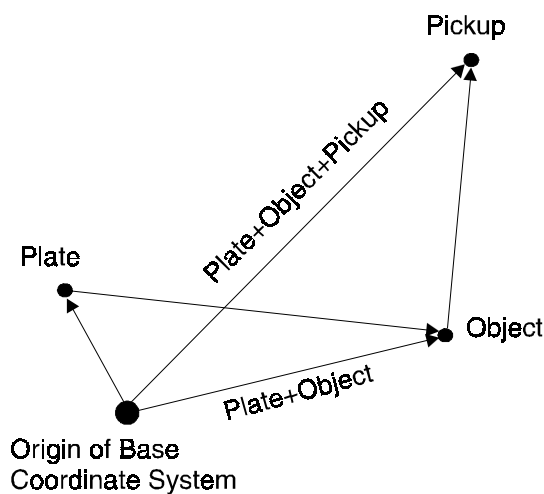
$$a1 + a2 = (1000, 1000, 0, 60, 0, 0)$$

$$a2 + a1 = (500, 1866, 0, 60, 0, 0)$$



## SYSTEM OVERVIEW

For example, “Plate” is the name of the transformation location representing the location of a base plate relative to the origin of the base coordinate system of the robot. “Object” is the relative transformation for the location of an object relative to the location of the plate. The compound transformation “Plate+Object” defines the location of the object relative to the origin of the base coordinate system of the robot. If the transformation location “Pickup” represents the final location relative to “Plate+Object”, the compound transformation “Plate+Object+Pickup” defines the location of pickup relative to the origin of the base coordinate system.



For example, to define a compound transformation move the robot to location **Plate** and enter the command: **HERE Plate**

> **HERE Plate**

Move the robot to location **Object** and enter the command: **HERE Plate + Object**

> **HERE Plate + Object**

Move the robot to location **Pickup** and enter the command: **HERE Plate + Object + Pickup**

> **HERE Plate + Object + Pickup**

Figure 1-3 Compound Transformation

As indicated in the example above, the compound transformation is defined by a combination of several transformation values separated by “+”.

## SYSTEM OVERVIEW

### 1.5 NUMERIC INFORMATION

Numeric information is a combination of numerals, variables, operators, and functions which return numeric values. Numeric expressions are used not only for mathematical calculations, but also as arguments for monitor commands or program instructions. Numeric values used in the AS system are divided into the four types described below:

#### 1.5.1 INTEGERS

Integers are values without fractional parts (whole numbers). Values with full precision ranges are from -16,777,216 to +16,777,216. Values that exceed this range are rounded to seven significant digits. Integer values are usually entered as decimal numbers, however, it may be more convenient to enter them in binary or hexadecimal notations.

#### 1.5.2 REAL NUMBERS

Real numbers have both the integer part and a fractional part which can range from  $-3.4E +38 \sim 3.4E +38$ . Like integers, real values are positive, zero or negative. They can be represented in scientific notation. Real values are stored with an accuracy of approximately seven digits, but actual values may have less precision caused by a calculation error.

#### 1.5.3 LOGICAL VALUES

Logical values have only two states, ON or OFF. These two states are also referred to as TRUE and FALSE respectively. A value of negative one (-1) is assigned for the TRUE or ON state and a value of zero (0) is assigned for the FALSE or OFF state.

#### NOTE

ON, OFF, TRUE, and FALSE are AS Language keywords.

- |              |                                     |
|--------------|-------------------------------------|
| > AA = ON    | Stores a value of -1 in variable AA |
| > BB = FALSE | Stores a value of 0 in variable BB  |
| > CC = -TRUE | Stores a value of 1 in variable CC  |

## SYSTEM OVERVIEW

### 1.5.4 ASCII VALUES

An ASCII value is the numeric value of one ASCII character. An ASCII value is specified by prefixing the character with an apostrophe (').

> X = 'A	Stores a value of 65 in variable X
> X = 'a	Stores a value of 97 in variable X

### 1.6 VARIABLE NAMES

Variable names must start with an alphabetic character and can contain only letters, numbers, periods, and underlines. The letters used in variable names can be entered either in lowercase or uppercase. The length of a name is limited to fifteen characters. AS Language commands should not be used and in some cases cannot be used as variable names because they cause ambiguity with the AS system keywords, but their abbreviations can be used. For example, the following names cannot be used:

3P	(first character is not alphabetic)
part#2	("#" prefix for precision location name)
random	(AS keyword)

Precision location names must be preceded by the symbol “#” to differentiate them from transformation location names. String variables must be preceded by the symbol “\$” to differentiate them from real and transformation variables.

pick	(transformation or real variable value)
#pick	(precision value)
\$count	(string variable)

A transformation location and precision location may have the same name, however, the same name may not be used for transformation values and real values. A defined variable may be used by any program in the system.

Array variables can be used for any type of information. Arrays consist of several values under the same name and these values are distinguished from each other by their index value. In order to designate array elements, attach an element number (index) enclosed by brackets to the array name. For example, “part[7]” indicates the seventh element of the array “part”. Indexes should be integers within the range 0 to 9999.

Location, real, string, and array are four types of variables within the AS system. These four variable types are explained on the following pages.

## SYSTEM OVERVIEW

### 1.6.1 LOCATION VARIABLES

A location variable (precision or transformation) is automatically defined when a value is assigned for the first time. Prior to this, the location name is undefined. If a program uses an undefined variable, an error occurs. The user defines location variables by using monitor commands or program instructions. The following are examples of location variable values:

Precision location value: name with joint angles

	JT1	JT2	JT3	JT4	JT5	JT6
#weld	90.000°	145.056°	-95.098°	90.000	°45.000°	0.000°

Transformation location value: name with joint angles and turning angles

	X	Y	Z	O	A	T
weld	60.000 mm	145.050 mm	-95.098 mm	90.000°	45.000°	0.000°

### 1.6.2 REAL VARIABLES

Real variables are defined using the assignment instruction (=). The format for assigning a real variable is:

```
Variable_name = numeric_value  
a = 6  
b = 7  
c = a + b
```

The variable on the left side may be either a scalar variable (i.e., “count”) or an array element (i.e., “x [2]”). A variable is defined automatically the first time it is assigned a value. If a program uses an undefined variable, an error occurs. The numeric value on the right side may be a constant, a variable, or a numeric expression. When an assignment instruction is processed, the value on the right side of the assignment instruction is first computed, then the value is assigned to the variable on the left side.

For example, the assigned value “x=3” assigns the value 3 to the variable “x”. If a variable on the left side has never been used it is defined automatically, and if it has already been assigned a value, its current value is replaced by a new assigned value. The above example is read as “assign 3 to x” and not “x is equal to 3”. The following example shows this difference: x = x + 1.

## SYSTEM OVERVIEW

If the example is a general equation, it is read as “x is equal to x plus 1”, which does not make sense mathematically. It must be read as “assign the value of x plus 1 to x”. In this case, the sum of the current value of “x” and 1 is calculated. In the next step, that value is assigned to “x” as a new value. Therefore, the result of the above assignment instruction is to increase the value of x by 1. In this example, the variable “x” should have been previously defined.

```
x = 3
x = x + 1
```

In the case above, the resulting value of “x” is 4.

### 1.6.3 CHARACTER STRING VARIABLES

The character information referred to in the AS system is indicated as a string of ASCII characters enclosed by quotation marks (“”). Since the quotation marks indicate the beginning and end of a character string, they cannot be included in the string. ASCII control characters (CTRL, CR, LF, etc.) also cannot be included in the string. For example, a command for printing (displaying on the screen) would be entered as:

```
PRINT “Kawasaki”
```

Character strings are defined by using the assignment instruction (=). The format for assigning a character variable is :

```
$string_variable    =    string_value
(name of variable)  (string expression)
```

The string variable on the left side may be either a scalar variable (ie., “\$name”) or an array element (ie., “\$line [2]”). A variable is defined automatically the first time it is assigned a value. If a program uses an undefined variable, an error occurs. The character string on the right side may be a constant, a string variable, or a string expression. When an assignment instruction is processed, the value on the right side is first computed, then the value is assigned to the variable on the left side. If the variable on the left side has never been used, it is defined automatically, and if it has already been used, its current value is replaced by a new assigned value.



## SYSTEM OVERVIEW

The following is an example of string variable assignment:

```
$First = "Kawasaki"  
$Last = " Robotics"  
$Name = $First + $Last + " Inc."
```

In the above example, the string variable \$Name is assigned the sum of \$First, \$Last, and the character string "Inc.". The command PRINT or TYPE \$Name returns the string value: Kawasaki Robotics Inc.

### 1.6.4 Arrays

An array is a group of values that share a single name. Location variables can be scalars or arrays. A location scalar is a single location value. 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. For example, "part[7]" refers to element 7 of the array "part." Indexes must be integers in the range of 0 ~ 9999. Three examples of arrays are described below:

Example 1:

PROGRAM	OUTPUT
HERE edge	edge[1]=120.456
DECOMPOSE edge[1]=edge	edge[2]=145.670
FOR i=1 to 6	edge[3]=-95.432
TYPE "edge["11,i,"]=",/D,edge[i]	edge[4]=90.456
END	edge[5]=45.000
edge[6]=10.018	

In the above example, the current location of the robot is defined as "edge". The DECOMPOSE instruction extracts component values of edge (XYZOAT) consecutively (1 through 6). The program instructions between the FOR and END statements are executed repeatedly and the TYPE instruction displays the component values of edge individually.

## SYSTEM OVERVIEW

Example 2:

```
FOR i = 2 to 6 STEP 2
DRAW 100, 10 * i + 7, 50
HERE weld[i]
END
```

In the above example, the robot moves 100 mm in the X direction, a calculated amount (10 \* i + 7) mm in the Y direction, and 50 mm in the Z direction, and define the location as weld[i].

The FOR statement, in this example increments the value of “i” in increments of two, for example:

i = 2, i = 4, i = 6.

Example 3:

PROGRAM	SUBPROGRAM	OUTPUT
Main()	Pg10()	Corner 1
\$POINT[1]="Corner1"	FOR i=1to3	edge
\$POINT[2]="edge"	JMOVE weld[i]	Corner2
\$POINT[3]="Corner2"	TYPE\$POINT[i]	
CALL pg10	END	

In the above example, the array is used as a string array. Each move of the robot displays the strings assigned in the main program.

## 1.7 NUMERICAL EXPRESSION

The numerical expression is a combination of numeric values and variables combined together with operators. The expressions are completed by the addition of functional modifiers to the numeric values and variables. All numerical expressions evaluated by the system result in a real value. The interpretation of the value depends on the context in which the expression appears. For example, an expression specified for an array index is interpreted as yielding an integer value.

### 1.7.1 OPERATORS

For describing expressions, arithmetic, logical, and binary, operators are provided. All of these operators combine two values to obtain a single resulting value, except three: the two operators (NOT and COM) operate on a single value and the operator (-) operates on one or two values. The operators are described on the following page.

---

**SYSTEM OVERVIEW**

Arithmetic Operators:	+	addition
	-	subtraction or negation
	*	multiplication
	/	division
	^	power (if $a^b$ , $a < 0$ results in error) $x = (-2)^2$ results in an error $x = -2^2$ assigns -4 to x
	MOD	remainder $x = 5 \text{ MOD } 2$ assigns the remainder of 5/2 (1 in this case) to x
	Relational Operator:	<
	<=, (= <)	less than or equal to
	==	equal to
	<>	not equal to
	>=, (= >)	greater than or equal to
	>	greater than
Logical Operator:	AND	logical AND
	NOT	logical complement
	OR	logical OR
	XOR	exclusive logical OR

The logical operators are used in Boolean operations such as logical OR ( $0+1=1$ ,  $1+1=1$ ,  $0+0=0$ ), logical AND ( $0x1=0$ ,  $1x1=1$ ,  $0x0=0$ ), and logical XOR ( $0+1=1$ ,  $1+1=0$ ,  $0+0=0$ ). The logical operators are not used for calculating numeric values, but for determining the logical state (TRUE or FALSE) of the conditional expression. If a numeric value is zero (0), it is considered to be FALSE (0). All nonzero values are considered to be TRUE(-1).

OPERATION	RESULT
0 AND 0	0 (FALSE)
1 AND 1	-1 (TRUE)
1 OR 0	-1 (TRUE)

---

## SYSTEM OVERVIEW

Binary Operator:	BAND	Binary AND
	BOR	Binary OR
	BXOR	Binary XOR
	COM	Binary Complement

The binary logical operators perform logical operations for each respective bit of two numeric values.

OPERATION	RESULT
5 BOR 3	7
0101 BOR 0011	0111
5 BAND 9	1
0101 BAND 1001	0001

Expressions are evaluated according to a sequence of priorities. Parentheses can be used to group the components of an expression and to control the order in which the operations are performed. When expressions containing parentheses are evaluated, the expression within the innermost pair is evaluated first, then the system works toward the outermost pair. Within parentheses, expressions are evaluated in the following order:

1. Evaluate functions and arrays.
2. Process power operator “ ^ ”.
3. Process unary operators “ - ” (single component).
4. Process multiplication “ \* ” and division “ / ” operators from left to right.
5. Calculate remainders (MOD operators) from left to right.
6. Process addition “ + ” and subtraction “ - ” operators from left to right.
7. Process relational operators from left to right.
8. Process COM operators from left to right.
9. Process BAND operators from left to right.
10. Process BOR operators from left to right.
11. Process BXOR operators from left to right.
12. Process NOT operators from left to right.
13. Process AND operators from left to right.
14. Process OR operators from left to right.
15. Process XOR operators from left to right.

The logical expressions result in a logical value TRUE or FALSE. A logical expression can be used as a condition in which the execution of a program or program steps is performed. When evaluating logical expressions, the value zero is considered FALSE and all nonzero values are considered TRUE. Therefore, all real values or real value expressions can be used as a logical value.

---

## SYSTEM OVERVIEW

For example, the following two statements have the same meaning, but the second statement is easier to understand.

IF x GOTO 10        (If the value of x is true, goto label 10 in the program)

IF x <> 0 GOTO 10 (If the value of x is not equal to 0, goto label 10 in the program)

### 1.8 MONITOR COMMANDS

**TEACH        location variable name**

The TEACH command is used in conjunction with the small teach pendant. With the small teach pendant connected, the TEACH command is entered at the monitor prompt.

The small teach pendant is used to jog the robot to the locations used in the specified program. When the RECORD key on the teach pendant is pressed, the location is placed into the system memory with the specified location variable name followed by a 0. Each time the RECORD key is pressed the number following the location variable name is increased by one.

For example, the command TEACH loc is entered at the monitor prompt, the first time the RECORD key on the small teach pendant is pressed, a location with the name loc0 is stored in the system memory. The next time the RECORD key of the small teach pendant is pressed, the location stored in system memory is loc1.

The TEACH command allows the programmer to record transformation locations without having to exit the work cell for each new location.

The HERE command stores the current robot location in the specified precision or transformation variable.

HERE #pallet

The POINT command defines the named location variable using an existing location variable. Component values may also be entered from the keyboard.

POINT a = b

Assigns component values of location variable b into location variable a.

POINT #a

Displays the current component of location variable #a. If the location variable is not defined, zeros are displayed.

---

## SYSTEM OVERVIEW

### 1.8.1 PROGRAM INSTRUCTIONS

The commands HERE and POINT may also be used in a program as program instructions. For example,

```
JMOVE loc1  
POINT loc2=loc1  
DRAW 347.28, ,479.0  
HERE loc3
```

In the above example, the robot moves to loc1. The POINT command then assigns component values of loc1 to loc2. The DRAW command moves the robot 347.28 mm in X, and 479 mm in the Z direction. Finally, loc3 is defined by the HERE command.

---

**SAFETY**

<b>2.0</b>	<b>SAFETY</b> .....	2-2
2.1	Introduction .....	2-2
2.2	Safety Conventions and Symbology .....	2-3
2.2.1	Warning/Caution Symbols .....	2-3
2.3	Safety Categories .....	2-4
2.3.1	Personal Safety .....	2-4
2.3.2	Safety During Operation .....	2-6
2.3.3	Safety During Programming .....	2-7
2.3.4	Safety During Inspection and Maintenance .....	2-8
2.4	Safety Features .....	2-9
2.5	Work Envelope Drawings .....	2-10
2.5.1	FS02N/FS03N .....	2-10
2.5.2	FS06L .....	2-11
2.5.3	FC06N/FS06N/FW06N/FS10C .....	2-12
2.5.4	FS10N .....	2-13
2.5.5	FS10E .....	2-14
2.5.6	FS10L .....	2-15
2.5.7	FS20C .....	2-16
2.5.8	FS20N .....	2-17
2.5.9	FS30L .....	2-18
2.5.10	FS30N/FS45C .....	2-19
2.5.11	FS45N .....	2-20
2.5.12	UB150 .....	2-21
2.5.13	UT100/150/200 .....	2-22
2.5.14	UX70 .....	2-23
2.5.15	UX100/120/150 .....	2-24
2.5.16	UX200 .....	2-25
2.5.17	UX300 .....	2-26
2.5.18	UZ100/120/150 .....	2-27
2.5.19	ZD130 .....	2-28
2.5.20	ZX130L .....	2-29
2.5.21	ZX130U .....	2-30
2.5.22	ZX165U .....	2-31
2.5.23	ZX200S .....	2-32
2.5.24	ZX200U .....	2-33
2.5.25	ZX300S .....	2-34

---

## 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 equipment, peripheral and robotic equipment that occupies 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 a Kawasaki approved training course that would be pertinent to each employee's specific job responsibilities.

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 federal governments, but are NOT designed to supplant or supersede any existing rules, regulations, or guidelines that may be in use. 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.

Two recommended sources for national and federal safety laws and regulations are:

1. OCCUPATIONAL SAFETY AND HEALTH STANDARDS, available from:  
U.S. Department of Labor  
Occupational Safety & Health Administration  
Office of Public Affairs - Room N3647  
200 Constitution Avenue  
Washington, DC 20210  
  
<http://www.osha-slc.gov/SLTC/robotics/index.html>
2. AMERICAN NATIONAL STANDARD FOR INDUSTRIAL ROBOTS AND ROBOT SYSTEMS-SAFETY REQUIREMENTS (ANSI/RIA R15.06-1992), available from:  
American National Standards Institute  
11 West 42nd Street  
New York, NY 10036  
  
<http://www.ansi.org/>

All safety related issues and descriptions, either presented in written or oral form from any representative of Kawasaki Robotics (USA), Inc., are intended to provide general safety precautions and procedures and, therefore, are not intended to provide all safety measures necessary for the protection of all personnel in the work environment.



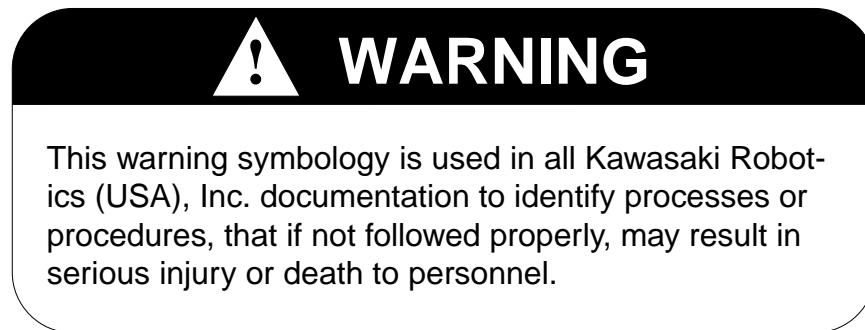
## SAFETY

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

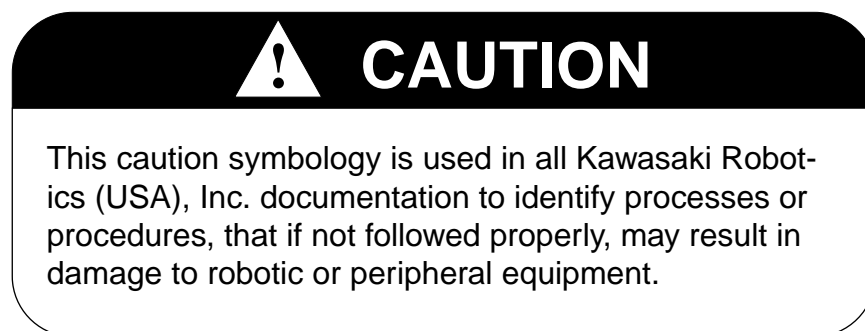
### 2.2 SAFETY CONVENTIONS AND SYMBOLOGY

#### 2.2.1 WARNING/CAUTION SYMBOLS

The following symbol is present in all Kawasaki Robotics (USA), Inc. documentation to signify to the user that proper guidelines, as set forth in the text, are designed to provide pertinent information for the protection of personnel:



The following symbol is present in all Kawasaki Robotics (USA), Inc. documentation to signify to the user that proper guidelines as set forth, are designed to provide pertinent information for the protection of robotic related equipment:



---

## SAFETY

### 2.3 SAFETY CATEGORIES

Personnel safety can be described in one of four categories:

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

A description of each follows in this section.

#### 2.3.1 PERSONAL SAFETY

Safety procedures must be an integral part of operational procedures for the operator, programmer, and maintenance person. These procedures must be followed explicitly and on a regular basis. Safety procedures are followed on a daily basis, they should become a regular part of everyday operational procedures, which are designed to protect the user. Some guidelines are presented in brief in the following section:

- Before operating or maintaining the robot or robot controller, be sure you fully understand and comprehend ALL maintenance, operating, and programming procedures, and ensure that ALL safety related precautions are taken and complied with before these procedures are attempted.
- AVOID wearing loose clothing, scarves, wrist watches, rings, and jewelry when working on the controller and robot. It is also recommended that if ties must be worn in your shop environment that they be the clip-on variety rather than tied ties.
- ALWAYS wear safety glasses or goggles and approved safety shoes for your shop conditions. Follow all applicable OSHA, NIOSHA, MSHA, local, state, federal, and plant safety specifications and procedures.
- 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.
- AVOID trap points in which personnel could become trapped between a moving device and any stationary devices.

---

## SAFETY

- Personnel should NEVER enter the work envelope during automatic operations.
- Ensure that ALL personnel are clear of the work envelope before initiating any motion commands for the robot.
- Before initiating any motion commands, KNOW beforehand how the robot will perform when that command is given.
- Be sure that the ENTIRE work area is free of any debris, tools, fixturing, lubricants, and cleaning equipment before operation of the robot is attempted.
- If any personnel observe unsafe working conditions, report them IMMEDIATELY to your supervisor or plant safety coordinator.
- ALL personnel should identify by name and function ALL switches, indicators, and control signals that could initiate robot motion.
- NEVER defeat, render useless, jumper out, or bypass any safety related device, whether mechanical or electrical in design.
- ALL safety devices approved for use in your plant must be properly installed and maintained to ensure personnel safety.
- NEVER attempt to stop or brake the robot during operation with your body or person.
- ONLY utilize E-stops to stop robot motion in emergency situations.

## SAFETY

### 2.3.2 SAFETY DURING OPERATION

- During operation of the robot, identify the maximum reach of the robot in ALL directions, which is referred to as the work envelope.
- ALWAYS keep your work area clean and free of any debris which includes, but is not limited to, oil, water, tool, fixturing, electronic test equipment, etc.
- During operations that involve the teach pendant, the ONLY person allowed in the work envelope is the teacher, or the person operating the teach pendant. The teach pendant has provisions to protect the operator. These safety provisions include an E-stop, trigger switch, and deadman switch.
- NEVER block the operator's path of retreat.
- During the teach operation of the robot ALWAYS have a path of retreat planned.
- AVOID pinch points.

---

## SAFETY

### 2.3.3 SAFETY DURING PROGRAMMING

- During operation of the robot, be sure you are able to identify the maximum reach of the robot in ALL directions, which is referred to as the work envelope.
- During teach operations the ONLY person allowed in the work envelope is the teacher, or the person operating the teach pendant. The teach pendant has provisions to protect the operator including E-stop, trigger switch, and deadman switch.
- AVOID pinch points.
- During point-to-point playback operations, be aware that the robot is ONLY cognizant 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.
- Playback accuracy and speed can affect the geometry of the path coordinates. Therefore, when changing accuracy or speed, ALWAYS test run the program at a slow speed or point-to-point mode before attempting the continuous path operation in the repeat mode.
- ALWAYS test run a new path program at a reduced speed or in point-to-point mode prior to attempting a high-speed playback operation in the repeat mode.

## SAFETY

### 2.3.4 SAFETY DURING INSPECTION AND MAINTENANCE

Before entering the work envelope to perform either inspection or maintenance procedures, turn off 3-phase power on the disconnect and tag and lockout 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 OFF the 3-phase power at the source, and tag and lockout the source disconnect.

- When removing an axis motor, be aware that the axis WILL fall if left unsupported. The brake assembly is in the servo drive motor, therefore, the axis of the robot will be unsupported if removed.
- When using the axis brake release switches in the controller, be aware that the axis MAY fall if left unsupported.
- Before working on pneumatic or high pressure water supplies, turn off supply pressure and purge ALL lines to remove any residual pressure.
- Assign ONLY qualified personnel to perform all maintenance procedures.
- Consult ALL available documentation before attempting any repair or service procedures.
- Use ONLY replacement parts approved by Kawasaki Robotics (USA), Inc.
- BEFORE attempting to adjust or repair a device in the robot controller that may have yellow interlock control circuit wires attached, locate the source of the power and remove it by disconnecting the appropriate disconnect at its source.
- During inspection and maintenance procedures, if your installation is equipped with safety fences and safety plugs, REMOVE and HOLD the safety plug while performing these operations. In addition, the safety procedures outlined above should be adhered to.

---

## SAFETY

### 2.4 SAFETY FEATURES

To safeguard the user, the Kawasaki robot system is equipped with many safety features. These safety items include:

- All E-stops are hard-wired.
- The multi function panel, small teach pendant, and operation panel are equipped with red mushroom-type detented E-stop push buttons. If an optional interface panel is installed, the E-stop from the operation panel is relocated to the optional interface panel.
- All robot axes are monitored by the robot controller for velocity and deviation errors.
- Robot velocities are constantly monitored by software. Should an over-velocity condition be detected, the robot will fault in a velocity error condition.
- Teach velocities and check mode velocities are limited to a maximum of 250 mm/sec (9.843 in/sec).
- All robot axes have software limits.
- JT1 is equipped with overtravel limit switches (JT2 and JT3 are optional).
- All F-series, U-series, and Z-series mechanical units have overtravel hardstops on the JT1, JT2, JT3, and JT5 axes.
- 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 are engaged when power is OFF at the robot controller.

**SAFETY**

**2.5 WORK ENVELOPE DRAWINGS**

**2.5.1 FS02N/FS03N**

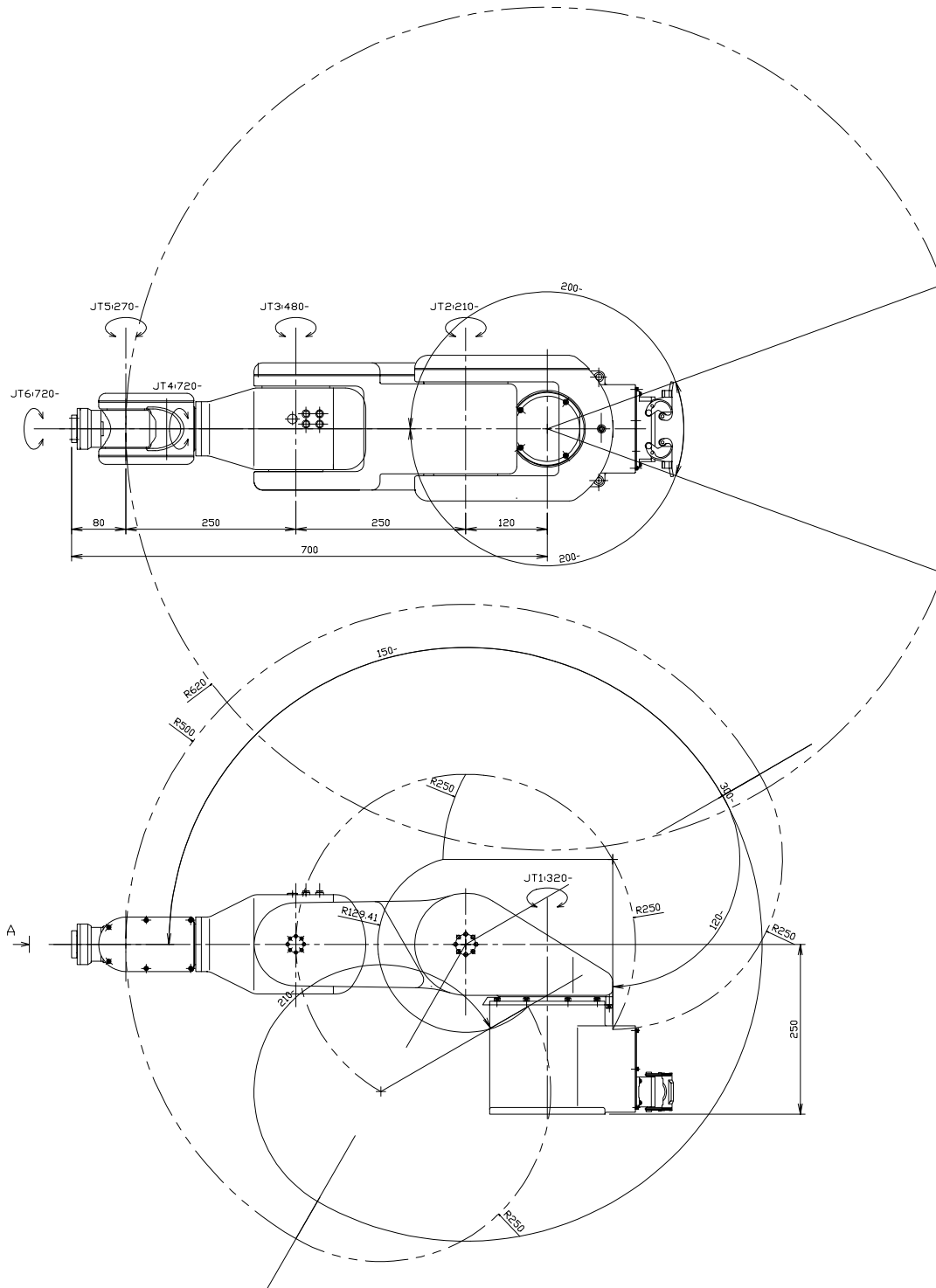


Figure 2-1 FS02N/FS03N Work Envelope



**SAFETY**

**2.5.2 FS06L**

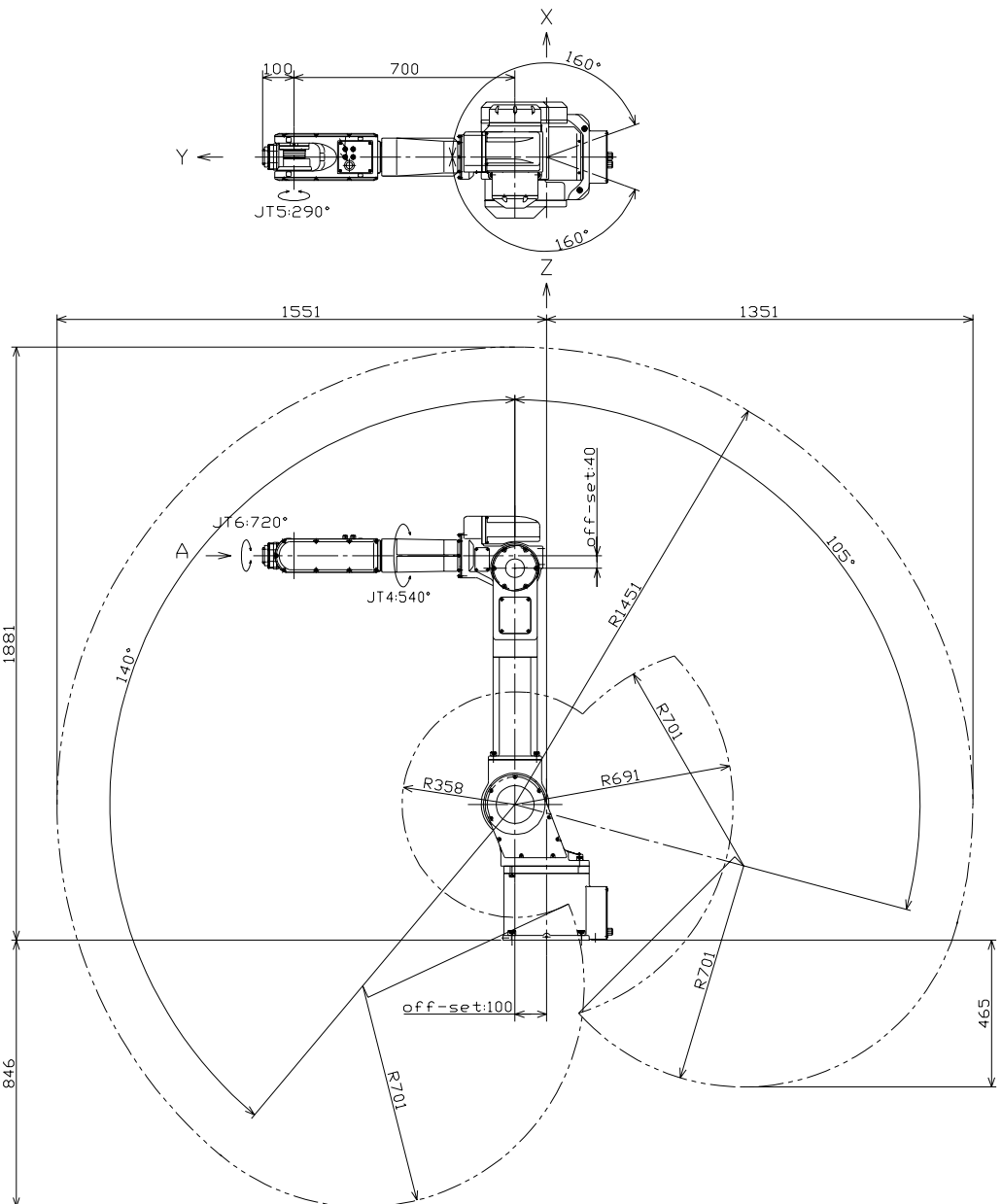
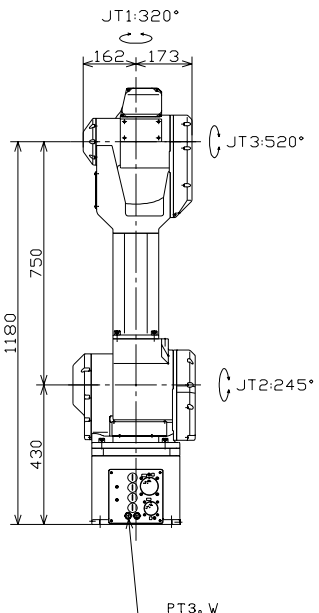


Figure 2-2 FS06L Work Envelope

**SAFETY**

**2.5.3 FC06N/FS06N/FW06N/FS10C**

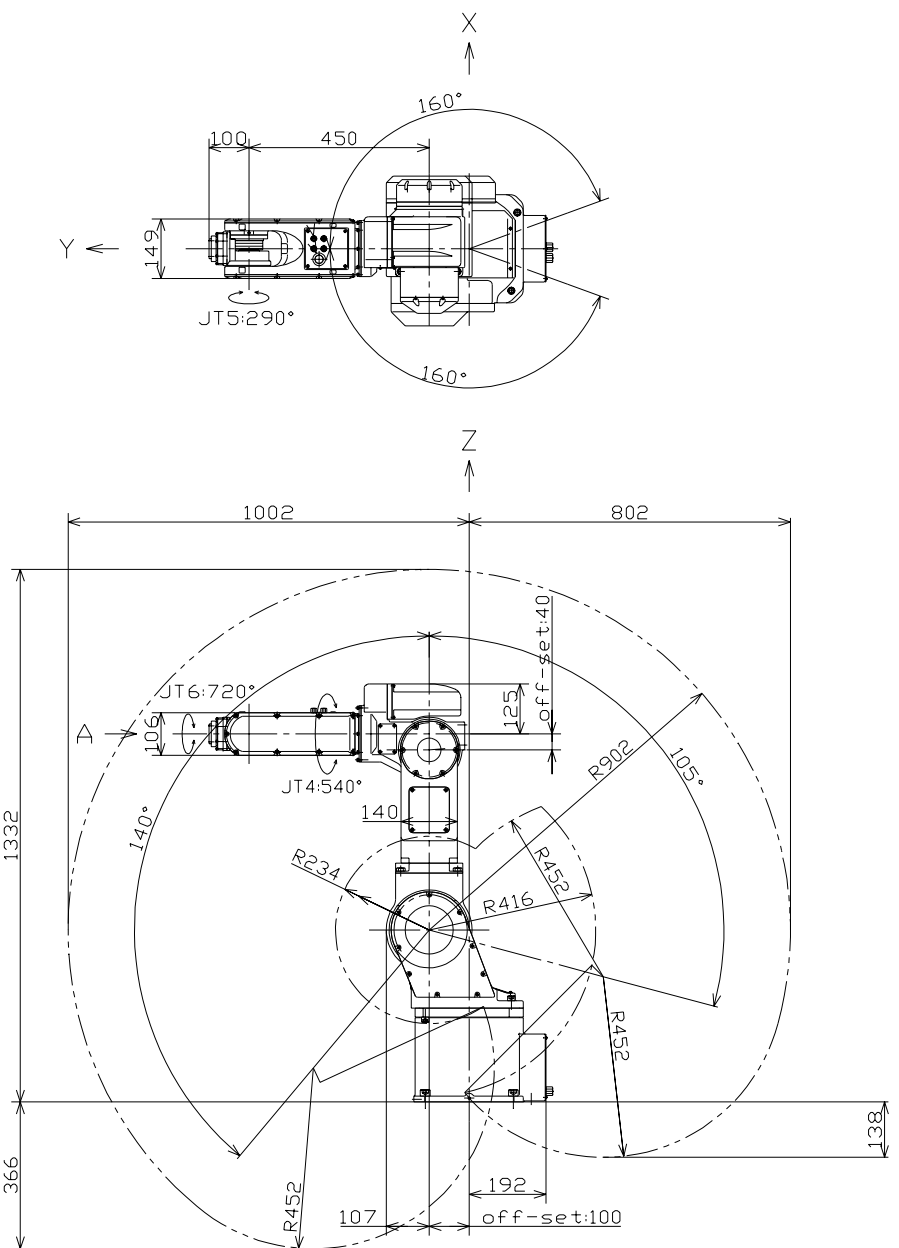
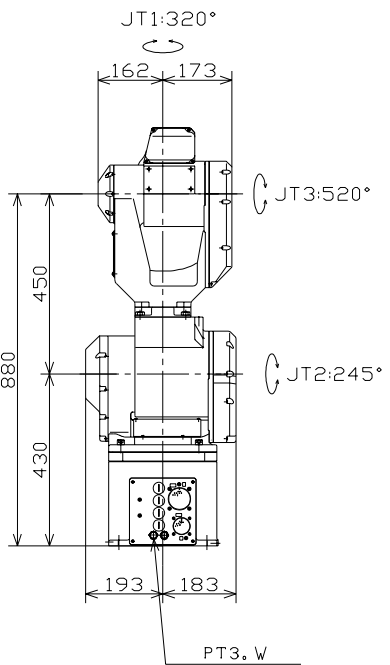


Figure 2-3 FC06N/FS06N/FW06N/FS10C Work Envelope

**SAFETY**

**2.5.4 FS10N**

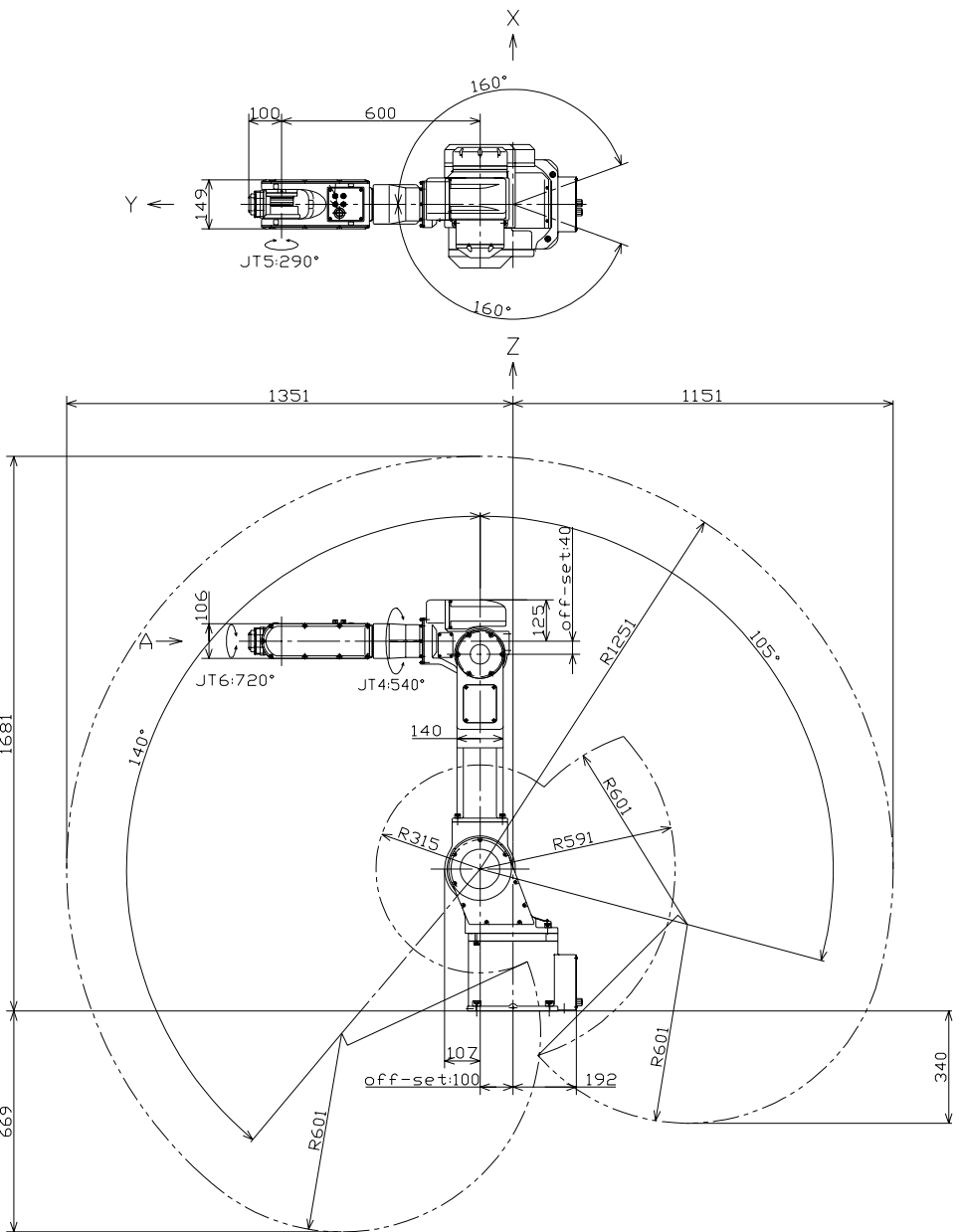
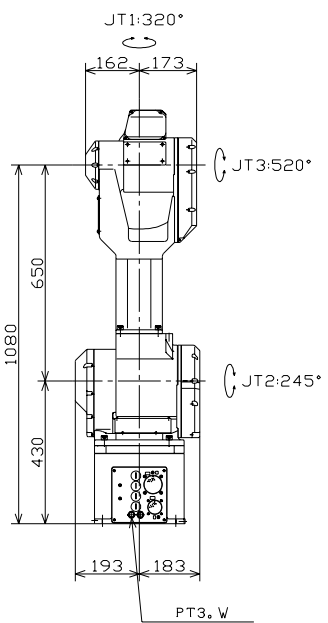


Figure 2-4 FS10N Work Envelope

**SAFETY**

**2.5.5 FS10E**

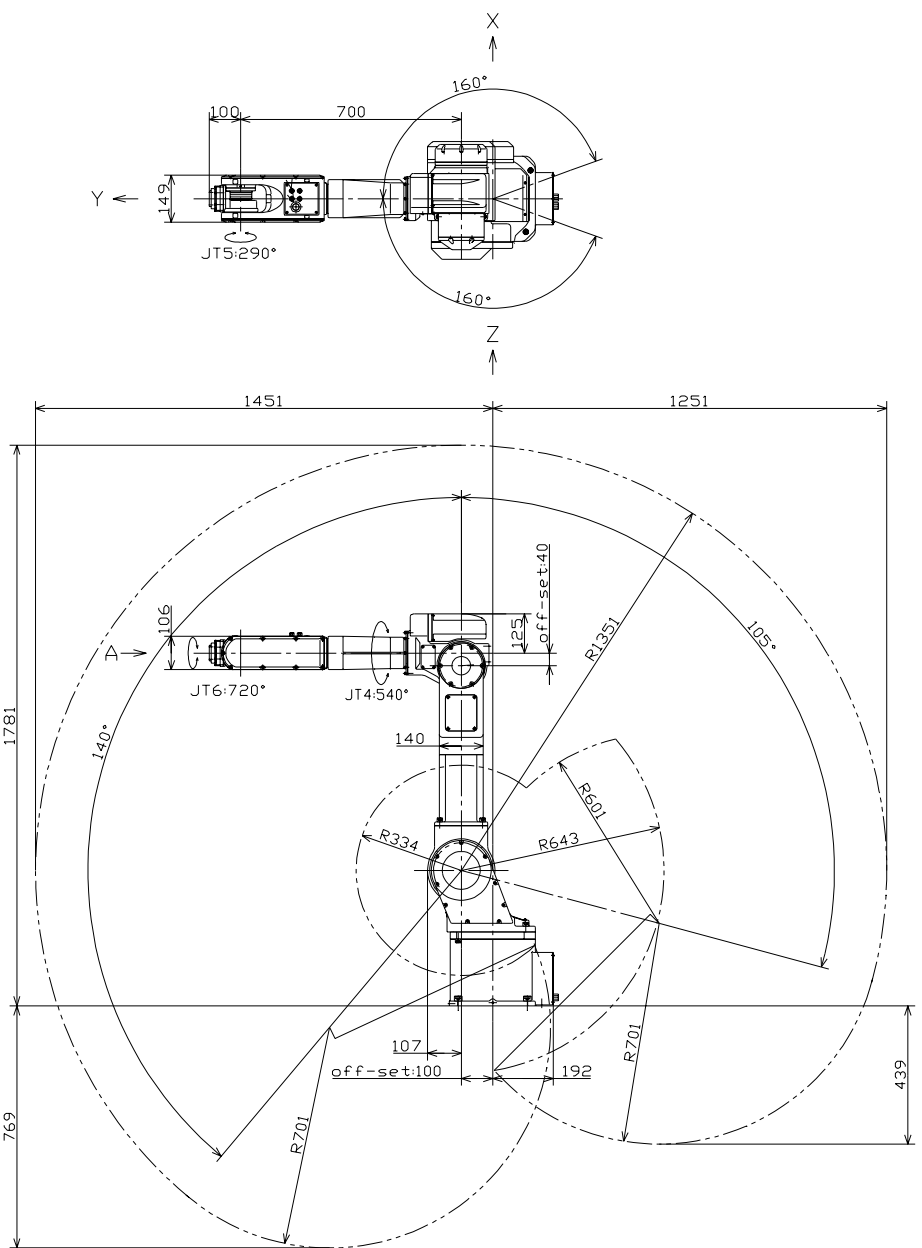
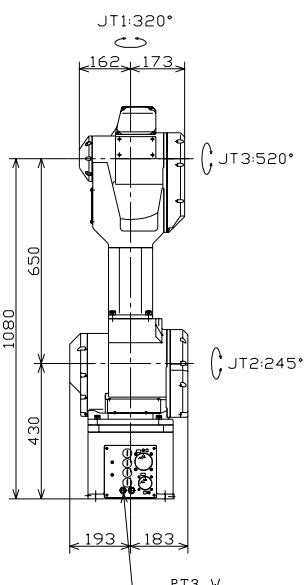
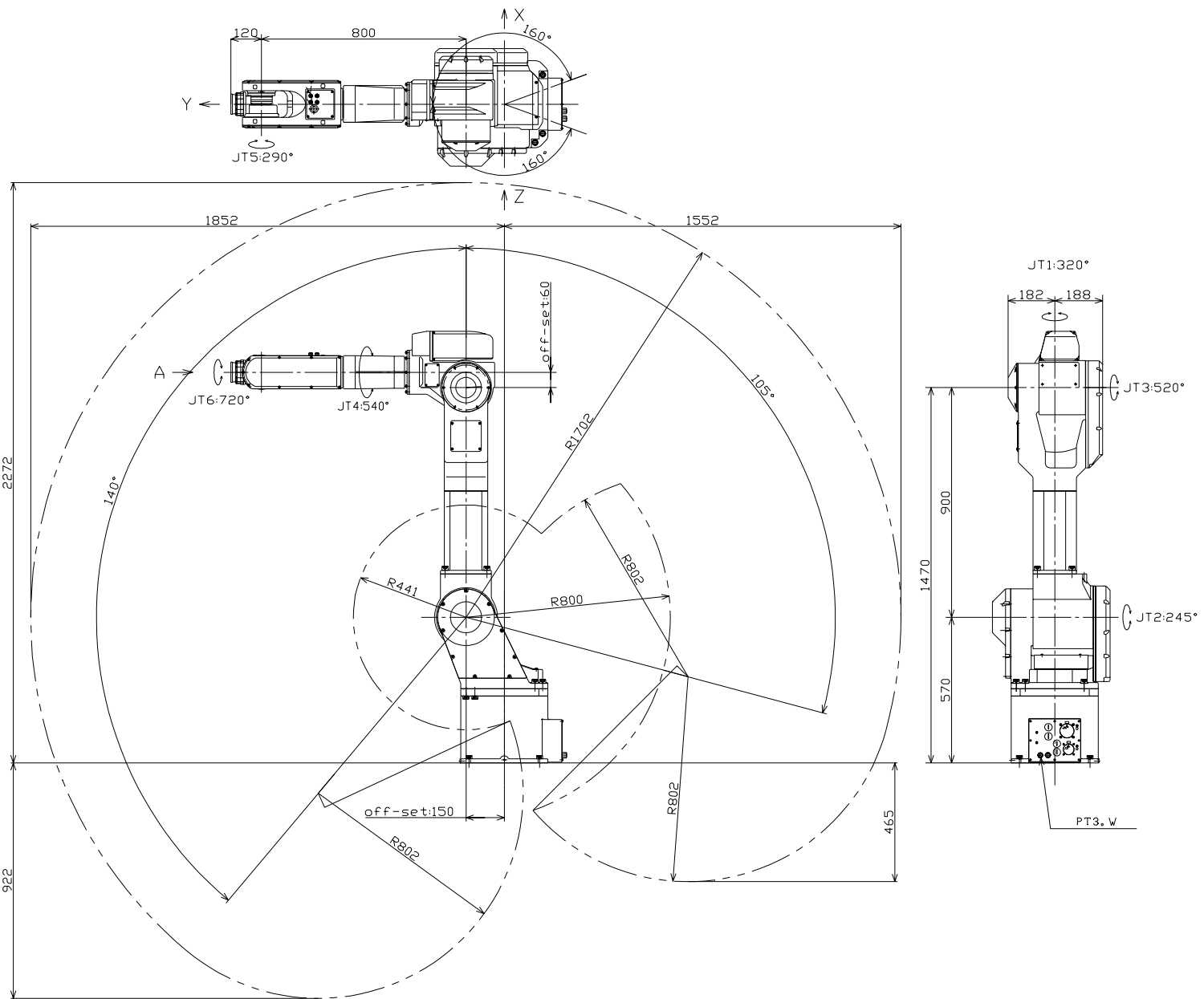


Figure 2-5 FS10E Work Envelope

**SAFETY**

**2.5.6 FS10L**



**Figure 2-6 FS10L Work Envelope**

**SAFETY**

**2.5.7 FS20C**

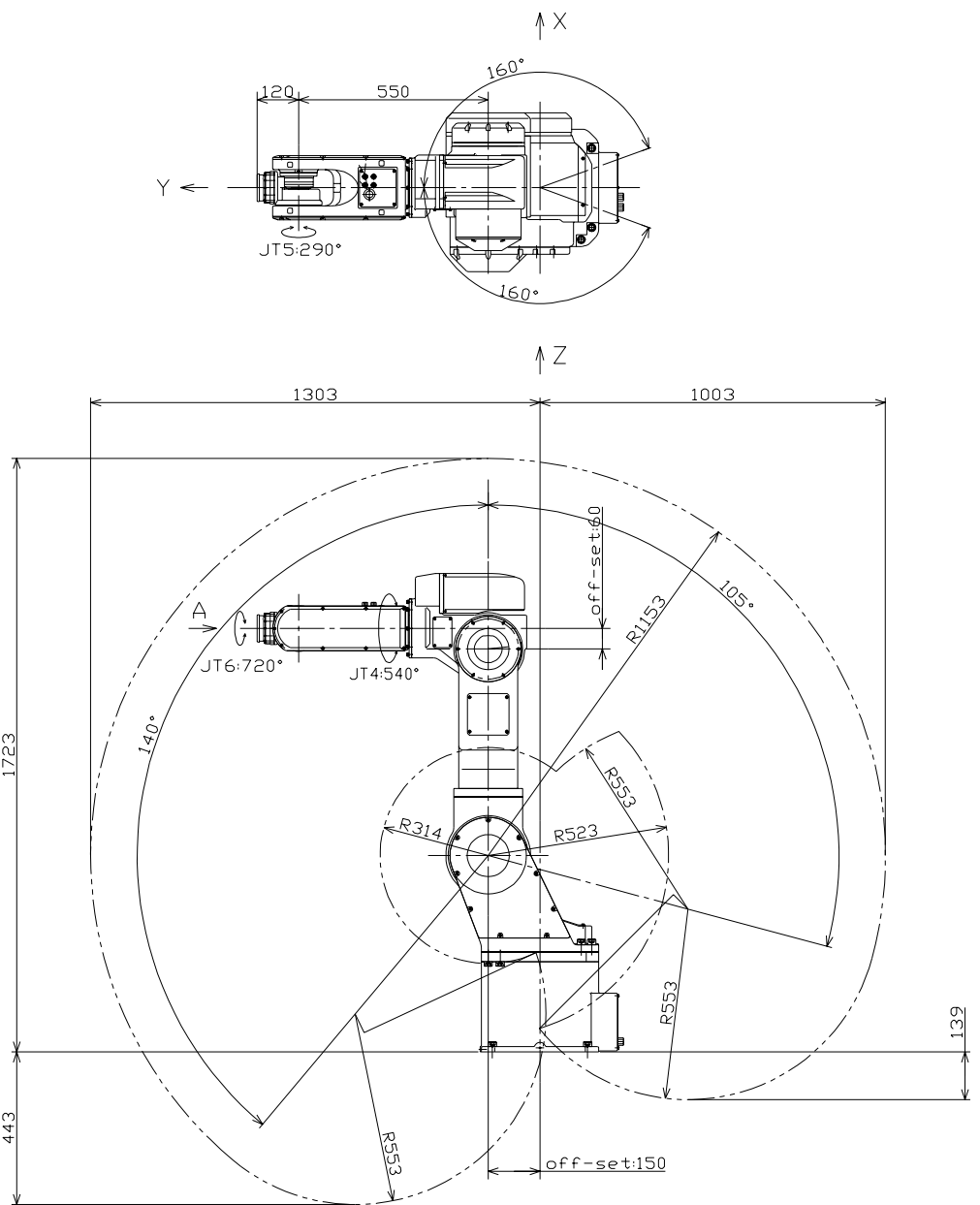
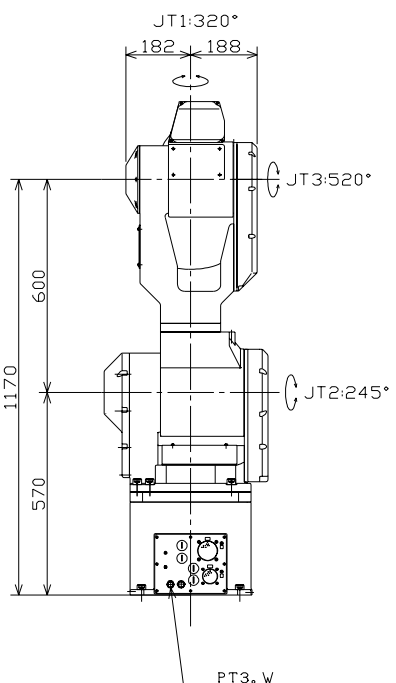


Figure 2-7 FS20C Work Envelope



**SAFETY**

**2.5.9 FS30L**

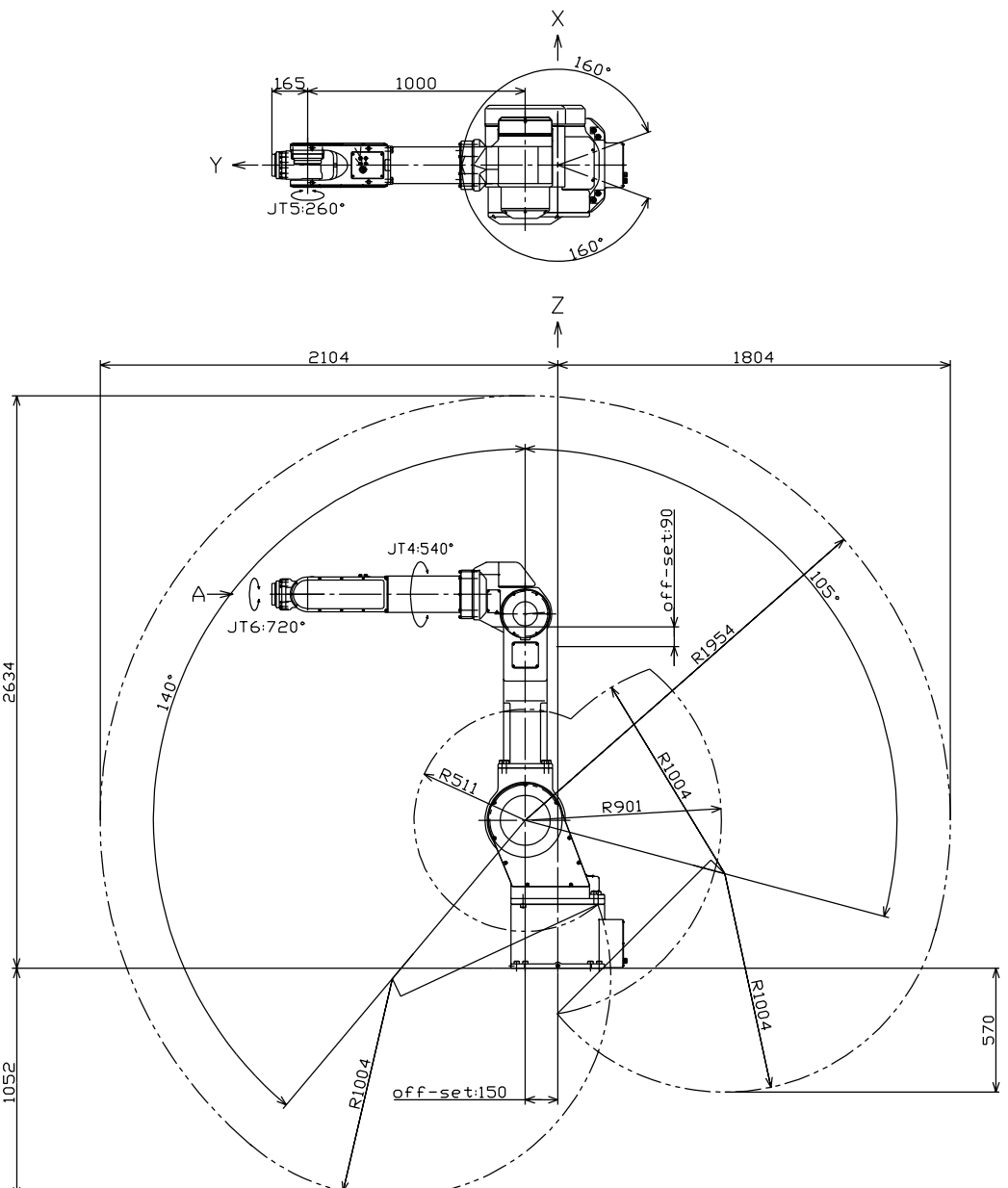
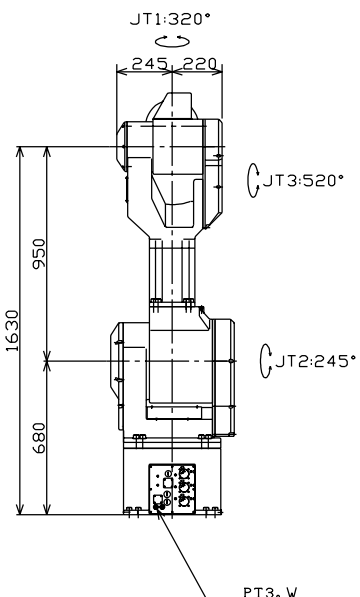


Figure 2-9 FS30L Work Envelope



**SAFETY**

**2.5.10 FS30N/FS45C**

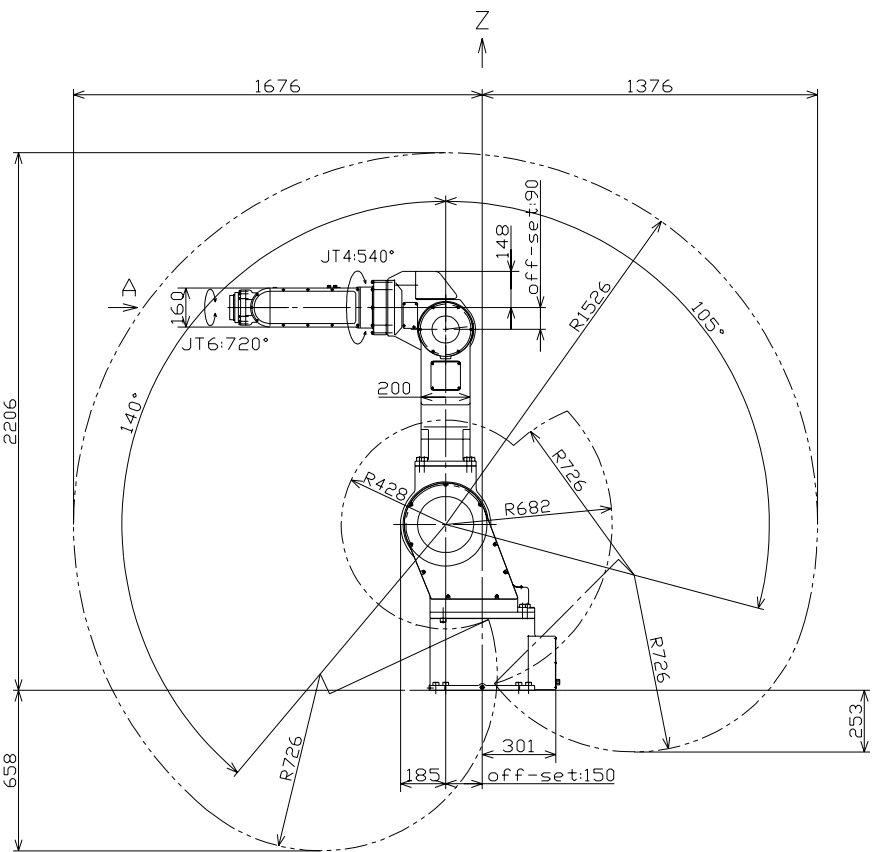
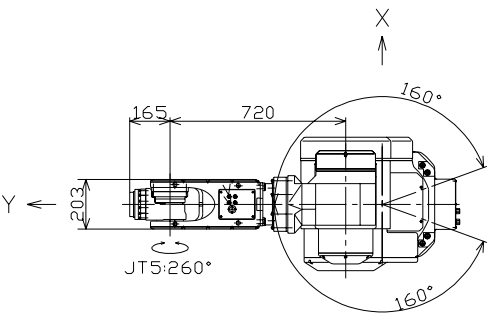
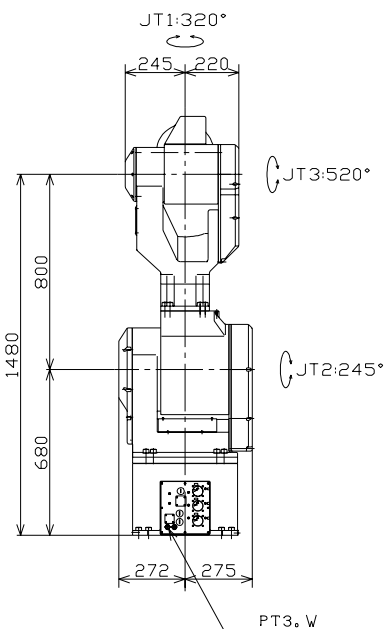


Figure 2-10 FS30N/FS45C Work Envelope

**SAFETY**

**2.5.11 FS45N**

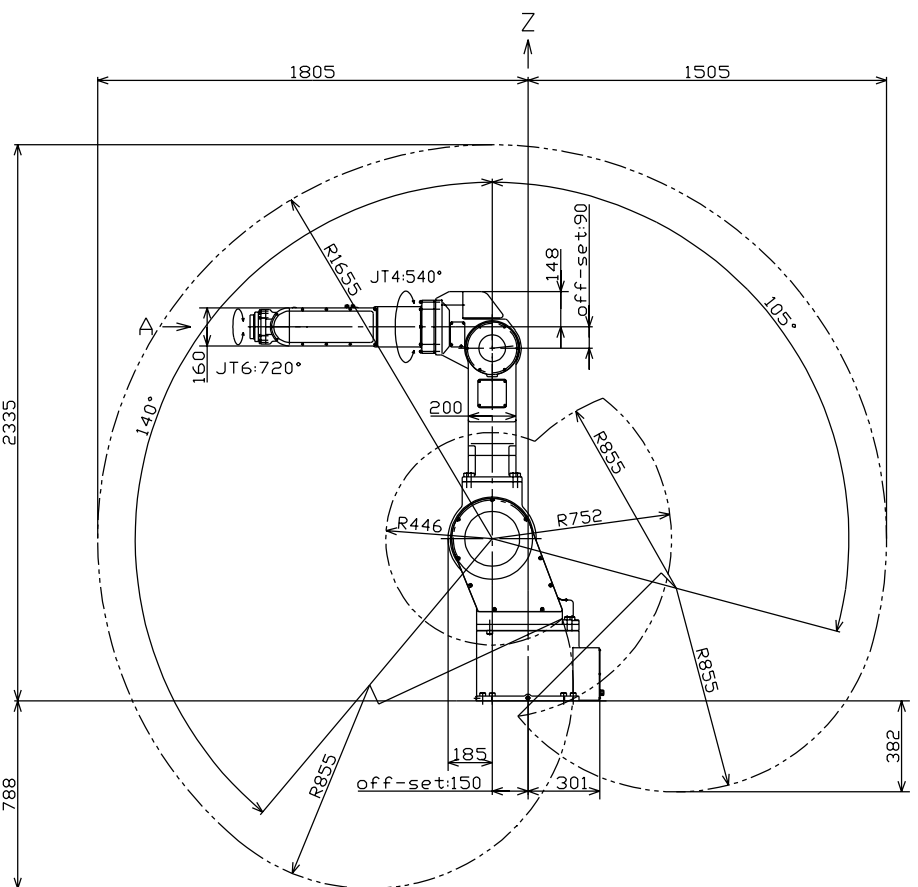
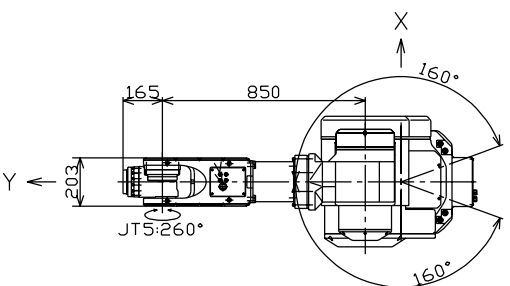
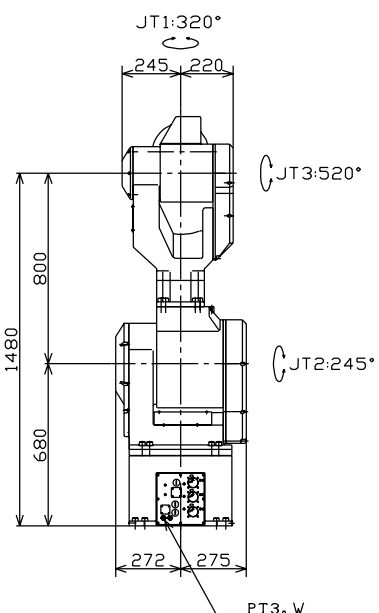


Figure 2-11 FS45N Work Envelope



**SAFETY**

**2.5.13 UT100/150/200**

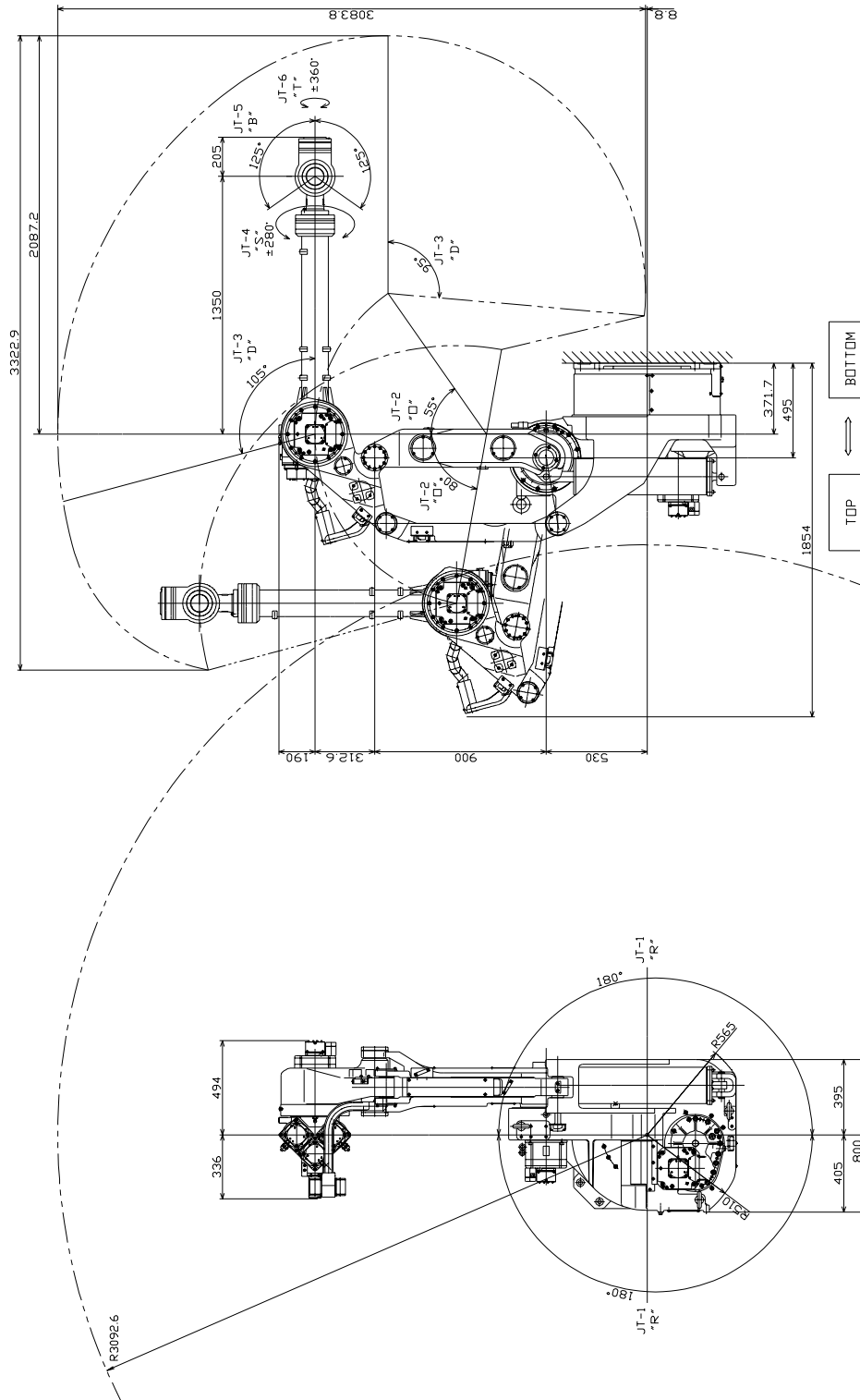


Figure 2-13 UT100/120/150 Work Envelope

**SAFETY**

**2.5.14 UX70**

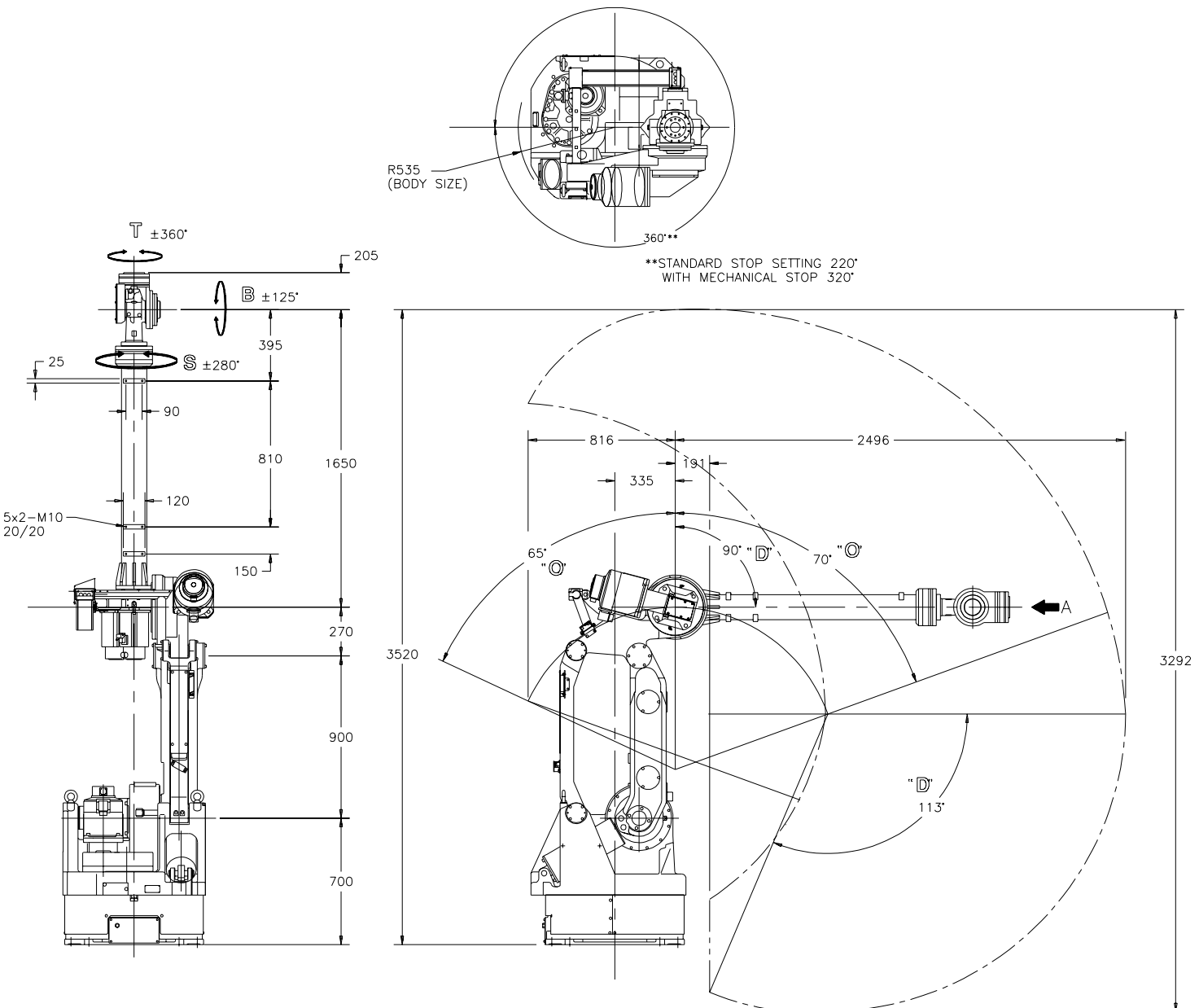


Figure 2-14 UX70 Work Envelope

**SAFETY**

**2.5.15 UX100/120/150**

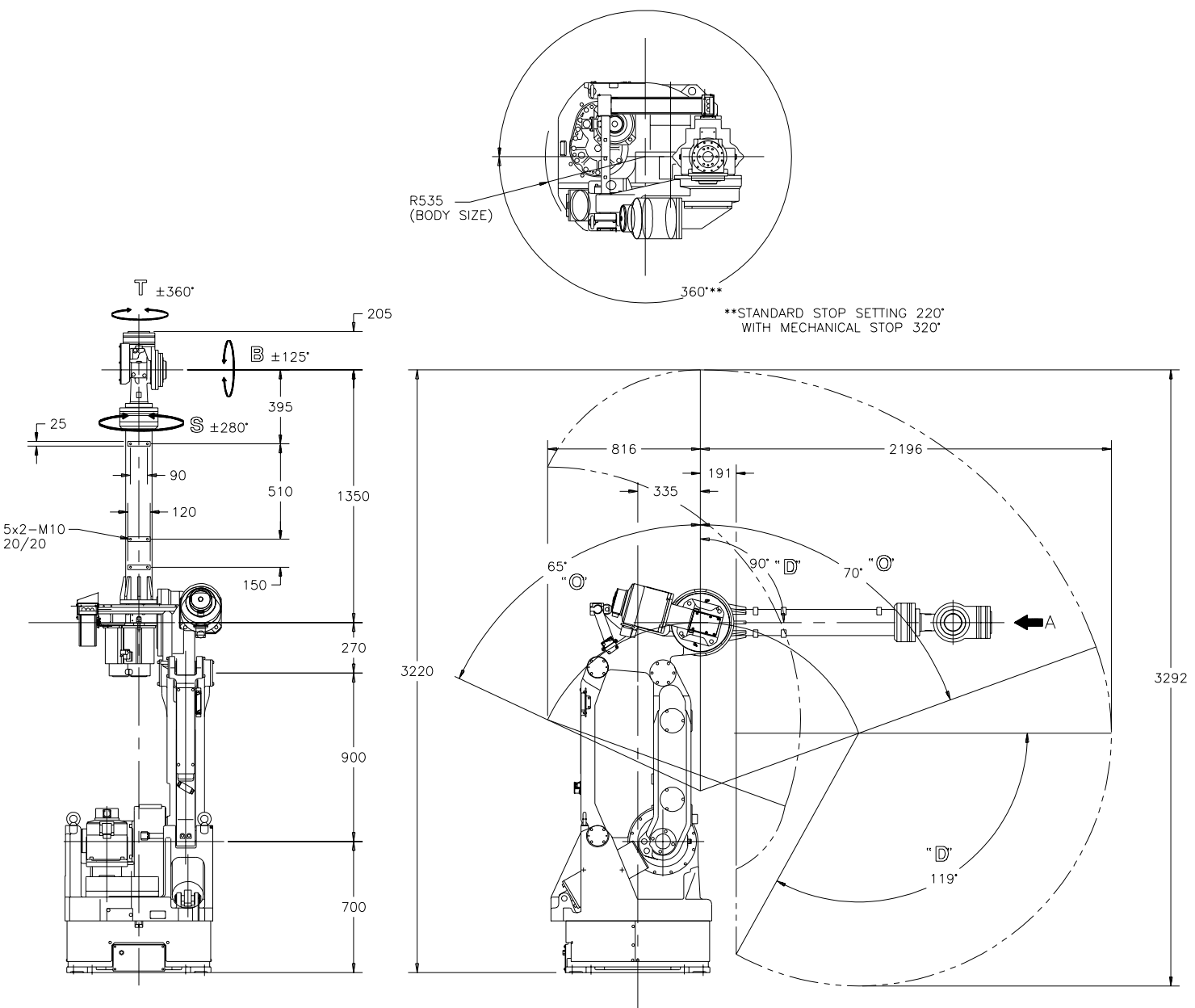


Figure 2-15 UX100/120/150 Work Envelope

**SAFETY**

**2.5.16 UX200**

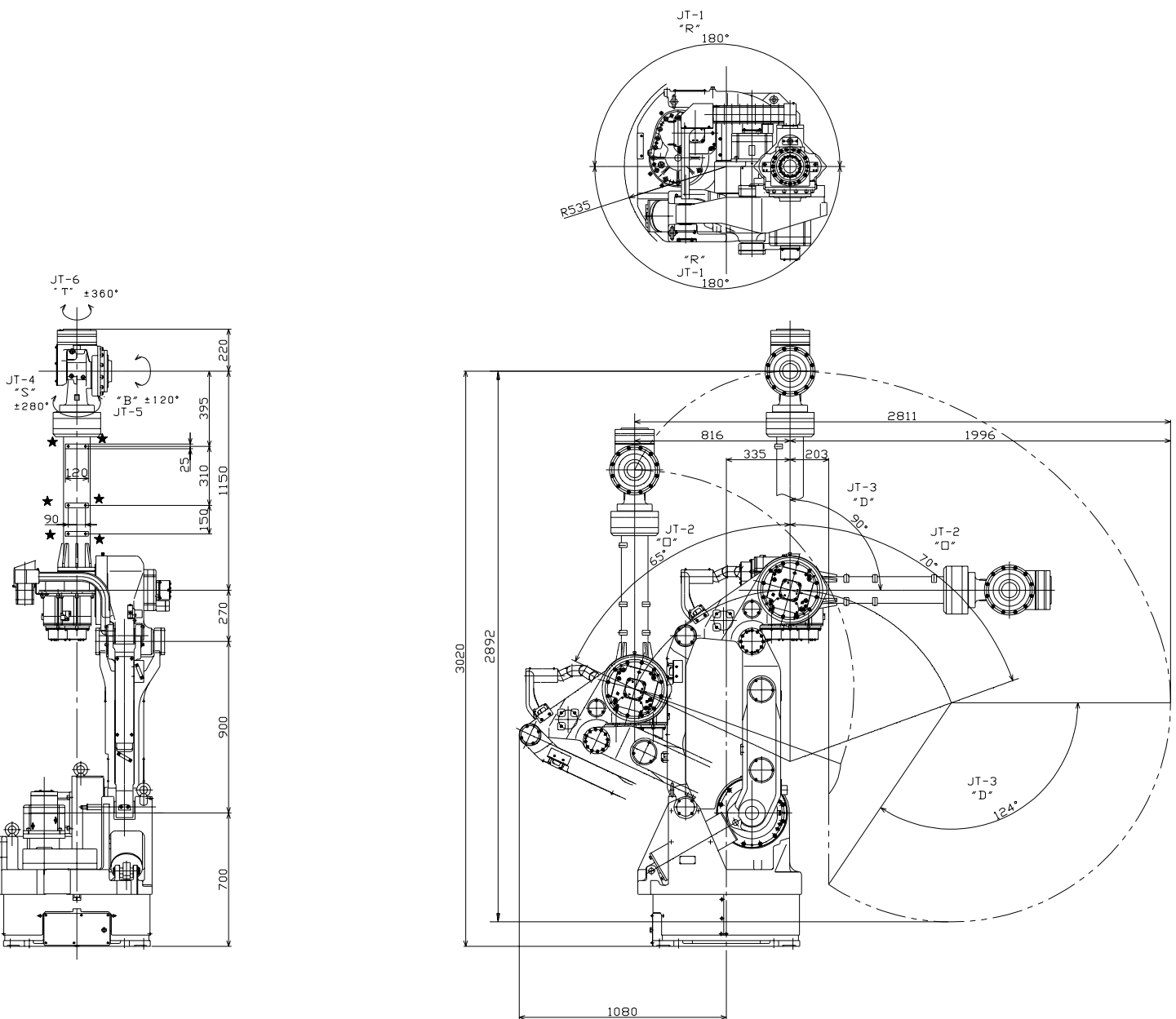


Figure 2-16 UX200 Work Envelope

**SAFETY**

**2.5.17 UX300**

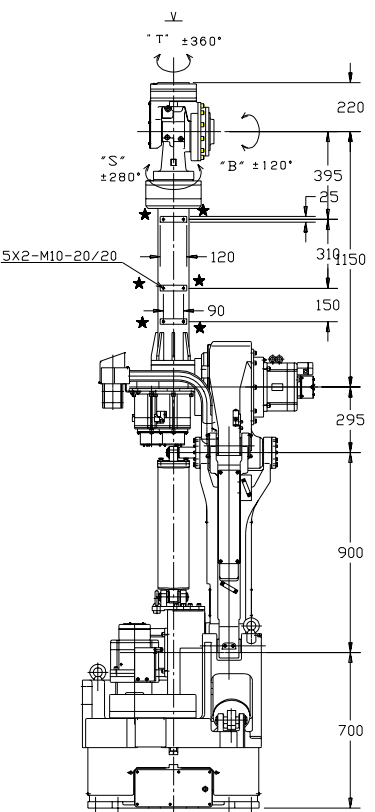
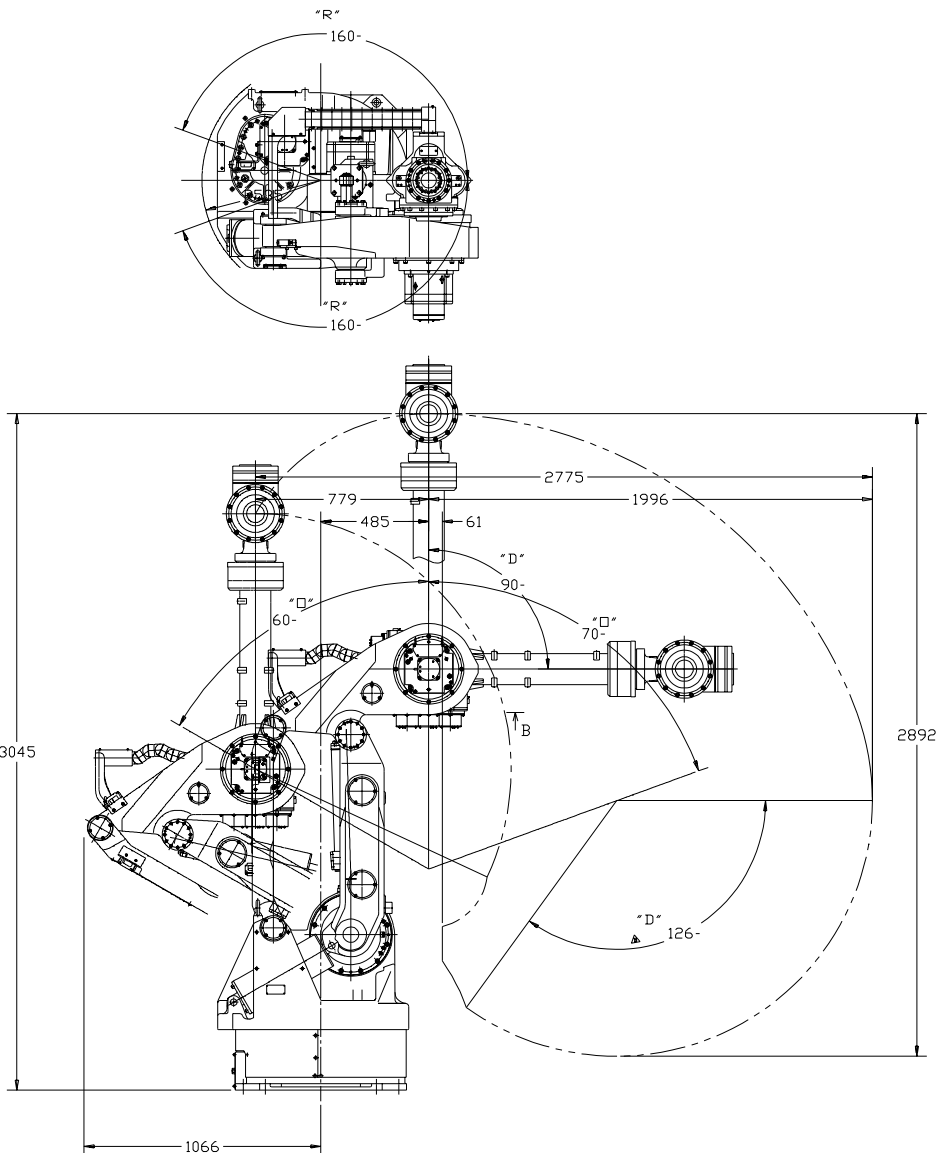


Figure 2-17 UX300 Work Envelope



**SAFETY**

**2.5.18 UZ100/120/150**

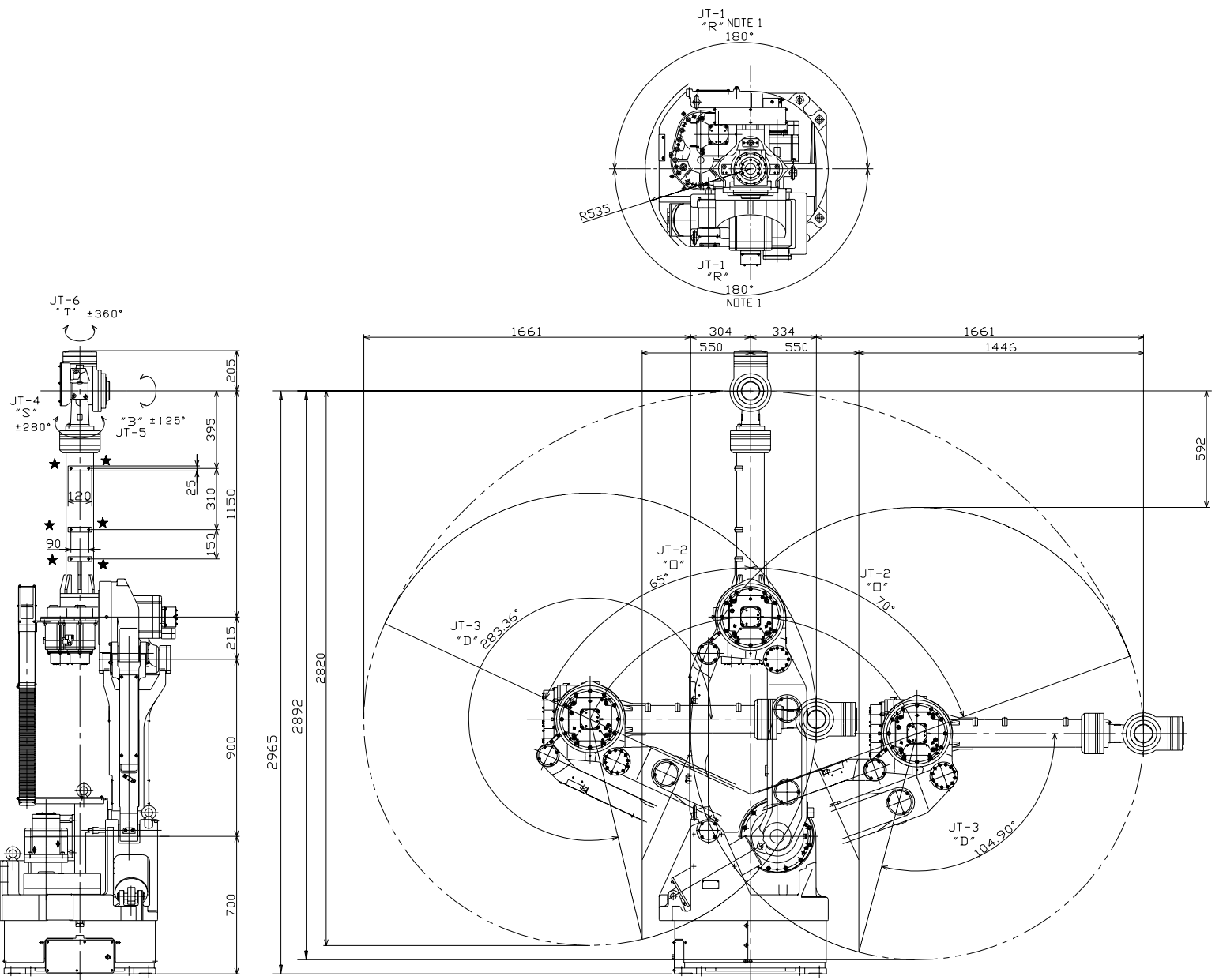


Figure 2-18 UZ100/120/150 Work Envelope

**SAFETY**

**2.5.19 ZD130**

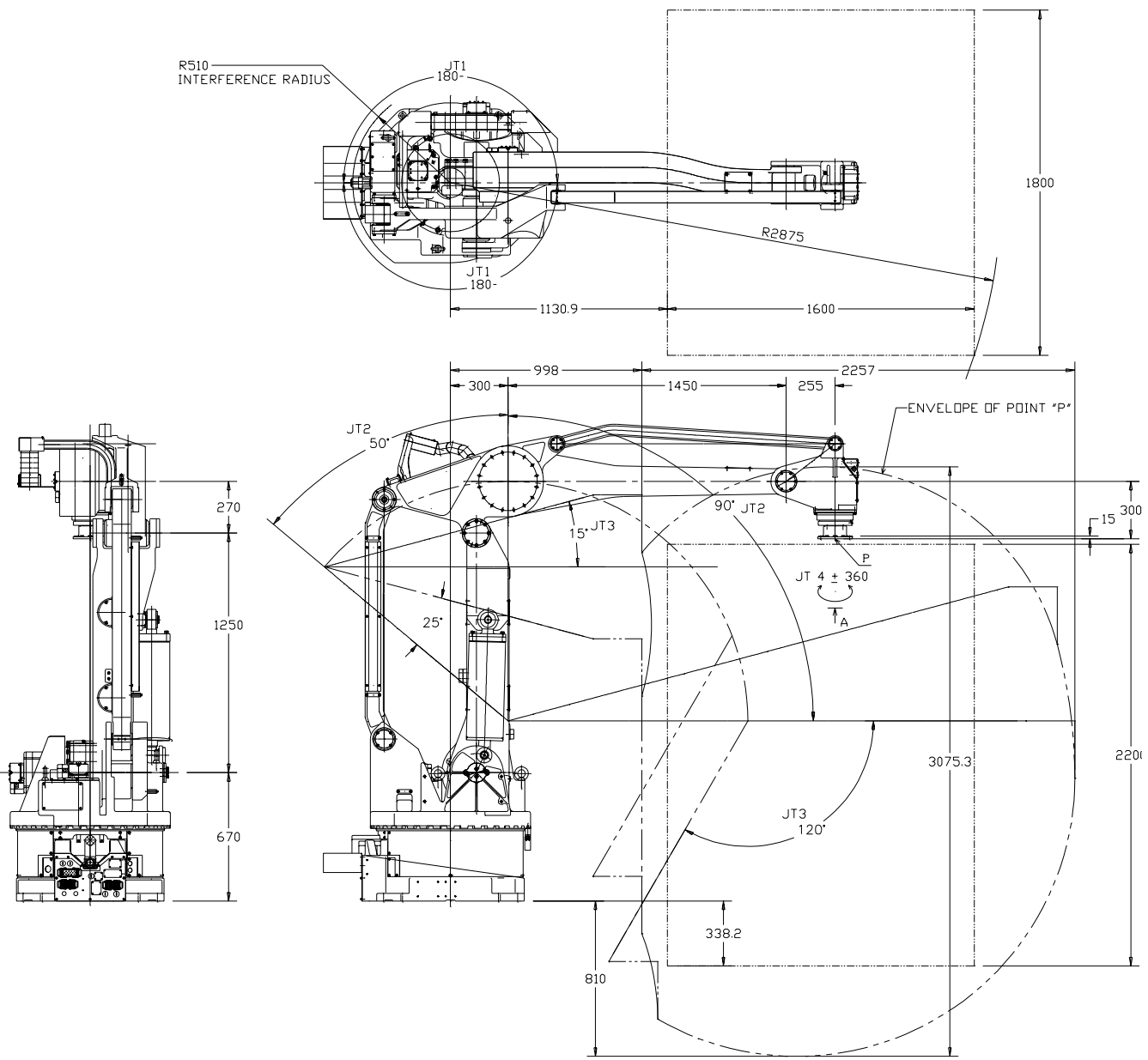


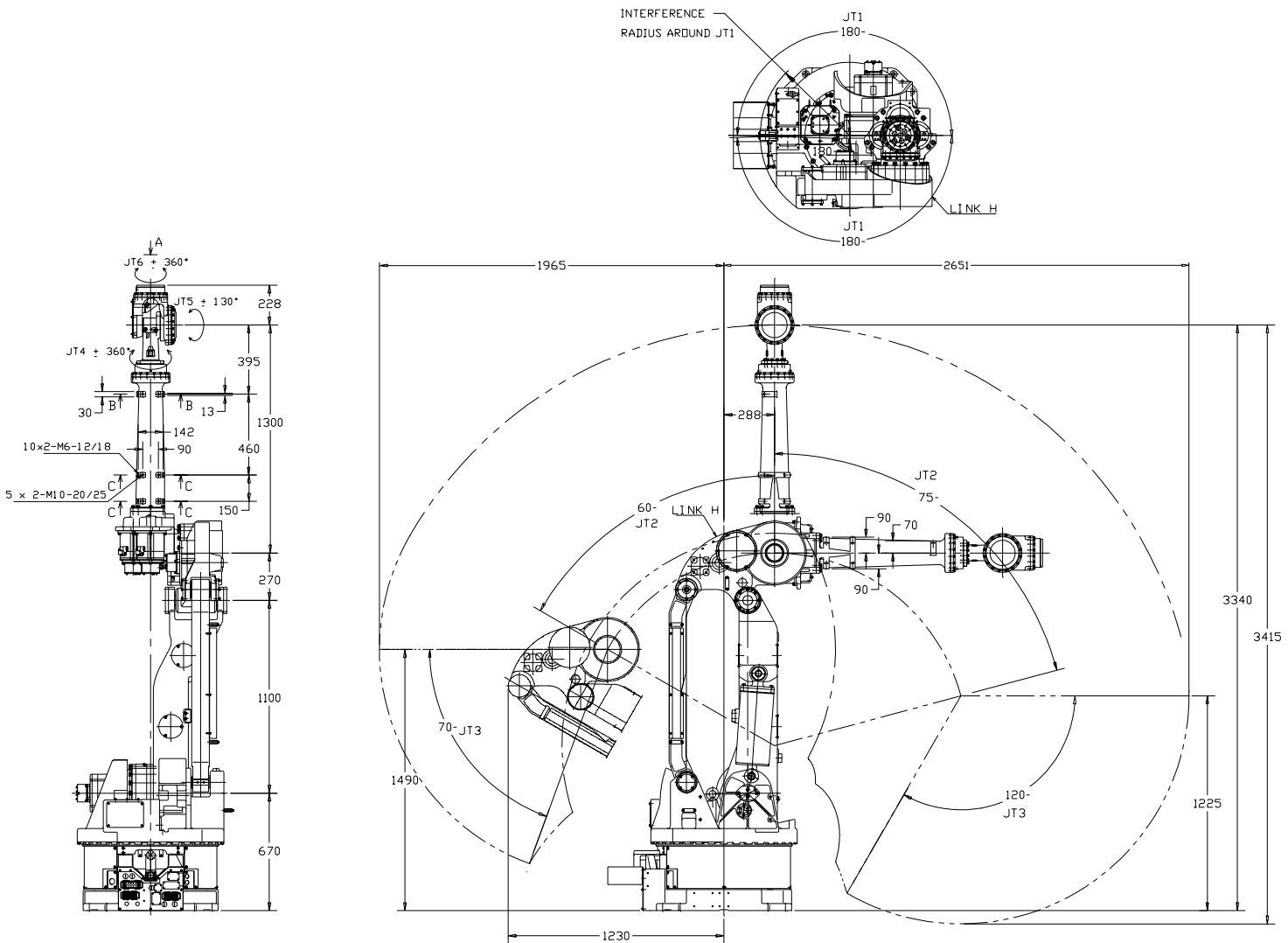
Figure 2-19 ZD130 Work Envelope





**SAFETY**

**2.5.22 ZX165U**



**Figure 2-22 ZX165U Work Envelope**

**SAFETY**

**2.5.23 ZX200S**

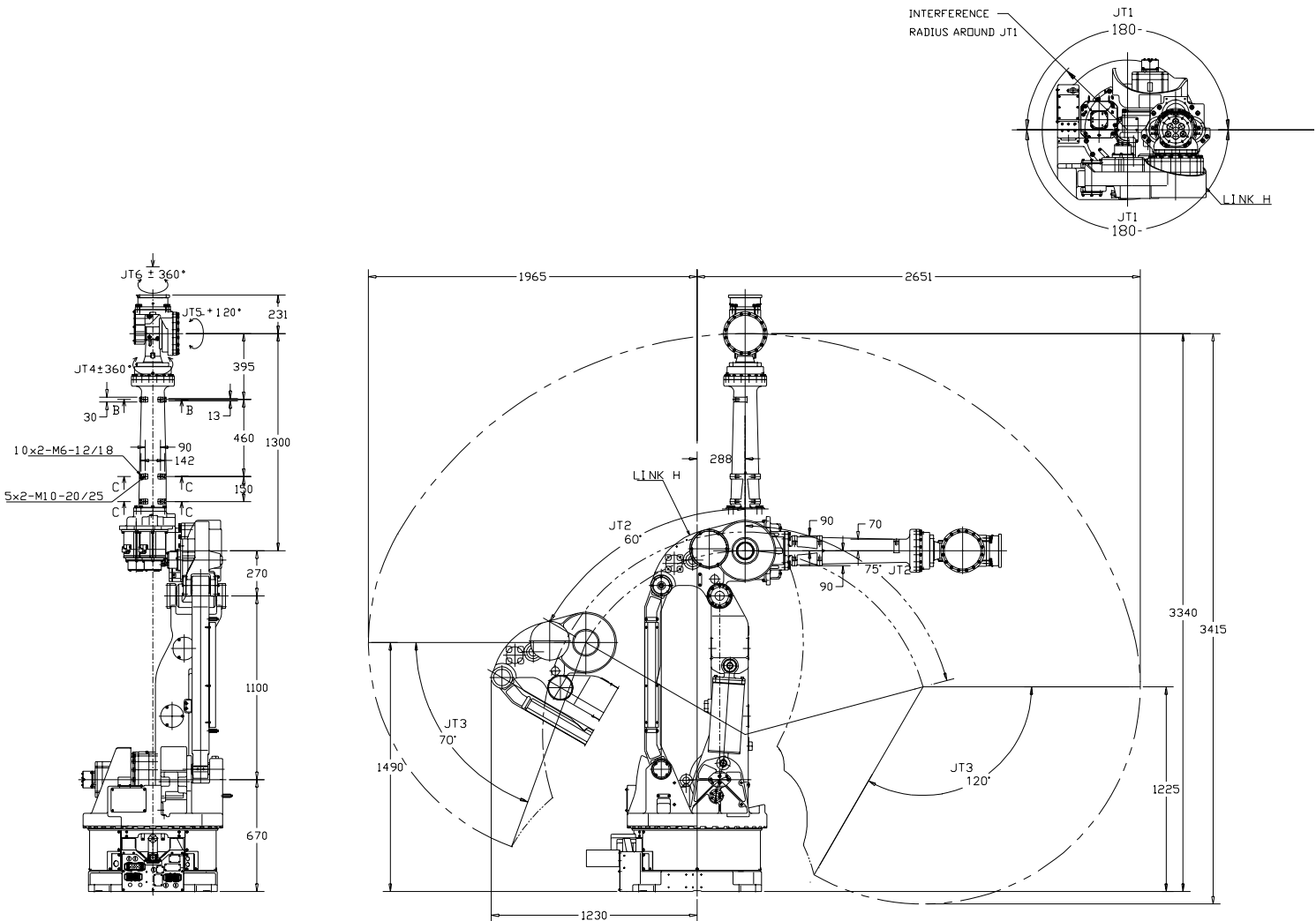


Figure 2-23 ZX200S Work Envelope

**SAFETY**

**2.5.24 ZX200U**

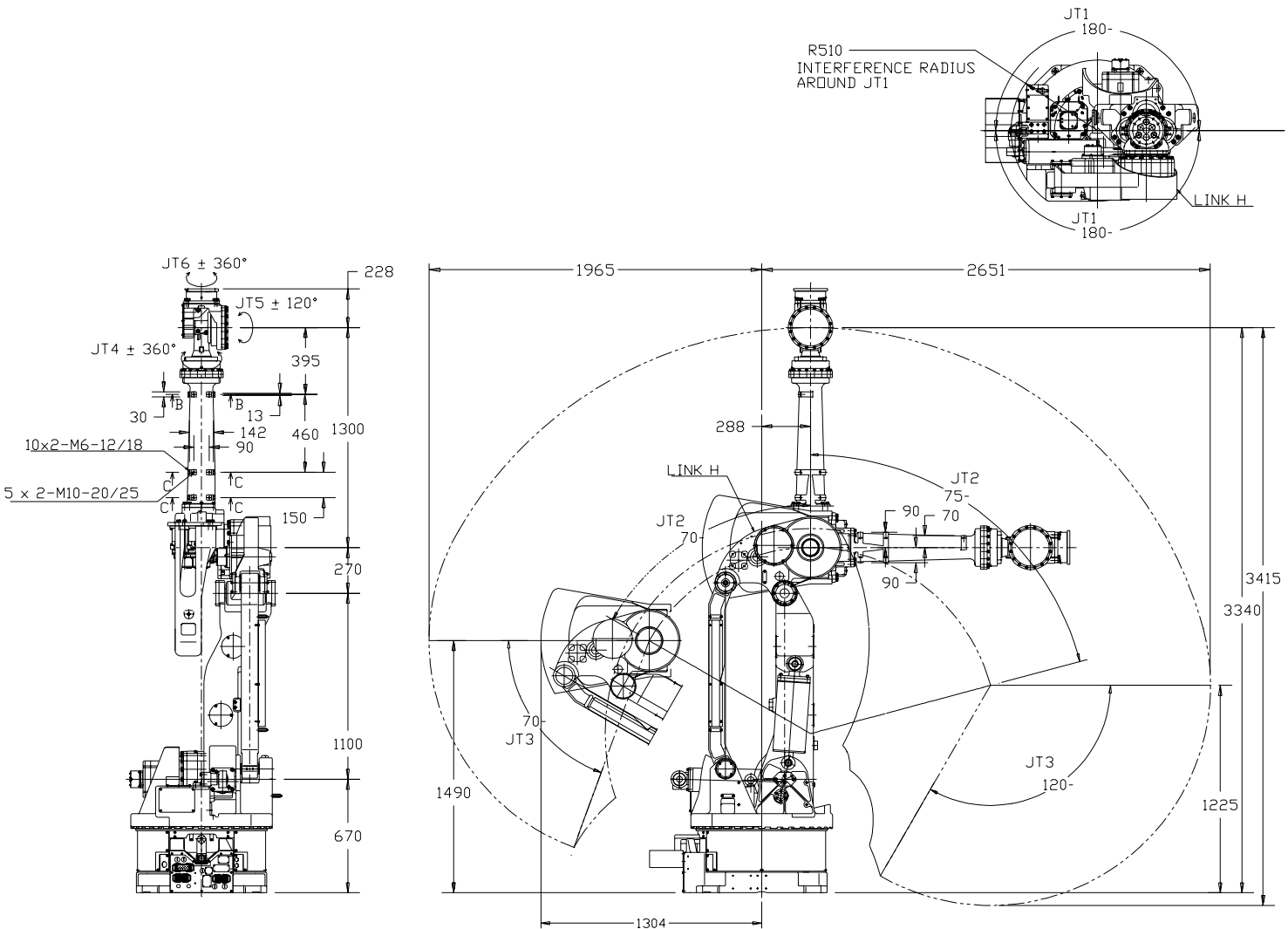


Figure 2-24 ZX200U Work Envelope

**SAFETY**

**2.5.25 ZX300S**

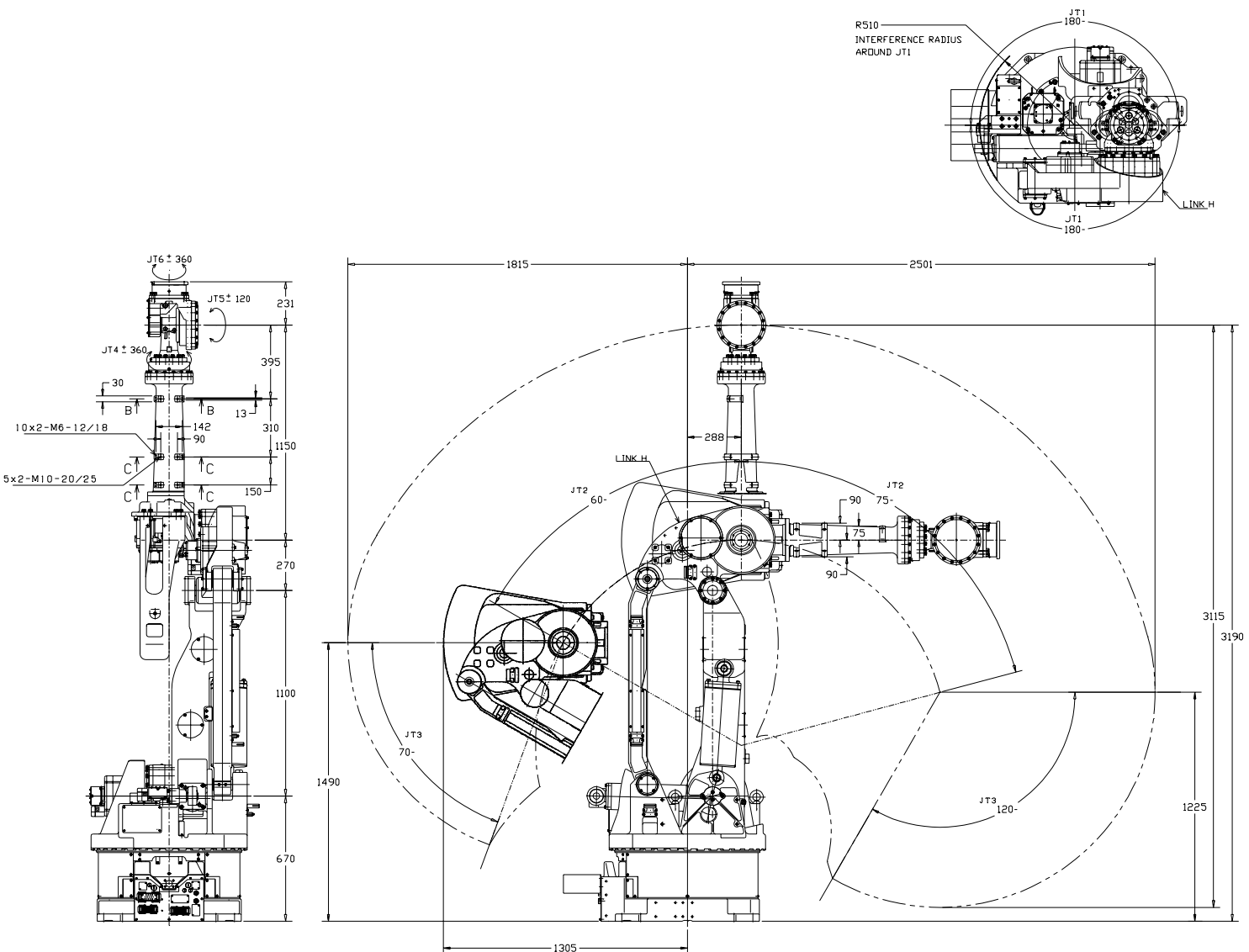


Figure 2-25 ZX300S Work Envelope



---

## POWER ON/OFF PROCEDURES

<b>3.0</b>	<b>POWER ON/OFF PROCEDURES</b> .....	3-2
3.1	Controller Power On/Off Procedures .....	3-2
3.1.1	Controller Power On Procedures .....	3-2
3.1.2	Controller Power Off Procedures .....	3-2
3.2	Servo Motor Power-On Procedures .....	3-7
3.2.1	Servo Motor Power-On in the Repeat Mode .....	3-7
3.2.2	Servo Motor Power-On in the Teach Mode .....	3-7
3.3	Methods for Stopping the Robot .....	3-8
3.3.1	Emergency Stop Switch .....	3-8
3.3.2	HOLD/RUN Switch .....	3-8
3.3.3	TEACH/REPEAT Switch .....	3-8

## **POWER ON/OFF PROCEDURES**

### **3.0 POWER ON/OFF PROCEDURES**

This unit provides the power ON/OFF procedures for the robot controller and servo motors. Refer to figures 3-1 through 3-7 during these procedures.

#### **3.1 CONTROLLER POWER ON/OFF PROCEDURES**

##### **3.1.1 CONTROLLER POWER ON PROCEDURES**

1. Ensure all personal are clear of the work cell and all safety devices are in place and operational.
2. Turn the HOLD/RUN switch to the HOLD position.
3. Place the controller main disconnect switch in the ON position. At this time the CONTROL POWER indicator lamp illuminates.

##### **3.1.2 CONTROLLER POWER OFF PROCEDURES**

1. Turn the HOLD/RUN switch to the HOLD position; the robot decelerates to a stop and the MOTOR POWER lamp turns off.
2. Press the EMERGENCY STOP switch. At this time the CYCLE START lamp turns off.
3. Place the controller main disconnect switch in the OFF position.

**POWER ON/OFF PROCEDURES**

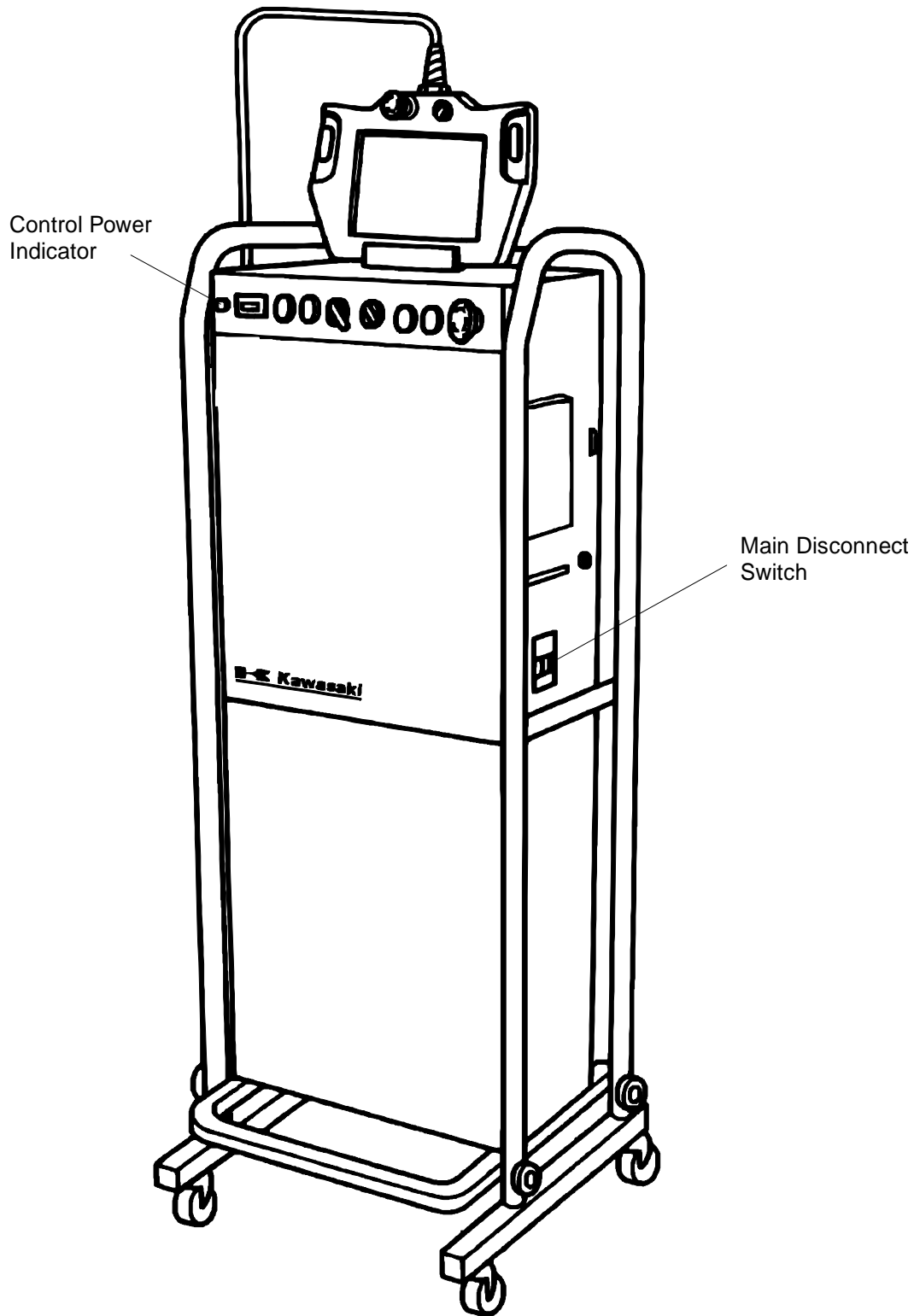


Figure 3-1 Standard C Controller

**POWER ON/OFF PROCEDURES**

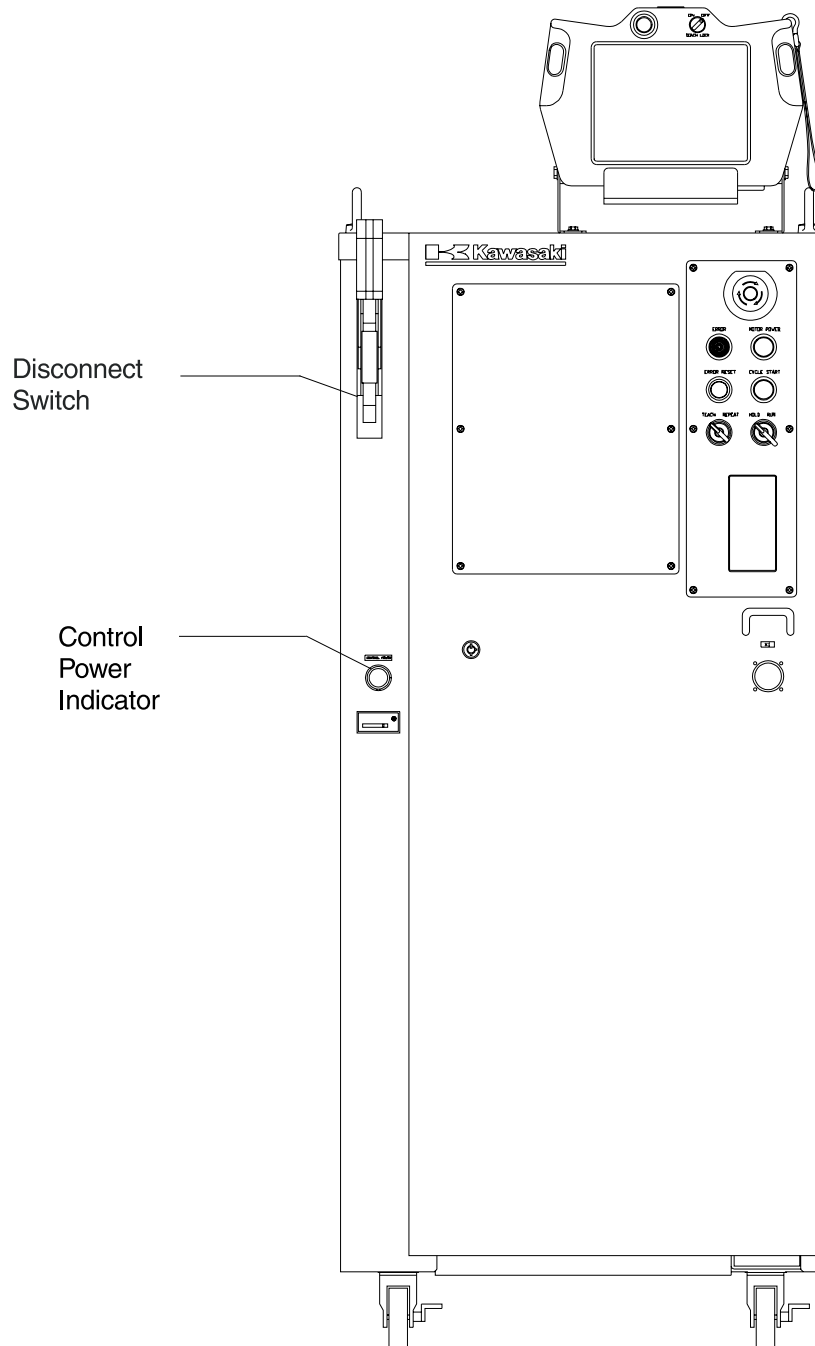


Figure 3-2 North American C Controller

**POWER ON/OFF PROCEDURES**

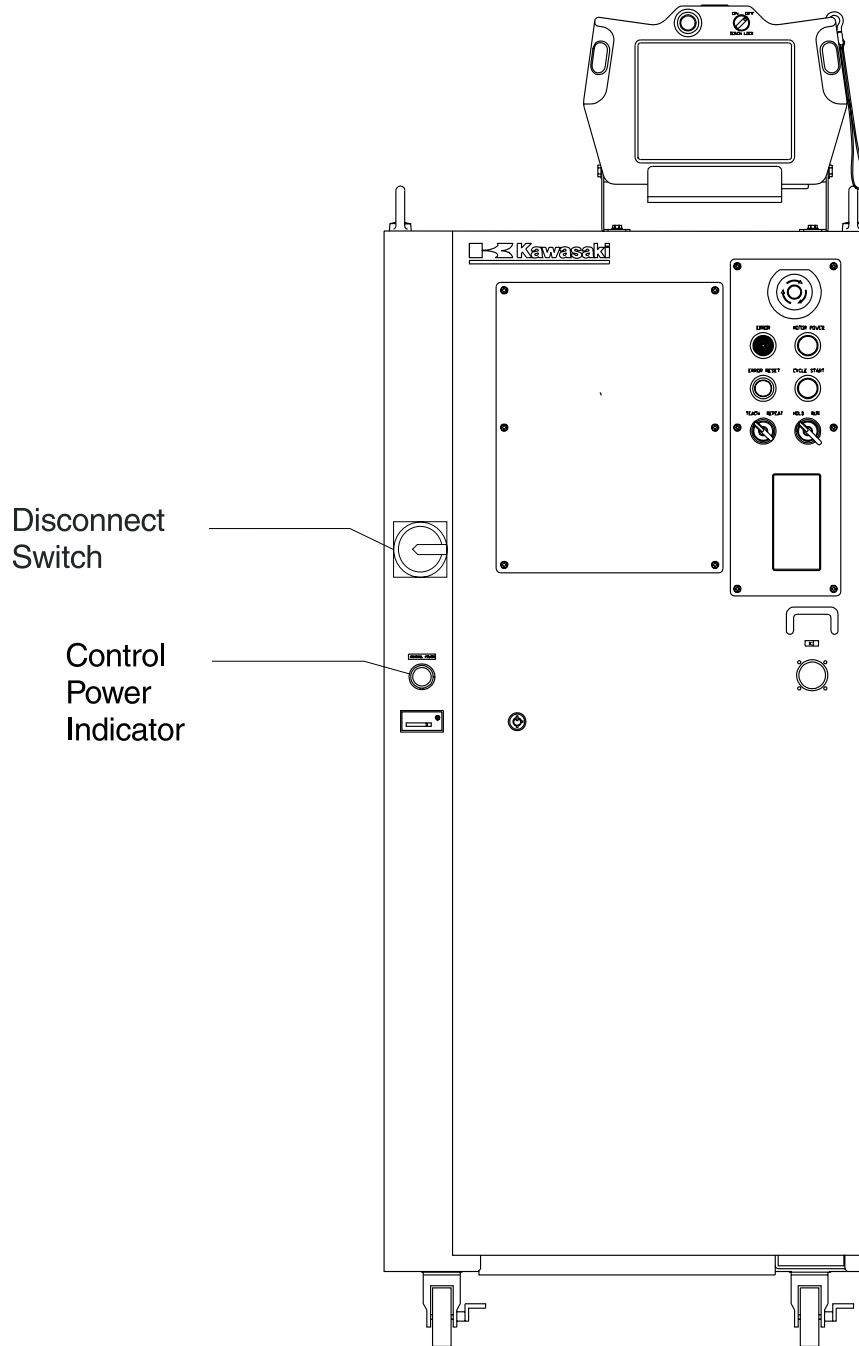


Figure 3-3 European C Controller

**POWER ON/OFF PROCEDURES**

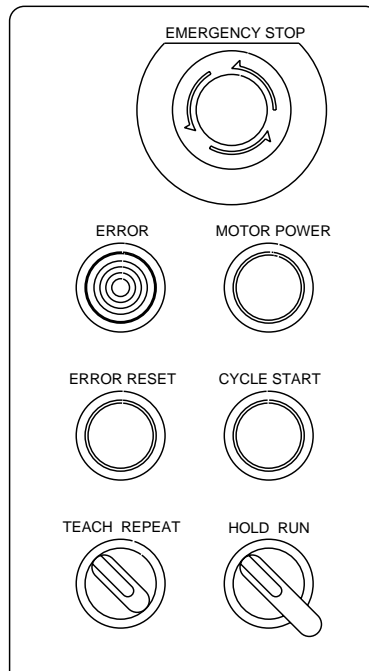


Figure 3-4 North American and European C Controller Switch Panel

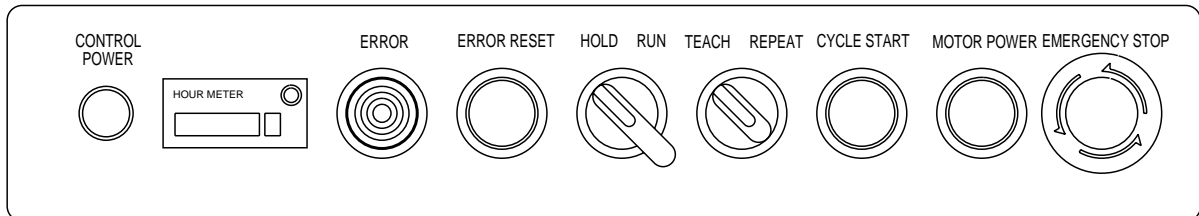


Figure 3-5 Standard C Controller Switch Panel

**POWER ON/OFF PROCEDURES****3.2 SERVO MOTOR POWER-ON PROCEDURES****3.2.1 SERVO MOTOR POWER-ON IN THE REPEAT MODE**

1. Place the TEACH LOCK switch on the multi function panel in the OFF position.
2. Place the TEACH/REPEAT switch in the REPEAT position.
3. Press the MOTOR POWER push button. The MOTOR POWER lamp illuminates.
4. Place the HOLD/RUN switch in the RUN position.
5. The robot is now ready to execute a program.

**3.2.2 SERVO MOTOR POWER-ON IN THE TEACH MODE**

1. Place the TEACH/REPEAT switch in the TEACH position.
2. Place the TEACH LOCK switch on the multi function panel in the ON position.
3. At the BLOCK TEACHING screen, press and hold one of the trigger (deadman) switches and press the MOTOR POWER push button. At this time the MOTOR POWER lamp illuminates.

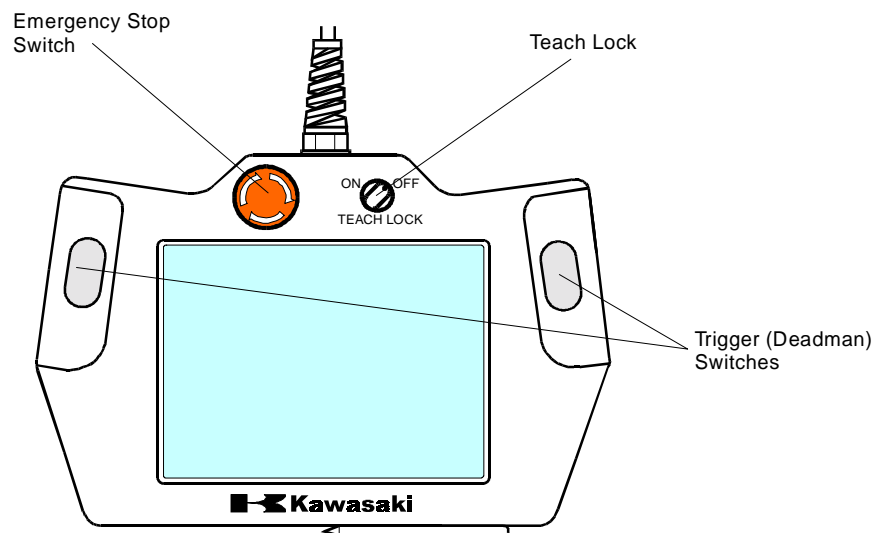


Figure 3-6 Multi Function Panel

---

## POWER ON/OFF PROCEDURES

### 3.3 METHODS FOR STOPPING THE ROBOT

One of three methods are used to stop robot motion. Each of these methods is described in the following sections.

#### 3.3.1 EMERGENCY STOP SWITCH

When the EMERGENCY STOP switch is pressed, motor power is turned off and the brakes are applied stopping the robot immediately. This places very high loads upon the robot and is only recommended for emergency situations. To stop the robot during non-emergency situations refer to section 3.3.2, HOLD/RUN SWITCH.

#### 3.3.2 HOLD/RUN SWITCH

When the HOLD/RUN switch is turned to the HOLD position the robot decelerates smoothly to a stop and the brakes are applied. This places the robot into a temporary stop condition. The motor power lamp turns OFF and the CYCLE START lamp remains ON. When the HOLD/RUN switch is again turned to the RUN position the robot continues the motion execution prior to HOLD. To create a permanent stop condition, press the EMERGENCY STOP switch or turn the TEACH/REPEAT switch to the TEACH position (the CYCLE START and MOTOR POWER indicator lamps turn off).

#### 3.3.3 TEACH/REPEAT SWITCH

When the TEACH/REPEAT switch is turned to the TEACH position motor power is turned off and the brakes are applied stopping the robot immediately. This places very high loads upon the robot and is only recommended for emergency situations. To stop the robot during non-emergency situations refer to section 3.3.2, HOLD/RUN SWITCH.



---

**AS LANGUAGE COMMANDS**

<b>4.0</b>	<b>MONITOR AND EDITOR COMMANDS</b> .....	4-3
4.1	Keyboard Display Control .....	4-4
4.1.1	Terminal Control .....	4-4
4.2	Editor Commands .....	4-5
4.3	Program and Data Control Commands .....	4-14
4.3.1	DIRECTORY Commands .....	4-15
4.3.2	LIST Commands .....	4-17
4.3.3	DELETE Commands .....	4-19
4.3.4	RENAME Command .....	4-20
4.3.5	XFER and COPY commands .....	4-21
4.4	Program and Data Storage Commands .....	4-22
4.4.1	FORMAT and FDIRECTORY Commands .....	4-23
4.4.2	SAVE Command .....	4-23
4.4.3	LOAD and FDELETE Commands .....	4-25
4.5	Program Control .....	4-26
4.5.1	SPEED and PRIME Commands .....	4-26
4.5.2	EXECUTE, STEP, and MSTEP Commands .....	4-28
4.5.3	ABORT, HOLD, CONTINUE, and STPNEXT Commands .....	4-30
4.5.4	KILL and DO Commands .....	4-31
4.6	Defining Locations, Limits, and Home Positions .....	4-31
4.6.1	Here, Teach, and Point Commands .....	4-32
4.6.2	TOOL and BASE Commands .....	4-36
4.6.3	ULIMIT and LLIMIT Commands .....	4-39
4.6.4	SETHOME Command .....	4-40
4.6.5	WHERE Command .....	4-41
4.7	System Information .....	4-43
4.7.1	ERRLOG and OPLOG Commands .....	4-43
4.7.2	STATUS, FREE, ID, and HELP Commands .....	4-44
4.8	System Control .....	4-47
4.8.1	SYSINIT, TIME, and ERESET Commands .....	4-47
4.8.2	SWITCH, HSETCLAMP, and ZSIGSPEC Commands .....	4-48
4.8.3	ZZERO Command .....	4-49
4.8.4	BATCHK, ENCCHK_EMG, and ENCCHK_PON Commands .....	4-51
4.8.5	SLOW_REPEAT, REC_ACCEPT, ENV_DATA, and ENV2_DATA Commands .....	4-52
4.8.6	CHSUM Command .....	4-55
4.9	Signal Commands .....	4-56
4.9.1	SIGNAL, PULSE, DLYSIG, and BITS Commands .....	4-56
4.9.2	I/O and RESET Commands .....	4-58
4.9.3	DEFSIG Command .....	4-60
4.10	Z-Series Robots AS Language Commands .....	4-62
4.10.1	1GV Arm ID Board Functions and Commands .....	4-62
4.10.2	Failure Prediction Function, Aux 124 (Option) .....	4-65
4.10.2.1	Failure Prediction Function Setup Procedure .....	4-65

---

**AS LANGUAGE COMMANDS**

4.10.3	Collision Detection Function .....	4-66
4.10.3.1	Setting Tool Weight Data Using AS Language WEIGHT Command .....	4-66
4.10.3.2	Range of Threshold .....	4-67
4.10.3.3	Setting Thresholds .....	4-68
4.10.3.4	Collision Detection AS Language Commands .....	4-73
4.10.3.5	COLR .....	4-74
4.10.3.6	COLRON/OFF .....	4-76
4.10.3.7	COLRJ .....	4-77
4.10.3.8	COLRJON/OFF .....	4-78
4.10.3.9	COLT .....	4-78
4.10.3.10	COLTON/OFF .....	4-78
4.10.3.11	COLTJ .....	4-79
4.10.3.12	COLTJON/OFF .....	4-79
4.10.3.13	COLMVON/OFF .....	4-80
4.10.3.14	COLCALON/OFF .....	4-81
4.10.3.15	WEIGHT .....	4-81
4.10.3.16	SETCOLTHID .....	4-82
4.10.3.17	COLINIT .....	4-83
4.10.3.18	COLSTATE .....	4-83
4.10.3.19	Collision Detection Error Code .....	4-85
4.10.3.20	Collision Detection Troubleshooting .....	4-85

## AS LANGUAGE COMMANDS

### 4.0 MONITOR AND EDITOR COMMANDS

The AS system is operational as soon as power is applied to the controller. To access the MONITOR mode, press the menu key and select the keyboard screen (Figure 4-1).

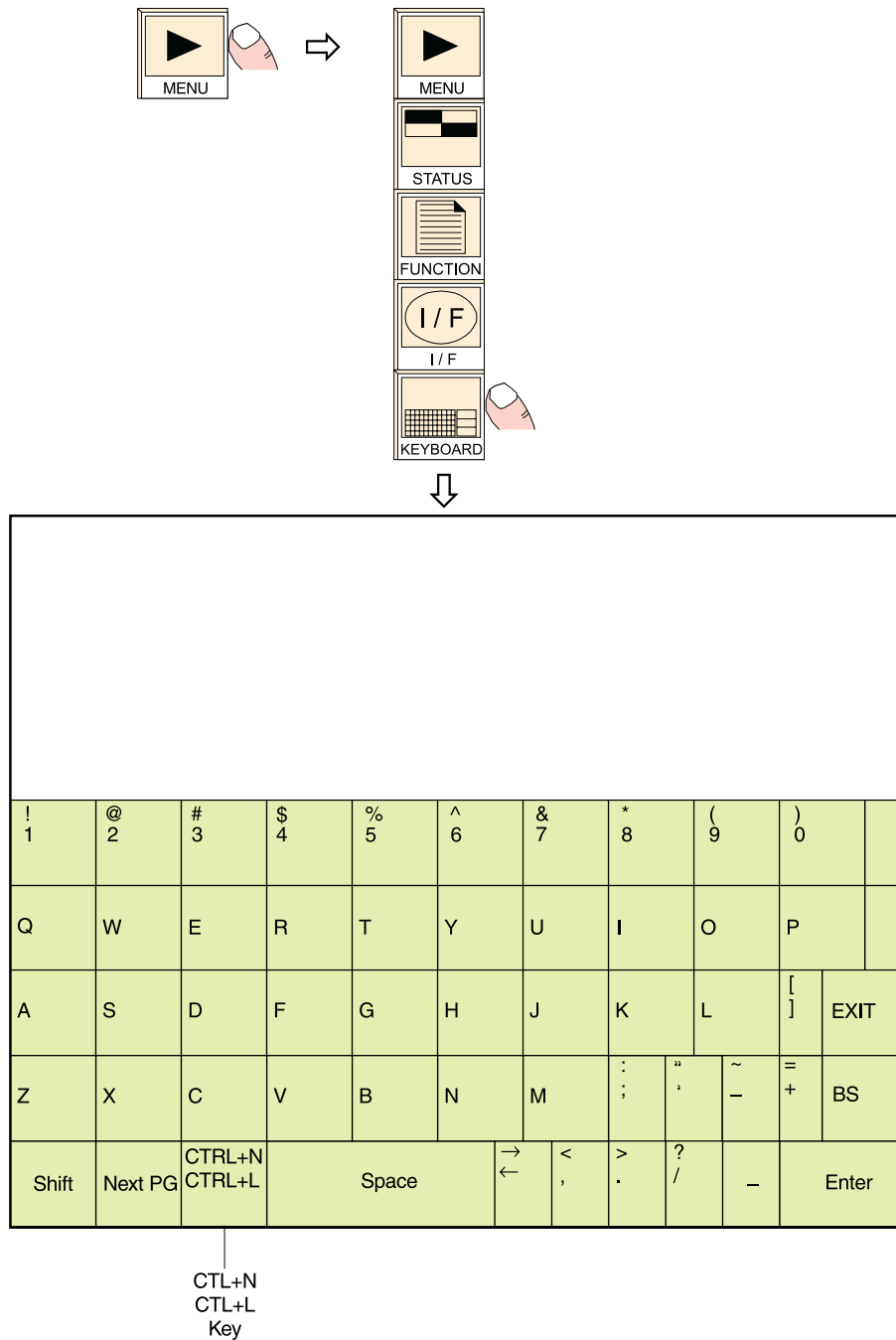


Figure 4-1 Keyboard Screen

---

## AS LANGUAGE COMMANDS

### 4.1 KEYBOARD DISPLAY CONTROL

**CTL+L/CTL+N** The CTL+L/CTL+N (last/next) key enables the programmer to display the contents of the last line entered to appear on the current line or change the the current line to the next (CTL+N) one in order of lines after the CTL+L key is used. The CTL+N key is only effective after CTL+L is used more than once.

CTL+L is available with the keyboard in the normal mode and CTRL+N is available with the keyboard in the shifted mode (Figure 4-1).

#### 4.1.1 TERMINAL CONTROL

The following terminal control commands are available for AS Language programming using the keyboard of a personal computer (PC) interfaced with the C controller (see unit 8):

- CTRL C** This command cancels the current input line. It is not used to terminate the program that is currently executing. This command is similar to pressing the EXIT key on the multi function panel keyboard.
- CTRL L** This command enables the contents of the line of code previously entered to display on the current input line. This operation can be used up to seven times to recover previously entered data.
- CTRL N** The CTRL N command is used in conjunction with the CTRL L command. The CTRL N command changes the contents of the current input line to the next one in the history of command inputs after the CTRL L command is used. This operation is effective after CTRL L is pressed more than once.
- CTRL Q** This command is used to resume the updating of displayed information after it was stopped with a CTRL S command.
- CTRL S** Stops the scrolling of output information displayed. This command is used to confirm output information. Output resumes when CTRL Q is entered.

## AS LANGUAGE COMMANDS

### 4.2 EDITOR COMMANDS

Editor commands are used after accessing the editor mode. The user must type “edit” in the MONITOR mode followed by the name of the program to edit or a new program name.

**ED**                    **program\_name,step**  
**EDIT**

**program\_name:**    Name of the program to edit. If the program does not exist, a new program is created. If a program name is not specified, then the last program edited is opened for editing.

**step:**                Optional step number to start editing. If the step is not specified, editing starts at the first step or the last step edited. If an error occurred during the last program run, the step where the error occurred is selected.

#### NOTE

The program that is executing cannot be edited or deleted. Program commands or instructions are entered in lowercase or uppercase characters. When listing or editing the program, keywords are displayed in uppercase characters

```
$Edit test  
  
 .PROGRAM test()  
1 LMOVE aa  
1?  
2 LMOVE bb  
2?  
3 HOME  
3?
```

## AS LANGUAGE COMMANDS

### S STEP

#### step\_number

The number of the step to edit. If the step number is not specified, the first step is selected. If the step number is greater than the number of steps in the program, a new step following the last recorded step in the program is selected.

```
$Edit test

.PROGRAM test()

1 LMOVE aa
1?
2 LMOVE bb
2?
3 JMOVE cc
3?s 6
6 LMOVE dd
6?
7 LDEPART 50
7?s
1 LMOVE aa
1?
```

### P PRINT

#### step\_count

The number of steps to display beginning with the current step. If the step number is not specified, only one step is displayed.

```
$Edit test

.PROGRAM test()

1 LMOVE aa
1?
2 LMOVE bb
2? p 2
2 LMOVE aa
3 LMOVE bb
4 SPEED 500 mm/s
4?
```

---

## AS LANGUAGE COMMANDS

### L LAST

Displays the previous step for editing.

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 LMOVE bb
2?
3 JMOVE cc
3? I
2 LMOVE bb
2?
```

### I INSERT

Inserts lines before the current step and all consecutive steps are renumbered. To terminate the insert mode, press the RETURN key twice.

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 LMOVE bb
2?i
2I ARC ON
3I LAPPRO start,50
4I LWS start
5I
5 LWE end,1,1
5?
```

**AS LANGUAGE COMMANDS****D  
DELETE****step\_count**

Deletes program steps beginning with the current step. All consecutive steps are renumbered. The `step_count` specifies the number of steps to delete beginning with the current step. If the step count is not entered, the current step is deleted.

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 ARC ON
2?d 3
2 ARC ON
2 LAPPRO start,50
2 LWS start
2 LWE end,1,1
2?
3 LDEPART 50
```

**F  
FIND****character\_string**

Searches the current program for the specified `character_string` from the current step to the last, and displays the first step that includes the string.

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 LWE end,1,1
2?
3 LDEPART 50
3?f home
6 HOME
6?
7 JMOVE clean
7?
```



**AS LANGUAGE COMMANDS****M**  
**MODIFY**                    **/existing\_characters/new\_characters**

Modifies the current step by replacing the existing characters specified with the new characters specified.

**/existing\_characters:**                    Characters to modify in the current step.

**/new\_characters:**                    Characters used to modify the existing characters.

```
$Edit test

  .PROGRAM test()
  1 LMOVE aa
  1?
  2 LWE end1,1
  2?
  3 LDEPART 50
  3?m/50/75
  3 LDEPART 75
  3?
```

**O**  
**OVER**

Places the cursor on the current step for editing.

Use the arrow keys to move the cursor within the step. The Back-space key removes the character preceding the cursor.

```
$Edit test

  .PROGRAM test()
  1 LMOVE aa
  1?
  2 LMOVE bb
  2?o
  2□ LMOVE bb
```

---

## AS LANGUAGE COMMANDS

### R REPLACE

#### character\_string

Replaces existing characters on current step with new characters.

character\_  
string:

New characters to replace the existing characters.

To use the REPLACE command:

- Use the spacebar to move the cursor under the first character to change.
- Press the “r” key, then the spacebar.
- Enter the new character(s) to replace existing characters in the step and press the RETURN key.

```
$Edit test  
  
  .PROGRAM test()  
1 LMOVE aa  
1?  
2 JMOVE bb  
2?  
3 LDEPART 75  
3?      r 50  
3 LDEPART 50  
3?  
4 JMOVE cc  
4?
```

---

**AS LANGUAGE COMMANDS****C  
CHANGE****program\_name, step\_number**

Opens the selected program for editing at the specified step

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 LMOVE bb
2?c welda1

  .PROGRAM welda1()

1 HOME
1?
2 JMOVE #START
2?
3 ARC ON
3?
```

**E  
EXIT**

Exits from the editor to the monitor mode.

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 LMOVE bb
2?e
$
```

---

## AS LANGUAGE COMMANDS

### **XD CUT**

#### **number of lines**

The XD command is used to remove (cut) a specified number of lines from a program and store them in the paste buffer.

Move the cursor to the first line to remove to the paste buffer and enter XD, the number of lines to cut, and press ENTER.

The number of lines specified, including the current line, are placed in the paste buffer and the program steps are renumbered accordingly.

When the XD command is used again, the contents of the paste buffer are overwritten.

### **XY COPY**

#### **number of lines**

The XY command is used to copy a specified number of lines from a program and store them in the paste buffer.

Move the cursor to the first line to copy to the paste buffer and enter XY, the number of lines to copy, and press ENTER.

The number of lines specified, including the current line, are placed in the paste buffer.

When the XY command is used the lines in the program are not affected.

### **XP PASTE**

Places the contents of the paste buffer into a program.

The steps in the paste buffer are placed in the program ahead of the step number where the XP command is entered.

The program steps are renumbered accordingly.

---

## AS LANGUAGE COMMANDS

**XQ  
PASTE** Places the contents of the paste buffer into a program in reverse order.

The steps in the paste buffer are placed in the program ahead of the step number where the XQ command is entered.

The program steps are renumbered accordingly.

**XS  
DISPLAY** Displays the contents of the paste buffer.

The XS command entered at step 2 displays the three steps in the paste buffer.

```
$Edit test

  .PROGRAM test()
1 LMOVE aa
1?
2 JMOVE bb
2?XS
--- Paste Buffer ---
[1]>  JMOVE #a
[2]>  JMOVE #b
[3]>  JMOVE #c
2 JMOVE bb
2?
```

---

## AS LANGUAGE COMMANDS

### 4.3 PROGRAM AND DATA CONTROL COMMANDS

DIRECTORY	Displays the names of all programs and variables.
DIRECTORY/P	Displays the names of programs.
DIRECTORY/L	Displays the names of locations.
DIRECTORY/R	Displays the names of real variables.
DIRECTORY/S	Displays the names of string variables.
LIST	Displays all program steps and variable values.
LIST/P	Displays all program steps.
LIST/L	Displays the value of specified locations.
LIST/R	Displays the value of specified real variables.
LIST/S	Displays the value of specified string variables.
DELETE	Deletes specified programs and related variables.
DELETE/P	Deletes specified programs.
DELETE/L	Deletes specified locations.
DELETE/R	Deletes specified real variables.
DELETE/S	Deletes specified string variables.
RENAME	Changes the name of a program.
XFER	Copies steps from one program to another.
COPY	Copies one or more programs to a new program.

## AS LANGUAGE COMMANDS

### 4.3.1 DIRECTORY COMMANDS

The DIRECTORY commands display the names of programs and variables residing in memory. If a program name is not specified when using the DIRECTORY command, all program names, location names, real variable names, and string variable names are listed. The screen stops at the end of each page until the spacebar is pressed, and continues to do so until all names have been listed. Pressing the ENTER key stops the listing.

The asterisk "\*" is a wild card character which represents any character. It is used with all program and data control commands except the RENAME command. The following illustration shows how the asterisk is used to list specific information. For example, if DIR w\* is typed, all programs beginning with "w" and subroutine programs called by the selected programs are listed. All locations and real variables used in the programs are displayed.

```
$Dir w*
Programs
weld1      weld2      weld3      weldtop
pg01      pg10
Location Variables
a1         a2         a3         b1
fixt.1     part1     part2     pos.1
#aa1      #aa2
Real Variables
deg        i          r1        st1
String Variables
$Count
```

---

**AS LANGUAGE COMMANDS****DIRECTORY/P**      **program\_name**

Displays the names of programs in memory.

program\_name:      Name of the program to be displayed.

```
$Dir/p
Programs
arc  sample  zz      pg0
pg1  pg2      pg10   pg555
```

**DIRECTORY/L**      **location\_name**

Displays the names of locations in memory.

location\_name:      Name of the location to display

```
$Dir/l
Location Variables
a    a1    a2    a3
bb   begin safe  zx
```

**DIRECTORY/R**      **real\_variable\_name**

Displays the names of real variables in memory.

real\_variable\_
name:      Name of the real variable to be displayed.

```
$Dir/r
Real Variables
deg   got300fix  i    p[]
r1    st      step
```



**AS LANGUAGE COMMANDS****DIRECTORY/S**      **string\_variable\_name**

Displays the names of string variables in memory.

string\_variable\_  
name:

Name of the string variable to be displayed.

```
$Dir/s  
String Variables  
$count      $name      $start
```

**4.3.2 LIST COMMANDS**

The LIST command displays program steps and the values of variables residing in memory. If a name is not specified when using the LIST command, all program names, location names, real variable names, and string variable names are listed. The screen stops at the end of the page until the spacebar is pressed and continues until all names have been listed.

**LIST/P**            **program\_name**

Displays all program steps.

program\_name:      Name of the program to be listed.

```
$Li/p test  
.PROGRAM test()  
1 SPEED 50 mm/s  
2 LMOVE part  
3 W1SET1= 50,190,21  
4 LMOVE fixt  
5 HOME  
.END
```

**AS LANGUAGE COMMANDS****LIST/L**                    **location\_name**

Displays the value of the specified locations.

location\_name:            Location variable name.

```
$Li/l part*
Location
      X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
part1 -130.3  390.49 -154.56  155.67  90.00  -2.65
part2 -230.3  490.40 -254.67  123.67 -45.00  1.00
part3  230.3  590.00  100.05  180.00  0.00  -5.06
```

**LIST/R**                    **real\_variable\_name**

Displays the value of the specified real variable.

real\_variable\_
name:                    Real variable to be listed.

```
$Li/r r1
Real
r1    = 100
```

**LIST/S**                    **string\_variable\_name**

Displays the value of the specified string variable.

string\_variable\_
name:                    Name of string variable to be listed.

```
$Li/s
String
$count = "Number of parts"
$name  = "Kawasaki"
```

## AS LANGUAGE COMMANDS

### 4.3.3 DELETE COMMANDS

The DELETE command is used to delete specified programs, location variables, real variables, and string variables from memory.

**DELETE**            **program\_name**

Deletes the specified programs.

**program\_name:**    Name of program to delete. All subroutines, locations, real variables, and string variables within the specified program are deleted unless a subroutine is called by another program. The current program on the program stack cannot be deleted.

To delete all programs starting with a specified character, use that character with an asterisk. If DEL s\* is typed, all programs starting with the letter "s" are deleted, including any subroutines and all variables called by those programs. Likewise, if DEL pg\* is typed, all programs starting with "pg" are deleted, including their subroutines and variables.

```
$Del base
Are you sure? (Yes: 1, No: 0) : 1
Base: Because select program is executing, can't delete it

$Del/p p*
Are you sure? (Yes: 1, No: 0) : 1
$
```

**DELETE/P**            **program\_name**

Deletes the specified programs.

**program\_name:**    Name of the program to delete.

**AS LANGUAGE COMMANDS****DELETE/L**            **location\_name**

Deletes the specified locations.

location\_name:        Name of location to delete.

**DELETE/R**            **real\_variable\_name**

Deletes the specified real variables.

real\_variable\_  
name:                 Name of real variable to delete.**DELETE/S**            **string\_variable\_name**

Deletes the specified string variables.

real\_variable\_  
name:                 Name of the string variable to delete.**4.3.4 RENAME COMMAND****RENAME**            **new\_program\_name = old\_program\_name**

Renames the current name of a program with a new program name. If the new program name already exists, the RENAME operation is aborted and an error message displayed.

```
$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
```

## AS LANGUAGE COMMANDS

### 4.3.5 XFER AND COPY COMMANDS

**XFER**                    **new program, start step=old program, start step, number of steps**

Copies steps from one program to another (or within the same program) and inserts them before the specified start step. The old program steps are not replaced or deleted by this command.

new program:            Name of the program to copy the specified steps into.

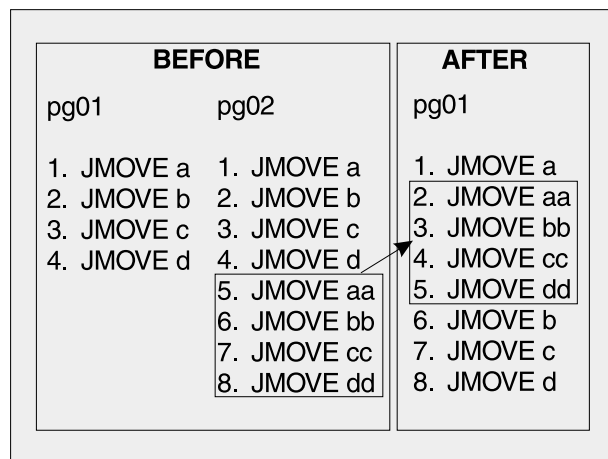
start step:              Step to insert specified steps before.

old program:            Name of the program to copy steps from.

start step:              Step to start coping from.

number of steps:        Number of steps to copy, including the start step.

Example:                XFER pg01,2=pg02,5,4



**COPY**                    **destination program name=source program name + source program name**

The copy command is used to copy a complete program or programs to a new program. The name of the destination program cannot be an existing program.

---

## AS LANGUAGE COMMANDS

### 4.4 PROGRAM AND DATA STORAGE COMMANDS

FORMAT	Initializes a PC card or a floppy disk.
FDIRECTORY	Displays the name of files stored on a PC card or a disk.
SAVE	Stores programs and variables in a specified file on a PC card or a disk.
SAVE/P	Stores programs in a specified file on a PC card or a disk.
SAVE/L	Stores locations in a specified file on a PC card or a disk.
SAVE/R	Stores real variables in a specified file on a PC card or a disk.
SAVE/S	Stores string variables in a specified file on a PC card or a disk.
SAVE/SYS	Stores system data in a specified file on a PC card or a disk.
SAVE/ELOG	Stores error log data in a specified file on a PC card or a disk.
LOAD	Loads programs and variables from a specified file into system memory.
LOAD/Q	Loads selected programs and variables from a specified file into system memory.
FDELETE	Deletes specified files on a PC card or a disk.

## AS LANGUAGE COMMANDS

### 4.4.1 FORMAT AND FDIRECTORY COMMANDS

**FORMAT** Initializes a PC card or a disk to accept files. The format command erases all data on a PC card or a disk. It also sets up a directory for keeping track of files as they are created. A new PC card or disk must be formatted before it can be used.

**FDIRECTORY** Displays names of files stored on the PC card or disk. The extensions indicate the type of file, such as system (AS), programs (PG), auxiliary (AU), location variables (LC), real variables (RV), weld data (WD), and string variables (ST). It also displays the size of the file, and the date and time the file was created.

```
$dir
File01.AS      22108   91-02-18   09:51
File02.AS       3678   91-02-18   09:59
File56.BPG     16108   91-02-23   11:30
File56.PG       108     91-02-29   11:51
ALL.AS        23576   92-02-15   12:24
Sample.PG      2500    92-04-25   07:43

869453 bytes free
$
```

### 4.4.2 SAVE COMMAND

**SAVE** **file\_name=program\_name**

Stores programs and variables in a specified file onto a PC card or a disk.

**file\_name:** Name of the file in which all programs, locations, and variables are stored.

**program\_name:** Name of the program stored in the specified file directory. If the name of the program is not specified, all data in memory is stored.

If a file with the same name already exists a "B" (backup) is added to the extension of the existing file. For example, if a file named TEST.AS exists on disk, and a backup is made to the PC card or disk using the same file name, the file extension is changed to TEST.BAS and a new file TEST.AS is created.

---

## AS LANGUAGE COMMANDS

**SAVE/P**                    **file\_name=program\_name**

Stores programs in a specified file on a PCcard or a disk.

file\_name:                Name of the file in which programs are stored.

program\_name:        Name of the program to store in the specified file directory. If the name of the program is not specified, all programs in memory are stored. If the file name has no extension, **PG** is added.

Example:                SAVE/P FENDER = pg01, weld, pg02 saves programs pg01, weld, and pg02 in the file FENDER.

**SAVE/L**                    **file\_name=program\_name**

Stores locations in a specified file on a PC card or a disk.

file\_name:                Name of the file in which locations are stored.

program\_name:        Name of the program to store in the specified file directory. If the name of the program is not specified, all locations from memory are stored. If the file name has no extension, **LC** is added.

Example:                SA/L #POS saves all precision points in the file #POS.

**SAVE/R**                    **file\_name=program\_name**

Stores real variables in a specified file on a PC card or a disk.

file\_name:                Name of the file in which real variables are stored.

program\_name:        Name of the program to store in the specified file directory. If the name of the program is not specified, all real variables from memory are stored. If the file name has no extension, **RV** is added.

Example:                SA/R REAL\_VAR saves all real values in the file REAL\_VAR.



---

## AS LANGUAGE COMMANDS

**SAVE/S**                    **file\_name=program\_name**

Stores character string variables in a specified file on a PC card or a disk.

file\_name:                Name of the file in which string variables are stored.

program\_name:          Name of the program store in the specified file directory. If the name of the program is not specified, all string variables from memory are stored. If the file name has no extension, **.ST** is added.

Example:                 S/S MESSAGES saves all string variables in the file MESSAGES.

**SAVE/SYS**                **file\_name**

Stores system data in a specified file on a PC card or a disk.

file\_name:                Name of the file in which system data is stored with an **.SY** extension.

**SAVE/ELOG**              **file\_name**

Stores system data in a specified file on a PC card or a disk.

file\_name:                Name of the file in which error log data is stored with an **.EL** extension.

### 4.4.3 LOAD AND FDELETE COMMANDS

**LOAD**                    **file\_name**

Loads programs and variables from a specified file into system memory.

file\_name:                Name of file to load from disk to memory. If an extension is not specified, **.AS** is assumed. All data in the specified file is loaded.

**LOAD/Q**                 **file\_name**

Loads selected programs and variables from a specified file into system memory. The user is prompted "Load this data? (1:Yes, 0:No, 2:Load all, 3:Exit)" before each type of data is loaded.

## AS LANGUAGE COMMANDS

**file\_name:** Name of file to load from the PC card to memory. If an extension is not specified, **.AS** is assumed. All selected data in the specified file is loaded.

An error message is displayed if a program in the file selected to load already exists in memory. If a variable is loaded from a file to memory and a variable of the same type and same name exists the old value is lost and the new value is stored. When system settings are loaded from a file the settings in memory are overwritten.

### **FDELETE**            **file\_name**

Deletes specified files from a PC card or a disk.

**file\_name:** Name of file to delete. When a file is deleted, its name and data are removed from the directory, and cannot be loaded from the PC card or disk into RAM.

**Example:** FDEL pg01.pg, deletes pg01. To delete all files from the PC card or disk enter FDEL \*.\*.

## 4.5 PROGRAM CONTROL

<b>SPEED</b>	Sets the monitor speed.
<b>PRIME</b>	Prepares a program for execution.
<b>EXECUTE</b>	Executes a robot control program.
<b>STEP</b>	Executes a single step of the program.
<b>MSTEP</b>	Executes a single robot motion step in the program.
<b>ABORT</b>	Stops execution after the current step is completed.
<b>HOLD</b>	Stops execution immediately.
<b>CONTINUE</b>	Resumes execution of the program.
<b>STEPNEXT</b>	Executes the next program step in step once mode.
<b>KILL</b>	Initializes the execution stack.
<b>DO</b>	Executes a single program instruction.

### 4.5.1 SPEED AND PRIME COMMANDS

#### **SPEED**            **monitor\_speed**

Sets the monitor (repeat conditions) speed in percentages. The range is from 0.01 to 100 percent. The new speed is not effective until the next robot motion. The robot speed is determined by the product of the monitor speed, and the program speed.

## AS LANGUAGE COMMANDS

**Example:** In program pg01 shown below, the program speed is 1,000 mm/s and the monitor speed is 50; therefore, the robot repeat speed is 500 mm/s (50 percent of 1000 mm/s). In program pg02, the program speed for steps 1 and 3 is SP9 and steps 2 and 4 is SP6, the monitor speed is 60; therefore, the robot repeat speed for steps 1 and 3 is 60 percent of SP9 and steps 2 and 4 is 60 percent of SP6.

\$list pg01	\$list pg02
.PROGRAM pg01()	.PROGRAM pg02()
1? speed 1000 mm/s alway	1 JOINT SP9 ACCU1 TIM0 OX WX
2? lmove aa	2 LINEAR SP6 ACCU1 TIM0 OX WX
3? lmove bb	3 JOINT SP9 ACCU1 TIM0 OX WX
4? lmove cc	4 LINEAR SP6 ACCU1 TIM0 OX WX
.END	.END
\$speed 50	\$speed 60

**PRIME**                    **program\_name, execution\_cycles, step\_number**

**program\_name:** The program name is optional. If omitted, the program specified by the last EXECUTE or PRIME command is selected.

**execution\_cycles:** Specifies the number of execution cycles. If omitted, one is assumed and the program executes once. If a negative number is entered, the program repeats continuously until 32,767 cycles are completed.

**step\_numberE**            The optional step number allows the user to specify the program step desired for beginning execution. If omitted, execution begins at the first executable step.

### NOTE

The PRIME command is used to prepare the system to execute a program. The PRIME command does not execute the program.

**Example:** prime prog01,5,3 puts prog01 on the stack. When the CYCLE START button is pushed, the command EXECUTE or CONTINUE is typed and entered; prog01 is executed five times starting at step 3.

---

## AS LANGUAGE COMMANDS

### 4.5.2 EXECUTE, STEP, AND MSTEP COMMANDS

**EXECUTE**            **program\_name, execution\_cycles, step\_number**

Executes a robot control program.

**program\_name:**    Name of the program to execute. If the program name is omitted, the current program is executed.

**execution\_cycles**    Specifies the number of execution cycles. If omitted, one is assumed and the program executes once. If a negative number is entered, the program repeats continuously until 32,767 cycles are completed.

**step\_number:**        Number of the step at which program execution is to begin. If omitted, execution begins at the first executable step.

**STEP**                **program\_name, repeat\_count, step\_number**

Executes one step of the program.

**program\_name:**    Name of program to execute. If the program name is omitted, the last executed program is selected.

**repeat\_count:**     Number of times execution of the program is repeated. If the count is omitted, it is set to one and the user may step through all the steps in the program only once. After the last step is executed, a "program completed" message is displayed and the step command becomes ineffective. To continue stepping through the program after the last step, the user must specify a repeat\_count greater than one.

**step\_number:**        Number of the program step to be executed. If all parameters are omitted, the next step is executed. The user can execute the desired step by typing all the parameters.

**Example:**            step pg01,1,5 executes step 5 of program pg01.

---

## AS LANGUAGE COMMANDS

**MSTEP**                    **program\_name, repeat\_count, step\_number**

Executes one robot motion instruction.

**program\_name:**        Name of the program to execute. If the program name is omitted, the current program is selected.

**repeat\_count:**        Number of times execution of the program is repeated. If the count is omitted, it is set to one and the user may step through all the steps in the program only once. After the last step is executed, a “program completed” message is displayed, and the step command becomes ineffective. To continue stepping through the program after the last step, the user must specify a **repeat\_count** greater than one.

**step\_number:**        Number of the program step to execute. If all parameters are omitted, the next step is executed.

**Example:**            If the robot is currently at step 4, the MSTEP command executes the two non-motion steps, SIGNAL 1 and DELAY 5, then the motion step 7 jmove bb.

```
4? JMOVE aa
5? SIGNAL 1
6? DELAY 5
7? JMOVE bb
```

## AS LANGUAGE COMMANDS

### 4.5.3 ABORT, HOLD, CONTINUE, AND STPNEXT COMMANDS

**ABORT** Stops execution of the robot control program. Execution is terminated after the current step is completed. If the robot is in motion, it is terminated after completion of the current motion. Program execution is resumed using the CONTINUE command.

**HOLD** Stops execution of the robot control program immediately. Execution is terminated and motor power remains ON. Program execution is resumed using the CONTINUE command.

**CONTINUE** **next**

Resumes execution of the robot control program terminated by the PAUSE, ABORT or HOLD commands, or as a result of an error. This command can also be used to initiate programs ready to be executed by use of the PRIME, STEP, or MSTEP commands.

**next** Optional argument “next” specifies execution starts from the next step. If “next” is omitted, execution starts from the current step.

The WAIT, SWAIT, and TWAIT commands can be skipped by using the CONTINUE next command.

**Example:** In program pg01, the operator can skip step 6 by typing CONTINUE next. In program pg02, step 4 cannot be skipped.

pg01	pg02
5? JMOVE cc	3? JMOVE aa
6? SWAIT 1001	4? DELAY 60
7? JMOVE dd	5? JMOVE bb

**STPNEXT** When REPEAT CONDITION, STEP CONT/STEP ONCE is set to STEP ONCE, the STPNEXT command is used to advance to the next step of the program.

---

## AS LANGUAGE COMMANDS

### 4.5.4 KILL AND DO COMMANDS

**KILL** Initializes the stack of the robot control program. If the program is stopped by the PAUSE or ABORT commands, or an execution error, the program stack is unchanged. Once the KILL command is executed, the CONTINUE command is ineffective since there is no longer a program on the stack. The KILL command can only be issued when program execution is stopped.

**DO** **instruction**

Executes a single program instruction from the monitor prompt, without creating a program. If the instruction is omitted, the DO command repeats the last instruction.

Example: > do jmove aa

Moves the robot in joint interpolated motion to location aa.

### 4.6 DEFINING LOCATIONS, LIMITS, AND HOME POSITIONS

HERE	Defines a location variable as the current robot location.
TEACH	Sets location values to a series of location variables.
POINT	Defines a location variable.
POINT/X	Sets the value of X component.
POINT/Y	Sets the value of Y component.
POINT/Z	Sets the value of Z component.
POINT/OAT	Sets the value of OAT components.
POINT/7	Sets the value of seventh axis component.
TOOL	Defines the location and direction of the tool tip.
BASE	Changes the robot base coordinate system.
ULIMIT	Sets the upper limit of the robot motion.
LLIMIT	Sets the lower limit of the robot motion.
SETHOME	Sets the home position.
WHERE	Displays the current robot position.

## AS LANGUAGE COMMANDS

### 4.6.1 HERE, TEACH, AND POINT COMMANDS

#### HERE `location_variable`

The HERE command defines the `location_variable` as the current robot location. The location variable defined with the HERE command can be a transformation location, precision location, or a compound transformation location. Entering the HERE command displays the location coordinates (XYZOAT) or joint angles, and the prompt asking the user if a change to the information is desired. If the displayed information is acceptable, the ENTER key is pressed to complete the storage of the location information. If the user desires to change any components, the new information is typed and the ENTER key is pressed.

Precision locations are defined by preceding the location name with a “#” symbol. Compound transformations are defined using the “+” symbol to make the location on the far right of the expression relative to the location on the left. If a transformation variable in the compound expression is not defined, an error occurs.

```
$here fixt.1
  X[mm]   Y[mm]   Z[mm]   O[deg]   A[deg]   T[deg]
  -130.3  390.49  -154.90  163.31  89.56   -2.65
CHANGE? (if not, hit RETURN only)
```

```
$here #fixt.1
      JT1 [deg]  JT2 [deg]  JT3 [deg]  JT4 [deg]  JT5 [deg]  JT6 [deg]
      25.657   18.490   -23.900  156.316   89.080   -2.650
CHANGE? (if not, hit RETURN only)
, , , ,90
      JT1 [deg]  JT2 [deg]  JT3 [deg]  JT4 [deg]  JT5 [deg]  JT6 [deg]
      25.657   18.490   -23.900  156.316   90.000   -2.650
CHANGE? (if not, hit RETURN only)
```



## AS LANGUAGE COMMANDS

### TEACH **location\_variable name**

Starts recording a series of locations with the name of an array or location variable each time the REC key on the multi function panel (MFP) is pressed. The teaching mode is disabled by pressing the RETURN key. Each time the REC key on the MFP is pressed, the current robot location is recorded in the specified variable and its name is issued a number beginning with zero, or a number the operator selects. Each additional specified location is then recorded and numbered consecutively.

```
$teach welda1
Go to teach Pendant
Location name : welda0
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
-140.4  509.90  -85.00  178.323  178.768  179.870

Location name : welda1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
-125.4  509.90  -85.00  178.530  178.768  -179.929
```

### POINT **location\_variable = location\_value**

**location\_variable:** Location variable name (precision variable, transformation variable, or compound transformation expression) to define.

**location\_value:** Existing location value of same type as **location\_variable**.

Defines the specified location variable value to equal the location on the right. The left location variable and the right location value must be the same type (transformation values or precision values). If the right location is not designated, and the left variable is defined, its component values are displayed and can be changed. If the left variable has not been defined (when the right location is omitted), all component values displayed are zero.

## AS LANGUAGE COMMANDS

Pressing the RETURN key sets all component values and the message “Change?” is displayed. At this point the user can change the values. If a precision variable is defined, its component values are displayed in joint angles. For transformation variables, values are displayed in XYZOAT.

```
$point welda2
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  Z[deg]
  0.000  0.000  0.000  0.000  0.000  0.000
CHANGE? (if not, hit RETURN only)

$point welda2=welda1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  Z[deg]
 -125.4  509.90 -85.00  178.530  178.768 -179.929
CHANGE? (if not, hit RETURN only)

$point welda2
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  Z[deg]
 -125.4  509.90 -85.00  178.530  178.768 -179.929
CHANGE? (if not, hit RETURN only)
```

## AS LANGUAGE COMMANDS

POINT/X  
POINT/Y  
POINT/Z  
POINT/OAT  
POINT/O  
POINT/A  
POINT/T  
POINT/7

**transformation\_variable = transformation\_value**

Sets the component (X, Y, Z, O, A, T or JT7) of the location on the left-hand side to be equal to the location variable on the right-hand side.

transformation\_  
variable:

Name of a transformation variable for which a value is set.

transformation\_  
value:

Transformation value from which the component value is obtained.

```

$point welda3
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
    0.000  0.000  0.000  0.000  0.000  0.000
CHANGE? (if not, hit RETURN only)
$point/x welda3 = welda1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
  -125.4  0.000  0.000  0.000  0.000  0.000
CHANGE? (if not, hit RETURN only)
$point/y welda3 = welda1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
  -125.4  509.90  0.000  0.000  0.000  0.000
CHANGE? (if not, hit RETURN only)
$point/z welda3 = welda1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
  -125.4  509.90  -85.00  0.000  0.000  0.000
CHANGE? (if not, hit RETURN only)
$point/oat welda3 = welda1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]
  -125.4  509.90  -85.00  178.530  178.768  -179.929
CHANGE? (if not, hit RETURN only)

```

```

$point weld1
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]  JT7[mm]
    0.000  0.000  0.000  0.000  0.000  0.000  0.000
CHANGE? (If not, hit RETURN only)
$Point/7 weld1 = weld2
  X[mm]  Y[mm]  Z[mm]  O[deg]  A[deg]  T[deg]  JT7[mm]
    0.000  0.000  0.000  0.000  0.000  0.000  120.000
CHANGE? (If not, hit RETURN only)

```

## AS LANGUAGE COMMANDS

### 4.6.2 TOOL AND BASE COMMANDS

#### TOOL **transformation\_value**

Defines the location and direction of the tool center point relative to the tool mounting flange of the robot.

transformation\_  
value:

Transformation value must be a previously defined variable or a compound transformation. If omitted, the current tool transformation value is displayed for modification.

If NULL is designated for transformation\_value, tool transformation value is set to "null tool". "Null tool" has its center on the tool mounting flange surface, and has its coordinate axes parallel to the respective coordinate axes of the last joint of the robot. ("Null tool" is represented by the transformation value [0, 0, 0, 0, 0, 0]). At the system initialization, tool transformation value is set to the null tool automatically.

After the tool transformation value is set, the values (XYZOAT) are displayed on the screen, and the message "change?" is displayed. To change component values, enter new values separated by commas and press the ENTER key.

```

$point t
  X[mm] Y[mm] Z[mm] O[deg] A[deg] T[deg]
  0.000 0.000 0.000 0.000 0.000 0.000
CHANGE? (if not, hit RETURN only)
,,315
  X[mm] Y[mm] Z[mm] O[deg] A[deg] T[deg]
  0.000 0.000 315.00 0.000 0.000 0.000
CHANGE? (if not, hit RETURN only)

$tool t
  X[mm] Y[mm] Z[mm] O[deg] A[deg] T[deg]
  0.000 0.000 315.00 0.000 0.000 0.000
CHANGE? (if not, hit RETURN only)
-1.4,3.2,,-80.6,41.82,95.2
  X[mm] Y[mm] Z[mm] O[deg] A[deg] T[deg]
  -1.400 3.200 315.00 -80.200 41.820 95.200
CHANGE? (if not, hit RETURN only)

```

## AS LANGUAGE COMMANDS

When an instruction to move the robot to a transformation location is executed, or when the robot is moved in teach mode, using base or tool coordinate, the tool transformation value is used in calculating the robot motion path and robot configuration.

If the transformation value specified as the argument of the TOOL command is modified after the TOOL command is issued, the change does not affect the robot motion until another TOOL command is issued.

### BASE

#### transformation\_value

Transformation or compound value that defines the base coordinates. If omitted, the current base transformation value is displayed for modification.

If NULL is designated for the parameter, the base value is set as “null base”. (“Null base” is indicated by the transformation value [0, 0, 0, 0, 0, 0].) When the system is initialized, the base transformation value is set as null base automatically.

After a new base transformation is set, the values (XYZOAT) and the message “Change?” is displayed. To change values, enter the new values separated by commas and press the ENTER key. If the parameter is omitted, the current value and the message “Change?” is displayed.

```
$base null
  X[mm] Y[mm] Z[mm] O[deg] A[deg] T[deg]
  0.000 0.000 0.000 0.000 0.000 0.000

CHANGE? (if not hit RETURN only)
  50,100,10
  X[mm] Y[mm] Z[mm] O[deg] A[deg] T[deg]
  50.000 100.000 10.000 0.000 0.000 0.000

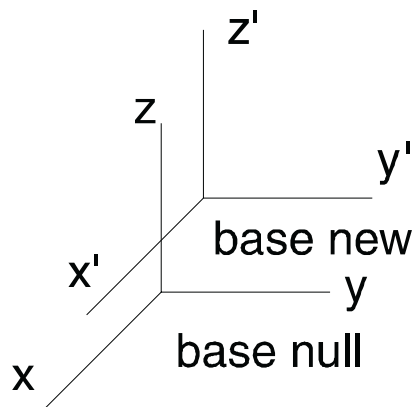
CHANGE? (if not hit RETURN only)
```

## AS LANGUAGE COMMANDS

The base coordinate system, defined by a transformation value, is the reference for robot motion to transformation locations or when the robot is moved manually in base coordinate mode. Changing the base coordinate system with the BASE command affects all locations recorded as transformations. After the BASE command is issued, if the transformation value used as the argument is changed, robot motion is not affected until the BASE command is issued again.

The BASE command has no effect on locations defined by a precision variable. The argument of the BASE command, or the BASE instruction, indicates the robot's displacement from the origin of the base coordinate system of the robot.

For example, the BASE command can be used when a fixture or existing part is relocated. Instead of changing all existing programmed points, the BASE command defines a new coordinate system in relation to the relocated fixturing or part.



## AS LANGUAGE COMMANDS

### 4.6.3 ULIMIT AND LLIMIT COMMANDS

**ULIMIT**                    **precision\_value** (joint angle)

Defines the upper limit (software limit) of robot range of motion .  
The Maximum line shows the maximum allowable limit for each joint. The second line shows the current software limit of each individual joint. Changes are made using the keyboard to enter new numeric values. The new values are separated by commas. Press the ENTER key to get back into the monitor mode.

\$ulimit	JT1	JT2	JT3	JT4	JT5	JT6
Maximum	160.000	140.000	150.000	270.000	180.000	360.000
Current	85.000	60.000	30.000	190.000	115.000	270.000
Change? (If not, hit RETURN only)						

**LLIMIT**                    **precision\_value** (joint angle)

Defines the lower limit (software limit) of robot range of motion .  
The Maximum line shows the maximum allowable limit for each joint. The second line shows the current software limit of each individual joint. Changes are made using the keyboard to enter new numeric values (ensure the negative sign {-} is included). The new values are separated by commas. The user must press the ENTER key to get back into the monitor mode.

\$llimit	JT1	JT2	JT3	JT4	JT5	JT6
Maximum	-160.000	-120.000	-150.000	-270.000	-180.000	-360.000
Current	-85.000	-50.000	-90.000	-190.000	-115.000	-270.000
Change? (If not, hit RETURN only)						

## AS LANGUAGE COMMANDS

### 4.6.4 SETHOME COMMAND

**SETHOME**            **accuracy, HERE**  
**SET2HOME**        **accuracy, HERE**

**accuracy:**            The acceptable range of the HOME position (in millimeters). If omitted it is assumed to be one (1) millimeter. This range determines when the dedicated outputs for the HOME positions turn on.

**HERE:**                If the argument HERE is specified, the HOME position is changed to the current robot joint positions. If the argument HERE is omitted, the current HOME value is displayed. In either case, the query "Change?" appears. To change components, enter new values separated by commas. Press the ENTER key to terminate the command.

```

$sethome 100
JT1   JT2   JT3   JT4   JT5   JT6   Accuracy
0.000 0.000 0.000 0.000 0.000 0.000   100.0
CHANGE ? (if no, hit RETURN only)
, , , ,-90 , , 50

JT1   JT2   JT3   JT4   JT5   JT6   Accuracy
0.000 0.000 0.000 0.000 -90.000 0.000   50.0
CHANGE ? (if no, hit RETURN only)

```



---

## AS LANGUAGE COMMANDS

### 4.6.5 WHERE COMMAND

**WHERE**            **display\_mode**

Displays the current robot location. If the display mode is omitted, the current location is displayed. When a display mode is specified, the current values are displayed continuously until the ENTER key is pressed.

display\_mode:

**WHERE**            Displays the current robot location in both joint angles and base coordinates (XYZOAT).

**WHERE 1**        Displays the current location in joint angles.

**WHERE 2**        Displays the current location in base coordinates (XYZOAT) (mm, deg).

**WHERE 3**        Displays the current instructed values (deg).

**WHERE 4**        Displays deviations from the instructed values (bit).

**WHERE 5**        Displays encoder value of each joint (bit).

**WHERE 6**        Displays speed of each joint (deg/sec).

The display on the next page (Figure 4-2) depicts the terminal screen when the WHERE command is issued. WHERE1 through WHERE6 have a continuously scrolling screen and display the current information.

### AS LANGUAGE COMMANDS

\$w						
	JT1	JT2	JT3	JT4	JT5	JT6
	11.998	12.735	-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	110.9811	158.083	93.324
\$w1						
	Joint value					
	JT1	JT2	JT3	JT4	JT5	JT6
	11.998	12.735	-49.549	27.190	-24.714	128.959
	⋮	⋮	⋮	⋮	⋮	⋮
\$w2						
	Transformation value					
	-270.379	1424.420	525.474	110.9811	158.083	93.324
	⋮	⋮	⋮	⋮	⋮	⋮
\$w3						
	Joint command					
	11.998	12.735	-49.549	27.190	-24.714	128.959
	⋮	⋮	⋮	⋮	⋮	⋮
\$w4						
	Joint position error					
	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮
\$w5						
	Joint encoder value					
	268464606	268487546	2681427546	268576237	268577166	268840020
	⋮	⋮	⋮	⋮	⋮	⋮
\$w6						
	Joint speed					
	JT1	JT2	JT3	JT4	JT5	JT6
	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮

Figure 4-2 Where Command Display Screen

## AS LANGUAGE COMMANDS

### 4.7 SYSTEM INFORMATION

ERRLOG	Displays a history of error conditions.
OPLOG	Displays a history of operations performed by the operator.
STATUS	Displays status information.
FREE	Displays amount of free memory.
ID	Displays robot model information and software version
HELP	Displays AS Language commands available to the operator

#### 4.7.1 ERRLOG AND OPLOG COMMANDS

**ERRLOG** The ERRLOG command displays a history of error conditions that are stored in system memory. A history of the last one thousand errors is retained. The display unit displays ten errors at a time starting from the most recent to the oldest. The user can alternate between the next screen and the previous screen by using function keys F4 and F5. The format of the error log is shown below:

<u>date</u>	<u>time</u>	<u>error code</u>	<u>error message</u>
12-01	15:40	(-203)	XXXXXXXXXXXXXXXXXXXXXXXXXX
12-01	15:23	(-600)	XXXXXXXXXXXXXXXXXXXXXXXXXX

To see additional ERRLOG or OPLOG errors, press the space bar or the F5 function key. To terminate the command, press the ENTER key, or the EXIT key on the keyboard.

**OPLOG** The OPLOG command displays a history of the last one hundred operations performed by the operator, and any messages related to operations. Like the ERRLOG command, ten operations or messages are displayed from the most recent to the oldest. To see the remaining operations, press the space bar or the F5 function key. To terminate the command, press the ENTER key, or the EXIT key on the keyboard. The format of the oplog is shown below:

<u>date</u>	<u>time</u>	<u>operation message</u>
12-01	15:54	where
12-01	15:40	ED DEMO
12-01	15:00	System Data Home-1 Set

## AS LANGUAGE COMMANDS

### 4.7.2 STATUS, FREE, ID, AND HELP COMMANDS

**STATUS** Displays status of the system and the current robot control program.

The status information is displayed in the following format:

Robot status: <status>  
Repeat speed: <speed>  
Total cycles  
Completed cycles: <cycles completed>  
Remaining cycles: <remaining cycles>  
Program name Step No.  
<name> <number>

```
$sta
Robot status:
Motor Power OFF
REPEAT mode

Environment:
Monitor speed (%) = 10.0
Program speed (%) ALWAYS = 100.0
ALWAYS Accu.[mm] = 1.0
Stepper status: Program is not running.
Execution cycles
Completed cycles: 3
Remaining cycles: Infinite
Program name Prio Step No.
test 0 1 WAIT sig(1001)
```

- **Robot status** <status>

The current robot status is one of the following:

Error state: An error has occurred; try the error reset operation.

Motor power off: Motor power is OFF.

Teach mode: Motor power is ON; the robot is controlled using the MFP.

## AS LANGUAGE COMMANDS

- Check mode: Motor power is ON; the robot is in the check mode.
- Repeat mode: Motor power is ON; the robot is controlled by the robot control program.
- Program running: Motor power is ON; the robot control program is running.
- Program waiting: Motor power is ON; the robot control program is running and in a wait condition (executing a WAIT, SWAIT, or TWAIT instruction).

- **Repeat speed** <speed>  
The current monitor speed (in percentages).
- **Completed cycles** <cycles completed>  
Execution cycles already completed.
- **Remaining cycles** <remaining cycles>  
Remaining execution cycles. If a “-1” execution cycle was specified in the EX-ECUTE command, “infinite” is displayed.

**FREE** Displays the size of memory area not currently used, both in percentages and in bytes.

```
$free
Total memory. 490584
Available memory size. 458680 bytes (93%)
```

**ID** Displays robot identification and software version information.

```
$ID
Robot name : UX120-E001    Num of axes 6    Serial No.1
Software version : version 23011GVE ... 99/08/30 16:10
Servo : SAOA00-UX120-01
Number of signals : output = 128 input = 128 internal = 256
Clamp number : 2 MOTION TYPE : 1 SERVO TYPE : 1
```

## AS LANGUAGE COMMANDS

### HELP

Displays the AS Language commands available to the operator. When the HELP command is entered followed by a space and a letter, all commands that begin with that letter are displayed.

Variations of the HELP command:

HELP/M	monitor commands
HELP/P	program instructions
HELP/F	function commands
HELP/PPC	PC program commands
HELP/MC	monitor commands
HELP/DO	DO 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 INPUT					
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.						

## AS LANGUAGE COMMANDS

### 4.8 SYSTEM CONTROL

SYSINIT	Initializes the entire system.
TIME	Sets the date and time.
ERESET	Resets the error condition.
SWITCH	Displays the system switch settings.
ON	Enables the system switches.
OFF	Disables the system switches.
HSETCLAMP	Sets the default clamp specifications.
ZSIGSPEC	Sets and displays the total number of I/O signals.
ZZERO	Displays or sets the zeroing data.
BATCHK	Enables/disables battery low voltage check.
ENCCHK_EMG	Sets the range to check position variance after an E-stop, when power is reapplied.
ENCCHK_PON	Sets the range of encoder deviation, for error display.
SLOW_REPEAT	Sets slow repeat mode speed.
REC_ACCEPT	Enables/disables recording and or changing programs.
ENV_DATA	Sets auto servo off timer and teach pendant connect/disconnect.
ENV2_DATA	Sets multi function panel and terminal connect/disconnect.
CHSUM	Clears check sum error.

#### 4.8.1 SYSINIT, TIME, AND ERESET COMMANDS

**SYSINIT** The system is initialized, and all programs, location variables, real variables and character string variables are erased. All system parameters under the DATA SET key (speed, accuracy, timers, system switches, etc.) are reset to default conditions. The **ERRLOG** and **OPLOG** data are not affected by this command.

**TIME** **date\_and\_time**

Sets the date and time. The format is yy-dd-mm and hh:mm:ss.

If the date and time are set by this command, the internal system variable used to set the present time is changed.

The respective element value allowed is as follows:

yy	year	(00-99)
dd	day	(01-31)
mm	month	(01-12)
hh	hours	(00-23)
mm	min.	(00-59)
ss	sec.	(00-59)

## AS LANGUAGE COMMANDS

If the parameters are omitted, the present time is displayed and the system awaits a change. To make a change, enter the present time or date in the format described above. To terminate the command, press the ENTER key.

**ERESET** Resets the current error condition (the same function as the ERROR RESET button on the control panel). The ERESET command is ineffective when an error occurs continuously.

### 4.8.2 SWITCH, HSETCLAMP, AND ZSIGSPEC COMMANDS

**SWITCH**            **switch\_name,...ON**  
                         **switch\_name,...OFF**

Displays (or changes) system switch settings.

switch\_name:        Name of a system switch to display or change. If omitted, all switch settings are displayed.

ON or OFF            If the argument ON is specified, the switches are enabled; if OFF is specified, switches are disabled. If omitted, the switch status is displayed without changing.

```
$switch cp, screen, messages, arc = on
$cp, screen, messages, arc off
```

**HSETCLAMP**        Assigns signal numbers to operate material handling clamps.

Additional clamp data is set in auxiliary function 114.

\$HSETCLAMP	CLAMP 1	CLAMP 2	CLAMP 3	CLAMP 4
	Spot weld	Handling	Not used	Not used
'ON' out. signal	24	24	24	24
'OFF' out. signal	0	0	0	0
	CLAMP 5	CLAMP 6	CLAMP 7	CLAMP 8
	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)				



## AS LANGUAGE COMMANDS

### ZSIGSPEC

This command sets and displays the default setting for the total number of input and output signals installed in the controller.

To use this command, type in ZSIGSPEC at the system prompt, and press the ENTER key. Type in the number of I/O signals, and press the ENTER key.

To display the current settings, type the ZSIGSPEC command, and press the ENTER key. To exit the ZSIGSPEC command without changing the data press the ENTER key again.

```
$zsigspec
  DO.  DI.  INT.
  128  128  256
Change? (If not, Press RETURN only.)
$
```

### 4.8.3 ZZERO COMMAND

#### ZZERO

**joint\_number**

The ZZERO command is used to set the encoder rotation count and to display or set the robot's zeroing data. The encoder rotation count is stored in the encoder and indicates the encoder revolution count from the zero position. The zeroing data is an absolute encoder value stored in memory that coincides with the mechanical zero position of each joint.

joint\_number:

When the ZZERO command is used to view or set the robot's zeroing data, the joint number specifies a specific joint. Entering a 0 indicates the zeroing data for all joints is set, while 1 through 7 indicate the zeroing data for the individual joint is set.

```
$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      65532      65510
Change? (If not, hit RETURN only)
```

## AS LANGUAGE COMMANDS

### NOTE

The robot axes can be zeroed individually but may require other axes to be positioned at zero. Refer to the C Series Controller Electrical Maintenance Manual, unit 8, Zeroing, for axes that must be positioned simultaneously before zeroing.

The first step in the zeroing process is to jog the robot to the position where the scribe marks, for the joint(s) to zero, are aligned. If the robot does not move because the current position is recognized as out of range, the robot is zeroed where it is, to allow jogging operations, and then rezeroed when the scribe marks are aligned. The next step in the zeroing process is to set the current encoder count offset to a 0° midpoint reference, this is done with the ZZERO 10\_ command. Two examples of the ZZERO 10\_ command, ZZERO 100 for all joints, and ZZERO 102 for joint 2 are shown below.

```
$ZZERO 100
**Encoder rot. counter reset (all 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 completed.
$
```

After the encoder rotation counter is reset, the ZZERO \_ command is used to set the joint(s) to 0°. An example of the ZZERO \_ command, ZZERO 0 for all joints is shown below.

```
$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 current values of all joints as zeroing data? (Enter 1 to set) 1
Setting completed.
$
```

## AS LANGUAGE COMMANDS

### 4.8.4 BATCHK, ENCCHK\_EMG, AND ENCCHK\_PON COMMANDS

**BATCHK** The BATCHK command is used to enable or disable the battery low voltage check at controller power up.

The batteries are attached to the card rack and connected to the 1HZ board. The batteries provide backup power for SRAM memory of data stored on the 1GA CPU board.

When the BATCHK command is entered, the screen prompts the user to enter “0:Ineffect” (battery check not performed) or “1:Effect” (battery check performed at power up).

**ENCCHK\_EMG** The ENCCHK\_EMG command is used to set a comparison range to check the robot’s position at an emergency stop versus the position when motor power is reapplied.

If the difference in positions exceeds the set value, a position offset error is displayed. 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 does not cause an error.

The purpose of this function is to prevent interference with fixtures, jigs, or the work piece when the robot is restarted after an emergency stop.

The range of data for the ENCCHK\_EMG command is 0.001 degree to 10.000 degrees for axes one to six and 0.001 mm to 100.000 mm for a seventh axis.

The default setting is 0 (error check is not performed).

\$ENCCHK_EMG						
	JT1	JT2	JT3	JT4	JT5	JT6
	0.000	0.000	0.000	0.000	0.000	0.000
Change? (If not, hit return only)						
\$						

## AS LANGUAGE COMMANDS

**ENCCHK\_PON** The ENCCHK\_PON command is used to set the range of encoder deviation allowed before an error is displayed at controller power up.

The encoder value at controller power down is compared to the encoder value at controller power up. If the difference is larger than the range set, a JT encoder abnormality error is displayed.

The range of data is from 0.001 degree to 10.000 degrees for axes one to six and from 0.001 mm to 100.000 mm for a seventh axis.

The default setting 2.0 degrees.

If the setting is too low, error messages may be displayed when the system is performing within design performance specifications.

```
$ENCCHK_PON
  JT1    JT2    JT3    JT4    JT5    JT6
  2.000  2.000  2.000  2.000  2.000  2.000
Change? (If not, hit return only)
$
```

### 4.8.5 SLOW\_REPEAT, REC\_ACCEPT, ENV\_DATA, AND ENV2\_DATA COMMANDS

**SLOW\_REPEAT** The SLOW\_REPEAT command is used to set the SLOW\_REPEAT mode speed of the robot from 1 to 25% of maximum speed.

A dedicated input signal must be assigned for the SLOW\_REPEAT mode function.

When this signal is ON, the robot operates at the speed set with the SLOW\_REPEAT command.

```
$SLOW_REPEAT
SLOW REPEAT MODE (1-25%)
(Enter only: No change ^C:Exit): Now      10 Change ?
```

## AS LANGUAGE COMMANDS

**REC\_ACCEPT** The REC\_ACCEPT command is used to set the status for entering new data.

When the REC\_ACCEPT command is entered, the display prompts the user to enable or disable the RECORD or PROGRAM CHANGE functions.

The RECORD option allows the user to prevent the recording of blockstep information by selecting disable.

When RECORD is disabled, blockstep program data cannot be changed and the error message "Set to RECORD ACCEPT" is displayed.

The PROGRAM CHANGE option allows the user to prevent the recording of AS Language information by selecting disable.

When PROGRAM CHANGE is disabled, AS Language program data cannot be changed and the error message "Program change inhibited. Set ACCEPT and operate again." is displayed.

In the example, the RECORD function was changed from disabled to enabled.

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

## AS LANGUAGE COMMANDS

### ENV\_DATA

The ENV\_DATA command is used to set the auto servo timer and identify if a teach pendant is installed.

When the ENV\_DATA command is entered, the display prompts 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 remains ON if no movement of the robot has occurred. The auto servo timer function is designed to save energy by using the brakes to maintain robot position.

If the AUTO SERVO OFF TIMER is not used electrical power and servo motors maintain robot position.

When the robot has not moved and the auto servo timer reaches its set value, the brakes are applied and servo motor power is removed.

The motor power light remains ON and the robot begins motion under the same conditions as if the auto servo timer did not remove power from the motors.

The ENV\_DATA command is used to identify if a TEACH PENDANT is connected.

The deadman buttons and the emergency stop button are hard-wired and a jumper (or a different user interface) must be installed if the teach pendant is disconnected.

In the example, the AUTO SERVO OFF TIMER is set to 600 seconds and a TEACH PENDANT is installed.

```
$ENV_DATA
AUTO SERVO OFF TIMER (0:Servo not off)
  (Enter only: No change ^C:Exit) : Now    0 Change? 600
TEACH PENDANT (0:Connect, 1:Disconnect)
  (Enter only: No change ^C:Exit) : Now    0 Change?
$
```

## AS LANGUAGE COMMANDS

### ENV2\_DATA

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 are hard-wired and a jumper (or a different user interface) must be installed if the multi function panel is removed.

The example below shows the display when the ENV2\_DATA command is entered.

```
$ENV2_DATA
PANEL (0:Connect, 1:Disconnect)
  (Enter only: No change ^C:Exit) : Now    0 Change?
TERMINAL (0:Connect, 1:Disconnect)
  (Enter only: No change ^C:Exit) : Now    0 Change?
$
```

### 4.8.6 CHSUM COMMAND

#### CHSUM

The CHSUM command is used when an abnormal check sum error (1019) is generated because the processor has calculated a difference between data when the controller was powered up compared to an expected value.

When this error occurs the programmer enters the CHSUM command and changes the CLEAR CHECK SUM ERROR setting to "EFFECT". With this setting, when controller power is cycled, the check sum error is cleared and the setting returns to "INEFFECT".

If the check sum error does not clear after cycling controller power, enter the CHSUM command again (cycle controller power) as shown in the example below. If the check sum error does not clear a message (as shown below) is displayed identifying additional troubleshooting.

```
$CHSUM
CLEAR CHECK SUM ERROR (0:Ineffect, 1:Effect)
  (Enter only: No change ^C:Exit) : Now    0 Change?
$CHSUM
Cannot clear sum check error. Check the following command or auxiliary data.
ZZERO
DEFSIG
:
:
```

---

## AS LANGUAGE COMMANDS

### 4.9 SIGNAL COMMANDS

SIGNAL	Turns ON (or OFF) signals.
PULSE	Turns ON a signal during the specified period of time.
DYLSIG	Turns ON a signal after the specified time has elapsed.
BITS	Sets a group of signals to be equal to the specified value.
IO	Displays I/O signal status.
RESET	Resets all external output signals.
DEFSIG	Sets or cancels signals for particular uses.

#### 4.9.1 SIGNAL, PULSE, DLYSIG, AND BITS COMMANDS

##### **SIGNAL**                    **signal\_number**

Sets external output signals or internal status signals to ON/OFF. Represents the number of an external output signal or an internal status signal. If this value is positive, the signal is turned ON; if negative, the signal is turned OFF.

signal\_number:        The signal number designates whether the signal is an external output or internal status signal. For external output signals, the standard number is from 1 to 32. Additional hardware is installed to increase the number to a maximum of 256. Internal status signal numbers range from 2001 to 2256. Input signals cannot be designated.

If the signal number is positive, the signal is turned ON; if negative, the signal is turned OFF. If "0" is given, no output signals are changed.

Multiple signal numbers are separated with comas, as shown in the example below.

Example:                SIG 4,5,-8,-10 turns outputs 4 and 5 ON and outputs 8 and 10 OFF.



---

## AS LANGUAGE COMMANDS

### **PULSE**                    **signal\_number, time (seconds)**

Turns ON the specified signal (external output or internal status) for the given period of time only.

**signal\_number:** Represents the number of the signal to turn on for the specified time. Numbers greater than 32, or negative numbers cannot be designated (0-32).

**time:** The period of time the signal is kept ON (in seconds). If omitted, 0.2 seconds is assumed.

**Example:** PULSE 4,3.56 turns output 4 ON for 3.56 seconds.

### **DLYSIG**                    **signal\_number, time (seconds)**

Turns the specified signal ON or OFF after a given time has elapsed.

**signal\_number:** Represents the number of an external output signal or an internal status signal. If this value is positive, the signal is turned ON; and if negative, the signal is turned OFF. Acceptable signal numbers are from 1 to 32, and from 2001 to 2032.

**time:** Time period after which the signal is turned ON or OFF (in seconds).

**Example:** DLYSIG 2,4.5 turns output 2 ON after 4.5 seconds has elapsed, where as DLYSIG -2,4.5 turns output 2 OFF after 4.5 seconds has elapsed.

## AS LANGUAGE COMMANDS

**BITS**                    **starting\_signal\_number, number\_of\_signals = value**

Sets a group of external output signals (or internal status signals) according to the given value. If the value is not designated, the current signal states are displayed.

starting\_signal\_  
number:                    The first signal to be set.

number\_  
of\_signals:                Number of signals (number of bits); maximum number allowed is sixteen.

value:                    The value which represents the desired signal states. If the binary notation of this value has more bits than the number of signals to be set, only the number of signals (specified by number\_of\_signals) from the smallest one (specified by starting\_signal\_number) are affected. If omitted, the current signal states are displayed in decimal notations.

This command turns ON/OFF one or more signals of external output or input status, according to the given value.

```
$bits 1,4=10
$io
 32-   1 1010 1010 0100 0100 0000 0000 0000 1010
1032-1001 0000 0000 0000 0000 0000 0000 0000 0000
2032-2001 0000 0000 0000 0000 0000 0000 0000 0000
$
```

### 4.9.2 I/O AND RESET COMMANDS

**IO**                        Displays the current states of all external and internal I/O signals. With the DISPIO\_01 system switch ON a “1” is displayed for the signals in the ON/HIGH state, and a “0” is displayed for the signals in the OFF/LOW state.

With the DISPIO\_01 SYSTEM switch OFF a “o” is displayed for the signals in the ON/HIGH state, and an “x” is displayed for the signals in the OFF/LOW state. An uppercase “O” or “X” indicates a dedicated signal.

## AS LANGUAGE COMMANDS

The signal numbers are displayed from right to left. The signal states are continuously updated until the RETURN key is pressed.

When IO or IO 1 is entered, the states of signals 1-32, 1001-1032, and 2001-2032 are displayed as shown below.

```
$io
 32-  1 1010 1010 0100 0100 0000 0000 0000 1010
1032-1001 0000 0000 0000 0000 0000 0000 0000 0000
2032-2001 0000 0000 0000 0000 0000 0000 0000 0000
$
```

When IO 2 is entered, the states of signals 33-64, 1033-1064, and 2033-2064 are displayed as shown below.

```
$io 2
 64- 33 1010 1010 0100 0100 0000 0000 0000 1010
1064-1033 0000 0000 0000 0000 0000 0000 0000 0000
2064-2033 0000 0000 0000 0000 0000 0000 0000 0000
$
```

When IO 3 or IO 4 is entered, the states of signals 65-96, 1065-1096, 2065-2096 or 97-128, 1097-1128, 2097-2128 are displayed.

If a controller is not configured for a range of signals a “-” is displayed for the signal number state as shown below.

```
$io 2
 64- 33 1010 1010 0100 0100 0000 0000 0000 1010
1064-1033 -----
2064-2033 -----
$
```

### RESET

Resets all external output signals to OFF. This command has no affect on dedicated signals.

## AS LANGUAGE COMMANDS

### 4.9.3 DEFSIG COMMAND

#### DEFSIG                      Input/Output

The DEFSIG command is used to display or set dedicated signals. The DEFSIG command without an argument displays a list of dedicated signals, conditions and signal numbers as shown below. The display mode does not allow the state of the signals, or the signal numbers, to be changed.

```
$DEFSIG
Dedicated signals set at present
EXT. MOTOR ON=1032
EXT. ERROR RESET=1031
EXT.CYCLE START=1030
MOTOR ON=32
ERROR=31
AUTOMATIC=30
  Condition : Panel switch in RUN.
  Condition : Panel switch in REPEAT.
  Condition : Repeat continuous.
  Condition : Step continuous.
CYCLE START=29
TEACH MODE=28
```

When input or output is specified, the dedicated input or output signals are displayed. The state of the signals, set or cancel and the signal numbers can be changed in this mode. To exit this mode type "e" for EXIT and press enter.

The DEFSIG OUTPUT display is shown below.

```
$DEFSIG OUTPUT
MOTOR ON Dedication cancel? (Enter 1 to set) 1
      signal number 32 Change? (1-128) (2001-2256)
ERROR Dedication cancel? (Enter 1 to cancel)
      signal number 31 Change? (1-128) (2001-2256)
CYCLE START Dedication cancel? (Enter 1 to cancel)1
AUTOMATIC Dedication cancel? (Enter 1 to cancel) e
$
```

## AS LANGUAGE COMMANDS

Output signals available are 1-128 and 2001-2256. Input signals available are 1001-128 and 2001-2256. When a signal number is assigned as a dedicated signal, it cannot be assigned to another dedicated signal or used as a general purpose IO signal.

Standard dedicated signals available are shown in table 4-1.

Table 4-1 Available Dedicated Signals

Dedicated Signals	
OUTPUTS	INPUTS
Motor power on	EXT.MOTOR ON
Prog. running	EXT.ERROR RESET
In error condition	EXT.CYCLE START
Automatic	EXT.PROGRAM RESET (EXRST)
Conditions:	EXT.program select (Jump)
Panel switch in RUN	Jump_ON
EXT_IT not set to hold	Jump_OFF
Panel switch in REPEAT	Jump_ST
Repeat continuous	EXT_IT (External Hold)
Step continuous	EXT.program.select (RPS)
TEACH LOCK OFF	RPS_ON
CYCLE START ON	RPS-ST
RGSO ON	Number of RPS Signals
Dryrun mode OFF	First signal number
CYCLE START	code (0 :Binary 1 :BCD)
TEACH mode	
HOME1	EXT_HOLD_RESET
HOME2	I/F PANEL PAGE1 SELECT
POWER ON	I/F PANEL PAGE2 SELECT
RGSO	EXT_HOLD_ERSET
DRYRUN	EXT.SLOW REPEAT MODE
WORKSPACE 1-9	Wire inching
ROBOT HOLD	External Weld ON
Ext.Prog select (RPS)enabled	Wire Retract
Positioner start	Positioner stop
Positioner speed	
WCR	
Weld ON	

## AS LANGUAGE COMMANDS

### 4.10 Z-SERIES ROBOTS AS LANGUAGE COMMANDS

For more information on the Z-series robot functions, refer to the *C Series Controller Operations and Programming Manual*.

#### 4.10.1 1GV ARM ID BOARD FUNCTIONS AND COMMANDS

The arm ID board is installed in the robot arm ID board box. The arm ID board stores model information, maintenance log information, and I/O signal settings. Functions and settings of the arm ID board are accessed from the controller.

The functions and settings of the arm ID board include:

- entries to the maintenance log
- display of the maintenance log
- deletion of maintenance log entries
- Settings of I/O signals to the robot arm

The AS Language commands used with the arm ID board are:

- MNTREC – maintenance log record
- MNTLOG – maintenance log display
- ARMIOSET – arm ID board I/O set

#### **\$MNTREC**      **robot\_number**

The MNTREC command is used to register maintenance log entries. If the robot number is omitted 1 is assumed. The maintenance log stores the last 100 entries. When over 100 entries are made the oldest entry is deleted.

Example:

```
$MNTREC <ENTER>  
Person in charge of record (input)? Joe Supervisor <ENTER>
```

```
Non of abnormality : 0 Memo input : 1? 1<ENTER>  
(Memo input) : JT1 motor replaced <ENTER>
```

content of registration

```
Person in charge : Joe Supervisor  
Memo : JT1 motor replaced
```

---

## AS LANGUAGE COMMANDS

Are you sure? (Yes:1, No:0) 1 <ENTER>  
arm ID board is busy.  
Writing ended.  
\$

### NOTE

Do not turn controller power off until "Writing ended" is displayed, or entry is not accepted.

---

**AS LANGUAGE COMMANDS****\$MNTLOG**            **robot\_number**

The MNTLOG command is used to display the contents of the maintenance log. If the robot number is omitted 1 is assumed. The maintenance log displays the last 100 entries, starting with the most recent. Press ENTER to stop the listing.

Example:

```
$MNTLOG <ENTER>
1-[00/06/24 12:03:00 Joe Supervisor]
[REPLACE JT1 HARNESS]
2-[00/04/12 14:20:32 Kawasaki]
[REPLACE JT2 MOTOR]
```

**\$ARMIOSET robot\_number,output\_signal\_No.,number\_of\_output\_signals,**  
**input\_signal\_No.,number\_of\_input\_signals**

The ARMIOSET command is used to allocate arm board parallel I/O signals. If the robot number is omitted 1 is assumed. Top signal range for output is 1 - 64. Number of output signals is 1 - 8. Top signal range for input is 1001 - 1064. Number of input signals is 1 - 24.

Example:

```
$ARMIOSET <ENTER>
                TOP SIGNAL,      SIGNAL NUMBER
OUTPUT SIGNAL 1                0
Change? (If not, Press RETURN only.)
6,8 <ENTER>

                TOP SIGNAL,      SIGNAL NUMBER
OUTPUT SIGNAL 6                8
Change? (If not, Press RETURN only.)
<ENTER>

                TOP SIGNAL,      SIGNAL NUMBER
INPUT SIGNAL 1001              0
Change? (If not, Press RETURN only.)
1012,24 <ENTER>

                TOP SIGNAL,      SIGNAL NUMBER
INPUT SIGNAL 1012              24
Change? (If not, Press RETURN only.)
<ENTER>
```

\$



## AS LANGUAGE COMMANDS

### 4.10.2 FAILURE PREDICTION FUNCTION, AUX 124 (OPTION)

This function establishes average current levels of motors during normal program operation and monitors motor current.

Reduction gear failure is gradual and motor current increases in small increments as the reduction gear approaches failure.

If the current levels exceed the established normal level, a warning message is displayed and an alarm signal is pulsed for one second.

If a program has varying operating conditions, such as payloads of different weights, that affect motor current this function can not be used.

Program teaching and verification must be completed before using this function. If changes are made to the program average current levels must be revised.

#### 4.10.2.1 FAILURE PREDICTION FUNCTION SETUP PROCEDURE

The AS Language command I2PG is used in programs to monitor motor torque (current). Each program must use the I2PG START pg (pg = program number) command at the point in the program current monitoring begins. The I2PG END command is used to end current monitoring. For usage of the I2PG command see the example below.

```
.PROGRAM pg00 ()
pg=1
;MAIN PROGRAM
;
100 HOME
BREAK
I2PG END ;End current monitoring
;
IF NOT SIG (2001) GOTO 100 ;Wait "move to pounce" signal
pg = BITS (2021,4) ;Assign bits decimal value to variable pg
I2PG START pg ;Start monitoring current for program number assigned to pg
CASE pg OF
VALUE 1:
CALL pg01
VALUE 2:
CALL pg02
VALUE 3:
CALL pg03
VALUE 4:
CALL pg04
END
GOTO 100
.END
```

---

## AS LANGUAGE COMMANDS

### 4.10.3 COLLISION DETECTION FUNCTION

The collision detection function is used to minimize damage to tooling and robotic equipment.

To activate the collision detection function, several parameters are set, using function screens or AS Language commands. Settings using function screens are covered in the *C Series Controller Operations and Programming Manual*. Settings using AS Language commands are covered in the following sections.

#### 4.10.3.1 SETTING TOOL WEIGHT DATA USING AS LANGUAGE WEIGHT COMMAND

##### **WEIGHT mass,center of gravity X Y and Z,inertia moment around X Y and Z**

The mass (weight), center of gravity location, and the inertia moment of the tool (Figure 4-3) are registered with the AS Language WEIGHT command. The arguments follow the command separated by commas (tool data is entered from the tool specification sheet or is available from the tool manufacturer).

The range of values are:

- load mass: 0 - load mass - kg
- center of gravity location: -9999.9 - 10000.0 - mm
- inertia moment: 0 - 999.99 - kg m<sup>2</sup>

Example:

```
WEIGHT 10, 0, 74.4, 4.5, 5.1, .21, .52
```

## AS LANGUAGE COMMANDS

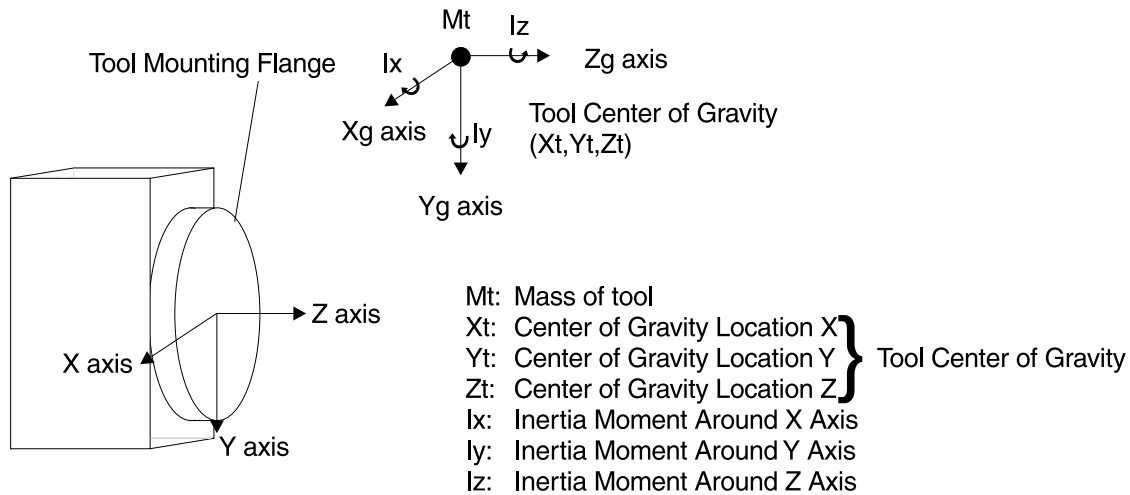


Figure 4-3 Tool Data

### 4.10.3.2 RANGE OF THRESHOLD

The two types of thresholds are collision detection and shock detection.

Each threshold is set for motion in teach mode and in repeat mode.

These thresholds are set using AUX 148 on the multifunction panel, AS Language monitor commands and program instructions (Table 4-2), or by automatic calibration using Aux 148-3.

## AS LANGUAGE COMMANDS

Table 4-2 Collision Detection Function Settings

AUX 148 COLLISION DETECTION FUNCTION			
	Threshold and Function Setting	AS Language Command/Instruction	Range
<b>Teach Mode (Check Mode)</b>	Set Threshold for Collision Detection	COLT	0 - 500%
	Collision detection	COLTON/COLTOFF	EFFECT/INEFFECT
	Set Threshold for Shock Detection	COLTJ	0 - 200%/msec
	Shock Detection	COLTJON/COLTJOFF	EFFECT/INEFFECT
<b>Repeat Mode</b>	Set threshold for Collision Detection	COLR	0 - 500%
	Collision Detection	COLRON/COLROFF	EFFECT/INEFFECT
	Set Thredhold for Shock Detection	COLRJ	0 - 200%/msec
	Shock Detection	COLRJON/COLRJOFF	EFFECT/INEFFECT
	Auto Calibration	COLCALON/COLCALOFF	EFFECT/INEFFECT

### 4.10.3.3 SETTING THRESHOLDS

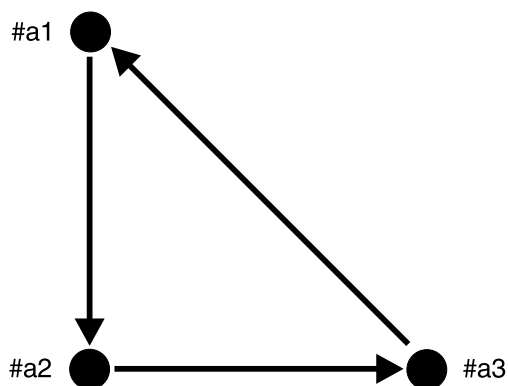
Nine threshold sets are available, in the example program in figure 4-4 and 4-5, three are set.

Each threshold set stores parameters for JT1 - JT6.

Example:

In the following example program, threshold 1 is set for the motion to #a1. Threshold 2 is set for the motion from #a1 to #a2. Threshold 3 is set for the motion from #a2 to #a3.

## AS LANGUAGE COMMANDS



Program collision\_detect

```

1 COLRON ;Set collision detection for repeat mode to ON.
2 SPEED 1400 mm/s ALWAYS ;Set program speed.
3 WEIGHT 10, 100, 40, 40, 5, 2, 0 ;Set the load of the tool.
4 COLR 60, 35, 60, 120, 140, 110 ;Set threshold 1 for repeat mode: JT1 through JT6.
5 JMOVE #a1 ;Move to location #a1.
6 COLR 38, 30, 58, 60, 80, 90 ;Set threshold 2 for repeat mode: JT1 through JT6.
7 SPEED 200 mm/s ;Change program speed for the next step.
8 LMOVE #a2 ;Move to location #a2.
9 DELAY 1 ;Delay robot movement for 1 second.
10 SPEED 300 mm/s ;Change program speed for the next step.
11 COLR 50, 48, 62, 82, 91, 98 ;Set threshold 3 for repeat mode: JT1 through JT6.
12 LMOVE #a3 ;Move to location #a3.
13 COLR 60, 35, 60, 120, 140, 110 ;Set threshold 1 for repeat mode: JT1 through JT6.
14 LMOVE #a1 ;Move to location #a1.
  
```

Figure 4-4 Collision Detection Sample Program

### AS LANGUAGE COMMANDS

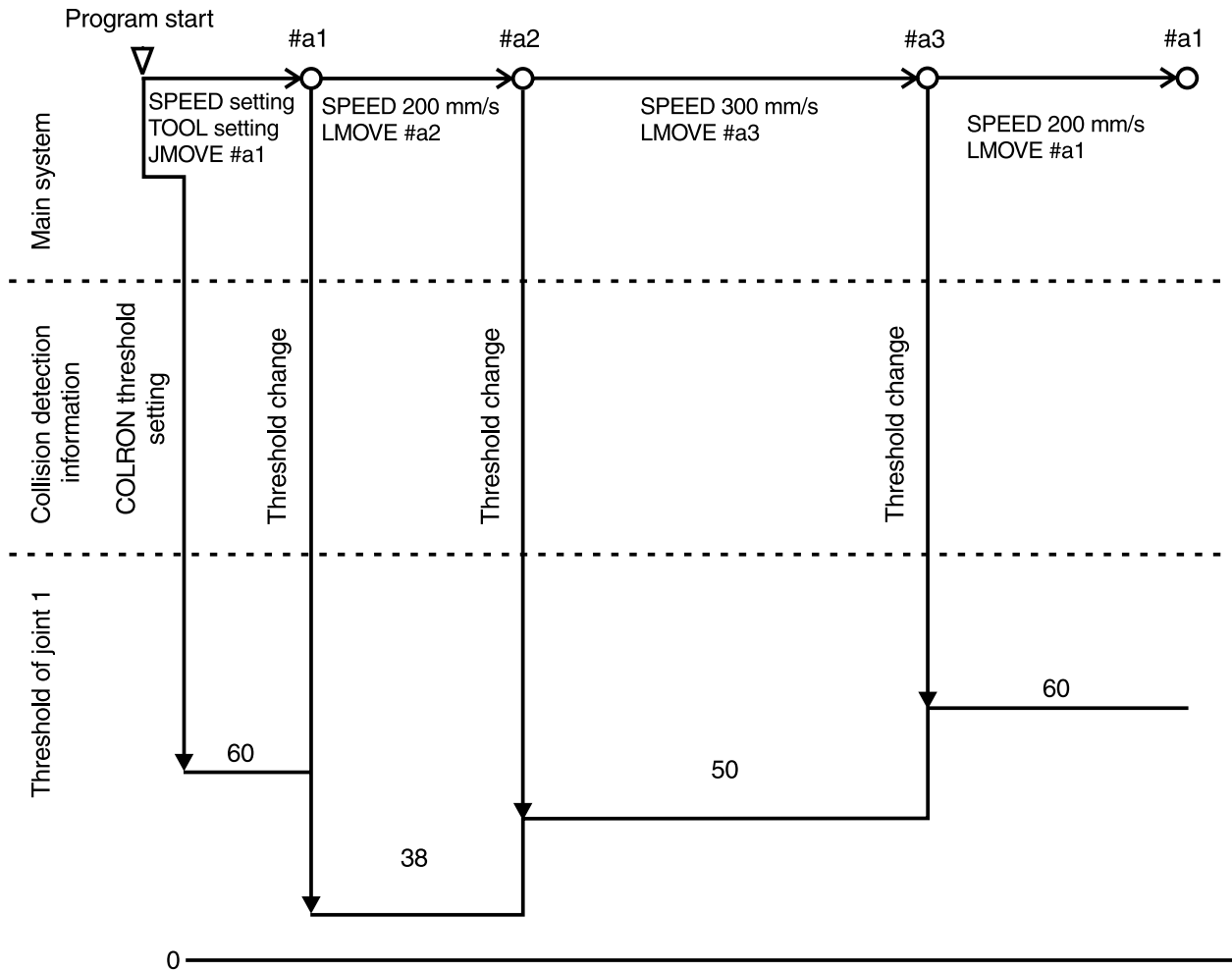


Figure 4-5 Collision Detection Sample Program Motion Diagram

---

## AS LANGUAGE COMMANDS

To obtain an accurate threshold setting tool registration data must be accurate.

To obtain threshold 1 delete the current threshold data in the collision\_detect program. Enter the CALCONON monitor command. Run the program collision\_detect several times, as it is normally run, with any peripherals and/or the work piece in place.

End the threshold setting with the CALCONOFF monitor command. Read the repeat mode threshold values using the COLR monitor command and enter the values in the collision\_detect program.

In the example program threshold 1 is effective for the entire program, unless changed for steps using the COLR and the COLRJ commands. Threshold 2 is effective during movement from #a1 to #a2 or from #a2 to #a1. Threshold 3 is effective during movement from #a2 to #a3.

To obtain threshold 1: execute the program collision\_detect.

To obtain threshold 2: execute the following program calib1.

Example:

```
.PROGRAM calib1()
WEIGHT 10,100,40,40,1,1.4,.52
SPEED 200 mm/s ALWAYS
JMOVE #a1
10
LMOVE #a2
DELAY 1
LMOVE #a1
DELAY 1
GOTO 10
.END
```

## AS LANGUAGE COMMANDS

To obtain threshold 3: execute the following program calib2.

```
PROGRAM calib2()
WEIGHT 10,100,40,40,1,1.4,.52
SPEED 200 mm/s ALWAYS
JMOVE #a2
10
LMOVE #a3
DELAY 1
LMOVE #a2
DELAY 1
GOTO 10
.END
```

The following example describes an alternate method for obtaining threshold data using the SETCOLHID instruction.

Nine threshold sets are available. The thresholds set by the SETCOLHID command are in effect unless changed by the COLR and the COLRJ commands using the specific threshold ID number.

Example:

```
PROGRAM chg_th()
1 SETCOLTHID 1
2 CALL collision_detect ;Motion block 1
3 SETCOLTHID 2
4 WEIGHT 8,10,0,25,2,3.5,4
5 C1MOVE #C1 ;Motion block 2
6 C1MOVE #C2 ;Motion block 2
7 C1MOVE #C3 ;Motion block 2
8 C1MOVE #a1 ;Motion block 2
9 COLR 30,20,40,30,30,30
10 JMOVE #home ;Motion block 3

PROGRAM collision_detect()
1 WEIGHT 10,100,40,40,1,1.4,.52
2 JMOVE #a1
3 LMOVE #a2
4 LMOVE #a3
5 LMOVE #a1
```



---

## AS LANGUAGE COMMANDS

Threshold ID number 1 is used for motion block 1.

Threshold ID number 2 is used for motion block 2.

Threshold for motion block 3 is set by the programmer using the COLR command.

### 4.10.3.4 COLLISION DETECTION AS LANGUAGE COMMANDS

The following section describes the AS Language monitor commands and program instructions used with the collision detection function.

COLR:	Monitor command and program instruction
COLRON/OFF:	Monitor command and program instruction
COLRJ:	Monitor command and program instruction
COLRJON/OFF:	Monitor command and program instruction
COLT:	Monitor command
COLTON/OFF:	Monitor command
COLTJ:	Monitor command
COLTJON/OFF:	Monitor command
COLMVON/OFF:	Monitor command and program instruction
COLCALON/OFF:	Monitor command and program instruction
WEIGHT:	Monitor command and program instruction
SETCOLTHID:	Monitor command and program instruction
COLINIT:	Monitor command
COLSTATE:	Monitor command

---

## AS LANGUAGE COMMANDS

### 4.10.3.5 COLR

#### COLR JT1\_threshold, JT2\_threshold, ..., JT6\_threshold(, ..., JTn\_threshold)

**COLR:** Is a monitor command and a program instruction. COLR sets the collision threshold for JT1 through JT6 for repeat mode, and has the option to set thresholds for addition axes (JTn).

range: 0 - 500%, a setting of 0 does not detect a collision.

When the power required to move the robot arm exceeds the threshold settings, a collision is detected and the robot stops. The lower the setting is the more sensitive the collision detection is. If the threshold settings are set to low the load torque of the arm may cause a mis-detection.

The recommend procedure for setting thresholds is to use the COLCAL command (automatically sets thresholds). If mis-detections occur when the COLCAL command is used to set thresholds, increase the settings using a factor of times 1.2.

#### Example:

The following example program moves the robot from #a1 to #a2 and from #a2 to #a3 (Figure 4-6) and changes the threshold in each section.

```
1 COLR 0,40,52,81,72,68
2 JMOVE #a1
3 JMOVE #a2
4 COLR 40,0,56,90,83,75
5 JMOVE #a3
```

JT1 does not detect a collision during the move from #a1 to #a2 and (Jt1 threshold set to 0) JT2 does not detect a collision during the move from #a2 to #a3 (Jt2 threshold set to 0).

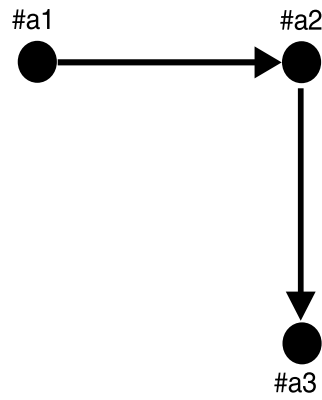
**AS LANGUAGE COMMANDS**

Figure 4-6 COLR Example Program Robot Movement

---

## AS LANGUAGE COMMANDS

### 4.10.3.6 COLRON/OFF

**COLRON/OFF** Is a monitor command and a program instruction. The COLRON command is used to start collision detection monitoring in repeat mode. The COLROFF command is used to stop collision detection monitoring in repeat mode.

When COLRON or COLROFF is executed as a monitor command, the change is effective until controller power is turned off.

Do not add a space between COLR and ON or OFF.

To display the status of COLRON/OFF use the COLSTATE monitor command.

#### Example:

The following example program moves the robot arm from #a1 to #a4 (Figure 4-7). Collision detection is effective only during the move from #a2 to #a3.

If a spot weld is programmed at a step collision detection is ineffective during the move.

```
1 COLR    38,40,52,81,72,68
2 COLROFF
3 JMOVE #a1
4 JMOVE #a2
5 COLRON
6 JMOVE #a3
7 COLROFF
8 JMOVE #a4
```

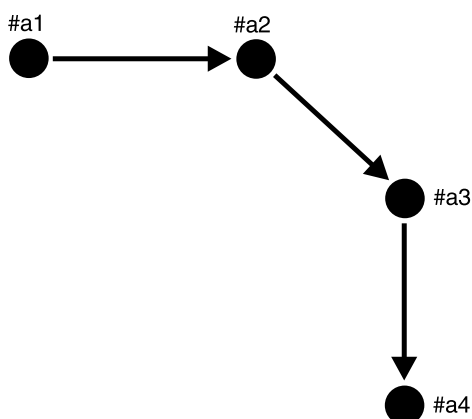
**AS LANGUAGE COMMANDS**

Figure 4-7 COLRON/OFF Example Program Robot Movement

**4.10.3.7 COLRJ**

**COLRJ**            **JT1\_threshold,JT2\_threshold,...,JT6\_threshold  
(,...JTn\_threshold)**

Is a monitor command and a program instruction. COLRJ sets the shock detection threshold for JT1 through JT6 for repeat mode, and has the option to set thresholds for addition axes (JTn).

range:            0 - 200%/msec, a setting of 0 does not detect shock.

The threshold values used for shock detection must be set using the COLCALON and COLCALOFF commands. Shock detection monitors the elapsed time of a current increase. If the amount of change exceeds the threshold settings, the robot stops. Shock detection reacts faster than collision detection, and is useful when increased sensitivity for collision detection is desired.

The recommend procedure for setting thresholds for shock detection, is to use the COLCAL command (automatically sets thresholds).

---

## AS LANGUAGE COMMANDS

### 4.10.3.8 COLRJON/OFF

**COLRJON/OFF** Is a monitor command and a program instruction. COLRJON is used to start shock detection monitoring in repeat mode. COLRJOFF is used to stop shock detection monitoring in repeat mode.

Shock detection increases the sensitivity of the collision detection function and is not suitable for fast or jerky robot motions.

Use the COLSTATE command to display the status of COLRJ ON or OFF.

### 4.10.3.9 COLT

**COLT** **JT1\_threshold,JT2\_threshold,...,JT6\_threshold**  
**(,...JTn\_threshold)**

Is a monitor command. COLT is used to set thresholds for JT1 through JT6 for teach mode, and has the option to set thresholds for addition axes (JTn).

range: 0 - 500%, a setting of 0 does not detect a collision.

This command is effective for teach and check modes only.

The COLT command is used in the same manner as the COLR monitor command.

### 4.10.3.10 COLTON/OFF

**COLTON/OFF** Is a monitor command. The COLTON command is used to start collision detection monitoring in teach and check modes. The COLTOFF command is used to stop collision detection monitoring in teach and check modes.

When COLTON or COLTOFF is executed as a monitor command, the change is effective until controller power is turned off.

Do not add a space between COLT and ON or OFF.

To display the status of COLTON/OFF use the COLSTATE monitor command.

---

## AS LANGUAGE COMMANDS

### 4.10.3.11 COLTJ

**COLTJ**                    **JT1\_threshold,JT2\_threshold,...,JT6\_threshold  
(,...JTn\_threshold)**

Is a monitor command. COLTJ sets the shock detection threshold for JT1 through JT6 for teach and check modes, and has the option to set thresholds for addition axes (JTn).

range:                    0 - 200%/msec, a setting of 0 does not detect shock.

The threshold values used for shock detection must be set using the COLCALON and COLCALOFF commands. Shock detection monitors the elapsed time of a current increase. If the amount of change exceeds the threshold settings, the robot stops. Shock detection reacts faster than collision detection, and is useful when increased sensitivity for collision detection is desired.

The recommend procedure for setting thresholds for shock detection, is to use the COLCAL command (automatically sets thresholds).

### 4.10.3.12 COLTJON/OFF

**COLTJON/OFF**            Is a monitor command. COLTJON is used to start shock detection monitoring in teach and check modes. COLTJOFF is used to stop shock detection monitoring in teach and check modes.

Shock detection increases the sensitivity of the collision detection function and is not suitable for fast or jerky robot motions.

Do not add a space between COLTJ and ON or OFF.

Use the COLSTATE monitor command to display the status of COLTJ ON or OFF.

## AS LANGUAGE COMMANDS

### 4.10.3.13 COLMVON/OFF

**COLMVON/OFF** Is a monitor command and a program instruction. The COLMVON command is used to execute the stress remove motion after a collision is detected. The COLMVOFF command is used when the stress remove motion after a collision is detected is not desired.

The stress remove motion causes the robot arm to move in the opposite direction of the robot path movement, when a collision is detected (Figure 4-8). This return motion is an arcing motion and may cause the tool to contact the work-piece or other peripherals (Figure 4-8).

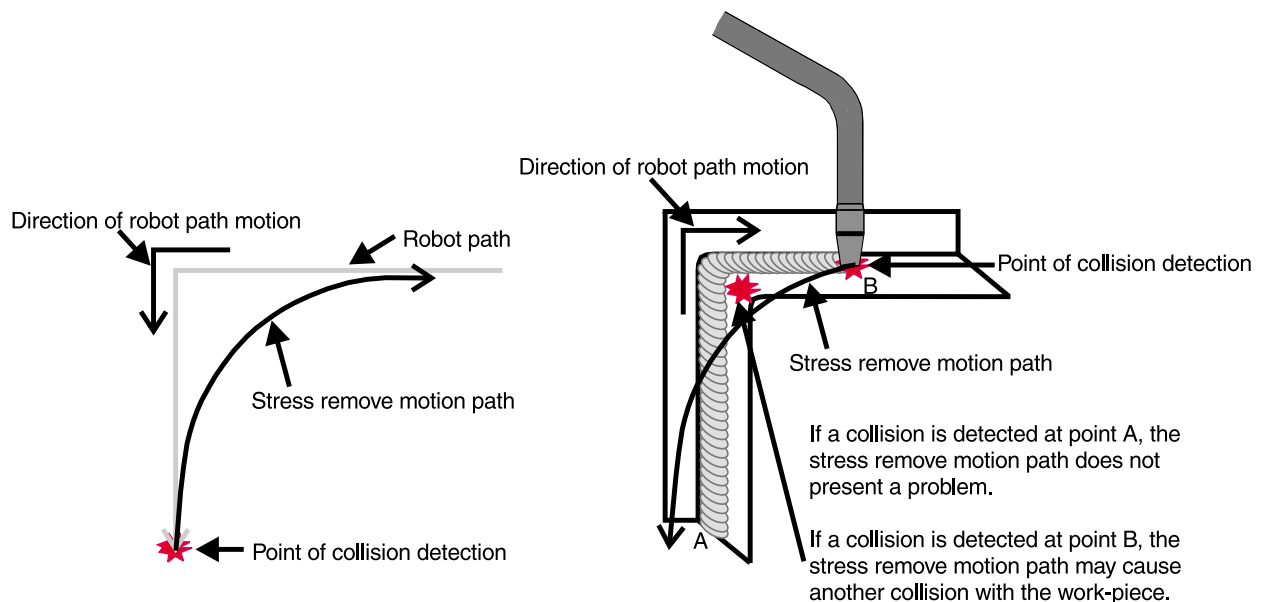


Figure 4-8 Stress Remove Motion Path

The slower the tool tip speed at a collision detection, the less stress remove motion is used by the robot.

When COLMVON or COLMVOFF is executed as a monitor command, the change is effective until controller power is turned off.

Do not add a space between COLMV and ON or OFF.

Use the COLSTATE command to display the status of COLMV ON or OFF.





---

## AS LANGUAGE COMMANDS

### 4.10.3.16 SETCOLTHID

**SETCOLTHID**      **threshold\_number**

Is a monitor command and a program instruction. The SETCOLTHID monitor command is used to display and set the designated threshold number. The SETCOLTHID program instruction is used to apply the threshold values of the designated threshold number to the robot motion program.

threshold\_number: Range is 1-9, this argument cannot be omitted.

When SETCOLTHID is used as a monitor command the value of the designated threshold number is displayed and can be changed. The collision detection and the shock detection values for the designated threshold are displayed separately for change. A message is displayed to allow the shock detection threshold to be continuously updated, if desired. Changes are effective for the robot program repeat motion.

When SETCOLTHID is used as a program instruction, it sets the threshold ID number (1-9) effective for the robot motion program. This command is effective until another SETCOLTHID command is executed in the robot motion program.

The SETCOLTHID command and program instruction executes the auto calibration of the threshold number designated and this threshold is continuously updated. Before executing the SETCOLTHID monitor command or program instruction, the operator must initialize the threshold with the COLINIT monitor command. The COLINIT monitor command sets the value of the designated threshold to the default factory setting (see section 4.10.3.17).

The value of threshold numbers 1-9 are displayed using the COLSTATE monitor command.

---

## AS LANGUAGE COMMANDS

### 4.10.3.17 COLINIT

#### **COLINIT**      **threshold\_number**

Is a monitor command. The COLINIT command is used to set collision detection parameters to default factory settings.

**threshold\_number:** Range is 1-9. The threshold number argument can be omitted. When the threshold number is omitted all threshold parameters are set to default factory settings.

When a threshold number is designated the settings for collision detection and shock detection for the designated threshold number are set to 0.

### 4.10.3.18 COLSTATE

#### **COLSTATE**   **threshold\_number**

Is a monitor command. The COLSTATE command is used to display current collision detection and shock detection parameters.

**threshold\_number:** Range is 1-9. The threshold number argument can be omitted.

The value of each axis threshold and the current setting of each collision detection mode are displayed (figure 4-9 and 4-10).

COLR:	Collision detection in repeat mode ON/OFF
COLRJ:	Shock detection in repeat mode ON/OFF
COLT:	Collision detection in teach mode ON/OFF
COLTJ:	Shock detection in teach mode ON/OFF
COLESCMV:	Stress remove motion ON/OFF
COLCAL:	Automatic calibration mode ON/OFF
VCOLR:	Collision detection in repeat mode ON/OFF (OFF during spot weld execution)
VCOLT:	Collision detection in teach mode ON/OFF (OFF during spot weld execution)

### AS LANGUAGE COMMANDS

```

$colstate

Current threshold for teach mode
  JT1/JT8  JT2/JT9  JT3/JT10  JT4/JT11  JT5/JT12  JT6/JT13  JT7/JT14
    15     35     42     120     110     160

Current threshold for shock detection
  JT1/JT8  JT2/JT9  JT3/JT10  JT4/JT11  JT5/JT12  JT6/JT13  JT7/JT14
    5       6       4       15       44       23

Current threshold for repeat mode
  JT1/JT8  JT2/JT9  JT3/JT10  JT4/JT11  JT5/JT12  JT6/JT13  JT7/JT14
    45     60     75     235     305     225

Current threshold for shock detection
  JT1/JT8  JT2/JT9  JT3/JT10  JT4/JT11  JT5/JT12  JT6/JT13  JT7/JT14
    3       5       5       11       11       15

COLR.....ON      VCOLR.....ON
COLRJ.....ON
COLT.....ON      VCOLT.....ON
COLTJ.....ON
COLESCMV....OFF  COLCAL.....OFF
    
```

Figure 4-9 COLSTATE Command Display without Threshold Number

```

$colstate 3

Current threshold for repeat mode
  JT1/JT8  JT2/JT9  JT3/JT10  JT4/JT11  JT5/JT12  JT6/JT13  JT7/JT14
    20     35     60     112     120     116

Current threshold for shock detection
  JT1/JT8  JT2/JT9  JT3/JT10  JT4/JT11  JT5/JT12  JT6/JT13  JT7/JT14
    3       4       3       6       7       6
    
```

Figure 4-10 COLSTATE Command Display with Threshold Number

---

## AS LANGUAGE COMMANDS

### 4.10.3.19 COLLISION DETECTION ERROR CODE

**ERROR CODE**     **-1902** JT\* Collision or abnormal disturb is detected

Occurs when the controller determines a collision has occurred because torque values for the motors cannot be determined.

- ⇒ Move the tool or the robot arm away from the obstacle.
- ⇒ Ensure the tool registration data is accurate.
- ⇒ If a collision did not occur and the error reoccurs, increase the threshold setting using a factor of times 1.2.

### 4.10.3.20 COLLISION DETECTION TROUBLESHOOTING

Constant mis-detection:

1. Ensure the tool registration data is accurate.
2. Reset thresholds using auto-calibration.
3. Manually reset the thresholds using a factor of times 1.2.

Collision detection reaction is slow:

1. Ensure the tool registration data is accurate.
2. Reset thresholds using auto-calibration.

Collision is not detected:

1. Use the COLSTATE command to determine if collision detection is set to ON and threshold values for the axes are not set to 0.

Stress remove motion is too large:

1. If the stress remove motion causes the tooling to interfere with the work-piece or peripherals, use the COLMVOFF command to make the stress remove motion ineffective.

## AS LANGUAGE COMMANDS

Table 4-3 Collision Detection AS Language Commands

Abbreviation	Command	Function	Type of Command	Format
	COLCALON/OFF	To obtain the optimum threshold	Monitor	COLCALON/OFF
	COLINIT	Return to default factory threshold settings	Monitor	COLINIT threshold number
	COLMVNON/OFF	Stress remove motion ON/OFF	Monitor/Program	COLMVNON/OFF
	COLR	Set collision detection threshold in repeat mode	Monitor/Program	COLR JT1 threshold - JT7 threshold
	COLRJ	Set shock detection threshold in repeat mode	Monitor/Program	COLRJ JT1 threshold - JT7 threshold
	COLRON/OFF	Collision detection ON/OFF	Monitor/Program	COLRON/OFF
	COLRJON/OFF	Shock detection ON/OFF	Monitor/Program	COLRJON/OFF
COLS	COLSTATE	Display status of collision detection	Monitor	COLSTATE threshold number
	COLT	Set collision detection threshold in teach mode	Monitor	COLT JT1 threshold - JT7 threshold
	COLTJ	Set shock detection threshold in teach mode	Monitor	COLTJ JT1 threshold - JT7 threshold
	COLTON/OFF	Collision detection ON/OFF	Monitor	COLTON/OFF
	COLTJON/OFF	Shock detection ON/OFF	Monitor	COLTJON/OFF
SETCOL	SETCOLTHID	Display and change data of the designated threshold number	Monitor	SETCOLTHID threshold number
SETCOL	SETCOLTHID	Apply the designated threshold number data to the robot program	Program	SETCOLTHID threshold number
WE	WEIGHT	Set the mass of the tool, the center of gravity location, and the inertia moment of the tool center of gravity rotation.	Monitor/Program	WEIGHT mass of tool, center of gravity location X, center of gravity location Y, center of gravity location Z, X-axis rotation inertia moment, Y-axis rotation inertia moment, Z-axis rotation inertia moment

---

**AS LANGUAGE PROGRAM INSTRUCTIONS**

<b>5.0</b>	<b>PROGRAM INSTRUCTIONS</b> .....	5-2
5.1	Program Instruction Syntax .....	5-2
5.2	Robot Motion Control .....	5-3
5.2.1	Motion Instructions .....	5-5
5.2.2	HOME Command .....	5-9
5.2.3	DRIVE and DRAW Commands .....	5-9
5.2.4	ALIGN Command .....	5-10
5.2.5	XMOVE and HMOVE .....	5-11
5.2.6	BREAK and BRAKE Commands .....	5-12
5.2.7	DELAY and STABLE Commands .....	5-13
5.2.8	ACCURACY and SPEED Commands .....	5-14
5.2.9	ACCEL and DECEL Commands .....	5-17
5.2.10	Clamp Control Commands .....	5-18
5.2.11	Configuration Commands .....	5-21
5.3	Program Control .....	5-21
5.3.1	GOTO and IF Commands .....	5-22
5.3.2	CALL and RETURN Commands .....	5-24
5.3.3	WAIT, SWAIT, and TWAIT Commands .....	5-26
5.3.4	PAUSE, HALT, and STOP Commands .....	5-27
5.3.5	SCALL, LOCK, ONE, RETURN Commands .....	5-28
5.3.6	ON, ONI, IGNORE, and EXTCALL Commands .....	5-29
5.4	Message Display Commands .....	5-33
5.4.1	PRINT and TYPE Commands .....	5-33
5.4.2	PROMPT Command .....	5-36
5.5	External Output Signal Control .....	5-37
5.5.1	SIGNAL, PULSE, and DYLSIG Commands .....	5-37
5.5.2	BITS Command .....	5-39
5.5.3	RESET and RUNMASK Commands .....	5-40
5.6	Definition of Variables .....	5-41
5.6.1	HERE and POINT Commands .....	5-42
5.6.2	DECOMPOSE Command .....	5-44
5.6.3	TOOL and BASE Commands .....	5-45
5.6.4	LLIMIT and ULIMIT Commands .....	5-45
5.6.5	TIMER and SWITCH Commands .....	5-46
5.7	Control Flow Structures .....	5-48
5.7.1	IF THEN ELSE Command .....	5-48
5.7.2	WHILE DO Command .....	5-50
5.7.2.1	Loop Control Variables .....	5-51
5.7.3	DO UNTIL Command .....	5-51
5.7.4	FOR TO Command .....	5-52
5.7.5	CASE Command .....	5-54

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.0 PROGRAM INSTRUCTIONS

#### 5.1 PROGRAM INSTRUCTION SYNTAX

The complete format of a valid AS program line is:

step number label <space> program instruction <space> argument ;comment

1?	100	JMOVE	pt1	;pounce
2?		JMOVE	pt2	;Anything after the semicolon
3?	line:	JMOVE	pt3	;is ignored by the system.

The step number is automatically inserted by the AS editor so all steps are numbered consecutively. The optional step label is used to identify a particular program line for a branching instruction elsewhere in the program. The optional label can be either a positive integer value or an alphanumeric name no longer than fifteen characters. If an alphanumeric label is used, it must be followed by a colon. The semicolon is used in the AS system to represent a comment. Any text that appears after the semicolon is ignored by the system.

One space is required between a program instruction, and a label, or an argument in a line. If more spaces are entered the entry is accepted and the extra spaces are automatically removed when the line is entered.

All AS instructions/commands in this unit are shown in uppercase letters with arguments shown in lowercase letters. Program instructions and their arguments are typed into a program using any combination of uppercase or lowercase letters. The AS editor converts the case of the input to conform to the convention of uppercase for all AS keywords and lowercase for all user variables.



---

**AS LANGUAGE PROGRAM INSTRUCTIONS****5.2 ROBOT MOTION CONTROL**

JMOVE	(JM)	Starts a joint interpolated motion.
LMOVE	(LM)	Starts a linear interpolated motion.
JAPPROACH	(JA)	Starts a joint interpolated motion to approach the location keeping the specified distance.
LAPPROACH	(LA)	Starts a linear interpolated motion to approach the location keeping the specified distance.
JDEPART	(JD)	Backs the tool the specified distance from the current location in a joint interpolated motion.
LDEPART	(LD)	Backs the tool the specified distance from the current location in a linear interpolated motion.
HOME	(HO)	Moves the robot to the home position.
DRIVE	(DRI)	Moves a single joint.
DRAW	(DRA)	Moves the robot by the amount given in XYZ components.
TDRAW	(TD)	Moves the robot by the amount given in tool XYZ components.
ALIGN	(AL)	Aligns Z-axis of the robot tool.
XMOVE	(X)	Moves to next location until specified signal is received
HMOVE	(HM)	Starts a hybrid interpolated motion
C1MOVE	(C1)	Starts move for a circular interpolated motion (option).
C2MOVE	(C2)	Ends move for a circular interpolated motion (option).
BREAK	(BRE)	Ensures current robot motion is completed.
BRAKE	(BRA)	Stops the current robot motion immediately.
DELAY	(DEL)	Stops robot motion for the specified period of time.

---

**AS LANGUAGE PROGRAM INSTRUCTIONS**

STABLE	(STA)	Waits for the robot to pause exactly for the specified period of time.
ACCURACY	(ACCU)	Sets the accuracy.
SPEED	(SP)	Sets the robot motion speed.
ACCEL	(ACCE)	Sets the acceleration factor.
DECEL	(DECE)	Sets the deceleration factor.
OPEN	(OPEN)	Opens clamp at the transition.
OPENI	(OPENI)	Opens clamp at coincidence.
CLOSE	(CLOSE)	Closes clamp at the transition.
CLOSEI	(CLOSEI)	Closes clamp at coincidence.
RELAX	(RELAX)	Turns clamp signals off at the beginning of next motion.
RELAXI	(RELAXI)	Turns clamp signals of at coincidence.
UWRIST	(UW)	Changes the robot configuration to position joint 5 in a positive angle.
DWRIST	(DW)	Changes the robot configuration to position joint 5 in a negative angle.
RIGHTY	(RI)	Changes the robot configuration to be right handed.
LEFTY	(LE)	Changes the robot configuration to be left handed.
ABOVE	(AB)	Changes the robot configuration to position elbow on the upper side.
BELOW	(BE)	Changes the robot configuration to position on the lower side.

---

**AS LANGUAGE PROGRAM INSTRUCTIONS****5.2.1 MOTION INSTRUCTIONS****JMOVE**                    **location, clamp\_number****LMOVE**

Starts the robot motion to the position defined by the given location.

JMOVE:                    Moves in joint interpolated motion.

LMOVE:                    Moves in linear interpolated motion.

location:                    Represents the destination (transformation value, precision value, location function, or compound transformation value).

The JMOVE instruction initiates a joint interpolated (point-to-point) motion. The robot moves in a path that interpolates respective joint angles between the current position and the destination position.

The LMOVE instruction initiates a linear interpolated motion and the robot tool center point moves along a straight line path.

clamp number:            The optional clamp number argument allows the programmer to designate a clamp to open or close.

a positive number opens the clamp and a negative number closes the clamp.

JMOVE start,2 ;moves to location start and opens clamp number 2.

JMOVE pick,-2 ;moves to location pick and closes clamp number 2.

If the optional clamp number is omitted, clamp signals are not changed.

**JAPPRO**                    **location, distance****LAPPRO**

Moves the tool to the location, offset the specified distance along the tool Z-axis.

JAPPRO:                    Moves in joint interpolated motion.

LAPPRO:                    Moves in linear interpolated motion.

location:                    Defines the destination (either transformation or precision value).

## AS LANGUAGE PROGRAM INSTRUCTIONS

**distance:** Distance along the tool Z-axis, between the specified location and the actual destination in millimeters. A positive distance sets the torch back (tool negative Z-axis) from the specified location; a negative distance offsets the tool forward (tool positive Z-axis) of the specified location.

In the JAPPRO and LAPPRO instructions, the tool direction is determined by the location argument, and the actual tool destination is set to approach the specified location by the given distance along the tool Z-axis (Figure 5-1). Refer to figure 5-2 for tool orientation.

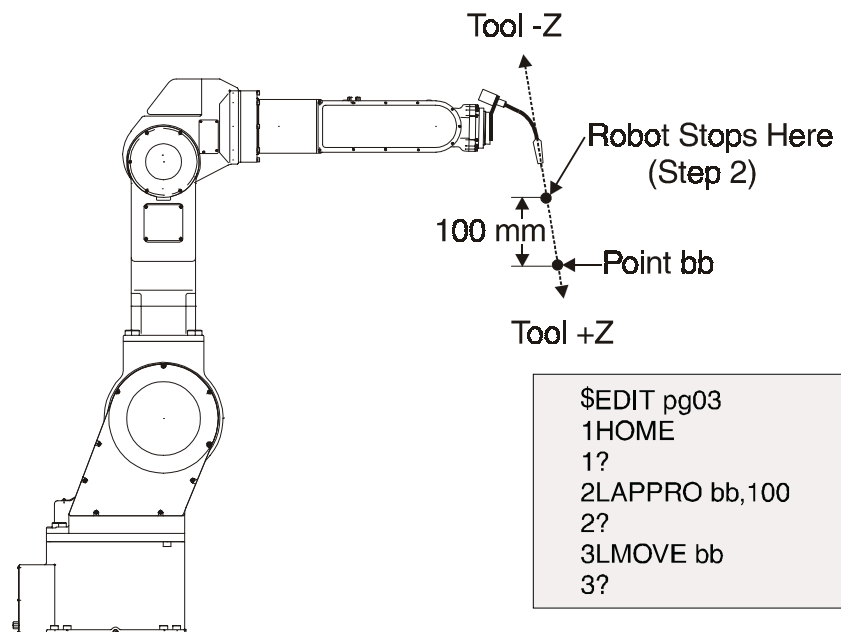


Figure 5-1 JAPPRO/LAPPRO Commands

In the preceding example, step 2 causes the robot to stop 100 mm away from location bb in the tool -z direction. The robot moves to location bb when step 3 is executed. If a negative value is specified in step 2, the robot moves 100 mm beyond location bb in the tool +z direction.

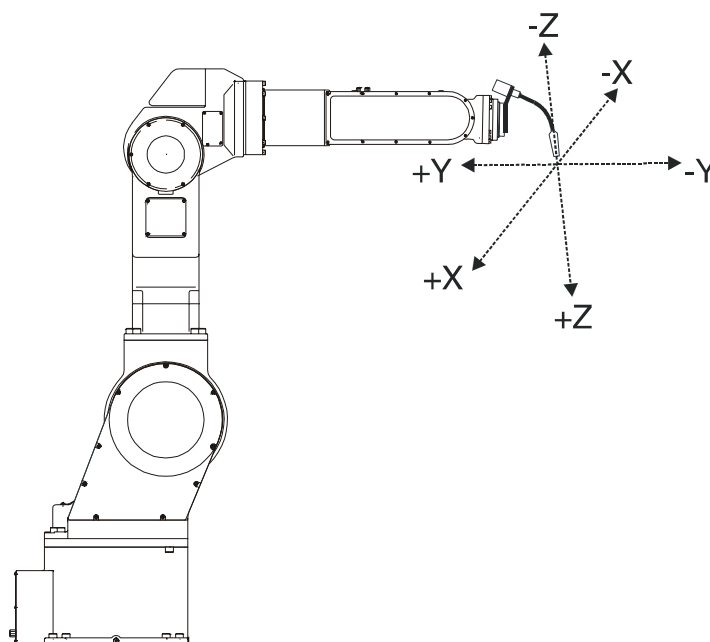
**AS LANGUAGE PROGRAM INSTRUCTIONS**

Figure 5-2 Tool Orientation

**JDEPART**  
**LDEPART****distance**

Moves the tool by the distance given along the current Z-axis of the tool.

JDEPART:  
LDEPART:Moves in joint interpolated motion  
Moves in linear interpolated motion.

Distance:

Distance along the tool Z-axis, between the specified location and the actual destination in millimeters. A positive distance sets the tool back (tool negative Z-axis) from the current location; a negative distance offsets the tool forward (tool positive Z-axis) of the specified location (Figure 5-3). Refer to figure 5-2 for tool orientation.

**AS LANGUAGE PROGRAM INSTRUCTIONS**

The LDEPART command at step 4, in the following example, moves the robot 100 mm away from the location bb (step3).

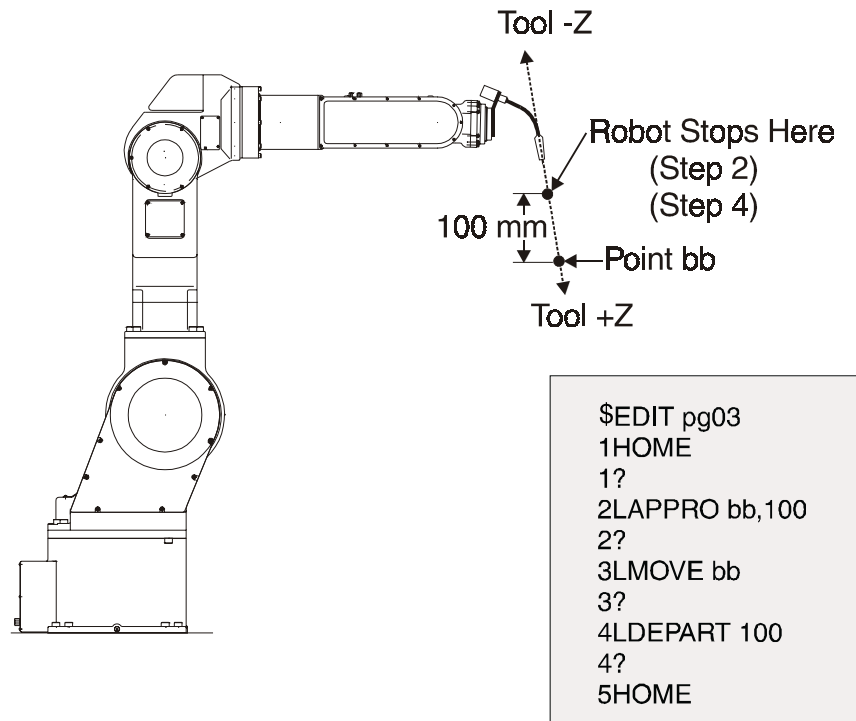


Figure 5-3 JDEPART/LDEPART Commands

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.2.2 HOME COMMAND

**HOME, HOME2** Begins a joint interpolated movement to the defined HOME or HOME 2 position. The HOME or HOME 2 positions must be defined using the SETHOME or SET2HOME monitor command, prior to executing this program instruction.

### 5.2.3 DRIVE AND DRAW COMMANDS

**DRIVE** `joint_number, change_amount, speed`

Moves a single joint of the robot independently.

**joint\_number:** Number of the joint to be moved.

**change\_amount:** Amount of the axis movement. For a rotational axis, specify this value in degrees, and for a traverse axis specify the distance in millimeters.

**speed:** Speed used for this motion. Similar to the program speed setting, the value is specified as a percentage of the repeat speed setting. If omitted, it is set to 100 percent.

The specified joint is moved by the `change_amount`. The motion speed is the combination of the speed specified in this instruction, and the current repeat speed.

For example, `DRIVE 2,50,75` moves joint 2 of the robot 50° at 75 percent of the repeat speed.

**DRAW, TDRAW** `X_variation, Y_variation, Z_variation`

`X_variation, Y_variation, Z_variation`

**DRAW:** DRAW moves the robot the amount specified in the X, Y, Z components of the base coordinate system.

**TDRAW:** TDRAW moves the robot for the amount specified in the X, Y, Z components of the tool coordinate system. If the X, Y, or Z variations are omitted, zero variation is assumed.

**X\_variation:** Distance in the X direction (in millimeters).

**Y\_variation:** Distance in the Y direction (in millimeters).

**Z\_variation:** Distance in the Z direction (in millimeters).

**AS LANGUAGE PROGRAM INSTRUCTIONS**

Example: DRAW 50,100,-50 moves the robot 50 mm in x, 100 mm in y, and -50 mm in the z-axis direction. The robot moves in a linear interpolated motion.

**5.2.4 ALIGN COMMAND****ALIGN**

Moves the robot so the tool Z-axis is parallel to the closest base coordinate axis of the base coordinate system.

This instruction is used for setting the tool direction parallel to a base axis before teaching a series of locations. It is also used for checking the tool coordinate definition.

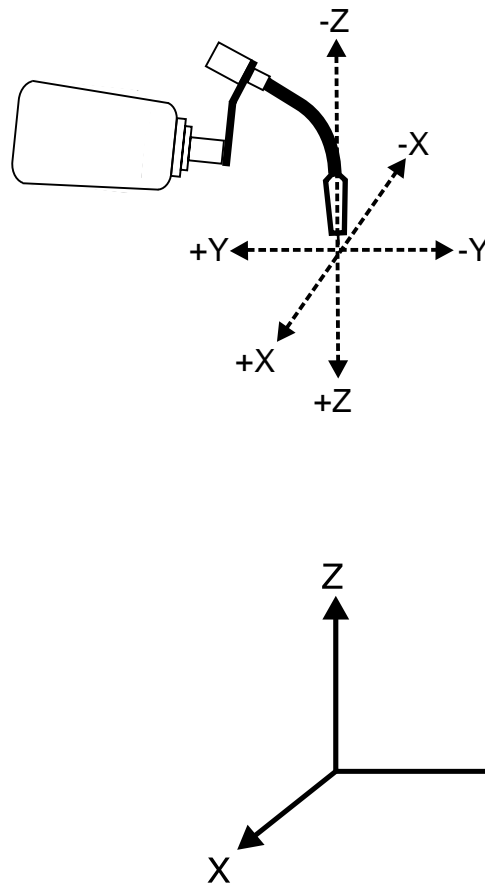


Figure 5-4 Alignment of Tool Z-Axis with Base



**AS LANGUAGE PROGRAM INSTRUCTIONS****5.2.5 XMOVE AND HMOVE****XMOVE                    location TILL signal\_number**

Starts a linear interpolated motion toward a specified location, until the specified signal number is received.

location:                Represents the destination (transformation value, precision value, location function, or compound transformation value).

signal\_number:        Number of the signal to monitor.

A positive number indicates the on state causes the transition.

A negative number indicates the off state causes the transition.

In the example below, the robot starts to move to location box.

If signal 1001 goes high, the robot changes direction and moves to location CC.

If the signal does not go high the robot completes the move to location box.

Example:                XMOVE box TILL 1001  
                              JMOVE CC

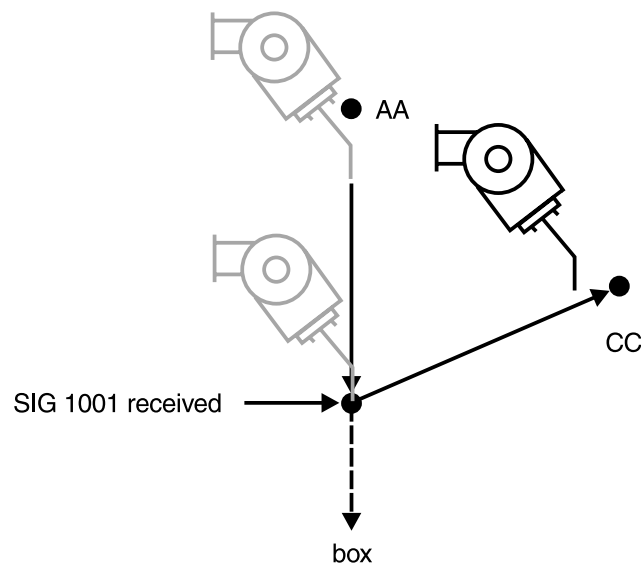


Figure 5-5 XMOVE Instruction

## AS LANGUAGE PROGRAM INSTRUCTIONS

### **HMOVE**                    **location, clamp\_number**

An HMOVE is a hybrid interpolated move used when a linear type motion is desired, but the configuration of the robot arm causes a singularity error.

A singularity error occurs when the wrist is aligned so movement of JT4 or JT6 result in the same movement of the TCP.

**location:**                    Represents the destination (transformation value, precision value, location function, or compound transformation value).

**clamp\_number:**            The optional clamp number argument allows the programmer to designate a clamp to open or close.

A positive number opens the clamp and a negative number closes the clamp.

**Example:**                    HOME  
                                  JMOVE #start  
                                  HMOVE pick,-1  
                                  LMOVE place,1  
                                  HMOVE end

### 5.2.6 BREAK AND BRAKE COMMANDS

**BREAK**                    Suspends execution of the next step until the current robot motion and all non-motion steps are completed.

This command has two effects:

1. The next program step is not executed until the robot reaches its destination.
2. The continuous path breaks between the current motion and the next motion; these two motions are not merged into one continuous path. The robot hesitates at the transition point.

In the following example, the BREAK command ensures the robot reaches a point 500 mm away from point *a* and then creates the location variable *pt1*. Without the BREAK command, the location *pt1* is created when the robot is in motion to the point 500 mm away from *a*.

## AS LANGUAGE PROGRAM INSTRUCTIONS

Example:

```
$EDIT breaker
1JMOVE a
1?
2DRAW ,,500
2?
3BREAK
3?
4HERE pt1
4?
```

**BRAKE** Stops current robot motion immediately and skips to the next step.

### 5.2.7 DELAY AND STABLE COMMANDS

**DELAY** **time**

**time:** Stops robot motion for the specified time, in seconds. The delay instruction is processed as a robot motion, but is considered a non-motion instruction. While the robot motion is stopped by the DELAY instruction, program execution of non-motion instructions continues, until the next motion instruction is encountered.

**Example:** Robot motion stops at location *bb* for five seconds and the SIGNAL command in step 4 is executed during the time delay. After five seconds, robot motion begins with the execution of step 5

```
$EDIT pg02
1JMOVE aa
1?
2LMOVE bb
2?
3DELAY 5
3?
4SIGNAL 1
4?
5JMOVE cc
```

#### NOTE

The time specified in the DELAY command begins when the tool center point enters the accuracy range of location *bb*.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### STABLE            time

time: Causes the robot to pause for a specified time when the robot reaches the programmed point. If the robot is placed in a hold condition, the time specified by the STABLE instruction is reset. When the robot is returned to run, the timing cycle starts over.

This instruction is not affected by the accuracy setting as the DELAY instruction. DELAY allows timing to start when the robot is within the accuracy range of the point. The STABLE instruction does not allow timing to begin until the robot reaches the exact location.

### 5.2.8 ACCURACY AND SPEED COMMANDS

#### ACCURACY            value ALWAYS

Sets robot positioning accuracy.

value: Distance for accuracy range in millimeters.

ALWAYS: If omitted, only the next motion instruction is affected. When the argument ALWAYS is specified, the accuracy setting is effective until the next ACCURACY instruction is executed.

In an AS Language program the default accuracy is used when accuracy is not defined at the beginning of the program.

Example: Accuracy is set at 100 mm for steps 2 through 4. The accuracy for step 6 is 1 mm and 100 mm for step 7, since **ALWAYS** is not specified in step 6.

```
$EDIT pg10
1ACCURACY 100 ALWAYS
1?
2LMOVE aa
2?
3LMOVE bb
3?
4JMOVE cc
4?
5ACCURACY 1
5?
6JMOVE dd
6?
7LMOVE aa
7?
```

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### **SPEED**                    **value ALWAYS**

Sets the robot motion speed for program steps.

**value:**                    The program step speed set as a percentage of monitor speed. Specify the speed in percentages within the range 0.01 to 100 percent. If mm/s (mm/second) or mm/min (mm/minute) is specified after the value, the absolute speed is set.

**ALWAYS:**                When the argument ALWAYS is specified, the setting is effective until the next SPEED instruction is executed. When not specified, only the next robot motion is affected.

This command sets robot motion speed as a percentage of monitor speed. The actual robot speed is determined by the product of the program speed (set by this instruction) and the monitor speed (set by the SPEED command in the monitor mode).

For a timed motion, enter the time value followed by an "s" (seconds).

When a value without a unit is designated, the value is accepted as a percentage of the maximum speed.

When an absolute speed is specified, it represents the tool tip speed in linear interpolated motion. Maximum tool tip speed is available from the robot specifications sheet for each robot model.

For joint interpolated motion, the joint speed is calculated and converted into an absolute value expressed in mm/s.

<b>Examples:</b>	<b>SPEED 50</b>	Sets the next motion speed to 50%.
	<b>SPEED 75 ALWAYS</b>	Sets the motion speed to 75% until changed by another SPEED command.
	<b>SPEED 20 mm/s</b>	Sets the tool tip speed to 20 mm per second for the next linear motion command.
	<b>SPEED 6000 mm/min</b>	Sets the tool tip speed to 6000 mm per minute for the next linear motion command.
	<b>SPEED 8s</b>	sets the time to complete the next motion.

## AS LANGUAGE PROGRAM INSTRUCTIONS

Monitor speed is set at 100 percent. Robot speed for steps 2, and 3 is 1,000 mm/s, and for step 4 **jmove cc**, the speed is converted to an equivalent value. Robot speed for step 6 is 30 percent, and the speed for step 7 is 1,000 mm/s since **always** is not specified in step 5.

```
$EDIT pg01
1SPEED 1000 MM/S ALWAYS
1?
2LMOVE aa
2?
3LMOVE bb
3?
4JMOVE cc
4?
5SPEED 30
5?
6JMOVE dd
6?
7LMOVE aa
7?e
$sp 100
```

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.2.9 ACCEL AND DECEL COMMANDS

#### ACCEL, DECEL    value ALWAYS

Sets the robot acceleration or deceleration.

value:                    Indicates a percentage of the maximum acceleration or deceleration.

The value range is from 0.01 to 100 percent. If the designated value is out of this range, the nearest value (either 0.01 or 100) is used.

ALWAYS:                If omitted, only the next motion is affected.

The acceleration when a robot motion starts and the deceleration when a robot motion ends are set by this instruction as the percentage of the maximum acceleration or deceleration.

Example:                The acceleration and deceleration are set at 70 percent of the maximum (100 percent).

```
$EDIT pg11
1ACCEL 70 ALWAYS
1?
2DECEL 70 ALWAYS
2?
3LMOVE aa
3?
4LMOVE bb
4?
5JMOVE cc
5?
```

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.2.10 CLAMP CONTROL COMMANDS

Clamp control parameters are set in auxiliary function 114.

#### **OPEN**                    **clamp\_number**

Is used to set the output signal to open the clamp at the beginning of the next robot motion.

clamp\_number:        Specifies the clamp to open. Eight clamps are available.

If the clamp\_number is omitted, clamp one is specified.

#### **OPENI**                    **clamp\_number**

Is used to set the output signal to open the clamp. The OPENI command causes a BREAK to occur if continuous path motion is in progress.

The output signal to open the clamp is sent at the completion of the current motion.

clamp\_number:        Specifies the clamp to open. Eight clamps are available.

If the clamp\_number is omitted, clamp one is specified.

Figure 5-7 shows the differences between the OPEN and OPENI commands.



## AS LANGUAGE PROGRAM INSTRUCTIONS

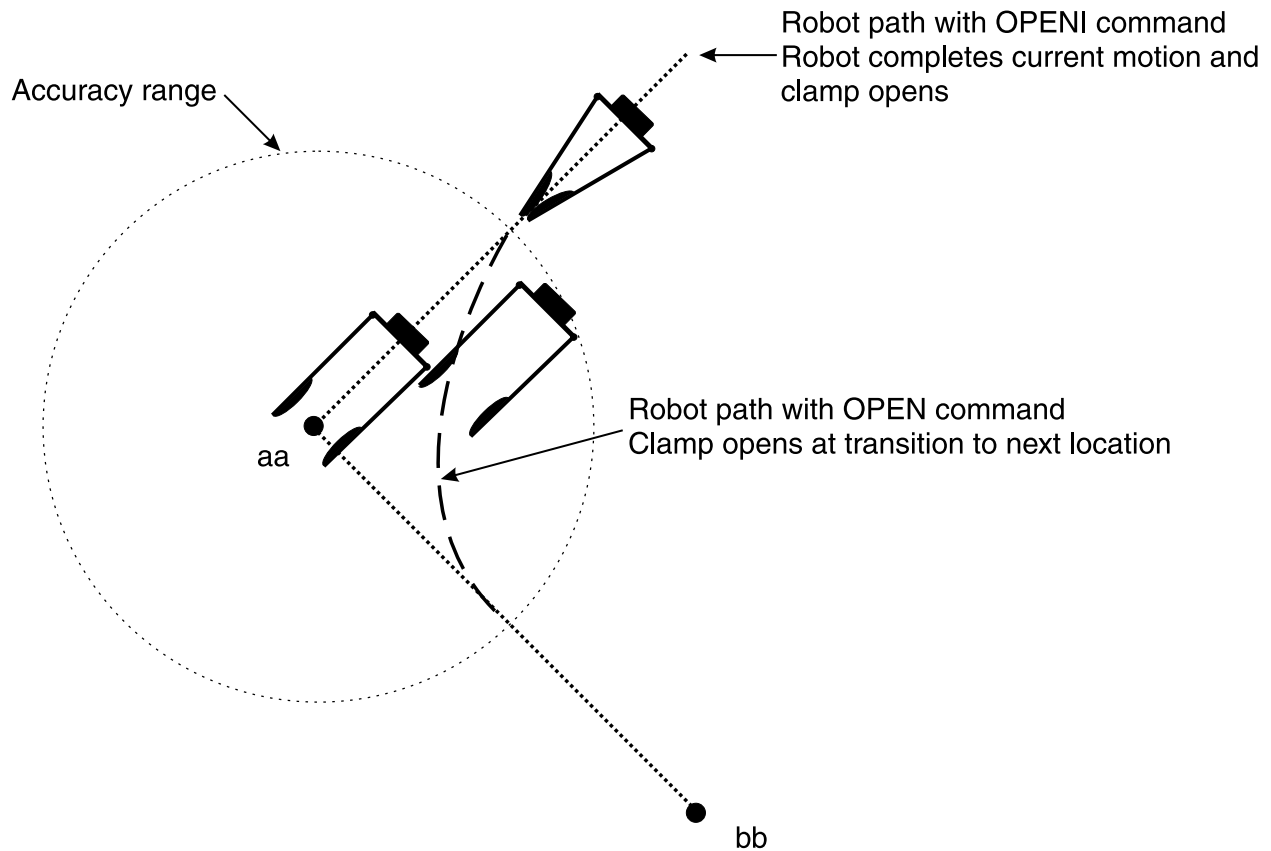


Figure 5-7 OPEN and OPENI Commands

### CLOSE

#### clamp\_number

Is used to set the output signal to close the clamp at the beginning of the next robot motion.

clamp\_number: Specifies the clamp to close. Eight clamps are available.

If the clamp\_number is omitted, clamp one is specified.

### CLOSEI

#### clamp\_number

The CLOSEI command is used to set the output signal to close the clamp. The CLOSEI command causes a BREAK to occur if continuous path motion is in progress.

The output signal to close the clamp is sent at the completion of the current motion.

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

clamp\_number: Specifies the clamp to close. Eight clamps are available.

If the clamp number is omitted, clamp one is specified.

### **RELAX**                    **clamp\_number**

Sets the output signals to release pneumatic pressure at the clamp at the beginning of the next robot motion.

clamp\_number: Specifies the clamp to be relaxed. Eight clamps are available.

If the clamp number is omitted, clamp one is specified.

### **RELAXI**                    **clamp\_number**

Sets the output signals to release pneumatic pressure at the clamp.

The RELAXI command causes a BREAK to occur if continuous path motion is in progress. The output signals to release pneumatic pressure to the clamp are set at the completion of the current motion.

clamp\_number: Specifies the clamp to be relaxed. Eight clamps are available.

If the clamp number is omitted, clamp one is specified.

### **UWRIST** **DWRIST**

Forces a configuration (posture) change during the next motion so the angle of Joint 5 has 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 for a precision location.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.2.11 CONFIGURATION COMMANDS

#### **RIGHTY**

#### **LEFTY**

Forces a robot configuration (posture) change during the next motion so 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 for a precision location.

#### **ABOVE**

#### **BELOW**

Forces a robot configuration (posture) change during the next motion so 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 for a precision location.

### 5.3 PROGRAM CONTROL

#### **GOTO**

Causes program execution to jump to a substitute program.

#### **IF**

Performs a conditional jump command.

#### **CALL**

Causes execution to branch to a subroutine.

#### **RETURN**

Returns control to the caller program.

#### **WAIT**

Causes execution to wait until the specified condition is set.

#### **SWAIT**

Causes execution to wait until the desired signal states are set.

#### **TWAIT**

Causes execution to wait until the specified time has elapsed.

#### **PAUSE**

Stops execution temporarily.

#### **HALT**

Stops execution and does not allow it to be resumed.

#### **STOP**

Terminates the current execution cycle.

#### **SCALL**

Calls a program assigned to a string variable.

#### **LOCK**

Sets subroutine priorities for process control programs.

#### **ONE**

Used in process control programs to call a program at an error.

#### **RETURNE**

Used in process control programs to return to the step after the step where an error occurred.

#### **ON**

Starts monitoring the signal for interrupt handling.

#### **ONI**

Starts monitoring the signal for immediate interrupt handling.

#### **IGNORE**

Cancels signal monitoring started by the ON or ONI instruction.

#### **EXTCALL**

Selects an external program number via RPS.

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.3.1 GOTO AND IF COMMANDS

**GOTO**                    label IF condition

This instruction causes execution to branch to the step which has the specified label. If a condition is given, execution branches only when the condition is TRUE. Otherwise, execution proceeds to the next step in order.

label:                    Label assigned to a program step which execution branches to. Labels can be a combination of alphanumeric characters up to fifteen characters long.

IF:  
condition:                Condition of the branch (represented in logical expression). If omitted, execution branches unconditionally.

Examples:                GOTO 100 causes execution to branch unconditionally to the step with label 100. If there is no line with the label 100 in the same program, an error occurs.

GOTO 200 IF  $n == 3$  causes execution to branch to a step with label 200 if variable  $n$  is equal to 3, otherwise, execution proceeds to the next step.

---

**AS LANGUAGE PROGRAM INSTRUCTIONS**

```
$EDIT test
1 100 HOME
1?
2JMOVE aa
2?
3x=x+1
3?
4GOTO 200 IF x>5
4?
5LAPPRO bb,50
5?
6LMOVE bb
6?
7LMOVE cc
7?
8LDEPART 50
8?
9GOTO 100
9?
10 200 HOME
10?
11BREAK
11?
12PRINT "Program complete"
12?
```

## AS LANGUAGE PROGRAM INSTRUCTIONS

### **IF**                    **condition GOTO label**

Causes execution to branch to the specified label if the condition is TRUE.

condition:            Logical expression evaluated during step execution.

Example:               $n==0, n>3, m+n>0$

GOTO:

label:                Label of program step to which execution branches to.

Examples:            IF N>3 GOTO 100

If the value of the integer variable n is greater than 3, execution branches to the step with label 100, otherwise, execution continues with the next step.

IF SIG(1001) AND SIG(-1002) GOTO 250

If input signal 1001 is ON and input signal 1002 is OFF, execution branches to the step with label 250, otherwise, execution continues with the next step.

### 5.3.2 CALL AND RETURN COMMANDS

#### **CALL**                **program\_name**

Causes execution to jump to the specified program (subroutine). After completion of the subroutine, execution returns to the step following the CALL instruction.

This instruction suspends execution of the current program. Execution branches to the first step of the subroutine program.

A subroutine cannot be called by both the robot control program and a PC program at the same time.

Multiple calls of a subroutine (calling another subroutine from the one being executed) are possible, up to twenty maximum.

A subroutine cannot call itself. The example on the next page illustrates the CALL instruction.

program\_name:      Name of the program to call.

## AS LANGUAGE PROGRAM INSTRUCTIONS

Example: In pg00 at step 2, if input signal 1001 is high, step 3 is executed (CALL pg02) and execution branches to pg02.

If input signal 1001 is low, step 5 is executed.

In pg02, step 3 causes execution to branch to pg07. After pg07 is executed, execution returns to the line following the call command in pg02.

The execution continues from step 4, and if input signal 1020 is high (step 6), the program branches to label 150 and pg07 is executed again.

If input signal 1020 is low, then execution returns to the line following the call command in pg00.

```
$LIST pg00
1 HOME
2 If SIG (1001) THEN
3 CALL PG02
4 END
5 HOME 2
```

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

```
$LIST pg07
1 JMOVE a1
2 C1MOVE a2
3 C2MOVE a3
4 C1MOVE a4
5 C2MOVE a1
```

## AS LANGUAGE PROGRAM INSTRUCTIONS

**RETURN** Terminates execution of the subroutine, and returns to the step following the call command in the program which called the subroutine.

If a call command in another program does not exist, as in the case of a subroutine executed by the EXECUTE instruction, the instruction is terminated as in executing a STOP instruction. If there are remaining cycles to execute, execution continues with the first step.

If no RETURN instruction exists in the subroutine, the control returns to the line following the call command in the program execution branched from.

The RETURN instruction is assumed at the end of a subroutine.

### 5.3.3 WAIT, SWAIT, AND TWAIT COMMANDS

**WAIT**                    **condition**

Causes execution to wait at the current step until the specified condition becomes TRUE.

condition:                Real value expression

This instruction is used to suspend program execution until the desired condition is encountered. A running WAIT instruction is canceled by the CONTINUE NEXT command.

Examples:                WAIT SIG (1001, -1003) causes program execution to wait until external input signal 1001 is turned ON and signal 1003 is turned OFF.

WAIT TIMER (1)>10 causes program execution to wait until the value of Timer 1 exceeds ten seconds.



---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### **SWAIT**                    **signal\_number**

Causes program execution to wait until the specified external (I/O) signals or internal (I/O) signals are set to the given states.

**signal\_number:**        Represents the number of an external or internal (I/O) signal. If assigned with a minus sign, the desired state is OFF.

If all specified signal states are satisfied, execution proceeds to the next step. If not satisfied, execution waits at the current step.

A running SWAIT instruction is canceled by the CONTINUE NEXT command. The same function as SWAIT is achieved using the WAIT instruction.

**Examples:**            SWAIT 1001, 1002 causes program execution to wait until both external input signals 1001 and 1002 are ON.

SWAIT 1, -2001 causes program execution to wait until the external output signal 1 is ON and the internal input signal 2001 is OFF.

### **TWAIT**                    **time**

Causes program execution to wait at the current step for the specified period of time (in seconds).

A running TWAIT instruction is canceled by the CONTINUE NEXT command.

**Examples:**            TWAIT 0.5, causes execution to wait for 0.5 seconds.

TWAIT fix, causes execution to wait for the time set by the variable fix.

## 5.3.4 PAUSE, HALT, AND STOP COMMANDS

**PAUSE**                    Stops program execution which can later be resumed.

This instruction stops execution and displays a program paused message on the terminal. Execution is resumed with the CONTINUE command. To check a program, insert a PAUSE command at any point to stop execution for a moment. Current values of variables may be checked.

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

- HALT** Stops program execution. Execution cannot be resumed.
- Regardless of remaining execution cycles, program execution is terminated, and a message appears on the terminal.
- Execution stopped by this instruction cannot be resumed with the CONTINUE command.
- STOP** Terminates the current execution cycle. If there are remaining cycles to be completed, execution returns to the first step, otherwise, execution ends. This instruction has a different effect than the HALT instruction.
- 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 program performs the same function as the STOP instruction.
- A main program is a program initiated by such commands as EXECUTE, STEP, and PCEXECUTE. A subroutine is a program called by another program with the CALL, ON, or ONI instruction.
- After a program has been stopped by a STOP instruction, it is impossible to resume execution with the CONTINUE command.

### 5.3.5 SCALL, LOCK, ONE, RETURNE COMMANDS

- SCALL** **program\_name**
- Similar to the CALL command. The SCALL command is used with a program name assigned to a string variable.
- program\_name: Name of the program to call.
- LOCK** **priority**
- Used in process control (PC) programs to set the execution priority of subroutines.
- priority: The priority setting range is 0 to 127. If subroutines called by the PC program do not have a LOCK priority assigned, 0 is assumed.

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### ONE **program\_name.**

Used in process control (PC) programs 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 use robot motion instructions.

**program\_name:** name of the program to call when an error occurs.

### RETURNE (return error) Is a PC program command used to return program execution to the step following the step where an error occurred.

The RETURNE command is used in conjunction with the ONE command to execute the next step of a program after an error occurs.

### 5.3.6 ON, ONI, IGNORE, AND EXTCALL COMMANDS

#### ON **signal\_number (CALL program\_name) or (GOTO label)** ONI **signal\_number (CALL program\_name) or (GOTO label)**

Starts monitoring the specified external input signal, and when the signal changes to the specified state, calls the specified subroutine, or jumps to the given label.

The ON instruction waits for completion of the current motion. The ONI instruction terminates the current robot motion.

**signal\_number:** Number of the input signal to monitor. Signal numbers from 1001 to 1032 (external input signals) or from 2001 to 2032 (internal signals) are used. A positive signal number specifies the signal change from OFF to ON causes an interruption, and a negative signal number specifies the signal change from ON to OFF causes an interruption.

**program\_name:** Name of subroutine program called when the signal change is detected. If omitted, execution proceeds to the next step.

**label:** Alphanumeric label assigned to the desired program step to branch to when the signal change is detected.

## AS LANGUAGE PROGRAM INSTRUCTIONS

When a RETURN instruction is encountered in the interruption-handling subroutine, execution returns to the step following the instruction being processed when the interruption occurred.

Signal monitoring is canceled in the following cases:

- An IGNORE instruction is executed for the signal(s) specified by an ON or ONI instruction.
- A corresponding interruption occurs.
- An ON (or ONI) instruction is executed for the same signal.

### NOTE

The interruption does not occur according to the state of the signal itself. The interruption occurs at the time the signal state changes. Assuming that the signal change from OFF to ON causes an interruption, and the signal is ON when an ON instruction is executed, the interruption does not occur until the signal is turned OFF then turned ON again.

To detect signal changes, the signal must be stable for at least 50 msec.

Signal monitoring begins immediately after an ON (or ONI) instruction is executed.

Once the ON/ONI command is executed, it must be re-processed to continue monitoring.

Signal changes are ignored while execution is stopped.

Examples:

#### **ONI -1001 CALL part**

Starts monitoring the external input signal 1001. When this signal changes from ON to OFF, the current robot motion is stopped at once, and execution branches to program part.

#### **ON test CALL delay**

Starts monitoring the external input signal indicated by the variable test. When the signal is detected, execution branches to program delay after the current step is completed.

## AS LANGUAGE PROGRAM INSTRUCTIONS

```
$LIST pg15  
1 HOME  
2 ONI 1001 CALL pg01  
3 ONI 1002 CALL pg02  
4 JMOVE aa  
5 JMOVE bb  
7 HOME
```

```
$LIST pg01  
1 JMOVE A1  
2 JMOVE B1  
3 JMOVE C1  
4 JMOVE D1
```

```
$LIST pg02  
1 JMOVE A2  
2 JMOVE B2  
3 JMOVE C2  
4 JMOVE D2
```

In program pg15 above, monitoring of inputs 1001 and 1002 begins when step numbers 1 and 2 are executed respectively. Any time after the first scan, if inputs 1001 or 1002 change state, a transition (TRUE to FALSE or FALSE to TRUE) must occur for an ON or ONI instruction to execute.

### NOTE

If the CONTINUE NEXT command is issued when the system is waiting for the RPS-ON signal to be turned ON, the EXTCALL instruction is terminated and execution proceeds to the next step.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### IGNORE `signal_number`

Cancels signal monitoring started by the ON or ONI instruction. Input signals to be monitored must be installed and connected.

`signal_number`: Represents the number of the signal within the range from 1001 to 1032 (external input) or from 2001 to 2032 (internal).

Example: **Ignore 1002** stops monitoring signal 1002 when an **ONI** instruction using that signal number is executed.

### EXTCALL

The EXTCALL instruction is used to select an external program number via the random program select (RPS) input signals.

The sequence of operation for the EXTCALL instruction is as follows:

1. RPS-ST signal is turned ON.
2. The program waits for the RPS-ON signal to be turned on.
3. When the RPS-ON signal is turned ON, a binary number is read from the external input signals allocated to external program numbers. The inputs used for external program numbers include binary RPS1 through RPS64. The EXTCALL instruction selects programs with a (Pg) prefix. If the external number is greater than or equal to 100, a program with a Pg prefix and a three digit number is selected (Pgxxx). If the number is in the range of 10 to 99, Pgxx is selected, and if the number is smaller than 10, program Pgx is selected.

#### NOTE

If the CONTINUE NEXT command is issued when the system is waiting for the RPS-ON signal to be turned ON, the EXTCALL instruction is terminated and execution proceeds to the next step

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

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.4 MESSAGE DISPLAY COMMANDS

PRINT	Displays a message in the upper area of the terminal display.
TYPE	Displays a message in the lower area on the terminal display.
PROMPT	Displays a message at the reverse image bar and requires user input from the keyboard.

#### 5.4.1 PRINT AND TYPE COMMANDS

**PRINT**  
**TYPE**

**data/format**

Displays data on the keyboard screen.

PRINT/TYPE 1: Displays data on a PC interfaced with the C controller (see unit 8).

PRINT/TYPE 2: Changes the current display to the keyboard screen.

Data consists of the following components separated by commas.

- Character string expression, such as “count=”.
- Real value expression (the value is calculated and displayed), such as count.
- Format information which controls the format of the output message, such as /D, /S.

The following format specifications are used to control the format of numeric values. The setting is effective for subsequent parameters until another format is given. In any format, if the value is too large to be displayed in the given width, asterisks (\*) fill the area. It is possible to display up to 128 characters in a line. To display more than 128 characters, use the format /S described on the following page.

/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.

---

**AS LANGUAGE PROGRAM INSTRUCTIONS**

**/Em.n** The value is displayed in scientific notation, whole numbers in the m digits field and the fraction in n digits field. The value of m should be larger than n by six or more, and smaller than thirty-two.

Example: 1.234E+02

**/Fm.n** The value is displayed in fixed point notation, whole numbers in the m digits field with the fraction in the n digits field.

Example: -1.234

**/Gm.n** If the value is 0.01 or greater 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.

**/Hn** 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 arguments are used to insert special characters between character strings.

**/Cn** A set of carriage returns (CR) and line feeds (LF) is output n times. If this argument is the first or the last in the PRINT instruction, n blank lines are displayed on the terminal, otherwise, n-1 blank lines.

**/S** This argument suppresses the output of (CR) and (LF) at the beginning of a message. This is effective only when /S is the first argument.

**/Un** The cursor is moved up by n lines.

**/Vn** If n=1, the upper area of the terminal is enlarged.  
If n=0, the lower area of the terminal is enlarged.

**/Xn** n spaces are output.



**AS LANGUAGE PROGRAM INSTRUCTIONS**

```
$LIST pg08
1 FOR n = 30 to 90 STEP 15
2 x = COS(n)
3 y = SIN(n)
4 TYPE /X10,"COS ",n," = ",/F5.2,x,/X10,"SIN ",n," = ",/f5.2,y
5 END

$      COS 30 = 0.87          SIN 30 = 0.50
      COS 45 = 0.71          SIN 45 = 0.71
      COS 60 = 0.50          SIN 60 = 0.87
      COS 75 = 0.26          SIN 75 = 0.97
      COS 90 = 0.00          SIN 90 = 1.00
```

In the above program, the type instruction is used to display the values of the trigonometric functions Sine and Cosine. The /X10 separates the output display by ten (10) spaces. The /F5.2 is used to specify the format in which the output is displayed. The first number, 5, indicates the whole field (including decimal point). The second number, 2, indicates the field allocated for the fractional part.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.4.2 PROMPT COMMAND

**PROMPT**            **“string message”, variable**

“string message”: The PROMPT command displays the given string text on the keyboard screen and waits for the user to input a value to assign to the variable. If the user presses the RETURN key or CTRL C, the variable is set to the last value entered.

PRINT/TYPE 1:      Displays data on a PC interfaced with the C controller (see unit 8).

PRINT/TYPE 2:      Changes the current display to the keyboard screen.

```
$LIST prompter
1 HOME 2
2 PROMPT "Please enter the number of parts", parts
3 JMOVE #pt1
4 FOR X = 1 TO parts
5 JAPPRO A, 200
6 LMOVE A
7 LMOVE AAI
8 LD 200
9 JM B
10 PRINT parts
11 END
12 HOME
```

## AS LANGUAGE PROGRAM INSTRUCTIONS

In the program prompter the robot moves to the HOME 2 position and displays the message “Please enter the number of parts”. Motion to point #pt1 does not begin until the user enters a value for the variable *parts*. The FOR TO END loop is a structure that results in the repeated execution of steps 5 through 10. This block of steps is repeated *n* times, where *n* is equal to the value entered for the variable *parts*.

### 5.5 EXTERNAL OUTPUT SIGNAL CONTROL

SIGNAL	Turns specified signals ON or OFF.
PULSE	Turns signal ON for a specified time.
DYLSIG	Turns signal ON after a specified time has elapsed.
BITS	Sets the states of a group of signals.
RUNMASK	Masks a group of signals. (These signals cannot remain ON after execution stops.)
RESET	Turns OFF all external output signals.

#### 5.5.1 SIGNAL, PULSE, AND DYLSIG COMMANDS

##### **SIGNAL**                    **signal\_number**

Turns the specified external output signals or internal I/O signals ON or OFF.

**signal\_number:**        A real value expression used to determine external binary output signal numbers or internal I/O signal numbers. Positive values turn signals ON. Negative values turn signals OFF. If the signal is zero, it is ignored and no signals are changed.

Signal numbers 1 to 256 are external binary output signals, and signals from 2001 to 2256 are internal I/O signals. Binary input signals (1001 to 1256) cannot be specified. For external output signals, only the signals which are installed and configured as outputs can be used. To check the current I/O signal configuration, use the IO command.

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### **PULSE**                    **signal\_number, time**

Turns the specified signal ON for a specified period of time.

signal\_number:    Number of an external output signal or an internal signal.

time:                Time in seconds for which the signal is kept ON. If omitted, it is set to 0.2 seconds.

### **DLYSIG**                    **signal\_number, time**

Turns ON/OFF the specified signal after a given time has elapsed.

signal\_number:    Number of an external output signal or an internal signal. Positive values indicate the signal is turned ON, and negative values indicate the signal is turned OFF. The signal number must be from 1 to 32 or from 2001 to 2032.

time:                Period of time (seconds).

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.5.2 BITS COMMAND

**BITS**                    **starting\_signal\_number, number\_of\_signals = value**

Arranges signals in a binary pattern and sets signal states according to the binary equivalent of the decimal value specified. The signals used in the BITS command can be either external output signals or internal signals.

**starting\_signal\_number:**            The first signal to be set. This signal is the least significant bit (LSB) in the binary pattern.

**number\_of\_signals:**                Number of signals (number of bits); maximum number allowed is sixteen.

**value:**                                Value which represents the desired signal states. If the binary notation of this value has more bits than the number of signals to set, only the number of signals, specified by **number\_of\_signals**, from the lowest number, specified by **starting\_signal\_number**, are affected. If omitted, the current signal states are displayed in decimal notation.

**Example:**                              In the example below, the output signal 1 is the least significant bit, the number of bits in the binary pattern is 4 (signals 1, 2, 3, and 4), and the decimal value is 10. The result of the command **BITS 1,4=10** sets external output signals 2 and 4 to the ON state as shown below.

\$BITS 1,4=10									
\$io									
32-1	1010	1010	0100	0100	0000	0000	0000	0000	1010
1032-1001	0000	0000	0000	0000	0000	0000	0000	0000	0000
2032-2001	0000	0000	0000	0000	0000	0000	0000	0000	0000

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.5.3 RESET AND RUNMASK COMMANDS

**RESET** Turns OFF all external output signals that are not dedicated.

**RUNMASK** **starting\_ signal\_ number, number\_ of \_signals**

Allows signals to be ON only while the program is executing. If the signal is turned ON by a SIGNAL, PULSE, or DLYSIG instruction, it is turned OFF when execution is interrupted.

starting\_ signal\_  
number:

Number of the first signal in the group of signals to mask. A negative value indicates that the mask function is canceled. These unmasked signals are not turned OFF when execution is interrupted.

number\_ of\_  
signals:

Number of signals masked; default is one.

Whenever execution stops, the masked signals are turned OFF immediately. Consider the case where execution is interrupted after the RUNMASK instruction is executed. If execution is resumed with a CONTINUE (DO or STEP) command, the signals are restored only while execution continues.

```
$LIST pg10
1 JMOVE aa
2 SIGNAL 1,3
3 JMOVE bb
4 PULSE 10,3
5 JMOVE cc
6 DLYSIG 15,5
7 RUNMASK 1,15
```

In the above example, output signals 1 and 3 turn ON when step 2 is executed. When step 4 is executed, output signal 10 turns ON for three seconds, then turns OFF. Output signal 15 turns ON five seconds after step 6 is executed. Output signals 1 through 15 are monitored by the instruction in step 7, which de-energizes or resets outputs 1 through 15 when program execution is stopped.

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.6 DEFINITION OF VARIABLES

HERE	Defines a location variable to be equal to the current robot location.
POINT	Defines a location variable.
POINT/X	Defines X component of a transformation variable.
POINT/Y	Defines Y component of a transformation variable.
POINT/Z	Defines Z component of a transformation variable.
POINT/7	Defines seventh axis component of a transformation variable.
POINT/OAT	Defines OAT components of a transformation variable.
DECOMPOSE	Extracts component values from a location.
TOOL	Sets system internal transformation value of the tool.
BASE	Changes base coordinate system.
LLIMIT	Sets the lower limit of the robot motion.
ULIMIT	Sets the upper limit of the robot motion.
TIMER	Sets a timer.
ON	Turns system switches ON.
OFF	Turns system switches OFF.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.6.1 HERE AND POINT COMMANDS

**HERE**                    **location\_name**

Sets the value of the location variable to be equal to the current robot location.

location\_name:        Specifies the name of a location variable (transformation or precision variable, or compound expression).

If the location variable is assigned in a compound transformation expression, the right-most variable is defined. If any transformation variable in the compound transformation expression is undefined, an error occurs.

```
$LIST pg55
1  DRIVE 1, 75, 50
2  HERE a
3  DRIVE 2, 100, 50
4  HERE b
5  JMOVE a
6  JMOVE b
```

In the above example, step 1, the DRIVE instruction is used to move a joint by the specified amount. The HERE command is used to set the current location values into the location name specified.



**AS LANGUAGE PROGRAM INSTRUCTIONS****POINT**                    **location\_name1 = location\_name2**

Assigns the value defined by the right-hand expression to the location on the left side of the assignment symbol (=).

location\_name1:    Name of the location variable or compound transformation expression to define.

location\_name2:    Previously defined location variable value used to define the location on the left side of the assignment symbol (=).

If the location on the right side is not defined, or if the location variable types differ, this instruction results in an error. If a compound transformation expression is given on the left side, the right-most variable is defined. When any other component variables are undefined, this instruction results in an error.

**POINT/X  
POINT/Y  
POINT/Z  
POINT/7  
POINT/OAT**

**transformation\_variable1 = transformation\_variable 2**

Assigns values of the components X, Y, Z, seventh axis, or OAT of the transformation value on the right-hand side to the corresponding component of the transformation variable on the left-hand side.

transformation\_variable1:    Name of a variable for component assignment.

transformation\_variable2:    Name of a variable providing component value.

```
$LIST/P pg60
1 Home
2 JMOVE aa
3 POINT bb = aa
4 POINT/Z bb = start
5 JMOVE bb
```

In the above example, the POINT command is used to assign the location value of aa to bb. The POINT/Z is used to assign the Z component value of location variable start to location variable bb.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.6.2 DECOMPOSE COMMAND

**DECOMPOSE:**    **array\_variable\_name [index] = location\_name or real\_variable\_name**

Extracts all component values of a location, or real variable value.

array\_variable\_  
name:

Name of a real variable array where the component values are stored.

index:

Optional number which indicates the first element to use. If omitted, zero (0) is assumed.

location\_name:

Name of a location whose components are extracted.

This instruction extracts component values of the specified location, and assigns these values to the consecutive elements of the named array.

If a given location is a transformation value, six elements corresponding to XYZOAT are defined. If it is a joint displacement value, elements corresponding to each of the joint components are defined.

real\_variable\_  
name:

Name of real variable whose value is assigned to the array.

```
$LIST/P pg62
1  HERE a
2  DECOMPOSE a[1] = a
3  FOR i = 1 to 6
4  TYPE "a [",i,"] = ", a[i]
5  END

$LIST/r a*
a[1] = 16.78
a[2] = 25.67
a[3] = 30.08
a[4] = 12.34
a[5] = 45.76
a[6] = 10.99
```

## AS LANGUAGE PROGRAM INSTRUCTIONS

In the example on the previous page, the current location of the robot is defined as *a*. The DECOMPOSE instruction extracts component values one through six of *a*. The program instructions between the FOR and END statement are executed repeatedly, and the TYPE instruction displays the component values of *a* individually.

### 5.6.3 TOOL AND BASE COMMANDS

#### TOOL

##### **transformation\_value**

Sets the system's internal transformation value indicating the position and direction of the tool tip, relative to the tool mounting flange.

If a NULL transformation value is given, the tool is set to be null tool (0,0,0,0,0,0).

This instruction causes a BREAK in the continuous path motion and the value of the tool transformation is changed at the next location.

For further information on this instruction, refer to the description of the TOOL monitor command, section 4.5.2.

#### BASE

##### **transformation\_value**

Changes the robot base coordinate system by the given transformation value.

If a NULL transformation value is given, the base coordinate system is set to be NULL base (0,0,0,0,0,0).

This instruction causes a BREAK in the continuous path motion, and changes the base coordinate system at the next location.

For further information on this instruction, refer to the description of the BASE command, section 4.5.2.

### 5.6.4 LLIMIT AND ULIMIT COMMANDS

#### LLIMIT, ULIMIT

##### **precision\_value (joint angle)**

Sets the value (degrees) of the lower limit/upper limit of the motion range of the robot. Changes to the limits of the robot affect all programs in memory.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.6.5 TIMER AND SWITCH COMMANDS

**TIMER**                    **timer\_number = time**

Sets the time of the specified timer.

**timer\_**  
**number:**                    Number of the timer to set. Ten timers are available, one through ten.

**time:**                      Time (in seconds) which is allocated to the timer.

When the instruction is executed, the specified timer is set to have the specified time. The value of the timer is obtained using the TIMER function.

**Example:**                    In this example the TIMER command is used to obtain a cycle time for the program in seconds.

```
$LIST pg_cycletime
1 TIMER 1 = 0
2 JMOVE aa
3 JMOVE bb
4 LMOVE cc
5 LDEPART 200
6 HOME
7 TYPE "cycle time = " ,/F5.3,Timer (1), " seconds"
```

```
cycle time = 5.678 seconds
```

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### SWITCH\_NAME ON

### SWITCH\_NAME OFF

Turns the specified system switches ON or OFF. The current setting of the system switch is displayed with the SWITCH command. In the example below, the CP and ARC switch is turned off.

```
$LIST pg08  
1 CP, ARC OFF  
2 JMOVE A1  
3 JMOVE B1  
4 JMOVE C1  
5 JMOVE D1
```

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.7 CONTROL FLOW STRUCTURES

IF THEN ELSE	Permits program instructions to be executed if the logical IF expression is true.
WHILE DO	Causes the repeated execution of a group of instructions while a given expression is true.
DO UNTIL	Provides a way to control the execution of a group of instructions based on a control expression.
FOR TO	Results in the repeated execution of a block of instructions.
CASE OF	Evaluates index variables and executes associated program instructions.

Control flow structures are special types of program instructions that consist of more than one line of code to form a group or block of steps. Depending on the structure used, these blocks can provide sequence control, decision making, looping, and the ability to select a set of actions from many possible sets. The following sections discuss the five control flow structures available with the AS Language.

#### 5.7.1 IF THEN ELSE COMMAND

```
IF THEN  
ELSE  
END           IF (logical expression) THEN  
  
                ... instruction(s) to execute  
                when the condition is true  
  
                ...  
                ELSE (optional)  
  
                ...  
                ... instruction(s) to execute  
                when the condition is false  
  
                ...  
                END  
                ... continue execution of program here
```

This block structure is an extension to the logical IF instruction. This block permits execution of more than one instruction if the logical expression being tested is true. The logical IF statement

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

permits only a single statement to be given after the logical expression. The block structure also permits an optional ELSE capability to the IF THEN ELSE block. It includes instructions to be executed whenever the logical condition is false.

When the logical expression is true, the instructions following the IF (logical expression) THEN statement, and preceding the ELSE statement are executed. Control of execution is then transferred to the statement following the END statement. When the logical expression is false, the instructions following the ELSE statement, and preceding the END statement are executed. Control of execution is then transferred to the step following the END.

Example:

For example, a segment of the AS program shown below sets the program speed to 10 percent if the value of the variable *n* is greater than five. If the variable is less than five, the program speed is set to 20 percent.

```
21  IF n>5 THEN
22  v=10
23  ELSE
24  v=20
25  END
26  SPEED v ALWAYS
```

The ELSE statement is optional. When this statement is omitted, and the logical expression is false, control of execution is transferred to the step following the END statement. For example:

```
51  IF x>0 THEN
52  pick = pick + x
53  count = count + 1
54  END
```

This block adds *x* to the variable *pick* and increments the counter count by one, only if the value of *x* is positive.

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.7.2 WHILE DO COMMAND

#### WHILE DO END    **WHILE** logical expression **DO**

```
...  
...    instruction(s) to execute while the condition is true  
...  
END  
...    continue execution of program here
```

The WHILE DO block causes the repeated execution of a group of instructions while a given expression is true. If the expression becomes false, execution of the END statement is processed. The general form of this block structure is shown above.

If the logical expression is true, the instructions following the WHILE DO statement and preceding the END statement are executed. Control of execution returns to the WHILE DO statement and re-tests the logical expression. This flow of execution continues repeatedly until the logical expression becomes false, or until another conditional statement inside the block causes a transfer out of the block structure. When the logical expression is false (which includes the first condition), control of execution resumes at the step following the END statement.

#### Example:

The following example uses a WHILE structure to monitor a combination of input signals to determine when a sequence of motions should stop. In this example, if the signal from either part feeder becomes zero (assumed to indicate the feeder is empty), then the repetitive motion of the robot stops and the program continues at step 27.

```
23    WHILE SIG (feeder1, feeder2) DO  
24    CALL part1  
25    CALL part2  
26    END  
27    HOME
```

#### NOTE

If either feeder is empty when the WHILE structure is first encountered, then execution immediately skips to step 27.



## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.7.2.1 LOOP CONTROL VARIABLES

Loop control variables are often used with WHILE DO control structures. The loop control variable is one whose value is continually tested in the logical expression. This variable must be initialized prior to entering the WHILE DO loop, for the loop to be executed the correct number of times. The value of the loop control variable must be updated within each loop execution. If the loop control variable is not updated, the logical expression may never become false and loop exit does not occur. Update of the loop control variable is usually the last executable instruction in the WHILE loop.

```
90   n=0
91   WHILE n<50 DO
92   JMOVE weld
93   n=n+1
94   END
```

The END statement is a non-executable statement. If control of execution is transferred to an instruction inside the block structure, the instructions are executed in a sequential manner, and the END statement is ignored. Control of execution may be transferred to a statement outside the block structure at any time. The WHILE DO statement cannot be the range statement in a DO loop, nor the statement used in a logical IF.

### 5.7.3 DO UNTIL COMMAND

```
DO
UNTIL      DO
...
...   statement(s) to execute
...
UNTIL logical expression
```

The DO UNTIL structure provides a way to control the execution of a group of instructions based on a control expression. The general format of this structure is shown above. When this structure is used, some action occurs within the group of enclosed instructions that changes the result of the logical expression from TRUE to FALSE. When FALSE, the structure is exited.

The DO UNTIL loop performs the same logic as the previous DO WHILE loop. The difference is, the DO UNTIL structure assures instructions in the body of the loop are executed at least once.

## AS LANGUAGE PROGRAM INSTRUCTIONS

No instructions are needed between the DO and UNTIL statements. When there are no instructions, the UNTIL criterion is continuously evaluated (stays in the loop) until it is satisfied, then program execution continues with the instructions following the UNTIL instruction.

Example:

The following example uses a DO structure to control a task that involves welding two parts on a carrier. The sequence assumes that the binary signal line `buffer.full` changes to the ON state when the parts buffer becomes full. (The robot then performs a different sequence of motions.)

```
50 DO
51 CALL weldparta1
52 CALL weldparta2
53 UNTIL SIG (buffer.full)
```

### 5.7.4 FOR TO COMMAND

**FOR TO STEP  
END**

```
FOR loop_variable = start value TO end value
STEP step_value
...
instruction(s) to execute
...
END
... continue execution of program
```

The FOR END loop is a structure that results in the repeated execution of a block of instructions. The general form of this control structure is shown above.

In the FOR END structure, the user must establish a counter variable (loop variable), and specify the range of values (start value TO end value) the variable takes during execution of the loop. At the beginning of each loop, the counter is assigned a new value. If the counter is within the range of values specified (start value TO end value), the block of instructions is executed once. The looping ends when the counter has gone through the entire range of values. The following are characteristics of the FOR TO STEP END structure:

## AS LANGUAGE PROGRAM INSTRUCTIONS

The block of instructions is preceded by a FOR statement at the top of the structure and followed by an END statement at the bottom of the structure.

The FOR statement names the counter variable and specifies the beginning and ending values of the counter. These two values are expressed as constants (variables that receive values before the start of the loop), or arithmetic expressions that compute a range of values for the counter variable.

The optional STEP statement specifies the amount (step value) the counter variable is increased or decreased after each loop. If its value is a positive number, the value of the counter increases with each execution. If the step value is negative, the counter decreases. If the STEP statement is omitted, the default increment is +1.

The FOR END structure does not loop if the start value is less than the end value and the step value is negative, or the start value is greater than the end value and the step value is positive.

Example:

```
FOR i = 2 to 6 STEP 2
DRAW 100, 10 * i + 7, 50
HERE weld[i]
END
```

In the above example, the robot moves 100 mm in X, a calculated amount ( $10 * i + 7$  mm) in the Y direction, and 50 mm in the Z direction, and defines the location as weld[i].

The FOR statement in this example increments the value of "i" in increments of two.

Example:

```
i = 2, i = 4, i = 6.
```

---

## AS LANGUAGE PROGRAM INSTRUCTIONS

### 5.7.5 CASE COMMAND

```
CASE OF
VALUE
ANY
END          CASE (index variable) OF
              VALUE number1 ,..... :
              program instructions
              ...
              ...
              VALUE number2,.....:
              program instructions
              ...
              ...
              ANY
              program instructions
              ...
              ...
              END
```

The CASE block structure selects a VALUE to match 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 value expression used to select the matching VALUE and perform the program instructions contained in the value. The CASE structure evaluates the *index variable*, and examines the VALUE statements sequentially to find the same VALUE as the *index variable*. After the instructions belonging to the selected VALUE statement are executed, control of execution resumes at the step immediately following the END statement (unless a GO TO statement is encountered).

If there is no VALUE match between the index variable and the VALUE(S), the ANY statement is executed.

If there is not an ANY statement the program continues at the step following the END statement

---

**AS LANGUAGE PROGRAM INSTRUCTIONS**

Example: The example program displays different messages according to the value of the index variable *x*. If the value is negative, a message indicating the value is negative is displayed, if the value is positive, one of the three different messages are selected using the CASE structure. Three cases are: even numbers below eleven, odd numbers below ten, and numbers greater than ten.

```
25   IF x<0 GOTO 10
26   CASE x OF
27   VALUE 0,2,4,6,8,10:
28   PRINT "The number" ,x, "is EVEN"
29   VALUE 1,3,5,7,9:
30   PRINT "The number" ,x, "is ODD"
31   ANY
32   PRINT "The number" ,x, "is larger than 10"
33   END
34   STOP
35   PRINT "Interrupting program because of negative value"
36   STOP
```

---

**AS LANGUAGE FUNCTIONS**

<b>6.0</b>	<b>FUNCTIONS</b> .....	6-2
6.1	Real Value Functions .....	6-2
6.1.1	True and False Functions .....	6-2
6.1.2	SIG and BITS Functions .....	6-2
6.1.3	TIMER Function .....	6-4
6.1.4	DISTANCE and DX, DY, DZ Functions .....	6-5
6.1.5	ASC and LEN Functions .....	6-6
6.1.6	INT, TASK, and AVE_TRANS Functions .....	6-6
6.1.7	BASE and TOOL Functions .....	6-7
6.1.8	ERROR Function .....	6-8
6.1.9	SWITCH Function .....	6-8
6.1.10	MAXVAL and MINVAL Functions .....	6-8
6.2	Location Value Functions .....	6-9
6.2.1	FRAME Function .....	6-9
6.2.2	TRANS and NULL Functions .....	6-12
6.2.3	SHIFT Function .....	6-13
6.2.4	HERE and #HERE Functions .....	6-14
6.2.5	#PPOINT Function .....	6-14
6.2.6	RX, RY, and RZ Functions .....	6-14
6.2.7	DEST and #DEST Functions .....	6-15
6.3	Arithmetic Functions .....	6-16

## AS LANGUAGE FUNCTIONS

### 6.0 FUNCTIONS

A function is an operator that creates a value. For example, if the line `x=SQRT 9` is executed in an AS Language program, the arithmetic function square root (SQRT) creates the value 3. The assignment instruction (=) stores the number 3 in the variable x. Functions are often combined with program instructions to create a value, however, some functions stand alone in a program.

#### 6.1 REAL VALUE FUNCTIONS

TRUE, ON	Returns the value -1.0 which represents TRUE (ON).
FALSE, OFF	Returns the value 0.0 which represents FALSE (OFF).
SIG	Returns the logical AND of the specified signal.
BITS	Returns the value corresponding to the bit pattern of the signals.
TIMER	Returns the current value of the timer.
DISTANCE	Returns the distance between two points.
DX, DY, DZ	Returns displacement component of transformation values.
ASC	Returns ASCII character value.
LEN	Returns string length.

##### 6.1.1 TRUE AND FALSE FUNCTIONS

<b>TRUE/ON</b>	Returns the value -1.0 which represents the logical value TRUE (ON). This function is useful to specify the logical condition TRUE.
<b>FALSE/OFF</b>	Returns the value 0.0 which represents the logical value FALSE (OFF). This function is useful to specify the logical condition FALSE.

##### 6.1.2 SIG AND BITS FUNCTIONS

<b>SIG</b>	<b>(signal_number,signal_number)</b>  Returns the logical AND of the specified binary signal states.  Calculates logical AND of all the specified binary signal states and returns the resulting value. If all the specified signal states are TRUE, the SIG function returns TRUE -1. Otherwise, the SIG function returns FALSE 0.
------------	---

## AS LANGUAGE FUNCTIONS

**signal\_number:** The signals 1 to 256 are external output signals, the signals 1001 to 1256 are external input signals, the signals 2001 to 2256 are internal bits or flags which may be set by the user. The external signals specified must be installed.

Signals specified by positive numbers are considered TRUE when they are ON, while signals specified by negative numbers are considered TRUE when they are OFF. If zero is specified as a signal number, it does not correspond to any signal, and is always considered TRUE. For example, if signal 1001 is ON, signal 1004 is OFF, and signal 20 is OFF, then the SIG function returns the following results:

```
$EDIT pg_sig
1 x = SIG (1001);assigns signal to variable
1?
2 y = SIG (1004);assigns signal to variable
2?
3 z = SIG(1001,-1004);assigns signals to variable
3?
4e

$LI/R
x = -1   y = 0       (Results)
z = -1
```

### **BITS** (starting\_signal\_number, number\_of\_signals)

The function reads consecutive binary signals and returns the value corresponding to the bit pattern on the specified binary signals.

**starting\_signal\_number:** The first signal number read. This is also the least significant bit.

**number\_of\_signals:** The number of signals to read, beginning with the starting signal number. When not specified, one is assumed. The maximum number of signals is sixteen.

**Example:** WAIT BITS (1021,4) waits for inputs 1021,1022,1023, and 1024. Based on the binary state of these signals, the BITS function returns a corresponding decimal value. If signal 1021 is ON, signal 1022 is OFF, signal 1023 is ON, and signal 1024 is OFF, it corresponds to a binary value of 0101. The binary value is converted to an equivalent decimal value, which is 5.



## AS LANGUAGE FUNCTIONS

### 6.1.3 TIMER FUNCTION

#### TIMER (timer\_number)

Returns the current value (in seconds) of the specified timer.

timer\_number: Represents the number of the timer to read. The number must range from 1 to 10 and be enclosed within parenthesis. Timer number 0 is used for the system clock and tracks the time elapsed since the system started.

The value returned indicates the elapsed time (in seconds) since the last execution of the TIMER instruction for the timer specified. If the specified timer is not assigned a value by the TIMER instruction, it has the same value as TIMER 0.

Example: The TIMER instruction on line 1 is used to set TIMER (1) to 0 seconds, and the TIMER function used on line 7 is used to obtain the cycle time for the program.

```
$EDIT pg67
1 TIMER(1)= 0
1?
2 JMOVE aa
2?
3 JMOVE bb
3?
4 LDEPART 200
4?
5 HOME
5?
7 TYPE "cycle time = " ,/F5.3,Timer (1), " seconds"
7?
8 e

$EX pg67

cycle time = 5.678 seconds
```

## AS LANGUAGE FUNCTIONS

### 6.1.4 DISTANCE AND DX, DY, DZ FUNCTIONS

#### **DISTANCE** (transformation\_value, transformation\_value)

Calculates the straight line distance between the two locations.  
Returns the distance of the two specified points in millimeters.

transformation\_  
value,  
transformation\_  
value:

Specifies the two locations for distance calculation.

The order of the two points does not affect the result.

Example: x=DISTANCE (start, HOME) assigns the straight line distance between location variable start and the HOME position to variable x.

#### **DX, DY, DZ** (transformation\_value)

Returns the displacement value (X,Y,Z) of the specified location.

transformation\_  
value:

Name of transformation location whose X,Y, or Z component is required.

Example: The transformation location start has a transformation value of:

X	Y	Z	O	A	T
125	250	-50	135	50	75

x=DX (start)            DX function returns   x = 125.00

y=DY (start)            DY function returns   y = 250.00

z=DZ (start)            DZ function returns   z = -50.00

Component values may be assigned to real variables, for example, y=DX(aa) assigns the x component value of location variable aa to real variable y.

These three components can also be obtained using the DECOMPOSE instruction. If the rotation components (O, A, T) are required, the DECOMPOSE instruction must be used.

---

## AS LANGUAGE FUNCTIONS

### 6.1.5 ASC AND LEN FUNCTIONS

**ASC** (string, character\_number)

Returns the ASCII value of the specified character in the given string as a real value.

string: Expression which contains the character whose ASCII value is required. If the string is a null string, -1 is returned.

character\_number: The number which indicates the character whose ASCII value is required. When omitted, or when 0 or 1 is specified, the first character of the string is selected.

Example: `w = ASC ("sample",2)` assigns the ASCII value of character a to real variable w.

**LEN** (string)

Returns the number of characters contained in the given string.

string: Represents either a constant, variable, or expression whose length is required.

Example: `r = LEN ("sample")` assigns a value of 6 to real variable r.

### 6.1.6 INT, TASK, AND AVE\_TRANS FUNCTIONS

**INT** (expression)

Is used to return the nearest integer value of the expression, rounded down from the left of the decimal point. For example:

`x = INT(0.123)`  
`y = INT(10.8)`  
`w = INT(-5.76)`  
`t = INT(3.125E+2)`  
`INT(cost+0.5)`

**0 is returned for x**  
**10 is returned for x**  
**-5 is returned for x**  
**312 is returned**  
**The value of cost, rounded to the nearest integer, is returned.**

## AS LANGUAGE FUNCTIONS

### **TASK** (task\_number)

Returns the execution status of a PC program. With the two arm option, a 1 or a 2 is entered to represent the robot. PC programs 1-3 are represented by 1001, 1002, and 1003.

robot\_number: TASK (1) represents robot program number 1 and TASK (2) represents robot program number two (with the two arm option).

Example: rbsta1 = TASK (1), assigns the status of robot program number 1 (pg1) to real variable rbsta1.

With the two arm option:  
robot program 1 (used for arm 1) is named pg1.  
robot program 2 (used for arm 2) is named pg2.  
pc1 is used for variables.  
pc2 is used for peripheral equipment control.  
pc3 is used for variables.

task\_number: TASK (1001) represent PC program 1.

Example: pcstat1 = TASK (1001), assigns the status of PC program 1 to the real variable pcstat1.

No execution, returns 0  
Program executing, returns 1  
Program in HOLD, returns 2  
Program waiting for stepper, returns 3

### **AVE\_TRANS** (transformation 1, transformation 2)

The AVE\_TRANS function returns the average X, Y, and Z component location of the two specified locations.

## 6.1.7 BASE AND TOOL FUNCTIONS

**BASE** Returns the location components of the defined BASE.

**TOOL** Returns the location components of the defined TOOL.

---

## AS LANGUAGE FUNCTIONS

### 6.1.8 ERROR FUNCTION

#### ERROR

Returns the error code number and message generated. For example, the command

TYPE \$ERROR(ERROR)

types the error code number and message currently generated.

### 6.1.9 SWITCH FUNCTION

#### SWITCH (switch name)

Is used to return a -1.0 (ON) or 0.0 (OFF) value of the identified switch.

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".

### 6.1.10 MAXVAL AND MINVAL FUNCTIONS

#### MAXVAL (real variable 1, real variable 2, ... etc.)

returns the highest value of the identified real variables.

Example:           if a=10, b=15, and c=20, the command

m = MAXVAL (a,b,c)

assigns a value of 20 to the real variable "m".

#### MINVAL (real variable 1, real variable 2, ... etc.)

Returns the lowest value of the identified real variables.

Example:           if a=10, b=15, and c=20, the command

m = MINVAL (a,b,c)

assigns a value of 10 to the real variable "m".

## AS LANGUAGE FUNCTIONS

### 6.2 LOCATION VALUE FUNCTIONS

FRAME	Returns the transformation value which represents the relative coordinate system.
TRANS	Returns the transformation value composed of the given components.
NULL	Returns a null transformation value.
SHIFT	Returns the transformation value generated by shifting the original location.
HERE	Returns the current transformation location of the tool center point.
#HERE	Returns the current precision location in joint angles of the robot.
#PPOINT	Returns the precision value composed from the given components.
RX, RY, RZ	Returns the transformation value which represents a rotation.
DEST	Returns the destination location as a transformation value.
#DEST	Returns the destination location as a joint displacement value.
Arithmetic Functions	Arithmetic functions and formats.

#### 6.2.1 FRAME FUNCTION

**FRAME**                    **(location1, location2, location3, location4)**

Returns the transformation values that represent the relative coordinate system.

location1,  
location2:                Transformation values used to determine the direction of the X-axis of the relative coordinate system. The direction of the X-axis is defined parallel to the line between location1 and location2.

location3:                Transformation value used to determine the direction of the Y-axis of the relative coordinate system. The direction of Y-axis is defined and the X-Y plane contains location1, location2 and location3.

location4:                Transformation value that is the origin of the relative coordinate system.

Locations used within the FRAME must be relative to the coordinate system using compound expressions. When the coordinate system requires modification, only the reference location needs to be redefined with the FRAME function (Figure 6-1).

The FRAME function is used when the orientation of the task is not parallel or perpendicular to the robot base coordinate system.

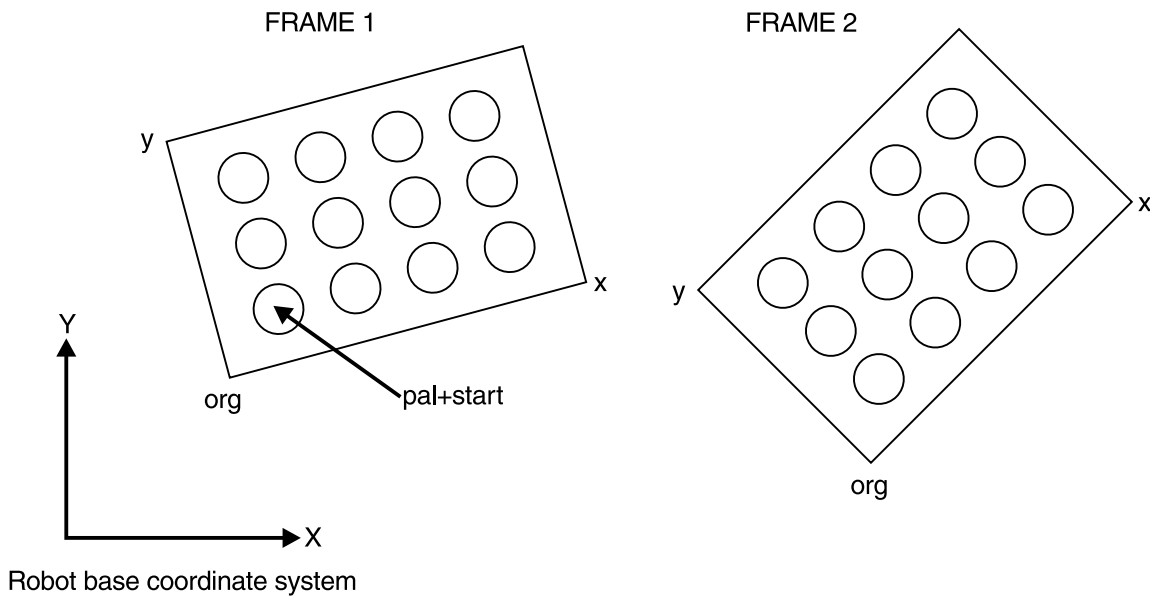
**AS LANGUAGE FUNCTIONS**

Figure 6-1 FRAME function

Example: To create the frame program in the example:

1. Teach the frame coordinate reference points org, x, and y.
2. At the monitor prompt (keyboard screen) enter POINT pal = FRAME (org, x, y, org).
3. Move the robot to the starting point and teach as pal+start.
4. Create the program as shown in the example.

**NOTE**

For accuracy, teach the locations org, x and y as far from each other as possible.

Tool definition and orientation is considered when defining locations for the FRAME function.

## AS LANGUAGE FUNCTIONS

```
lay.max = 3
col.max = 2
row.max = 3
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 BY 100*row,75*col,200*lay)
      POINT put_pt = pal+put
      JAPPRO #a, 100
      LMOVE #a
      LDEPART 100
      JAPPRO put_pt, 100
      LMOVE put_pt
      LDEPART 100
    END
  END
END
```

To change the frame orientation from the FRAME 1 to the FRAME 2 position (Figure 6-1), only the frame orientation points org, x, and y are retaught.



---

## AS LANGUAGE FUNCTIONS

### 6.2.2 TRANS AND NULL FUNCTIONS

**TRANS** (X\_component, Y\_component, Z\_component, O\_component, A\_component, T\_component)

Returns the transformation value consisting of the displacement components (X, Y, Z) and the rotation components (O, A, T). The transformation value is calculated from the assigned component values. The obtained transformation value is used in defining location transformations, motion requirements, or in compound transformation expressions.

X, Y, Z\_

Component:

Defines the displacement of X, Y, Z respectively.

O, A, T\_

Component:

Defines the rotation components of O, A, T respectively.

The value returned is assigned to a real variable and is used to modify the base coordinates.

Example:

```
POINT totaloffset = TRANS (offsetx,offsety,offsetz) BASE  
totaloffset
```

**NULL**

Returns a null transformation value of which all components are zero.

The null transformation value represents a location of which all displacement components are zero ( $X=Y=Z=0$ ), and all rotation components are zero ( $O=A=T=0$ ). The null transformation is convenient for setting a coordinate system back to its original value when repetitive changes are required.

## AS LANGUAGE FUNCTIONS

### 6.2.3 SHIFT FUNCTION

**SHIFT** (transformation\_location BY X\_shift, Y\_shift, Z\_shift)

Returns the transformation value representing the shifted position of the specified transformation value.

transformation\_location: Transformation location name whose value is changed.

X\_shift

Y\_shift

Z\_shift: Represents the value added to the respective position components of the specified transformation value. Values added are positive or negative.

The transformation value returned by this function has X, Y, Z position components with respective shift values added.

The rotation components of the specified transformation value remain unchanged. If any shift amount is omitted, zero is assumed.

The following example program is a palletizing routine using the SHIFT function.

Example:

```
POINT put = start
FOR col = 0 TO 5
  FOR row = 0 TO 10
    POINT put = SHIFT(start BY 100*row, 75*col, 0)
    JAPPRO #a, 100
    LMOVE #a
    LDEPART 100
    JAPPRO put, 100
    LMOVE put
    LDEPART 100
  END
END
```

In the example the “start” location is assigned to the location “put”. The two FOR TO END control flow structures are nested to fill one row before shifting to fill the next row. X\_shift (row) is 100 mm, Y\_shift (col) is 75 mm, and Z\_shift is 0 mm.

Six rows per column and eleven columns are filled to complete the nested loops.

## AS LANGUAGE FUNCTIONS

### 6.2.4 HERE AND #HERE FUNCTIONS

#### HERE

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".

$$\text{dist} = \text{DISTANCE} (\text{HERE}, \text{spot})$$

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 utilizing the calculation in a logical evaluation.

#### #HERE

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.

### 6.2.5 #PPOINT FUNCTION

#### #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.

### 6.2.6 RX, RY, AND RZ FUNCTIONS

**RX** (angle)

**RY** (angle)

**RZ** (angle)

The RX, RY, and RZ functions return transformation values that represent rotation around the specified axis of the base coordinate system.

## AS LANGUAGE FUNCTIONS

Example:           POINT crnr = RX(45)

Assigns a value 45 degrees of rotation about the X-axis of the base coordinate system to the transformation location crnr.

### 6.2.7 DEST AND #DEST FUNCTIONS

**DEST**                   Returns the value of transformation location, which is the planned destination location of the current robot motion.

The location where the robot stops, and the location the DEST function returns are not always identical. If robot motion is interrupted for some reason (the HOLD/RUN switch is turned to the HOLD position) the robot stops immediately. The DEST function returns the robot to the original destination location.

**#DEST**                   Returns the value of the precision location, which is the planned destination location of the current robot motion.

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 stops immediately. The #DEST function returns the robot to the original destination location.

The DEST function is useful to continue the motion that was interrupted because of an ONI signal\_number CALL program instruction. By inserting the following steps in an interrupt subroutine, the interrupted motion can be resumed.

POINT new = HERE   ;Stores current location in the variable new.

POINT old = DEST   ;Stores the destination location in the variable old.

JDEPART 50           ;Backs the tool away by 50 mm.

JMOVE old           ;Starts motion to the planned destination.

---

## AS LANGUAGE FUNCTIONS

### 6.3 ARITHMETIC FUNCTIONS

ABS:	Returns the absolute value of a numerical expression.
SQRT:	Returns the square root of a numerical expression.
PI:	Returns the constant $\pi$ .
SIN:	Returns the sine value.
COS:	Returns the cosine value.
ATAN2:	Returns the arctangent value.
RANDOM:	Returns a random number.

COMMAND	FUNCTION	FORMAT
ABS	Returns the absolute value of a numerical value.	ABS (value)
SQRT	Returns the square root of a numerical value.	SQRT (value)
PI	Returns the ratio of the circumference of a circle to its diameter; (3.1415...).	PI
SIN	Returns the sine (sin) of a given angle.	SIN (value)
COS	Returns the cosine (cos) of a given angle.	COS (value)
ATAN2	Returns the value of an angle (in degrees) whose tangent (tan) equals v1/v2.	ATAN2 (v1,v2)
RANDOM	Returns a random number from 0.0 to 1.0.	RANDOM

---

**CREATING AND EXECUTING PROGRAMS**

<b>7.0</b>	<b>CREATING AND EXECUTING PROGRAMS</b> .....	7-2
7.1	Types of AS Language Programs .....	7-2
7.1.1	Robot Control Programs .....	7-2
7.1.2	PC Program (Process Control) .....	7-2
7.1.3	SLOGIC Programs .....	7-2
7.2	Program Step Format .....	7-3
7.3	Program Execution .....	7-4
7.3.1	Executing Robot Control Programs .....	7-4
7.3.1.1	Stopping Robot Control Programs .....	7-5
7.4	Executing PC Programs .....	7-5
7.5	Flowcharting .....	7-6
7.6	Compound Transformations and Tool Dimensions .....	7-10
7.7	Sample Programs .....	7-13

---

## CREATING AND EXECUTING PROGRAMS

### 7.0 CREATING AND EXECUTING PROGRAMS

An AS Language program is a collection of instructions used to direct 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 are 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 I/O 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.

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 I/O signals can be used in PC programs.

##### 7.1.3 SLOGIC PROGRAMS

SLOGIC programs are used to provide logic instructions for remote I/O 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.

---

## CREATING AND EXECUTING PROGRAMS

### 7.2 PROGRAM STEP FORMAT

Each step of an AS Language program uses 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 inserted after the last step of the program.

2. Label:

Step numbers cannot be used as branch destinations with the program instruction GOTO.

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.



## CREATING AND EXECUTING PROGRAMS

### 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 your 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

To execute a robot control program, select the desired program.

If the program to execute is already selected, the monitor command EXECUTE can be used. This runs the program one time and a message is displayed indicating the program was completed.

The EXECUTE command can be used with the optional argument of a program name to run a program not on the stack.

The number of times a program is to execute is entered after the program name. To specify the program to run continuously a -1 is entered.

The PRIME command, in conjunction with the CYCLE START switch, can be used when executing programs. The PRIME command is used to specify the step of the program where execution starts, if no number is specified with the PRIME command, program execution starts 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.

---

## CREATING AND EXECUTING PROGRAMS

To run a program using the controller switch panel or multi function panel switches, refer to the *C Series Controller Operations and Programming Manual*.

### 7.3.1.1 STOPPING ROBOT CONTROL PROGRAMS

An EMERGENCY STOP switch (located on the multi function panel and controller cabinet) are used anytime an operator needs to stop robot motion immediately.

It is recommended the EMERGENCY STOP switches are not used as a routine method of stopping robot motion.

When an EMERGENCY STOP switch 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 completes the current instruction and stops after a normal deceleration to the taught location.

Motor power remains ON when the ABORT command is issued. Motor power can then be turned OFF by pressing an EMERGENCY STOP switch.

To stop a running program using the controller switch panel or multi function panel switches, refer to the *C Series Controller Operations and Programming Manual*.

## 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 after the current step is completed.

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.

## CREATING AND EXECUTING PROGRAMS

### 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 programs are then written to meet the objectives identified in the flowchart.

Figure 7-1 shows an example of a working flowchart.

**CREATING AND EXECUTING PROGRAMS**

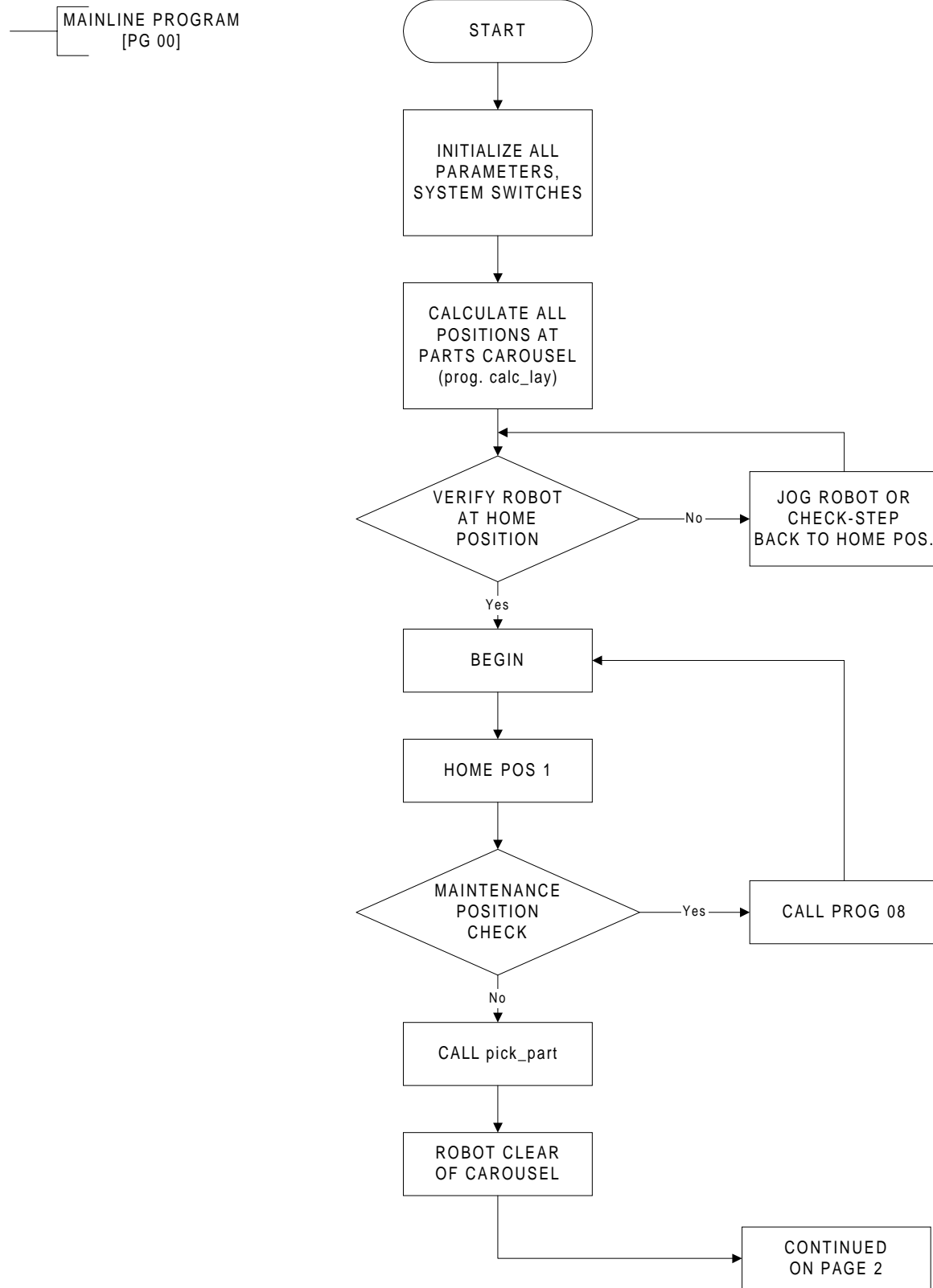


Figure 7-1 Working Flowchart

**CREATING AND EXECUTING PROGRAMS**

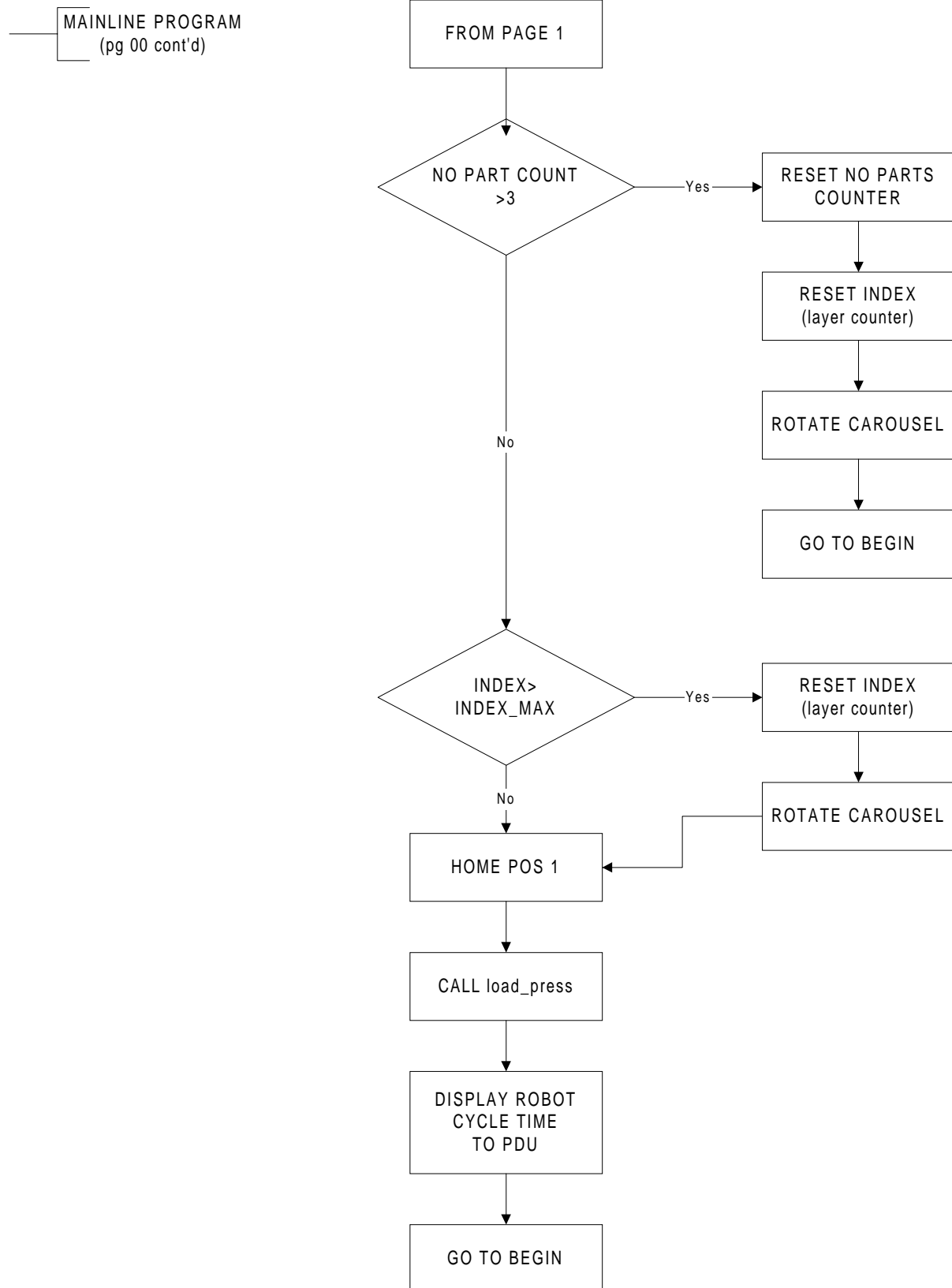


Figure 7-1 Working Flowchart (continued)

### CREATING AND EXECUTING PROGRAMS

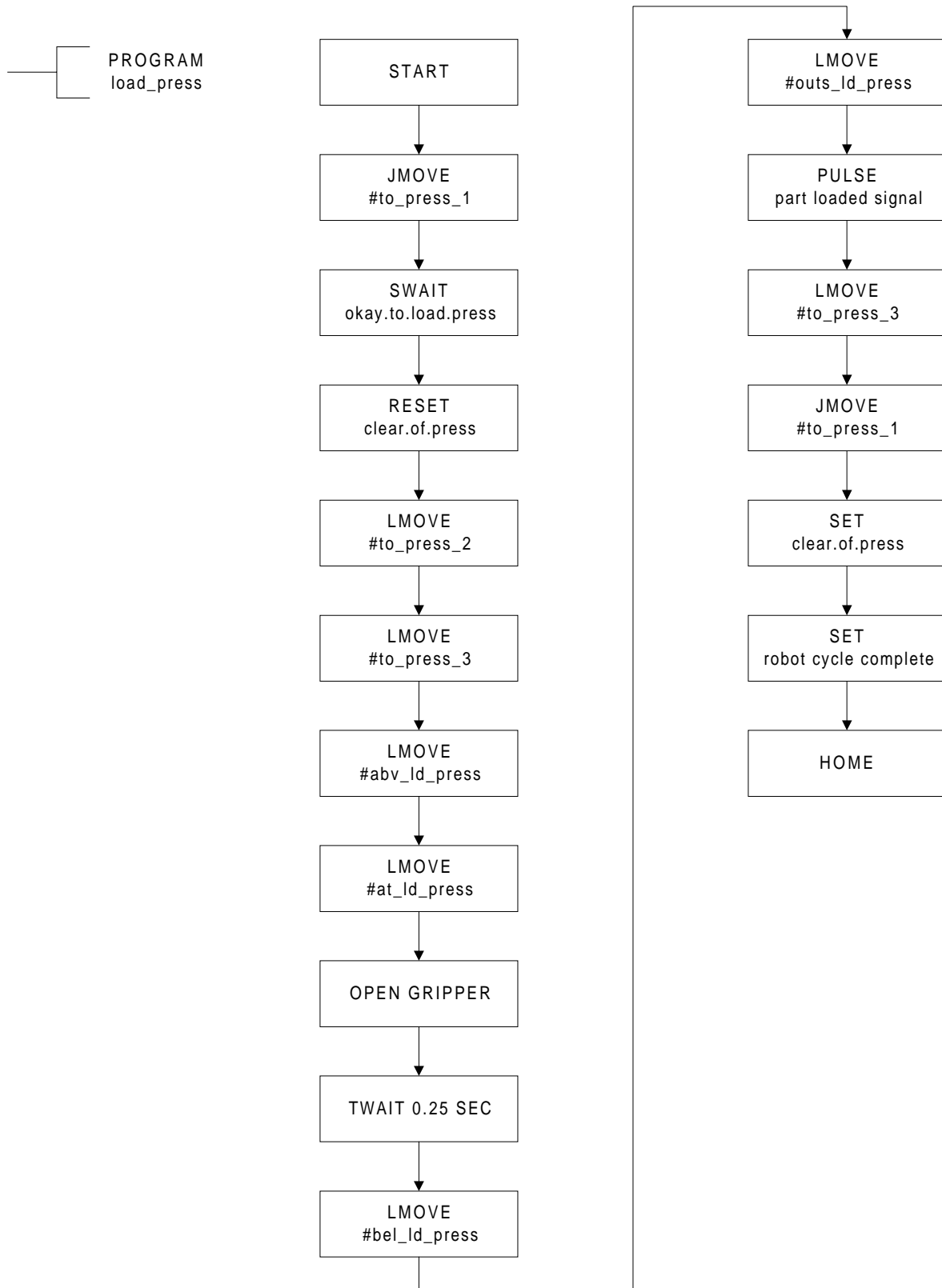


Figure 7-1 Working Flowchart (continued)

## CREATING AND EXECUTING PROGRAMS

### 7.6 COMPOUND TRANSFORMATIONS AND TOOL DIMENSIONS

The controller's ability to compute compound transformations can be utilized to calculate the tool dimensions of an installed tool.

To accomplish this, a precise reference point is required to be set inside the work envelope (preferably along the y-axis). To work with an AS Language defined tool, the monitor command **QTOOL OFF** must be entered.

To ensure the controller is not calculating dimensions based on a previously installed tool, the monitor command **TOOL NULL** must be entered.

Figure 7-2 shows an example of the NULL tool center point and the fixed reference point.

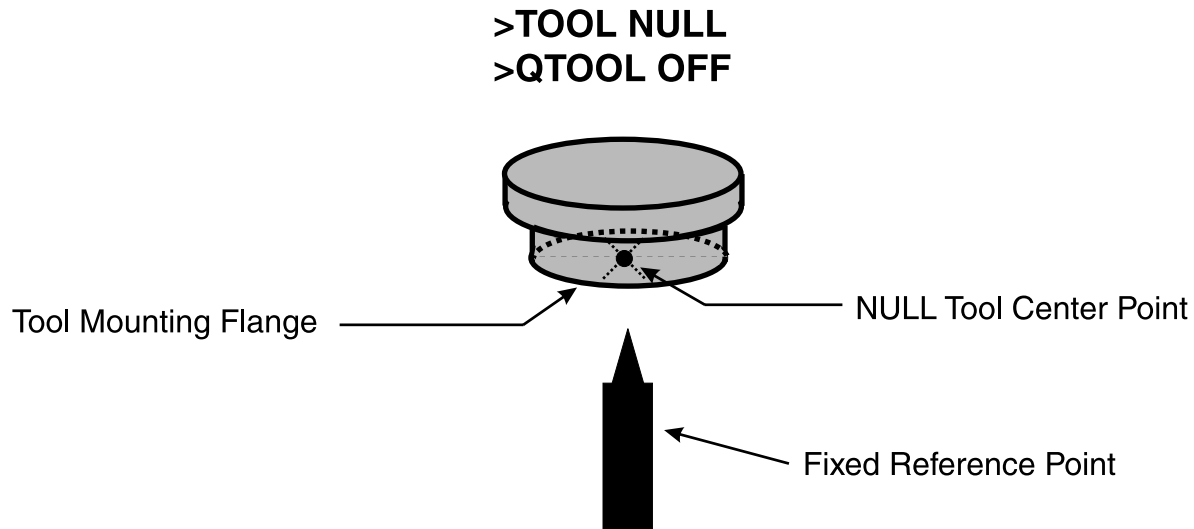


Figure 7-2. Tool Transformation Preparation

The monitor instruction **DO ALIGN** is used to ensure that the tool mounting flange is parallel with the x and y axes of the base coordinate system.

After the tool mounting flange has been aligned, the robot should be jogged in the base mode to maintain this orientation.

Jog the robot so that the center of the tool mounting flange is at the fixed reference point as shown in figure 7-3.

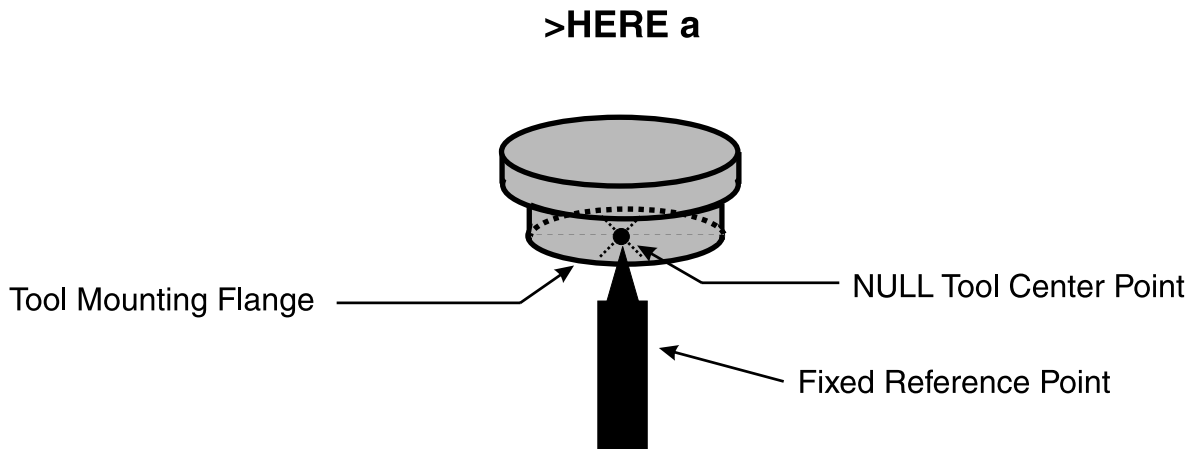
**CREATING AND EXECUTING PROGRAMS**

Figure 7-3 Defining the Location a

Jog the robot away from the fixed reference point and install the tool to dimension.

After the tool is installed, jog the robot so the new tool center point is at the fixed reference point.

The tool must be oriented so the z tool coordinate axis is perpendicular to the base coordinate x and y axes.

The monitor command **HERE a+b** is entered to define and record the compound transformation location b (Figure 7-4).



## CREATING AND EXECUTING PROGRAMS

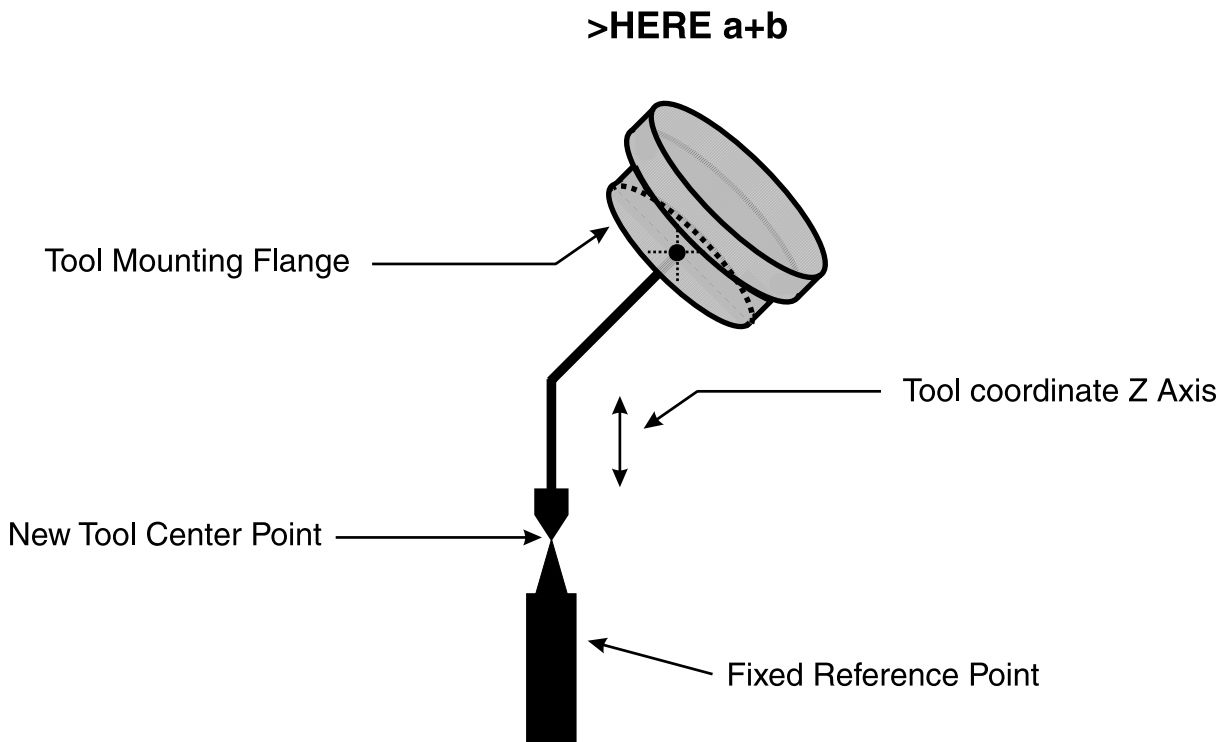


Figure 7-4 Defining the Compound Transformation a+b

The monitor command **POINT t= -b** is entered to assign the inverse of the compound transformation b to the variable t.

The inverse of b is used because the compound transformation b is the location of the NULL tool center point relative to the fixed reference point.

The monitor command **TOOL t** is entered to assign the value of variable t to an AS Language tool.

To confirm the new tool dimensions are recorded properly, jog the robot away from the fixed reference point.

Select the jogging TOOL mode and jog the robot using the rx, ry, and rz keys. The tool center point remains fixed while the robot arm rotates about that point.

Ensure the path from the robot's current position to the fixed reference is clear. Set the repeat condition speed to 10%. Place the robot in the repeat mode and enter the monitor command **DO JM a**, the tool center point travels to the fixed reference point.

---

## CREATING AND EXECUTING PROGRAMS

### 7.7 SAMPLE PROGRAMS

The following AS Language program is an example of a “mainline” program. Programs like this are used to call other subroutine programs based on signal inputs from another peripheral device, usually a PLC.

#### **.PROGRAM PG00()**

##### **1 ; Mainline program**

*The ; identifies a comment only, anything to the right of the ; is not evaluated as a program instruction. Mainline program identifies the type of program to the reader of the program code.*

##### **2 ;**

*The ; is used to add a space for readability.*

##### **3 100 HOME**

*100 is a label to be used with GOTO statements. HOME is a location that the robot travels to when this line of code is processed. The HOME position has unique characteristics regarding how it is defined, the commands necessary to move the robot to the HOME position, and the dedicated output signals that can be assigned to it.*

##### **4 BREAK**

*The BREAK command stops the program from processing ahead and ensures the robot reaches the HOME position before output signals are set.*

##### **5 SIGNAL 1,-2,3,5,6**

*The SIGNAL command sets output signals 1 (clear to advance transfer), 3 (clear to open clamp), 5 (zone 1 clear), and 6 (zone 2 clear) to the ON condition. Signal 2 (clear to retract transfer) is set to the OFF condition.*

---

## CREATING AND EXECUTING PROGRAMS

- 6 ;** The ; is used to add a space for readability.
- 7 WAIT BITS (1021,4)** The WAIT command causes the program to suspend execution until the specified condition (BITS 1021,4) is satisfied. The BITS command is used to specify that 4 signals be evaluated for their binary value, the first signal is 1021 (binary value of 1) the fourth signal is 1024 (binary value of 8), if any combination of these signals are present, the program continues processing.
- 8 pg = BITS (1021,4)** In this line of code, the binary value of signals 1021 through 1024 (BITS (1021,4)) are assigned (=) to the real variable pg.
- 9 SIGNAL -3** The SIGNAL command is used to set output signal 3 (clear to unclamp) to the off condition.
- 10 CASE pg OF** The CASE pg OF command is a control flow structure that evaluates the real variable pg. If the value of pg matches the VALUE statement, the program instructions following the instruction are processed. If none of the VALUE statements match the value of the variable pg, the program processes the first instruction after the END statement identifying the last step of the control flow structure. As long as the value of pg matches a VALUE statement within the control flow structure the program continues to process inside the structure.
- 11 VALUE 8:** In this line of code, if the VALUE of pg is 8, the next line of the program is processed. If the value of pg is not 8, then the program scans past this line of code and continue to evaluate the VALUES for a match with pg.
- 12 CALL pg8** If the program reaches this line of code, the instruction CALL pg8 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg8.

---

## CREATING AND EXECUTING PROGRAMS

- 13 VALUE 10:** In this line of code, if the VALUE of pg is 10, the next line of the program is processed. If the value of pg is not 10, then the program scans past this line of code and continues to evaluate the VALUES for a match with pg.
- 14 CALL pg10** If the program reaches this line of code, the instruction CALL pg10 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg10.
- 15 END** The END command identifies the end of the control flow structure, CASE pg OF...VALUE xxx. If the value of the specified variable is not matched within the control flow structure the program proceeds to process the first instruction after the END statement.
- 16 ;** The ; is used to add a space for readability.
- 17 IF NOT SIG (1001) GOTO 100** In this line of code, the signal state of input signal 1001 (Signal 1001 move to pounce) is evaluated for an off or on condition. If input signal 1001 signal is on, the next program instruction is processed. If input signal 1001 is off, the GOTO 100 section of the instruction is processed and the program searches for the label 100 and continues processing beginning with that instruction.
- 18 pg = BITS (1021,4)** In this line of code, the binary value of signals 1021 through 1024 (BITS (1021,4)) are assigned (=) to the real variable pg.

---

## CREATING AND EXECUTING PROGRAMS

### 19 CASE pg OF

The CASE pg OF command is a control flow structure that evaluates the real variable pg. If the value of pg matches the VALUE statement, the program instructions following the instruction are processed. If none of the VALUE statements match the value of the variable pg, the program processes the first instruction after the END statement identifying the last step of the control flow structure. As long as the value of pg matches a VALUE statement within the control flow structure the program continues to process inside the structure.

### 20 VALUE 1:

In this line of code, if the VALUE of pg is 1, the next line of the program is processed. If the value of pg is not 1, then the program scans past this line of code and continues to evaluate the VALUES for a match with pg.

### 21 CALL pg1

If the program reaches this line of code, the instruction CALL pg1 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg1.

### 22 VALUE 2

In this line of code, if the VALUE of pg is 2, the next line of the program is processed. If the value of pg is not 2, then the program scans past this line of code and continues to evaluate the VALUES for a match with pg.

### 23 CALL pg2

If the program reaches this line of code, the instruction CALL pg1 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg1.

### 24 VALUE 3

In this line of code, if the VALUE of pg is 3, the next line of the program is processed. If the value of pg is not 3, then the program scans past this line of code and continues to evaluate the VALUES for a match with pg.

---

## CREATING AND EXECUTING PROGRAMS

- 25 CALL pg3** If the program reaches this line of code, the instruction CALL pg3 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg3.
- 26 VALUE 4** In this line of code, if the VALUE of pg is 4, the next line of the program is processed. If the value of pg is not 4, then the program scans past this line of code and continues to evaluate the VALUES for a match with pg.
- 27 CALL pg4** If the program reaches this line of code, the instruction CALL pg4 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg4.
- 28 VALUE 5** In this line of code, if the VALUE of pg is 5, the next line of the program is processed. If the value of pg is not 5, then the program scans past this line of code and continues to evaluate the VALUES for a match with pg.
- 29 CALL pg5** If the program reaches this line of code, the instruction CALL pg5 is executed. The CALL command causes the program to execute the specified subroutine, in this example, pg5.
- 30 END** The END command identifies the end of the control flow structure, CASE pg OF...VALUE xxx. If the value of the specified variable is not matched within the control flow structure the program proceeds to process the first instruction after the END statement.
- 31 GOTO 100** The command GOTO 100 causes the program to search for the label 100 and continue processing beginning with that instruction.
- .END**

---

## CREATING AND EXECUTING PROGRAMS

The following AS Language program is an example of a welding clamp subroutine program. Programs like this are used to communicate with a weld controller. Welding clamp specifications are defined in auxiliary function 114.

### **.PROGRAM pg90()**

- |                                  |   |
|----------------------------------|---|
| <b>1 ; Weld Sequence Program</b> | The ; is used to identify a comment. Any information to the right of the ; is not processed as part of the program. "Weld Sequence Program" identifies the type of program for readers.   |
| <b>2 ;</b>                       | The ; is used to place a space in the program for ease of reading.  |
| <b>3 IF SIG(-23) THEN</b>        | The instruction IF SIG(-23) THEN is part of a control flow structure. If signal 23 (weld initiate output) is off, then the program instructions inside the structure is processed. The last step of the structure is defined by an END command. If signal 23 is on when this instruction is processed, the program processes the first step after the control flow structure. |
| <b>4 PULSE 23, 0.2</b>           | The pulse command is used to turn output signal 23 (weld initiate output) on for a period of 0.2 seconds.   |
| <b>5 END</b>                     | The END command is used it identify the last step of the IF THEN structure.   |
| <b>6 SWAIT -1018</b>             | The SWAIT command is used to evaluate the input signal 1018 (weld complete signal, sent from the weld controller). If signal 1018 is off, the program continues processing. If signal 1018 is on, the program stops processing at this step until signal 1018 changes to the off state.   |
| <b>7 TIMER 1 = 0</b>             | The command TIMER 1 = 0 is used to assign (=) the value of 0 to timer number 1. This initializes timer 1 to a value of zero. Timer 1 begins timing immediately after this instruction is processed.   |

---

## CREATING AND EXECUTING PROGRAMS

### 8 WHILE TIMER (1) <= 2 DO

The instruction WHILE TIMER (1)<= 2 DO is part of a control flow structure. If the value of timer 1 is less than or equal to 2, then the program instructions inside the structure are processed. The last step of the structure is defined by an END command. If the value of timer 1 is greater than 2 when this instruction is processed, the program processes the first step after the control flow structure.

### 9 IF SIG(1018) THEN

The instruction IF SIG(1018) THEN is part of a control flow structure. This structure is “nested” inside the WHILE DO structure. When structures are nested, the program evaluates the innermost structure first. If input signal 1018 (weld complete signal from the weld controller) is on, then the program instructions inside the structure are processed. Because of the previous WHILE TIMER (1) DO instruction the program monitors the weld complete signal for 2 seconds. The last step of the structure is defined by an END command. If signal input 1018 is off when this instruction is processed, the program processes the first step after the control flow structure.

### 10 RETURN

When the RETURN instruction is processed, program processing exits pg90 and returns to the program that originally called it.

### 11 END

The END command is used to identify the last step of the nested IF THEN structure.

### 12 END

The END command is used to identify the last step of the WHILE DO structure.

### 13 Signal 7

This command turns output signal 7 (time delay exceeded) on.

### 14 SWAIT 1018

The SWAIT 1018 instruction causes the program to wait for input signal 1018, the weld complete signal from the weld controller. When signal 1018 is on, the program continues processing the next instruction. When signal 1018 is off, the program waits at this step until 1018 turns on or the wait condition is overridden.



## CREATING AND EXECUTING PROGRAMS

### 15 SIGNAL -7

This command turns output signal 7 (time delay exceeded) off.

### 16 RETURN

When the RETURN instruction is processed, program processing leaves pg90 and returns to the program that originally called it.

### .END

---

**PROGRAMMING WITH PERSONAL COMPUTER**

<b>8.0 PC INTERFACE .....</b>	<b>8-2</b>
8.1 PC Interface Overview .....	8-2
8.2 PC Interface Configuration .....	8-3
8.3 Installing Kawasaki AS MONITOR Software .....	8-3
8.4 PC Operation .....	8-5

---

## PROGRAMMING WITH PERSONAL COMPUTER

### 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 needs 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 is designed to communicate serially over a standard RS232C, D shell, 9 pin female to 9 pin female, null modem cable interface with a personal computer (PC). The RS232C connector cable is purchased locally or 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

---

## PROGRAMMING WITH PERSONAL COMPUTER

### 8.2 PC INTERFACE CONFIGURATION

To interface a PC with the C-series controller the following items are required:

- Personal computer running DOS 3.x, or higher, with a 3.5 inch 1.44MB disk drive (user supplied)
- RS232C, D shell, 9 pin female to 9 pin female, null modem cable interface option, or user supplied cable.
- 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 installs 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 interface cable between the robot and the PC is properly connected.
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 protect.
4. Turn on the PC and allow it to go through its power-up sequence. If the PC automatically boots into Windows the user must 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.

---

## PROGRAMMING WITH PERSONAL COMPUTER

6. Confirm the diskettes 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 ensure 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 Operating System.

1. Ensure that the interface cable between the robot and the PC is properly connected.
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 protect.
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

## PROGRAMMING WITH PERSONAL COMPUTER

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 keyboard can be accessed with the PC.

Some notebook PC's may not be set up to read/write to SRAM PC Cards. If this is the case with the device you are using, specify the proper Windows™ install and setup drivers.

Open the Notepad text editor and add the following lines, in the following order, to the end of the Config.sys file:

```
device=c:\windows\system\csmapper.sys  
device=c:\windows\system\carddrv.exe/slot=n ("n" represents the number or PCMCIA  
slots available in your computer, on most notebooks "n"=2)
```

Save your changes and restart the computer. When the computer reboots, access Windows Explorer and verify that "n" new drives (one for each PCMCIA slot) are available.



### CAUTION

The notebook computer should not be connected when initializing the controller or when SRAM cards are formatted by the controller. It is possible for the controller to format the hard drive of the notebook when these operations are performed.

---

## PROCESS CONTROL PROGRAMS

<b>9.0</b>	<b>PROCESS CONTROL PROGRAMS .....</b>	<b>9-2</b>
9.1	Process Control Commands .....	9-2
9.1.1	PCSTATUS Command .....	9-2
9.1.2	PCEXECUTE, PCABORT, PCCONTINUE, PCSTEP, and PCKILL Commands	9-3
9.1.3	PCSCAN Command.....	9-4

## PROCESS CONTROL PROGRAMS

### 9.0 PROCESS CONTROL PROGRAMS

A process control (PC) program is an AS Language program 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 evaluate variables, perform mathematical calculations, execute logic instructions, and set external or internal signals.

PC programs cannot use instructions that cause robot motion.

The BASE and TOOL instructions are not available for PC programs.

PC programs are 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

PCSTATUS	Displays the status of the specified PC program.
PCEXECUTE	Executes the specified PC program.
PCABORT	Stops execution of the specified PC program.
PCCONTINUE	Resumes execution of the PC program.
PCSTEP	Executes a single step of a PC program.
PCKILL	Initializes the PC program execution stack.
PCSCAN	Specifies PC program process time.

#### 9.1.1 PCSTATUS COMMAND

**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 to display.

PC program\_number:

PC program number range is 1 to 3. When the PC program number is omitted, 1 is assumed (Figure 9-1).



## PROCESS CONTROL PROGRAMS

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, PCABORT, PCCONTINUE, PCSTEP, AND PCKILL COMMANDS

**PCEXECUTE**      **program\_name, execution\_cycles, starting\_step**

The PC EXECUTE command is used to execute a PC program. This command is identical to the EXECUTE monitor command except that this command executes a PC program instead of a robot control program.

**PCABORT**      **PC\_number**

The PCABORT command is used to stop the execution of PC programs.

PC\_number      Specifies the number of the PC program to abort, 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.

**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.

PC\_number:      Specifies the number of the PC program to continue execution.

NEXT:      When used with the NEXT argument, the WAIT instruction currently executing by the PC program is skipped and the execution continues at the next step.

---

## PROCESS CONTROL PROGRAMS

The PCCONTINUE command is identical to the CONTINUE monitor command, except is used with process control programs instead of robot control programs.

### **PCSTEP**            **PC program\_name, execution\_cycles, starting\_step**

The PCSTEP command is used to execute a single step of a PC program.

PC program\_name: Is the name of the PC program to execute one step at a time.

execution\_cycles: Is the number of cycles for the PC program to execute with the PCSTEP command.

starting\_step: Is the program step number to execute first.

### **PC KILL**            Removes the current PC program from active status.

A PC program cannot be removed from the PC program stack if it is executing.

The PCKILL command is identical to the KILL monitor command, except that it handles process control programs instead of robot control programs.

### 9.1.3 PCSCAN COMMAND

#### **PCSCAN**            **time**

The PCSCAN command is used to specify the length of time used to process the PC program one time.

time: Is specified in increments of one second.

If the PCSCAN command is not used, the PC program is processed at the CPU speed.

---

## ERROR CODES/TROUBLESHOOTING

<b>10.0 ERROR CODES/TROUBLESHOOTING .....</b>	<b>10-2</b>
10.1 Error Recovery .....	10-2
10.2 Error Codes .....	10-5
10.3 Troubleshooting Flowcharts .....	10-84

## ERROR CODES/TROUBLESHOOTING

### 10.0 ERROR CODES/TROUBLESHOOTING

This unit provides error recovery flowcharts, error code information, and error code troubleshooting flow charts. In addition, typical causes and remedies for the errors are also provided.

#### 10.1 ERROR RECOVERY

Figure 10-1 shows troubleshooting processes that may be helpful if the controller becomes unresponsive to commands or an error code has been encountered that cannot be cleared. Troubleshooting should begin with confirmation of basic integrity of the system: ensure that the power supply is on and meeting supply requirements, all cables are correctly attached, all circuit boards are properly installed and fully seated, all peripheral equipment is wired correctly, software is properly configured, etc.

**ERROR CODES/TROUBLESHOOTING**

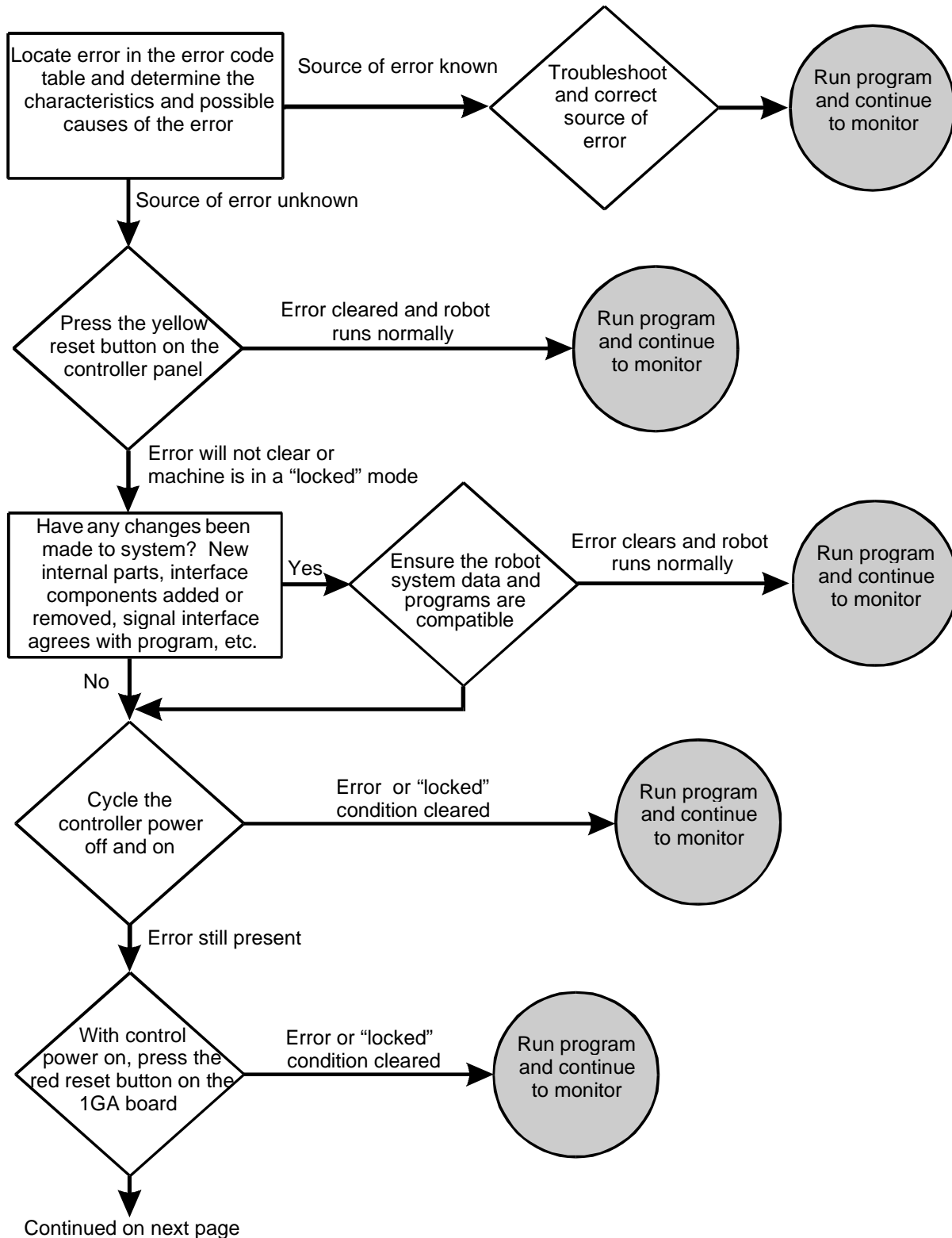


Figure 10-1 Trouble Shooting Process

**ERROR CODES/TROUBLESHOOTING**

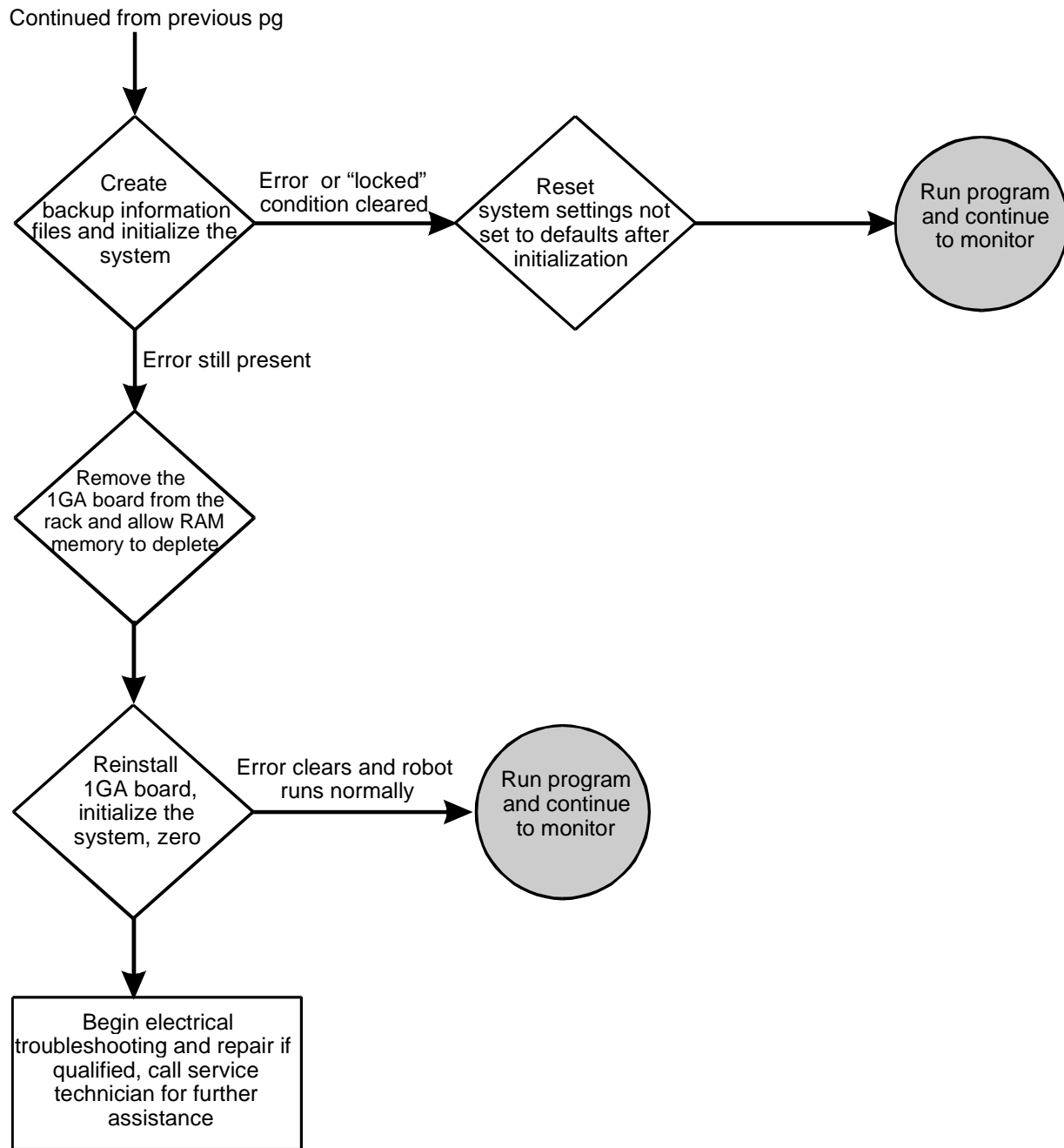


Figure 10-2 Trouble Shooting Process (cont'd)

---

## ERROR CODES/TROUBLESHOOTING

### 10.2 ERROR CODES

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 in numerical order by code number with the message that is displayed with the associated code. An expanded explanation of the message is provided along with possible methods to clear or prevent the specific error. Troubleshooting information is preceded by a ⇒ symbol.

**ERROR CODE -50**            Warning! Cannot move along straight line in this configuration.

Joint speed may exceed maximum, joints 4 and 6 aligned.

⇒ *Change angle of joint 5, slow the speed, change to joint mode.*

---

**ERROR CODE -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.*

---

**ERROR CODE -100**        Matrix Calculation Error.

The vector element of the matrix cannot be operated because of 0.

⇒ *Change and recalculate value.*

---

**ERROR CODE -101**        Turn off motor power.

Motor power cannot turn on according to command and instruction.

⇒ *Turn motor power OFF and execute command and instruction.*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -102**      Application is changed. Turn OFF & ON the control power.

The robot configuration was changed from spot welding/material handling to a paint/sealant application using AUX function 907.

⇒ *Turn controller power OFF then ON.*

---

**ERROR CODE -200**      Cannot execute a program because motor power is OFF.

Program will not start because motor power is not on.

⇒ *Turn motor power ON.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -207**      Turn to HOLD at HOLD/RUN sw.

Occurs when an 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.*

---



---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -208** Teach pendant is not connected.

Hardwired 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 CODE -211** Cannot edit a program because the TEACH LOCK switch is ON.

Programs cannot be edited if the TEACH LOCK switch is ON.

⇒ *Turn TEACH LOCK switch OFF.*

---

**ERROR CODE -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.*

---

**ERROR CODE -302** Can't continue. Use EXEC.

The CONTINUE command is not permitted because of program selection status.

⇒ *Use the EXECUTE command to start program.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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 enter the command.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -318**            Waiting retract or extend pos. input signal.

One of the following operations were attempted when the robot was processing a spot weld sequence (waiting for the Retract/Extend signal after the weld complete signal had been received).

1. Cycle start (including EXECUTE, CONTINUE command).
2. Program selection or step change.
3. Record.

⇒ *Input the Retract/Extend detection signal to the robot or press the WX key and wait override key on the multi function panel.*

---

**ERROR CODE -319**            Spot sequence is running.

Step change, program change, or program execution were attempted while the spot weld sequence was executing (after Retract/Extend input signal and weld complete signal have been received). For example, the robot is executing move delay time after weld complete.

⇒ *Perform step change, program registration or program execution after the spot welding sequence.*

---

**ERROR CODE -320**            Cannot operate because teach pendant in operation.

You cannot perform functions on the personal computer while the multi function panel is in use.

⇒ *Perform functions on the PC after the multi function panel has completed operation.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -324**          Cannot execute with MC instruction.

MC instruction could not be carried out.

⇒ *Use instruction which can be executed by MC instruction.*

---

**ERROR CODE -325**          Cannot execute the instruction in robot program.

Command and instruction cannot be used or executed.

⇒ *Rewrite the command or instruction.*

---

**ERROR CODE -326**          Cannot delete because used by another command.

Cannot COPY, DELETE or XFER command because current step is being executed.

⇒ *Perform DELETE commands after execution ends.*

---

**ERROR CODE -327**          Used in programs.

The variable used with the program was deleted.

⇒ *Confirm the deleted variable.*

---

**ERROR CODE -328**          Used in editor.

When the program was used in the editor, the program was deleted.

---

**ERROR CODE -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/PC KILL the program, then delete.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -350**            Illegal input data.

Input data from AS Language monitor command is improper for the instruction.

⇒ *Enter data that is within acceptable range.*

---

**ERROR CODE -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.*

---

**ERROR CODE -353**            Input data is too big.

Data entered for the POINT or HERE commands exceeds the allowable range.

⇒ *Enter data that is within acceptable range.*

---

**ERROR CODE -360**            Illegal WHERE parameter.

Occurs if data entered with the WHERE command is not an integer between 1 and 6.

⇒ *Ensure that data is within acceptable range.*

---

**ERROR CODE -361**            Illegal PC number.

Unused

---

**ERROR CODE -365**            Illegal Robot number.

Unused

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -367**      Illegal priority.

Priority level designation is wrong.

⇒ *Input a correct priority level.*

---

**ERROR CODE -368**      Invalid coordinate value.

Upper limit value is less than value of lower limit.

⇒ *Input the correct upper limit coordinates.*

---

**ERROR CODE -371**      External axis type and gun data mismatch.

The servo gun setting does not correspond to the type of external axis (set in AUX 160) and type of gun (set in AUX 114). This error is detected when motion begins for a step with a clamp signal set.

⇒ *Confirm that the external axis and gun type data settings are correct.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -402**          Ambiguous statement.

Displayed when an abbreviation is entered incorrectly or has missing letters.

⇒ *Enter the correct abbreviation or entire command.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -405**          Statement cannot be executed.

Occurs when the AS Language instruction entered was not acceptable for the mode of operation. For example: a monitor command was entered in the editor mode.

⇒ *Use instructions and commands that are compatible with the input mode.*

---

**ERROR CODE -406**          Not a program instruction.

An instruction was entered into a program that is not a valid program instruction, i.e., a monitor command, editor command, etc.

⇒ *Refer to the AS Language Manual for proper use of AS Language commands.*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -407**      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.*

---

**ERROR CODE -408**      Missing argument.

Displayed when a DO command is not followed by an acceptable program instruction.

⇒ *Correct the input and re-enter.*

---

**ERROR CODE -410**      Illegal expression.

A real number expression must be present for processing DECOMPOSE command(s). Also displayed when incorrect numerical information is entered with arguments.

⇒ *Ensure correct format and numerical expressions are entered.*

---

**ERROR CODE -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.*

---

**ERROR CODE -412**      Illegal argument of function.

Occurs when function and argument are not in correct format.

⇒ *Use correct format for functions and arguments.*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -413** Invalid variable (or program) name.

Displayed when illegal variable or program name is entered from the editor or monitor modes. For example: JM ##a (to many precision symbols) or CALL #a (not an acceptable program name).

⇒ *Define program names and variables correctly.*

---

**ERROR CODE -414** Illegal variable type.

Displayed when illegal variable is entered from editor or monitor modes. For example: b = #a + b (combining location and real variables using an arithmetic operator).

⇒ *Use compatible variable type for commands or instructions.*

---

**ERROR CODE -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.*

---

**ERROR CODE-416** Missing parenthesis.

Occurs when parentheses are not entered as a pair, containing both a left and right parenthesis.

⇒ *Enter parentheses in left and right pairs.*

---

**ERROR CODE -417** Expected to be a binary operator.

A non-binary operator has been entered where a command expected a binary operator.

⇒ *Input a binary operator.*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -419**      Illegal qualifier.

Displayed when monitor commands are followed by unexpected qualifiers. For example: LIST W (expected characters would include P, L, and R for program, location, and real variables; "W" causes error).

⇒ *Use only acceptable qualifiers.*

---

**ERROR CODE -420**      Invalid label.

Occurs in the editor mode when a GOTO instruction is combined with a reserved character. For example: GOTO #a ("#" is a reserved character).

⇒ *Do not use reserved characters in label identification.*

---

**ERROR CODE -421**      Invalid name.

Displayed when an unidentified program, file, variable, etc., is used in a command.

⇒ *Use only names that have been defined.*

---

**ERROR CODE -422**      Missing expected character.

Occurs when commands or instructions are entered with an incorrect format. For example: TOOL a=b (TOOL a would be correct, "=b" causes error).

⇒ *Use correct format for commands and instructions.*

---

**ERROR CODE -423**      Illegal switch name.

Displayed when a system switch is incorrectly identified. For example: SWITCH light (there is no system switch named "light").

⇒ *Use only available switch names.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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 CODE -425**      Illegal format qualifier.

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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -430**      Cannot define as array.

A non-array variable with the same name as the array variable that was attempted to be created, already exists.

⇒ *Use a different name for the new variable. Refer to the AS Language Manual for proper use of AS Language commands.*

---

**ERROR CODE -431**      Dimension exceeds 3.

Attempted to create an array variable with more than 3 dimensions.

⇒ *Refer to the AS Language Manual for proper use of AS Language commands.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -432**          Different dimensional array exist.

An array variable with same name but a different number of dimensions already exists.

⇒ *Change the name of the new variable or provide the same number of dimensions as the existing variable. Refer to the AS Language Manual for proper use of AS Language commands.*

---

**ERROR CODE -433**          Array variable exist.

Attempted to create a non-array variable with the same name as an existing array variable.

⇒ *Confirm array variable exists. Select new name for non-array variable or select array variable.*

---

**ERROR CODE -434**          Non array variable exist.

Attempted to create an array variable with the same name as an existing non-array variable.

⇒ *Confirm non-array variable exists. Select new name for array variable or select non-array variable.*

---

**ERROR CODE -435**          Array variable expected.

A non-array variable exists with the same name as the array variable that was specified for the decompose instruction.

⇒ *Select a new name for the array variable that was specified for the decompose instruction.*

---

**ERROR CODE -440**          Local variable expected.

A subroutine call to a program with a local variable found a program with the same name but without a local variable.

⇒ *Refer to the AS Language Manual for proper use of AS Language commands.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -441** Unexpected suffix.

No data was provided in the brackets of an array variable.

⇒ *Refer to the AS Language Manual for proper use of AS Language commands.*

---

**ERROR CODE -442** Mismatch of arguments at subroutine call.

The order of the local variable in the called program and the order of the local variable in the actual program are different.

⇒ *Check the order of the local variable in the actual program and modify the subroutine call accordingly. Refer to the AS Language Manual for proper use of AS Language commands.*

---

**ERROR CODE -443** Mismatch of argument type at subroutine call.

The type of argument, i.e. location variable, real variable, is different than the argument in the origin program.

⇒ *Correct the type of the argument in the called program or origin program.*

---

**ERROR CODE -450** Control structure error.

Displayed when an illegal program control flow structure is evaluated.

⇒ *Use correct syntax and components in control flow structures.*

---

**ERROR CODE -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/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -453**      Step:xxx Cannot terminate DO with END.

This error is displayed when the control flow structure DO...UNTIL is entered with an END statement .

⇒ *Use correct syntax and components in control flow structures.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -459**      Step:xxx Too many control structures.

Occurs when 11 layers of control flow structure are exceeded.

⇒ *Limit control flow structure to 11 layers.*

---

**ERROR CODE -460**      Variable (or program) already exists.

Displayed when a variable is entered that is already part of the system memory. For example: a location named "weld1" is entered when a program named "weld1" is already in the memory.

⇒ *Do not use variable names for more than one item.*

---

**ERROR CODE -461**      Variable of different type already exists.

Displayed when a variable is entered that is already part of the system memory. For example: a location named "weld1" is entered when a program named "weld1" is already in the memory.

⇒ *Do not use variable names for more than one item.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -464** Internal buffer over.

Attempted to solve a complex mathematical equation in a program.

⇒ *Rewrite to reduce the complexity of the operations required to solve the equation.*

---

**ERROR CODE -465** Undefined Variable (or program).

Attempt call a subroutine or process a variable that does not exist.

⇒ *Verify the variable or program exists.*

---

**ERROR CODE -466** Illegal clock value.

Time or date was entered in the wrong format.

⇒ *Re-enter the values correctly; time - military, date - yy/mm/dd.*

---

**ERROR CODE -470** Expect “=”.

The argument is missing the necessary “=”.

⇒ *Check the argument and correct it.*

---

**ERROR CODE -471** Expect “)”.

The argument is missing the necessary “)”.

⇒ *Check the argument and correct it.*

---

**ERROR CODE -472** Expect “]”.

The argument is missing the necessary “]”.

⇒ *Check the argument and correct it.*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -473**      “Expect “TO”.

The argument is missing the necessary “TO”.

⇒ *Check the argument and correct it.*

---

**ERROR CODE -474**      “Expect “BY”.

The argument is missing the necessary “BY”.

⇒ *Check the argument and correct it.*

---

**ERROR CODE -475**      Expect “:”.

The argument is missing the necessary “:”.

⇒ *Check the argument and correct it.*

---

**ERROR CODE -476**      “Expect ”ON/OFF”.

An instruction other than ON/OFF was used with a SYSTEM SWITCH.

⇒ *Check the ON/OFF status SYSTEM SWITCH and input it correctly.*

---

**ERROR CODE -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.*

---

**ERROR CODE -494**      Program is interlocked by another procedure.

Attempt to edit a running program was made.

⇒ *Stop program execution before editing the program.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -499** Invalid statement.

Occurs when the program is executed and instructions are encountered that cannot be processed as AS Language commands.

⇒ *Use correct syntax and components for AS Language commands.*

---

**ERROR CODE -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.

---

**ERROR CODE -514** Device is not ready.

Unused

---

**ERROR CODE -523** Illegal file name.

Unused

---

**ERROR CODE -543** Data read error.

Unused

---

**ERROR CODE -545** Record inhibited. Set ""record accept"" and operate again.

Displayed when an attempt was made to enter data, but the RECORD INHIBIT system switch was set to inhibit.

⇒ *Change the setting of the RECORD INHIBIT system switch.*

---

**ERROR CODE -551** Cannot open the file.

Unused

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -581**          Stop of process.

*(Option: Host communication I correspondence.)*  
*Refer to the host communication manual for details.*

---

**ERROR CODE -583**          Receive not data after receive request.

*(Option: Host communication I correspondence.)*  
*Refer to the host communication manual for details.*

---

**ERROR CODE -584**          Too long receive data (MAX=255 character).

*(Option: Host communication I correspondence.)*  
*Refer to the host communication manual for details.*

---

**ERROR CODE -585**          Abnormal data (EOT) received in communicating.

*(Option: Host communication I correspondence.)*  
*Refer to the host communication manual for details.*

---

**ERROR CODE -586**          Time out.

*(Option: Host communication I correspondence.)*  
*Refer to the host communication manual for details.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -591**            Illegal device number.

Two types of selections are possible for the serial port. This error occurs when port numbers other than the sensor port are specified.

*(Option: Host communication / correspondence.)  
Refer to the host communication manual for details.*

---

**ERROR CODE -596**            Cannot attach terminal.

The prompt instruction was executed by two or more programs at the same time.

*⇒ Do not execute the prompt instruction from two or more programs at the same time.*

---

**ERROR CODE -597**            Cannot attach communication port.

The RECEIVE instruction and the SEND instruction were executed by two or more programs at the same time.

*⇒ Do not execute these instructions from two or more programs at the same time.*

---

**ERROR CODE -598**            Cannot execute on this terminal.

Attempt was made to execute a command that cannot be used at that terminal. Some commands can only be used at the multi function panel and some only at the PC.

*⇒ Use the above command from the proper terminal.*

---

**ERROR CODE -599**            Waiting input data for PROMPT. Connect input device.

The input device specified by the prompt command, i.e., PC or MFP, was not connected.

*⇒ Verify that the specified device is connected.*

---

**ERROR CODE -600**            Motor power OFF. Displayed whenever an emergency stop is encountered.

*⇒ Reset emergency stop button and reapply motor power.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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 function 114.*

---

**ERROR CODE -612** Weld fault input.

Displayed when the controller receives a weld fault signal.

⇒ *Check operation of welding equipment, ensure signal numbers are correctly set.*

---

**ERROR CODE -613** Retract pos. monitor error.

Occurs when the input signal for the retract operation of a spot welding application is not received.

⇒ *Check the operation of welding equipment, check the signal number setting in auxiliary function 114-10.*

---

**ERROR CODE -614** Extend pos. monitor error.

Occurs when the input signal for the extend operation of a spot welding application is not received.

⇒ *Check the operation of welding equipment, check the signal number setting in auxiliary function 114-10.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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 number setting in auxiliary function 114-11.*

---

**ERROR CODE -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 CODE -631** Cannot achieve desired pressure.

The gun does not obtain the set closing pressure within 5 seconds after the gun starts closing.

1. Tip wear measurement is not executed.
2. Incorrect taught point (closing pressure is not defined for the recorded position).
3. Closing pressure is set too high.

⇒ *Perform the tip wear measurement.*

⇒ *Reteach the point.*

⇒ *Decrease the closing pressure.*

---

**ERROR CODE -632** Gun chip[tip] stick.

Indicates that the tips are stuck. More power is required to open the gun, following a weld, than the stuck detection value, set in AUX 114-41, WELDING DETECTION(0:NO CHK) [kgf].

If the stuck detection value is set below the power level required for normal operation, the gun cannot operate properly and the error is set.

⇒ *If the gun is stuck, release it manually.*

⇒ *If the gun is not stuck, verify that the stuck detection value (AUX 114-41, WELDING DETECTION(0:NO CHK) [kgf]) is set correctly. This setting may need to be increased.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -633**          Copper plate abrasion over the limit. step=\*

The copper backing plate wear exceeds the copper plate wear limit (AUX 114-12). The error is detected when the gun executes a weld.

⇒ *Replace the copper backing plate.*

---

**ERROR CODE -634**          Not dedicated encoder and brake power off signal.

The gun separation accept signal is not received during a gun change.

⇒ *From AUX 111 Dedicated Input Signal, set the gun separation accept signal number (encoder brake power supply OFF control signal).*

---

**ERROR CODE -641**          Now servo gun is disconnected or a different gun.

The robot executes a step when a gun is not attached to the tool changer or the gun number in the tool changer is different than the gun number for the program step.

- ⇒ *Execute the step in manual mode or manually install the correct gun. In the teach or check mode, verify connection of the gun by pressing the connect/separate button.*
  - ⇒ *If the correct gun is attached to the tool changer, confirm that the gun signal number (AUX 114-41, GUN CONNECTION SIGNAL) is correct. From the MONITOR (INPUT SIGNAL) screen, confirm that the signal number is received.*
  - ⇒ *Inspect the connectivity from the gun to the 1FG board.*
  - ⇒ *Temporarily set the gun connect signal number (AUX 114-41, GUN CONNECTION SIGNAL) to 0 (when the gun connect number is set to 0, the error is not detected); ensure that the correct gun is in the tool changer. Repair the gun connection and set the gun connect signal number as soon as possible.*
-

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -642** Calibration is not completed.

The controller terminates the calibration operation after a gun change.

⇒ *Reset the error and perform a manual gun change.*

---

**ERROR CODE -643** Measure of chip abrasion (stage 1) was not executed.

When the tip wear measurement program stage 2 (reference plate) is attempted, without first performing stage 1.

⇒ *Execute tip wear measurement program stage 1 prior to executing stage 2.*

---

**ERROR CODE -644** Work sensing signal (gun\_chip[tip] touch sig) is not established.

A workpiece thickness measurement is attempted, when the gun tip touch signal (set in AUX 114-41) has not been received.

⇒ *Set the gun tip touch signal number (AUX 114-41).*

---

**ERROR CODE -645** Cannot weld because of abnormal thickness.

The thickness of the workpiece is out of tolerance.

Probable causes:

1. Defective workpiece.
2. Incorrect positioning of the workpiece
3. A gap between the two panels being welded.
4. Current tip wear (AUX 114-41) and actual tip wear are different.

⇒ *Inspect the workpiece. Measure the thickness and positioning; replace workpiece and/or correct abnormal positioning.*

⇒ *Execute the tip wear measurement program. Manually measure the tip wear; compare these measurements to the current tip wear data (AUX 114-41).*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -646** Servo welding gun mechanical parameter is not established.

When connecting a new gun, the mechanical parameters are not defined.

⇒ *Define the parameters in AUX 114-12 SPOT WELD GUN DEFINITION and AUX114-42 SERVO WELDING MECHANICAL PARA.*

---

**ERROR CODE -653** Illegal DOUBLE OX output.

When Double type signals are used, if either output in the pair is turned ON, the other turns OFF. An instruction to turn both outputs ON or turn both outputs OFF (OX=+1,2 or OX=-1,2) causes this error.

⇒ *Modify using the teaching screen.*

---

**ERROR CODE -654** Cannot use DOUBLE OX.

When Double type signals are used, if either output in the pair is turned ON, the other turns OFF. An instruction (BITS, PULSE, DELSIG, etc.) to change the state of either output causes this error to occur.

⇒ *Modify using the teaching screen.*

---

**ERROR CODE -660** Gun chip abrasion over the limit.

The maximum tip wear limit (AUX 114-41 MAX ABRASION MOVING/FIXED[mm]) is exceeded.

⇒ *Replace the tips.*

---

**ERROR CODE -662** Start point position error for circle.

The robot was stopped after it began executing a circular motion and moved 4 mm or more away from its calculated path (jogging or brake slippage, etc.). Attempting to continue execution of the path from this position will result in this error.

⇒ *Move the robot closer to the calculated path or restart the program a step prior to the circular path.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -671**          Cannot execute in check back mode.

The program reached an instruction which cannot be executed in check back mode.

⇒ *Execute after selecting a step that can be checked backward. If possible, check forward.*

---

**ERROR CODE -672**          Cannot execute in ONE program.

The instructions specified in the ONE program area not valid for this type of program.

⇒ *Refer to the AS Language manual for proper use of the AS Language commands.*

---

**ERROR CODE -673**          Angle between JT2 and JT3 is out of range at start location.

Movement from the current position to the start location position would cause JT2 and JT3 to move beyond their limits.

⇒ *Modify the position of the start location of the program.*

---

**ERROR CODE -674**          Angle between JT2 and JT3 is out of range at end location.

Movement from the current position to the end location position would cause JT2 and JT3 to move beyond their limits.

⇒ *Modify the position of the end location of the program.*

---

**ERROR CODE -675**          Terminal is not connected.

A PRINT, TYPE, or PROMPT command is specified for display on a PC that is not connected.

⇒ *Verify the PC connections or modify the commands for display on the MFP.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -676**          Cannot input /output to multi function panel.

A PRINT, TYPE, or PROMPT command is specified for display on the MFP, although noMFP connection is present.

⇒ *Verify the MFP connections or modify the commands for display on the PC.*

---

**ERROR CODE -691**          Cannot change two or more guns at the same step.

The operator attempts to execute a step with more than one gun change.

⇒ *Edit the program so that there is only one gun change per step.*

---

**ERROR CODE -692**          Gun is connected other.

The tool changer attempts to connect to a gun that is already connected to another tool changer or is missing.

⇒ *Select a different gun or disconnect the gun from the other tool changer.*

**NOTE**

The following are conventional robot errors; however, when they occur for the servo gun axis, the following additional causes are possible.

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -801**      No program step. The step specified for execution does not exist.

⇒ *Select valid step numbers for execution.*

---

**ERROR CODE -802**      Nonexistent label.

Occurs when executing the GOTO command and the destination label is not defined.

⇒ *Ensure valid labels are used within the program.*

---

**ERROR CODE -803**      Undefined variable. Variable data for a specific argument in a command is not defined.

⇒ *Ensure variables are properly defined.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -808** Illegal value. The numeric value entered exceeds the upper or lower limits of the acceptable range.

⇒ *Enter data that is within acceptable range.*

---

**ERROR CODE -809** Undefined array suffix.

Attempt to process an array variable with undefined suffixes, i.e., attempting to process A=B[C,D,E], but C, D, E have no values.

⇒ *Rewrite the program to define the array suffix before attempting to process the array.*

---

**ERROR CODE -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 calculations.*

---

**ERROR CODE -812** Character string is too long.

Character strings associated with arithmetic or comparative operators or the LEN function are too long.

⇒ *Correct the program.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -813**          Illegal exponential operation.

Numeric values that have exponents must be positive in value.

⇒ *Correct equations in program.*

---

**ERROR CODE -814**          Expression too complicated.

A numeric calculation too complex to be evaluated was encountered.

⇒ *Simplify mathematical equations.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -821**      Illegal argument value.

Displayed when the parameter specified for an command or instruction is illegal. For example: TOOL 2112 (after the TOOL command a defined transformation location or null is expected, 2112 causes error).

⇒ *Use correct argument values.*

---

**ERROR CODE -822**      Illegal joint number.

The joint numbered entered does not exist or is in the wrong format.

⇒ *Retype the command using the correct format.*

---

**ERROR CODE -823**      Illegal signal number.

This is displayed when the SIG or BITS command is used and the specified signal number is beyond the range permitted by the system configuration.

⇒ *Use acceptable signal numbers for system configuration.*

---

**ERROR CODE -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.*

---

**ERROR CODE -825**      Illegal signal number.

When the RUNMASK, SIGNAL, BITS, PULSE, or SWAIT commands are used with a signal number that exceeds the range permitted by system configuration, this error is displayed.

⇒ *Check signal number specified in instruction and ensure it is within system configuration.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -829**      Illegal signal number.

This error is 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 the instruction and ensure it is within system configuration.*

---

**ERROR CODE -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.*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -840**      Too many subroutines.

Occurs when more than 20 subroutines are nested with EXTCALL or CALL instructions.

⇒ *Do not exceed 20 nested subroutines.*

---

**ERROR CODE -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.*

---

**ERROR CODE -846**      No program exist.

Pertinent program does not exist in DIRECTORY and LIST commands.

⇒ *Correct program or make program.*

---

**ERROR CODE -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 too low.

⇒ *Set lower software travel limits to an acceptable range.*

---

**ERROR CODE -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 too high.

⇒ *Set upper software travel limits to an acceptable range.*

---

**ERROR CODE -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 too low.

⇒ *Set lower software travel limits to an acceptable range.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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.*

---

**ERROR CODE -854**            Current position of jt\* is out of range.

This error can occur when two guns are used. When a program is taught with one gun, and a second gun is used to perform the program, the second gun may not be able to execute a program step. This may be due to differences in the physical size or mechanical characteristics of the two guns.

⇒ *Reteach the point(s), using a configuration which allows the robot to reach the desired position with both guns.*

---

**ERROR CODE -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 accept location.*

---

**ERROR CODE -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 accept location.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -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 accept location.*

---

**ERROR CODE -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.*

---

**ERROR CODE -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.*

---

**ERROR CODE -872** Cannot execute motion instruction in PC program.

A PC program cannot 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 CODE -873** Illegal auxiliary data number.

The value selected for auxiliary data (speed, timer, tool etc.) exceeds allowable range.

⇒ *Correct the value of auxiliary data.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -874**          No circular location.

Program circular motion instructions must have C1 moves followed by either a C1 or C2 move.

⇒ *Correct program instructions.*

---

**ERROR CODE -875**          No C1MOVE(CIR1) ins.

Program circular motion instructions must have C2 moves preceded by a C1 move.

⇒ *Correct program instructions.*

---

**ERROR CODE -876**          Cannot create circle.

Circular interpolation moves cannot be processed because the points identified are too narrow or are on a straight path.

⇒ *Correct program instructions.*

---

**ERROR CODE -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 instructions to match software configuration.*

---

**ERROR CODE -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 instructions to match software configuration.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -896**      Option is not set up, can't execute.

⇒ *Contact customer service for option specifications after confirming the purchase specification.*

---

**ERROR CODE -900**      Arc failure.

The signal (current detection: WCR) that indicates the robot is executing the weld did not return from the welder for 1 sec. or more after welding started.

⇒ *Check for insulating debris that prevents welding.*  
⇒ *Confirm that adequate supply of wire is available.*  
⇒ *Confirm that wire is not stuck to the welding tip.*

---

**ERROR CODE -901**      Wire stuck.

The wire is stuck to the base metal at the weld end.

⇒ *Cut the wire.*  
⇒ *Change weld conditions if problem occurs frequently.*

---

**ERROR CODE -902**      Electric pole stuck.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -909**      Watch-Dog Error RS485 Special Communication Board.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -910**      Work not detected (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -911**      Undefined sensing direction. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -912**          Insufficient sensing points. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -913**          Mother or daughter work does not exist. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -914**          Number of sensing points exceeded. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -915**          Illegal work appointment. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -916**          Illegal sensing points appointment. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -917**          Wire check failed. (Touch sensing)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -919**          No RS485 Special Communication board.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -920**          Illegal welding condition number.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -921**          Weld data not set up.

⇒ *Contact KRI Customer Service.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -922** Weld data out of range.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -934** No weld data base.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -935** Cannot change condition.

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -951** No RTPM board. (RTPM)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -960** To many taught points for RTPM. (RTPM)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -961** RTPM arc sensor error. (RTPM)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -962** Out of RTPM tracking value. (RTPM)

⇒ *Contact KRI Customer Service.*

---

**ERROR CODE -963** Out of RTPM tracking capacity. (RTPM) *↳ Contact KRI Customer Service.*

---

**ERROR CODE -964** RTPM current deviation error. (RTPM)

⇒ *Contact KRI Customer Service.*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -990**          No welding interface board.

The welding interface board was not detected at control power on.

- ⇒ Turn control power off and confirm that the weld interface board is mounted.
  - ⇒ Confirm that the weld interface board is not loose. Insert the board firmly.
- 

**ERROR CODE -999**          No welding interface board.

The welding interface board was not detected at control power on.

- ⇒ Turn control power off and confirm that the weld interface board is mounted.
  - ⇒ Confirm that the weld interface board is not loose. Insert the board firmly.
- 

**ERROR CODE -1003**        Data base error.

The program storage area of the system memory has been damaged and is not linking data correctly.

- ⇒ Turn on the 1GA board switch SW2-8 to initialize the memory, do not use AUX100 or SYSINIT command. Reload the teach data.
  - ⇒ Check that system is properly isolated from electrical noise.
  - ⇒ Check the memory backup battery. Replace if necessary.
  - ⇒ Replace the 1GA board if the error re-occurs.
-

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1012** Command position of jt-x has suddenly changed.

### Spot welding and material handling

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 200 mm/sec.

- ⇒ *Check for singularity condition during robot motion and reteach to correct.*
- ⇒ *Check for loose or defective servo system encoders, harnesses, and circuit boards.*

### Servo welding gun

Occurs when corrections for deflection cause movement of the robot to exceed the range of motion of one or more axes. This condition is not limited to the gun axis; any axis, including the gun axis, can cause this error.

- ⇒ *Change the posture of the robot.*
  - ⇒ *Decrease the speed in the step.*
  - ⇒ *Reduce the amount of the deflection correction in the step by using AS Language SGREFLEX command.*
  - ⇒ *Reteach the point and change the configuration of the posture of the robot if needed.*
- 

**ERROR CODE -1014** Commanded position of jt-x is out range.

### Spot welding and material handling

The commanded position for the joint identified has exceeded the software limits. Condition is monitored in both the check and repeat modes.

- ⇒ *Correct taught positions to avoid the software limits.*
- ⇒ *Adjust the software limits to provide the necessary work envelope.*

### Servo welding gun

If a taught point is near a servo gun or arm axis range of motion limit, the combination of tip wear and deflection correction can prevent the robot from reaching the taught point.

- ⇒ *Do not teach points that are near axes range of motion limits.*
-

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1017**      Angle between JT2 and JT3 is out of range.

The commanded position for JT2 and JT3 cannot be reached due to the nature of the mechanical links that exist between these points for the U-series, EH, and ES robots. This error does not apply to JS-series robots. The main cause of this error is that the taught positions, including positions through which the robot moves, are bad.

⇒ *Change the taught positions.*

---

**ERROR CODE -1019**      Check sum error of system data.

The check sum of the system data of the AS software was changed when the system information such as model number, number of axis, and option setting, was downloaded. When the error occurs in situations other than downloading, the error is caused by defective memory back-up, defective 1GA board, or memory error from noise.

- ⇒ *Use Aux 78 CLEAR CHECK SUM ERROR or CHSUM command to reset the data. When the error cannot be reset using the check sum commands, the command with the abnormality is shown. Rewrite the command and use CHSUM or Aux 78 to clear the error.*
- ⇒ *Check the memory backup battery. Replace if necessary.*
- ⇒ *Replace the 1GA board if the error re-occurs.*
- 

**ERROR CODE -1022**      RAM battery low voltage <board name>.

When control power is applied or motor power is turned on, a voltage check is performed on the batteries that maintain SRAM memory when power is off. This message is displayed when a voltage of 2.5 VDC or less is detected (normal is 3.6 VDC).

- ⇒ *Back up system and program data, check and replace batteries as required.*
- ⇒ *Check batteries for defective connections. Repair the defective connection.*
- ⇒ *Defective battery voltage monitoring circuit. Replace the circuit board.*
-

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1025** AS Flash memory sum check error.

A check sum error of AS system data in flash memory on 1GA board occurred when the power was turned on. The check sum data is created when the FCHK command is executed and is recorded in flash memory during download. Main causes of the error are:

1. When the AS system was downloaded, the FCHK command was not executed.
2. The addressing of the FCHK command was wrong.
3. The flash memory and 1GA board are defective.
4. The system data in the flash memory is damaged.

⇒ *Confirm the content of the command as\_load.cmd file in the IC card if error occurs immediately after downloading the AS system. Download system again. If error continues after download, exchange the 1GA board.*

---

**ERROR CODE -1026** Servo Flash memory sum check error.

A check sum error of the servo system in flash memory on 1GA board occurred when the power was turned on. The check sum data is created when the FCHK command is executed and is recorded in flash memory during download. Main causes of the error are:

1. When the servo system was downloaded, the FCHK command was not executed.
2. The addressing of the FCHK command was wrong.
3. The flash memory and 1GA board are defective.
4. The system data in flash memory is damaged.

⇒ *If the error occurs immediately after download, confirm the content of as\_load.cmd in the PC card and download again. If the error persists, change the 1GA board.*

---

**ERROR CODE -1051** Cannot execute in this robot arm.

Unused

---

---

**ERROR CODES/TROUBLESHOOTING****ERROR CODE -1100** CPU Error (Code = \*\*\*\*).

The 1GA board CPU has stopped (detected with the AS software). This error is caused by defective AS or servo software, defective hardware, or noise related malfunction.

⇒ *With a PC connected enter "\$save/flt filename" to save the fault data, and send to KHI.*

⇒ *Confirm which board the error occurred on by referring to the message that appears on the PC. This error does not appear on the multi function panel or the small teach pendant.*

*I/O BUS ERROR: error in the address of the 1FR or 1GW board.*

*PSB BUS ERROR: error with the 1FP or 1HP power sequence board.*

*VME BUS ERROR: error in the address of the boards that use the VME bus.*

⇒ *If the initialization prompt appears, answer "no", then reload software. If the error returns, initialize the system and reload software. If the multi function panel or PC are not functional, system initialization cannot be performed.*

⇒ *If this error occurs during certain specific operations it may indicate a defect in the AS system.*

⇒ *Replace the 1GA board if the error cannot be reproduced or cleared.*

---

**ERROR CODE -1101** Main CPU BUS error.

On the 1GA board, a bus error (in the VME bus line, data processing was not able to be done normally) has occurred (detected with the AS software). This error is caused by defective AS or servo software, defective 1GA board, or noise related malfunction.

⇒ *With PC connected enter "\$save/flt filename" to save the fault data, and send to KHI.*

⇒ *Confirm which board the error occurred on by referring to the message that appears on the PC. This error does not appear on the multi function panel or the small teach pendant.*

*I/O BUS ERROR: error in the address of the 1FR or 1GW board.*

*PSB BUS ERROR: error with the 1FP or 1HP power sequence board.*

*VME BUS ERROR: error in the address of the boards that use the VME bus.*

⇒ *If the initialization prompt appears, answer "no", then reload software. If the error returns, initialize the system and reload software. If the multi function panel or PC are not functional, system initialization cannot be performed.*

⇒ *If this error occurs during certain specific operations it may indicate a defect in the AS system.*

⇒ *Replace the 1GA board if the error cannot be reproduced or cleared.*

---

---

**ERROR CODES/TROUBLESHOOTING****ERROR CODE -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. This error is caused by defective AS or servo software, defective 1GA board, or noise related malfunction.

⇒ *With a PC connected enter "\$save/flt filename" to save the fault data, and send to KHI.*

⇒ *Confirm which board the error occurred on by referring to the message that appears on the PC. This error does not appear on the multi function panel or the small teach pendant.*

*I/O BUS ERROR: error in the address of the 1FR or 1GW board.*

*PSB BUS ERROR: error with the 1FP or 1HP power sequence board.*

*VME BUS ERROR: error in the address of the boards that use the VME bus.*

⇒ *If the initialization prompt appears, answer "no", then reload software. If the error returns, initialize the system and reload software. If the multi function panel or PC are not functional, system initialization cannot be performed.*

⇒ *If this error occurs during certain specific operations it may indicate a defect in the AS system.*

⇒ *Replace the 1GA board if the error cannot be reproduced or cleared.*

---

**ERROR CODE -1200** Encoder board is not installed.

Unused

---

**ERROR CODE -1201** Power sequence board is not installed.

The first address of the No.1 power sequence board (1FP/1HP board) cannot be read when control power is turned ON. Main causes for this error include:

1. The power sequence board (1FP/1HP board) is not installed in the correct card slot.
2. Jumper or dip switch settings of the power sequence board are incorrect.
3. Defect in the power sequence board.

⇒ *Ensure the 1FP/1HP board DSW1 setting is configured as power sequence board No. 1 and is installed in the 1FP/1HP card rack slot. Check and replace the board if necessary.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1202** No2 power sequence board is not installed.

The first address of the No.2 power sequence board (1FP/1HP board) cannot be read when the control power is turned ON. Main causes for this error include:

1. The power sequence board (1FP/1HP board) is not installed in the correct card slot.
2. Jumper or dip switch settings of the power sequence board are incorrect.
3. Defect in the power sequence board.

⇒ *Ensure the 1FP/1HP board DSW1 setting is configured as power sequence board No. 2 and is installed in the 1FP/1HP card rack slot. Check and replace the board if necessary.*

---

**ERROR CODE -1203** No x-M I/O board is not installed.

Cannot read the first addresses of I/O boards (1FR/1GW board, etc.) based on the number of signals that have been set with DO (output point), or DI (input point) of the ZSIGSPEC command, when the control power supply is turned on. This error is not used when a 1FS board is installed. Main causes include:

1. Incorrect value set with the ZSIGSPEC command.
2. Error in board address of I/O board (1FR/1GW board, etc.).
3. Defect of I/O board (1FR/1GW board, etc.).

⇒ *Set the correct number of signals with the ZSIGSPEC command. I/O cards are typically mounted in the card rack from right to left in ascending order. Because of VME buss communication, this is not critical. The robot can run without I/O boards. The maximum number of signals in the software set with the ZIGSPEC command is DO, DI=256, INT=512, but DO and DI are restricted by hardware and other options.*

⇒ *Set board address jumpers correctly.*

⇒ *Replace the 1FR/1GW board.*

---

**ERROR CODE -1204** Option SIO port is not installed.

The 1GA board sets more than seven axes at control power ON, but the IC (SIO) for serial communications for command line 3 and 4CH communication with the servo board is not installed. This error typically does not occur because SIO is installed on standard 1GA boards but may not be on the prototype boards. Main causes of this error include more than seven axes set with a prototype board with SIO not mounted or a defect in the 1GA board.

⇒ *Replace the 1GA board.*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1205** Power sequence board any error.

An error signal, not classified through the error summary, is generated by the 1FP/1HP board and not recognized by the AS software. Main causes include:

1. The error detection function of the power sequence board does not correspond to the error processing function of the AS software.
2. Defect in the power sequence board.
3. Defect in the 1GB board.
4. Defect in the wiring between the 1FP/1HP board and the 1GB board.  
(XGB-CN1 ↔ XHZ-CN4)

⇒ *Replace the power sequence board.*

⇒ *Repair/Replace the 1FP/1HP board to the 1GB board wiring harness.*

⇒ *Install the correct version of the AS software.*

---

**ERROR CODE -1206** Built-in sequence board is not installed.

Installation of the built-in sequencer board is checked when the control power supply is turned ON. This error occurs when the built-in sequencer board is not installed.

⇒ *Install the built-in sequencer board.*

---

**ERROR CODE -1208** RI/O board is not installed.

This error occurs when the control power supply is turned on and the first address of the RI/O board is not read (1GW, 1FS etc.).

1. The RI/O board is not installed.
2. Defect in the RI/O board.

⇒ *Ensure the RI/O board installed.*

⇒ *Replace the RI/O board.*

---



---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1209** RI/O board initialize error.

This error occurs when the control power supply is turned on and the RI/O board does not successfully initialize (1FS board).

1. Incorrect dip switch settings on the RI/O.
2. Defect in the RI/O board software.

⇒ *Set dip switch correctly on RI/O board.*

⇒ *Reinstall the RI/O board software.*

---

**ERROR CODE -1247** Axis setting data incorrect.

The command channel line for the external axis was incorrectly set using AUX 901, External Axis Set.

⇒ *Input the correct value.*

---

**ERROR CODE -1248** Number of Axis Changed! SYSINI.

The number of axis assigned to the robot has changed.

⇒ *Change the number of axes and initialize the system.*

---

**ERROR CODE -1249** Servo parameter Changed! Control power turn OFF & ON.

Servo parameters in system data changed during LOAD.

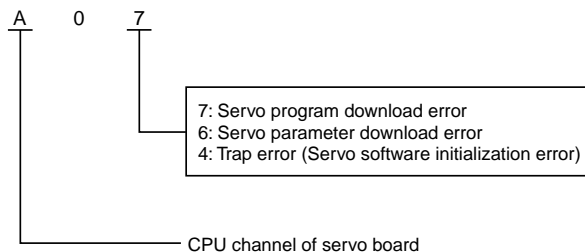
⇒ *Turn control power supply OFF and ON.*

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1250** Servo board (X) Initialize error.

The servo software was not successfully loaded from the FLASH memory on the 1GA board to the servo board when control power was turned ON. The contents of (X) is an alphanumeric code as indicated below:



CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)	
A	First 1GB PC Board A Unit	Master	
B	First 1GB PC Board B Unit	Master	
C	First 1GB PC Board C Unit or Second 1GB P Board A Unit or One Axis Amplifier	Slave	
	Second 1GB P Board B Unit or One Axis Amplifier		Slave

Main causes include:

1. Defect in the 1GB servo board.
2. Defect of the harness between power sequence board and servo board.
3. Error in mother board jumper setting.
4. The servo software is not in the flash memory on the 1GA board.  
(XGB-CN1↔XHZ-CN4)
5. Malfunction caused by noise, etc.
6. Servo software and AS software versions incompatible.
7. Versions of ROM in servo software and servo board not compatible.

⇒ Replace the 1GA, servo board, and each harness.

⇒ Confirm the servo software and the servo board monitor ROM version and install the corresponding servo software again. Confirmation of the version can be confirmed by AUX 90, Software Version Display, or ID command.

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1251** Servo board (X) communication error.

Communications in the command line of each CPU of the servo board (1GB board) failed twice consecutively. The command line always communicates every 4 msec. When each CPU of the servo board stops, this error might be detected. Main causes include:

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
	or	
	One Axis Amplifier	
D	Second 1GB P Board B Unit	Slave
	or	
	One Axis Amplifier	

1. Defect in the 1GA board.
2. Defect in the 1GB servo board.
3. Defect in each communication harness.
4. Encoder defect if JT3 (A) or JT4 (B) are displayed.
5. Noise malfunction.
6. Mismatch of AS and servo software
7. Improper setting of AUX 928; servo type 2.

⇒ Ensure software versions are correct.

⇒ Ensure AUX 928 setting is correct.

⇒ *Replace the 1GA, servo board, and each harness.*

## ERROR CODES/TROUBLESHOOTING

### ERROR CODE -1252(A), -1253(B), -1254(C), -1255(D)

Servo board (x) hardware error code = xxxx.

A hardware error was detected by the 1GB board with no corresponding error in the AS software. The internal error code number that the AS software detected is displayed in xxxx. The unit name of the servo board displayed at x in the error message corresponds to the CH column in the table below.

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
	or One Axis Amplifier	
D	Second 1GB P Board B Unit	Slave
	or	
	One Axis Amplifier	

Main causes of this problem include the following abnormalities:

1. Defect in the servo software.
2. Noise malfunction.
3. Defect in the harness between servo board and power sequence board.
4. Defect of the servo board.
5. Versions of servo and AS software incompatible.
6. Malfunction in the servo or AS software.

⇒ *Install upgraded versions of servo and AS software.*

⇒ *Replace the 1GA board, the servo board, and each harness.*

⇒ *When this error occurs, contact KRI to report the details.*

## ERROR CODES/TROUBLESHOOTING

### ERROR CODE -1256(A), -1257(B), -1258(C), -1259(D)

Servo board (x) software error code = xxxx.

A hardware error was detected by the 1GB board with no corresponding error in the AS software. The internal error code number that the AS software detected is displayed in xxxx. The unit name of the servo board displayed at x in the error message corresponds to the CH column in the table below.

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
D	or	Slave
	One Axis Amplifier	
	Second 1GB P Board B Unit	
D	or	Slave
	One Axis Amplifier	
	One Axis Amplifier	

Main causes of this problem include the following abnormalities:

1. Defect in the servo software.
2. Noise malfunction.
3. Defect in the harness between servo board and power sequence board.
4. Defect of the servo board.
5. Versions of servo and AS software incompatible.
6. Malfunction in the servo or AS software.

⇒ *Install upgraded versions of servo and AS software.*

⇒ *Replace the 1GA board, the servo board, and each harness.*

⇒ *When this error occurs, contact KRI to report the details.*

**ERROR CODE -1260**      Option changed! SYSINI.

Unused

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1261** Servo board (x) parameter setting error.

After a setting was changed with AUX 976 servo parameter, the data sent to the servo board was different than the data received two or more times. The unit name of the servo board displayed at x in the error message corresponds to the CH column in the table below.

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
	or	
	One Axis Amplifier	
D	Second 1GB P Board B Unit	Slave
	or	
	One Axis Amplifier	

Main causes of this error include:

1. Defect in the 1GA board.
2. Defect in the servo board.
3. Defect of each communication harness.
4. Versions of servo and AS software incompatible.
5. Malfunction by noise.

⇒ *Replace the 1GA board, the servo board and each harness.*

⇒ *Install correct versions of AS and servo software.*

⇒ *If error cannot be reset, reset data with AUX 976.*

⇒ *If error cannot be reset, cycle control power.*

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1300** Servo CPU-(x) watch dog error.

The watch dog circuit on the 1GB board has detected a software problem. This is caused by a defective servo board or a problem with the servo software. The unit name of the servo board displayed at x in the error message corresponds to the CH column in the table below.

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
	or	
	One Axis Amplifier	
D	Second 1GB P Board B Unit	Slave
	or	
	One Axis Amplifier	

⇒ *Replace the servo board.*

---

**ERROR CODE-1306** Servo board command error

Servo software has returned the error code to the AS software. Main causes include:

1. Malfunction in the servo or AS software.
2. Noise malfunction.
3. Defect in the harness between servo board and power sequence board.
4. Defect of the servo board or the 1GA board.
5. Versions of servo and AS software incompatible.

⇒ *Install correct versions of servo and AS software.*

⇒ *Replace the 1GA board, the servo board, and each harness.*

---

---

**ERROR CODES/TROUBLESHOOTING****ERROR CODE -1308** Motor power off.

Occurs when software turns motor power on (K1 or K2 contactors on) but there is no signal feedback (MCON) indicating motor power is on. Main causes include:

1. Defect in the power sequence board (1FP/1HP board).
2. Defect in the relay board (1FY/1HY board).
3. Malfunction in the error detection circuit and error processing of each board.
4. Auxiliary contact of magnet contactor (K1, K2) for motor power is defective.
5. Malfunction with the servo software or the AS software.
6. Defect in the harnesses between XHY-CN3 and K1/K2.

⇒ *Replace the power sequence board.*

⇒ *Replace the relay board.*

⇒ *Replace the magnetic contactor.*

⇒ *Repair or replace the harnesses between XHY-CN3 and K1/K2.*

---

**ERROR CODE -1333** Monitor ID of servo board mismatch!

The monitor ROM software version of each unit on the 1GB board is different, caused by a defect of the 1GB board.

⇒ *Replace the 1GB board.*

---

**ERROR CODE -1334** Servo control line error.

Brake open command is not returned within 2 seconds when the servo control on (SVCN) signal is sent from the AS software to the servo board (1GB) through the EPLD register of the power sequence board (1FP/1HP). Main causes include:

1. Defect in the 1FP/1HP power sequence board.
2. Defect in the 1GB servo board.
3. Incorrect 1GB board dip switch settings.
4. Defect in the harness between the two boards. (XGB-CN1↔XHZ-CN4)
5. Defect in the harness between the 1GC/1GD power block and K3 contactor. (X1SA↔MS, X1-SA↔XGC/XGD-CN10/CN12)
6. Defect in the harness between the 1GB servo board and the 1GC power block. (XGB-CN12↔XGC/XGD-CN8, XGB-CN13↔XGC/XGD-CN9)
7. Defect in the 1GC/1GD power block.

⇒ *Repair or Replace harnesses as needed.*

⇒ *Exchange the power sequence board and the servo board, etc.*

⇒ *Check 1GB board dip switch settings.*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1336** Safety gate circuit open.

The safety gate branch of the safety circuit is open due to a missing safety plug or defective circuit.

- ⇒ *Reinsert the safety plug.*
  - ⇒ *Repair open circuit.*
  - ⇒ *Replace the power sequence board (1FP/1HP).*
- 

**ERROR CODE -1337** Two MC lines are not consistent.

A problem has been detected with the MC lines controlling the K1/K2 contactors, due to an open safety circuit, loose connectors or defects of the 1HP, 1HY, or 1HZ boards, or a defect of the MFP.

- ⇒ *Check the safety circuit.*
  - ⇒ *Check connections to the 1HP, 1HY, and 1HZ boards, and MFP.*
  - ⇒ *Replace the 1HP, 1HY, or 1HZ boards, or MFP as necessary.*
- 

**ERROR CODE -1338** K1 and/or K2 works wrong.

Feedback from the auxiliary contacts of the K1/K2 contactors was not received within a fixed time, when the contactor was energized. Main causes are safety circuit failures, K1/K2 auxiliary contact fault (welded, etc.), loose connectors or defects of the 1HP, 1HY, or 1HZ boards.

(XHP-CN3↔XHY-CN4↔1HY↔XHY-CN3↔K1/K2)

- ⇒ *Check the safety circuit.*
  - ⇒ *Check connections to the 1HP, 1HY, and 1HZ boards.*
  - ⇒ *Replace the 1HP, 1HY, or 1HZ boards as necessary.*
-

---

**ERROR CODES/TROUBLESHOOTING****ERROR CODE -1401** Amp over current jt x-M.

The feedback current from a current sensor in the power block exceeded 144% of the maximum instantaneous motor current rating. This error can be caused by the following abnormalities:

1. Short in the U, V, W from the power block to motor and ground wire.  
(power block↔X4/X5↔robot)
2. Defect in the motor.
3. Defect in the power block.
4. Defect in the servo board (1GB board, etc.)

⇒ *Check the connection for the U, V, W and ground line to the power block. Replace the separation harness if necessary.*

⇒ *Replace the motor, 1GB board, or power block.*

---

**ERROR CODE -1407** AMP power unit error.

General servo system error. This error indicates that a servo system error has occurred. Main causes include a defect in the connection of the harness between the power sequence board and the servo board.

⇒ *Check the servo error codes that follow this error for additional information. Check the harness connections between the 1HP board and 1GB board. Replace the 1HP board.*

---

**ERROR CODE -1413** Regenerative resistor overheat or disconnect.

Unused

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1420**      Current detector type (x) mismatch!

When the control power supply is turned ON, the ID code data of the 1GM board and the AS software installed on the 1GB board do not correspond. The unit name of the servo board displayed at x in the error message corresponds to the CH column in the table below.

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
D	or	Slave
	One Axis Amplifier	
	Second 1GB P Board B Unit	

Main causes include:

1. 1GM board is not suitable for the AS software (robot model).
2. The AS software does not correspond to the 1GM board.

⇒ *Install the 1GB board equipped with 1GM board.*

⇒ *Load the robot system data when this error occurs immediately after initializing.*

⇒ *Re-initialize and reload the robot/system data if this error occurred immediately after loading the data.*

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1500** Motor overload jt-x.

### Spot welding and material handling

The current feedback from the power block exceeded maximum continuous ratings longer than the time allowed. Main causes include:

Mechanical -

1. The robot arm has contacted an external item hindering movement.
2. The harness is caught in the robot arm.
3. The decelerator, the gear, or the bearing are damaged.
4. Gear decelerator backlash is too narrow.
5. Payload weight exceeds robot specifications for capacity.
6. Robot motion pattern exceeds ratings of the motor.
7. Motor brake is not released.

Electrical-

1. Motor power line U, V, and W phase are disconnected.  
(1GC↔power block↔X4/X5↔robot)
2. The brake line is disconnected or brake drive circuit is damaged.  
(T1↔TB1↔V1↔XHY-CN1↔1HY↔XHY-CN2↔XGB-CN14↔1GB↔  
XGB-CN6↔X5↔robot)
3. The power block is damaged.
4. Defect in the 1HY board.
5. Defect in the 1GB servo board.
6. Malfunction of the motor.

⇒ *Check the decelerator, etc. for mechanical failure. Replace as necessary.*

⇒ *Re-teach the robot motion as necessary.*

⇒ *Check the harness and servo unit for electrical malfunctions and replace if necessary.*

### Servo welding gun

1. Clamping pressure time is excessive.
2. Distance between weld points is short; the motor does not have time to cool sufficiently between welds.
3. Clamping pressure setting exceeds the servo gun motor power capacity.

⇒ *If the weld points are close to each other, set a time delay. This provides a cool-down period for the servo gun motor.*

⇒ *Set the clamping force lower (see servo gun manufacturer's specifications).*

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1501** Overheat or motor harness is disconnected (x).

Motor thermal circuit is open due to motor overheat or harness problem. Not all robot models utilize thermal switches in the motors. The unit name of the servo board displayed at x in the error message corresponds to the CH column in the table below.

CH	To Communication	Relay to Power Sequence Board (1FP/1HP PC Board)
A	First 1GB PC Board A Unit	Master
B	First 1GB PC Board B Unit	Master
C	First 1GB PC Board C Unit	Slave
	or	
	Second 1GB P Board A Unit	
	or	
	One Axis Amplifier	
D	Second 1GB P Board B Unit	Slave
	or	
	One Axis Amplifier	

Main causes include:

1. Disconnected thermal line.  
(1GB↔XGB-CN6↔XTH↔X4↔robot)
2. Defective connection in the separation harness.
3. Defect in the servo board.
4. When the thermal is built-in:
  - The robot rated weight capacity is exceeded.
  - The ambient temperature exceeds limits for use.
  - Constant execution of abrupt high speed direction changes.
  - Defect robot cooling fan or cooling air purge system.
  - Servo system holding the robot arm in high load position for extended periods.
  - Defect in the thermal switch.

⇒ *When the thermal switch opens, error cannot be reset until cooling is complete.*

⇒ *Confirm continuity of thermal line.*

⇒ *Replace the harness or servo board as necessary.*

⇒ *Use the auto servo off function, AUX 91, to prevent the servo system holding a high load arm position for extended periods.*

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1503**      Speed error jt-x.

Joint speed calculated with encoder value exceeded a regulated value. In the Repeat mode: the rated joint speed \* 1.2. In the Teach/Check modes: 250mm/sec. \* 1.5, at the equivalent radius position for a rotating joint or actual command value speed for a linear joint. This error is caused by the following abnormalities:

1. Disconnected motor power line U, V, and W phase, and defective power block.
2. Wiring error in motor power line and encoder line.
3. Disconnected encoder single line, short circuit and defective main body of encoder.
4. Robot dependent upon singularity motion.
5. Defect in the 1GB servo board and 1GB power block.
6. Moment of inertia exceeded the motor torque.

⇒ *Check for disconnection, short circuit, and proper wiring. Replace the harness and encoder if necessary.*

⇒ *Replace the servo unit.*

⇒ *Rewrite program teach data such as speed and position in the case of singularity motion.*

---

---

**ERROR CODES/TROUBLESHOOTING****ERROR CODE -1504**      Position envelope error jt-x.**spot welding and material handling**

The difference between the current value from the encoder and the command value in the AS software exceeded a regulated value which varies by robot model. Main causes include singularity motion or the following abnormalities:

Mechanical -

1. The robot arm has contacted an external item hindering movement.
2. The harness is caught in the robot arm.
3. The decelerator, the gear, or the bearing are damaged.
4. Gear decelerator backlash is too narrow.
5. Payload weight exceeds robot specifications for capacity.
6. Robot motion pattern exceeds ratings of the motor.
7. Motor brake is not released.

Electrical-

1. Motor power line U, V, and W phase are disconnected.  
(1GC↔power block↔X4/X5↔robot)
2. The brake line is disconnected or brake drive circuit is damaged.  
(T1↔TB1↔V1↔XHY-CN1↔1HY↔XHY-CN2↔XGB-CN14↔1GB↔  
XGB-CN6↔X5↔robot)
3. The power block is damaged.
4. Defect in the servo board.
5. Malfunction of the motor.

⇒ *Check the decelerator, etc. for mechanical failure. Replace as necessary.*

⇒ *Re-teach robot motion as necessary.*

⇒ *Check the harness and servo unit for electrical malfunctions and replace if necessary.*

⇒ *Correct the teach data in case of singularity motion.*

**Servo welding gun**

The thickness of the workpiece at the weld point is greater than the thickness when the point was taught. The thickness difference needed to set this error is approximately 10mm, depending on the gun type.

⇒ *Inspect the workpiece; replace it if necessary.*

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1505**      Velocity envelope error jt-x.

The robot's current velocity is unable to keep up with the velocity command signal.

⇒ *Refer to Error Code -1504: Position envelope error jt-x*

---

**ERROR CODE -1506**      Commanded speed error jt-x.

Unused.

---

**ERROR CODE -1507**      Commanded acceleration error jt-x.

Unused.

---

**ERROR CODE -1510**      Encoder harness broken jt-x.

The encoder signal is lost.

⇒ *Machine signal or separate signal harness*

⇒ *+12 VDC or +5 VDC at the 1FG board*

⇒ *1GB or 1FG board*

---



---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1511** Encoder battery voltage low [Servo (A)].

Voltage of the encoder back up battery on the 1FG board decreased to 3.2 volts, or the encoder battery alarm signal (BAL\_AL) was disconnected. This error is only detected at control power on or motor power on. This is caused by the following:

1. Discharged batteries.
2. Defective encoder battery board (1FG board).
3. Defective encoder.
4. Short in the harness from the encoder battery backup board to the encoder.
5. Battery discharge.
6. Defective servo board (1GB board).

This error is reported by servo board unit A.

⇒ *Exchange the battery. There is a possibility that internal encoder data may be lost. Check the robot zeroing.*

⇒ *Check each harness for short circuit.*

⇒ *Replace the encoder, the servo board, or the encoder battery backup board, if necessary.*

---

**ERROR CODE -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.

⇒ *Check connections at encoder, 1FG board, and separation harness.*

⇒ *Check for noise and shield integrity.*

⇒ *Check connections, jumpers, and switches on 1GB board.*

⇒ *Check for continuity in the machine harness and separation harness.*

⇒ *Replace 1FG board.*

⇒ *Replace the 1GB board.*

⇒ *Replace the encoder.*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1516** Encoder data abnormal jt-x.

When control power is switched ON, the current encoder value is compared to the stored encoder value when the power was turned OFF. If the difference is more than the value set by AUX 43, Encoder Error Range, or the ENCCCHK\_DATA command, this error occurs. Normal occurrence is due to the following:

1. Back up battery failure due to disconnection or discharged battery.
2. Control power was turned off during robot motion by an abnormal power supply, etc., causing the robot to stop in a position other than the last position memorized by the AS software.
3. Motor replacement/encoder replacement.
4. Robot arm or motor operated by force when control power was off.
5. Robot was initialized.

⇒ *Data may have been lost due to backup battery. See error -1511. Exchange the battery. There is a possibility that internal encoder data may be lost. Check the robot zeroing.*

⇒ *Check each harness for short circuit.*

⇒ *Replace the encoder, the servo board or the encoder battery backup board if necessary.*

---

**ERROR CODE -1517** Cannot read initial data encoder jt-x.

### Spot welding and material handling

Immediately after control power on, steady encoder data was not able to be read. This error is caused by a disconnected encoder signal line or a short circuit, or a defect in the main body of the encoder.

⇒ *Check for disconnection or short circuit for encoder signal line.*

⇒ *Replace the encoder.*

⇒ *Replace the servo board (1GB board).*

### Servo welding gun

The gun is disconnected from the tool changer, without performing a software disconnect.

⇒ *Connect the gun manually and cycle the controller power (OFF/ON).*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1518**      Miscount of encoder data jt\*.

The servo gun or tool changer harness is defective.

⇒ *Inspect the servo gun and tool changer harnesses; replace if necessary.*

---

**ERROR CODE -1521**      Mismatch ABS and INC encoder of jt-x.

Occurs when there is a large difference between the value in the serial encoder data and the data calculated 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.*

⇒ *Check for continuity in the machine harness and separation harness.*

⇒ *Replace 1FG board.*

---

**ERROR CODE -1524**      Encoder line error of jt-x.

Not used.

---

**ERROR CODE -1550**      Encoder initialize error jt-x.

The encoder is not able to fix the absolute position at control power on. The scanning of absolute data is begun immediately after a shift from the state of non-operation or back-up operation to normal encoder operation. BUSY=1 is output until the absolute position has been determined.

⇒ *In the case of a conveyor encoder, ensure power-up speed is below 300 RPM.*

⇒ *Check connections at encoder, 1FG board, jumpers, and switches on 1GB board. ⇒ Replace encoder, replace 1GB board, replace 1FG board.*

⇒ *Check for continuity in the machine harness and separation harness.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1553** Encoder response error jt-x.

**Spot welding and material handling**

Occurs when the encoder does not respond to data request signal from the 1GB board.

- ⇒ *Check connections at encoder, 1FG board, and separation harness.*
- ⇒ *Check connections, jumpers, and switches on 1GB board.*
- ⇒ *Replace encoder, replace 1GB board.*
- ⇒ *Check for continuity in the machine harness and separation harness.*
- ⇒ *Replace 1FG board.*
- ⇒ *Check for noise malfunction and shield integrity.*

**Servo welding gun**

The servo gun or tool changer harness is defective.

- ⇒ *Inspect the servo gun and tool changer harnesses; replace if necessary.*
- 

**ERROR CODE -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.*
  - ⇒ *Check for continuity in the machine harness and separation harness.*
  - ⇒ *Replace 1FG board.*
- 

**ERROR CODE -1555** Encoder data conversion error jt-x.

Occurs when the M-code data from the encoder has an error pattern.

- ⇒ *Replace encoder, replace 1GB board.*
  - ⇒ *Check connections at encoder, 1FG board, and separation harness.*
  - ⇒ *Check connections, jumpers and switches on 1GB board.*
  - ⇒ *Check for continuity in the machine harness and separation harness.*
  - ⇒ *Replace 1FG board.*
-

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1556** Encoder ABS-track error jt-x.

The absolute data and the incremental data in the encoder are different. ABSALM=1 is output from the encoder when this error occurs.

- ⇒ *Defective encoder.*
  - ⇒ *Defective 1GB or 1FG boards.*
  - ⇒ *Open or short-circuited encoder harness.*
- 

**ERROR CODE -1557** Encoder INC-pulse error jt-x.

Occurs when the encoder A and B incremental pulses are abnormal. The encoder INALM bit sent with the serial data=1.

- ⇒ *Replace encoder, replace 1GB board.*
  - ⇒ *Check connections at encoder, 1FG board, and separation harness.*
  - ⇒ *Check connections, jumpers, and switches on 1GB board.*
  - ⇒ *Check for continuity in the machine harness and separation harness.*
  - ⇒ *Replace 1FG board.*
- 

**ERROR CODE -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.

- ⇒ *Replace the encoder harness.*
- 

**ERROR CODE -1559** Power module error jt-x.

An error signal from the IPM module in the power block was detected. This error may be caused by:

1. Defect in the power block.
2. Defect in the servo board.
3. Short-circuit of the motor power line U, V, and W phase.
4. Defective cooling fan in the servo unit.
5. Defect in the harness between the servo board and the power blocks.  
(XGB-CN12↔XGC/XGD-CN8, XGB-CN13↔XGC/XGD-CN9)

- ⇒ *Replace the servo unit.*
  - ⇒ *Check for short circuit in the motor harness and replace if necessary.*
-

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1561** Current sensor disconnect [Servo (x)].

The current sensor cable between the servo board and the power blocks is not connected. The x indicates the servo board unit reporting the error. Check the following harnesses as indicated below.

Unit A - 1GB-CN9↔power block CN1, CN2 CN3.

Unit B - 1GB-CN10↔power block CN4, CN5, CN6.

Unit C - 1GB-CN11↔power block CN7.

This error may be caused by a defect in the 1GB board.

⇒ *Check connection and continuity in the current sensor cable between the 1GB board and the power block. (XGB-CN9↔XGB-CN10)*

⇒ *Replace the 1GB board if necessary.*

---

**ERROR CODE -1563** Servo unit 12V DC error [Servo (A)].

The +/-12V supply to the 1GB board is below specified limits of +10.75V and -10.4V. This error may be caused by a defect in the harness between the 1GB board (XGB-CN8), the mother board (XHZ-CN2) and the AVR, or a defect in the 1GB board. This error is reported by servo board unit A.

⇒ *Check the voltage and replace the 1GB board and the AVR as necessary.*

---

**ERROR CODE -1567** Regenerative resistor error [Servo (A)].

Current was sent to regenerative resistors for six or more consecutive seconds. Main causes include:

1. Abrupt direction changes at high speed.
2. Burned out resistors or defective power block.
3. Defect in the servo board (1GB board).
4. Defective connection or harness between the servo board and the power block (1GB-CN13 ↔ power block CN9).
5. Defect in the optional second regenerative resistance unit (when used).

This error is reported by servo board unit A.

⇒ *Avoid abrupt high speed direction changes. Change accuracy, speed, add points.*

⇒ *Install optional second regenerative resistance unit.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1568** Servo unit P-N low voltage [Servo (A)].

The voltage between P-N supplied to the power block is 60VDC or less at servo on.

Main causes include:

1. Defect of K1, K2, K3.
2. Defect in the relay board (1FY/1HY board).
3. Defect in the power sequence board (1FP/1HP board).
4. Defect in the power unit.
5. Defect in the servo board.
6. Defect in the motor power circuit (diode bridge, K3, etc).
7. Defect in circuit breaker F2 or F2 is in the OFF position.
8. Defect in the harness between the operation unit and the servo units or the connections.

⇒ *Check the motor power circuit and the equipment, replace as necessary.*

⇒ *Check the power sequence board, the relay board, and the servo unit, and replace as necessary.*

---

**ERROR CODE -1569** Servo unit P-N- high voltage [Servo (A)].

The voltage between P-N supplied to the power block exceeded 410 VDC. This error is caused by:

1. Defect in regenerative resistance control circuit, overheat failure of regenerative resistance unit, defect in the voltage monitoring circuit, or defect in the power block.
2. Defect of servo board (1GB board)
3. Defect in the regenerative resistance unit.
4. Defect in the servo board (1GB board).
5. High inertia loads due to tool weight or program data.

This error is reported by servo board unit A.

⇒ *Reteach program steps to eliminate sudden high speed direction changes (dynamic shock).*

⇒ *Replace the servo unit and the regenerative unit.*

⇒ *Add the second regenerative resistor unit.*

---

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1570** Regenerative resistor over-heat [Servo (A)] or controller hot.

The thermal switch for the power block regenerative resistor unit, or the optional regenerative resistor unit reached 140° C (284°F), or the heat sink thermal switch reached 90°C (194°F), or the controller cabinet thermal switch reached 70° C (158° F). This error is reported by servo board unit A.

- ⇒ *Defective cooling fans.*
  - ⇒ *Insufficient clearance for air circulation.*
  - ⇒ *Power block connector X5-SA disconnected, or defect in the wiring harness to the controller cabinet thermal switch, or to the optional regenerative resistor unit.*
  - ⇒ *High inertia loads.*
  - ⇒ *High ambient controller temperature.*
  - ⇒ *Defective power block.*
  - ⇒ *Defective regenerating resistor unit.*
  - ⇒ *Defective 1GB board.*
  - ⇒ *Defect in the following harnesses: 1GB-CN13↔power block CN9, 1GC-CN21↔optional regenerative resistor unit.*
-



---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1600**      Uncoincidence error jt-x.

During the execution of a program, all joints must reach total coincidence within a given accuracy and time (approx. 5 seconds) This error occurs if the difference between the designated value and current value exceeds the established accuracy. Typical causes are listed below.

Mechanical-

1. Damaged bearings.
2. Insufficient gear backlash.
3. Motor brake not released.
4. Arm movement restricted by harnesses or peripheral devices.

Electrical-

1. Defective servo board (1GB board).
2. Defective power block.
3. Disconnected motor power or brake lines.
4. Defective encoder or encoder harness.

Program data-

1. Wrist positions at the beginning and end of a path that utilize the same XYZ coordinate.

- ⇒ *Repair mechanical or electrical failures.*  
⇒ *Check interference from external devices.*  
⇒ *Modify taught wrist positions as required.*
- 

**ERROR CODE -1601**      Limit switch of joint-x is ON.

This error occurs when the software can specify the axis that has caused the over travel condition. If the joint number cannot be specified, ERROR CODE 1602 Limit switch is broken, is displayed. Main causes include:

1. Software limits are set incorrectly.
2. Axis was moved using the manual brake release and the limit switch was turned ON.
3. The limit switch was turned ON by overshoot from inertia.

- ⇒ *Check and exchange the harness and the limit switch.*  
⇒ *Set the software limits to an appropriate value.*  
⇒ *Change the installation angle of the axis restriction limit switch to an appropriate value.*  
⇒ *Replace the power sequence board if necessary.*
-

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1602**      Limit switch signal line is broken.

This error occurs when the software cannot specify the axis that has caused the over travel condition. If the joint number can be identified, the “ERROR CODE 1601 Limit switch of jt-x is ON.”

- ⇒ *Set the software limits to an appropriate value.*
  - ⇒ *Change the installation angle of the axis restriction limit switch to an appropriate value.*
  - ⇒ *Check harness and limit switch and replace as necessary.*
  - ⇒ *Check the power sequence board and replace if necessary.*
- 

**ERROR CODE -1610**      Torch is interfered.

The limit switch for torch interference detection has been tripped.

- ⇒ *Move the robot away from the interference to close the limit switch. Inspect the torch for damage.*
  - ⇒ *Determine the cause of the interference before continuing operation.*
- 

**ERROR CODE -1735**      GROUP is not primed.

The GROUP instruction is not programmed before a motion instruction, with external axis system.

- ⇒ *This error occurs if the program is interrupted after a GROUP instruction and before the motion commands pertaining to the GROUP instruction are complete. When restarting the program prime the program before the GROUP instruction.*
  - ⇒ *In check mode, if the GROUP instruction is used, start checking before the GROUP instruction.*
-

---

**ERROR CODES/TROUBLESHOOTING****ERROR CODE -1800** AC primary power off.

This error occurs when there is an instantaneous decrease in the primary power to the AVR for control power supply (AC130-145V or less for 20-30 msec from 0 phase). The main causes for this error are:

1. NFB for the control power supply was turned OFF.
2. AC200/220V supplied to AVR for the control power supply caused the instantaneous decrease.
3. Defective AVR for control power supply.
4. Defective NFB for control power supply or NFB tripped.
5. Defect in the primary power supply.
6. Defective power sequence board.
7. Defect in the relay board.

⇒ *It is normal for this error to occur when NFB for control power supply is turned OFF.*

⇒ *Check power supply circuit in AVR for the control power supply and NFB.*

⇒ *Confirm that primary power is supplied according to specifications.*

---

**ERROR CODE -1801** 24VDC power source is low.

24VDC to the power sequence board has dropped below 21.6V. Main causes include the following abnormalities:

1. Defect in the AVR for control power supply.
2. Defect in the power sequence board.
3. Defect in the relay board.
4. Short circuit in the motor power on circuitry, the EMERGENCY STOP switch and axis restriction limit switch circuits.
5. Short in machine valve and sensor circuits.

⇒ *Check the power supply, machine valve, and the sensor circuits for short circuits.*

⇒ *Check the AVR, the power sequence board, and the relay board and replace as necessary.*

---

---

## ERROR CODES/TROUBLESHOOTING

**ERROR CODE -1802**      Primary power source is high.

Voltage level of the AVR for the control power supply was too high (AC2667-277V or more for 1-2 sec.). The main causes of this error include a defect in the AVR, the power sequence board, the relay board or the power supply circuit.

- ⇒ *Verify the power supply to the controller are within ratings.*
  - ⇒ *Check power supply circuit to the AVR.*
  - ⇒ *Confirm the supply power is within specifications.*
  - ⇒ *Confirm transformer tap settings are correct.*
- 

**ERROR CODE -1803**      Primary power source is low.

Voltage level of the AVR for the control power supply was too low (AC150-158V or less for 1-2 sec.). The main causes of this error include an instantaneous drop in the power supply; a defect in the AVR or NFB, the power sequence board, the relay board or the power supply circuit.

- ⇒ *Verify the power supply to the controller is within ratings.*
  - ⇒ *Check the AVR and NFB.*
  - ⇒ *Confirm supply power is within specifications.*
  - ⇒ *Confirm transformer tap settings are correct.*
-

---

**ERROR CODES/TROUBLESHOOTING**

**ERROR CODE -1804**      5 VDC or  $\pm 12$  VDC is abnormal.

5 VDC or +/-12 VDC supplied to the 1GA board is out of specifications.

+5 VDC: less than +4.85 VDC - more than +5.45 VDC

+12 VDC: +10.75 VDC or less

-12 VDC: -10.4 VDC or more

Main causes include:

1. Defective DC power supply.
2. Defective 1GA board.
3. Power supply contacts in the mother board are bad.
4. Defective MFP, small teach pendant, or harness.
5. Defective servo board or power block.
6. Defective 1FG board.
7. Short in the harness between the servo unit and the 1FG board.
8. Short in the optional circuit boards, i.e., vision, etc.
9. Insufficient DC power supply capacity to support optional boards.

⇒ *Replace the AVR and each board for the control power supply.*

⇒ *Check for short circuits in the MFP, the operation panel, and separation harness and replace as necessary.*

---

**ERROR CODE -1805**      Memory is locked because of AC\_FAIL.

The memory was accessed during the controller shut down due to a power supply abnormality (ACFAIL).

⇒ *Cycle the controller power OFF and ON.*

---

## ERROR CODES/TROUBLESHOOTING

### 10.3 TROUBLESHOOTING FLOWCHARTS

The following flowcharts are arranged in the numerical order of the error code. Refer also to the corresponding error code information in section 10.2 during troubleshooting procedures.

**ERROR CODES/TROUBLESHOOTING**

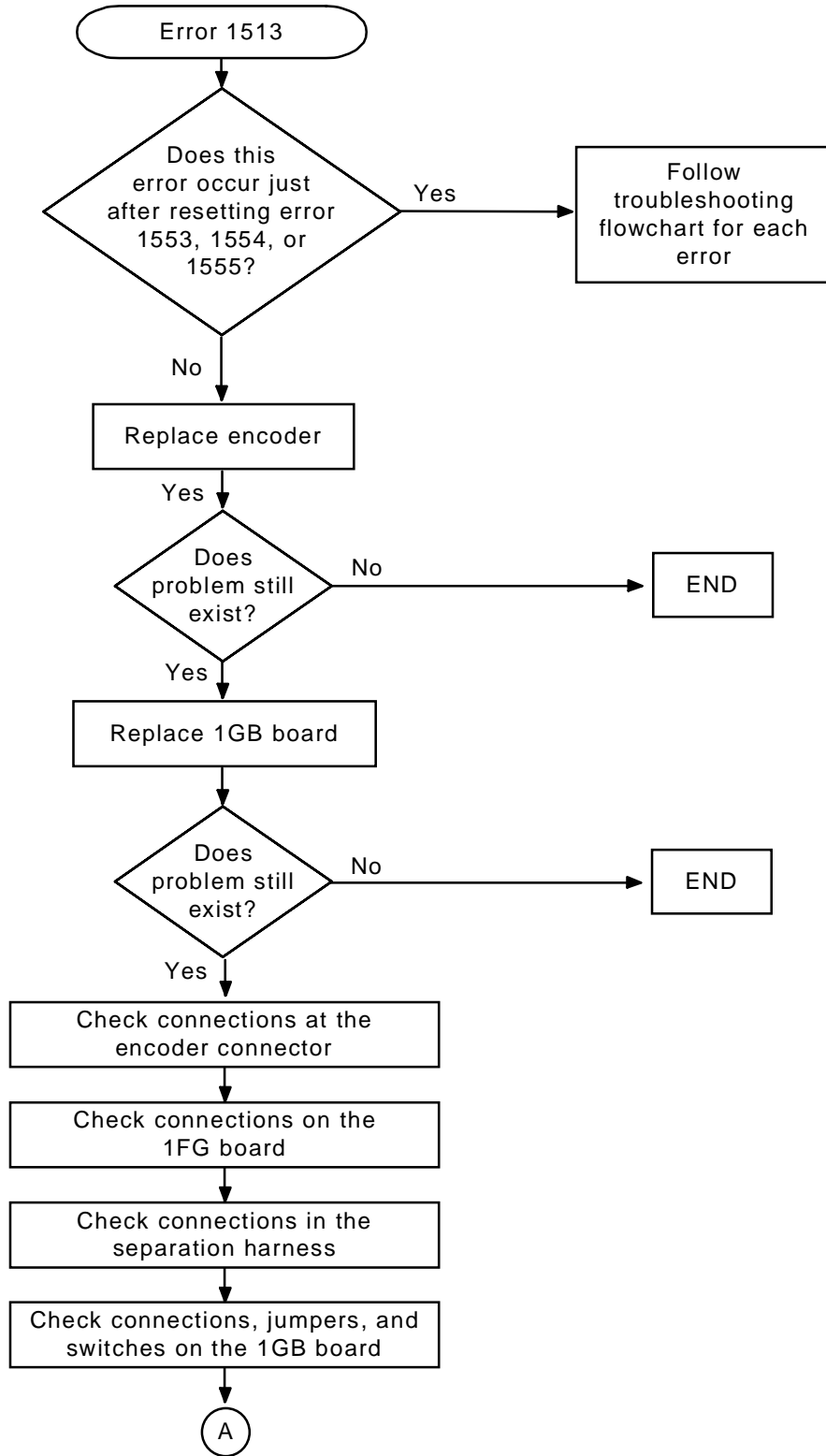


Figure 10-3 Error 1513 Flowchart (1)

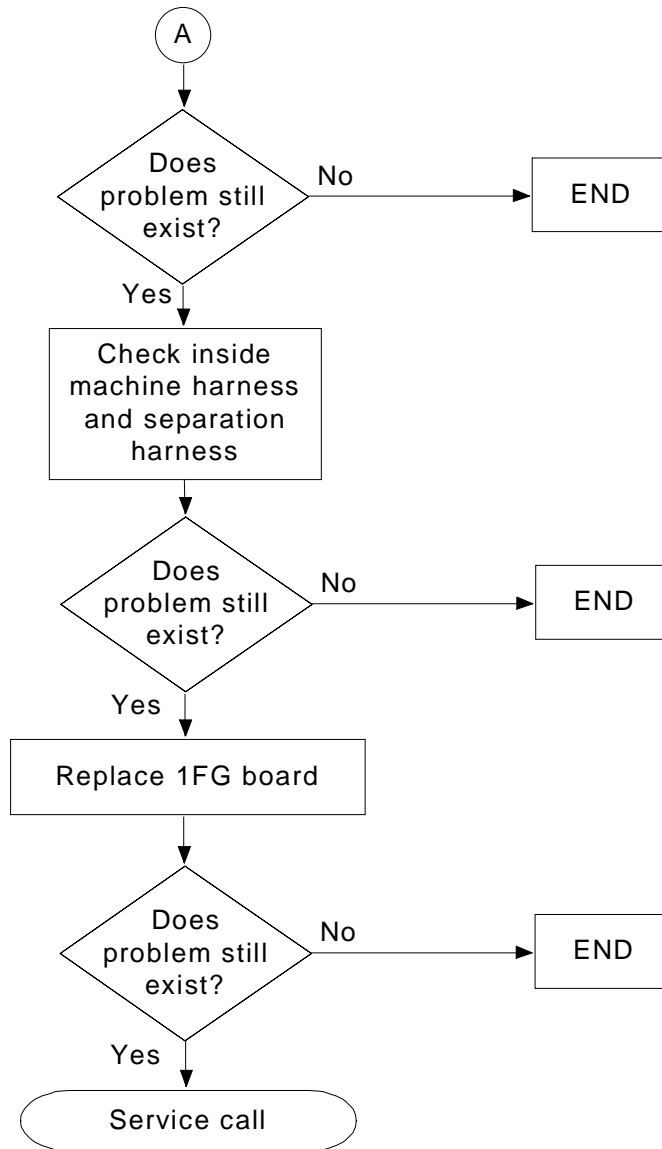
**ERROR CODES/TROUBLESHOOTING**

Figure 10-4 Error 1513 Flowchart (2)



## ERROR CODES/TROUBLESHOOTING

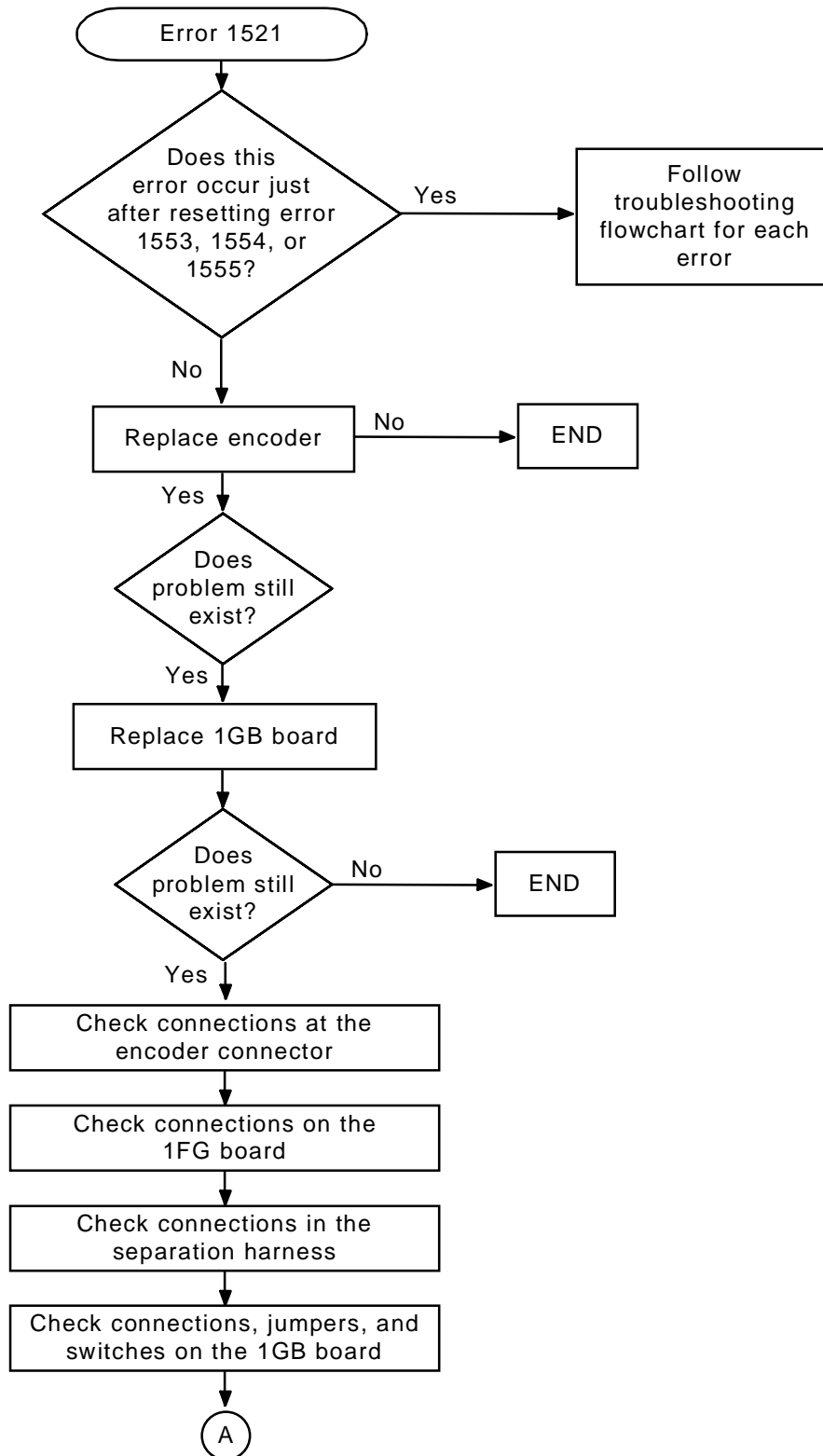


Figure 10-5 Error 1521 Flowchart (1)

## ERROR CODES/TROUBLESHOOTING

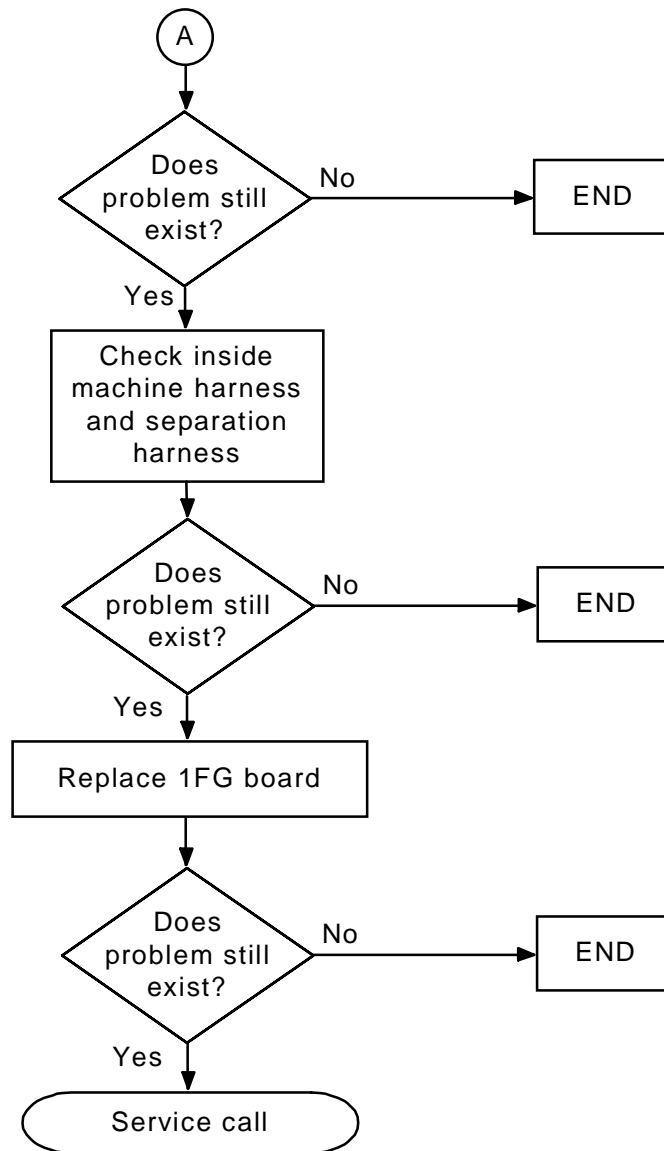


Figure 10-6 Error 1521 Flowchart (2)

**ERROR CODES/TROUBLESHOOTING**

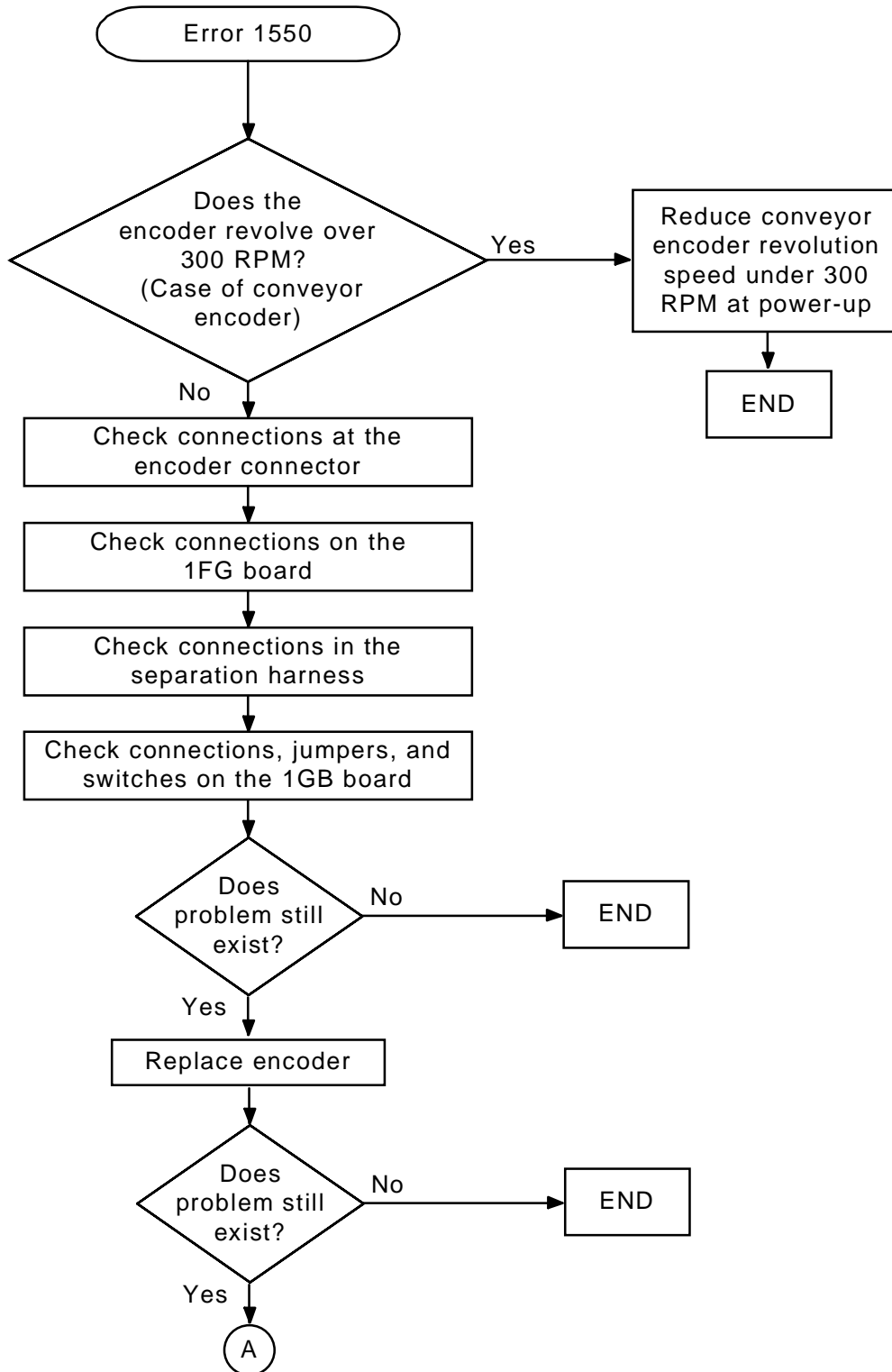


Figure 10-7 Error 1550 Flowchart (1)

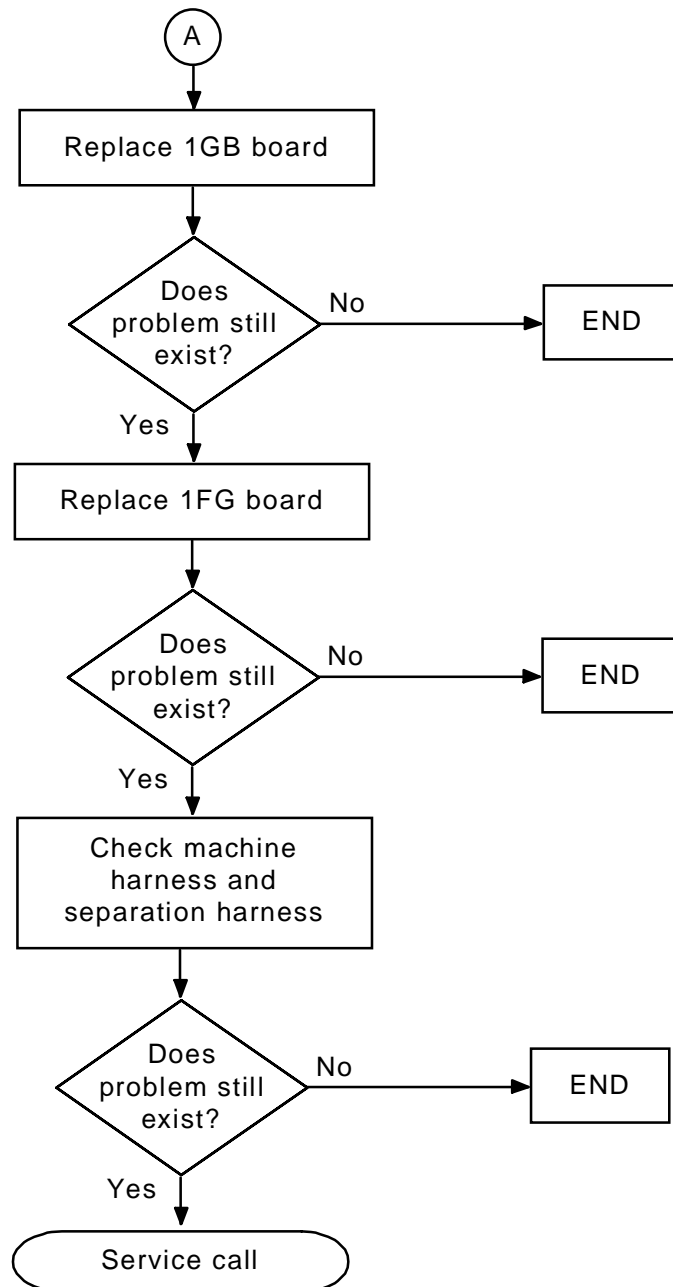
**ERROR CODES/TROUBLESHOOTING**

Figure 10-8 Error 1550 Flowchart (2)

**ERROR CODES/TROUBLESHOOTING**

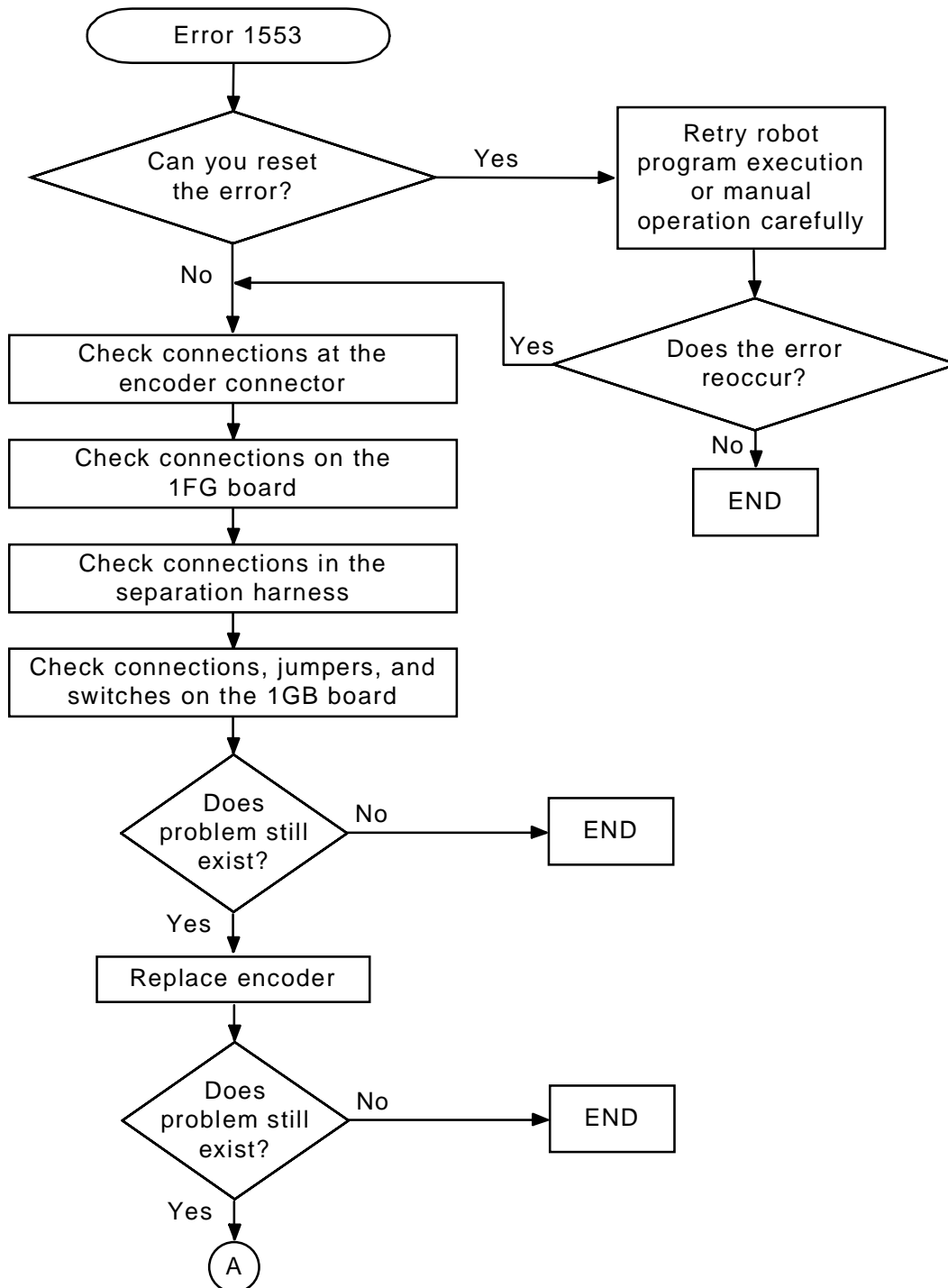


Figure 10-9 Error 1553 Flowchart (1)

**ERROR CODES/TROUBLESHOOTING**

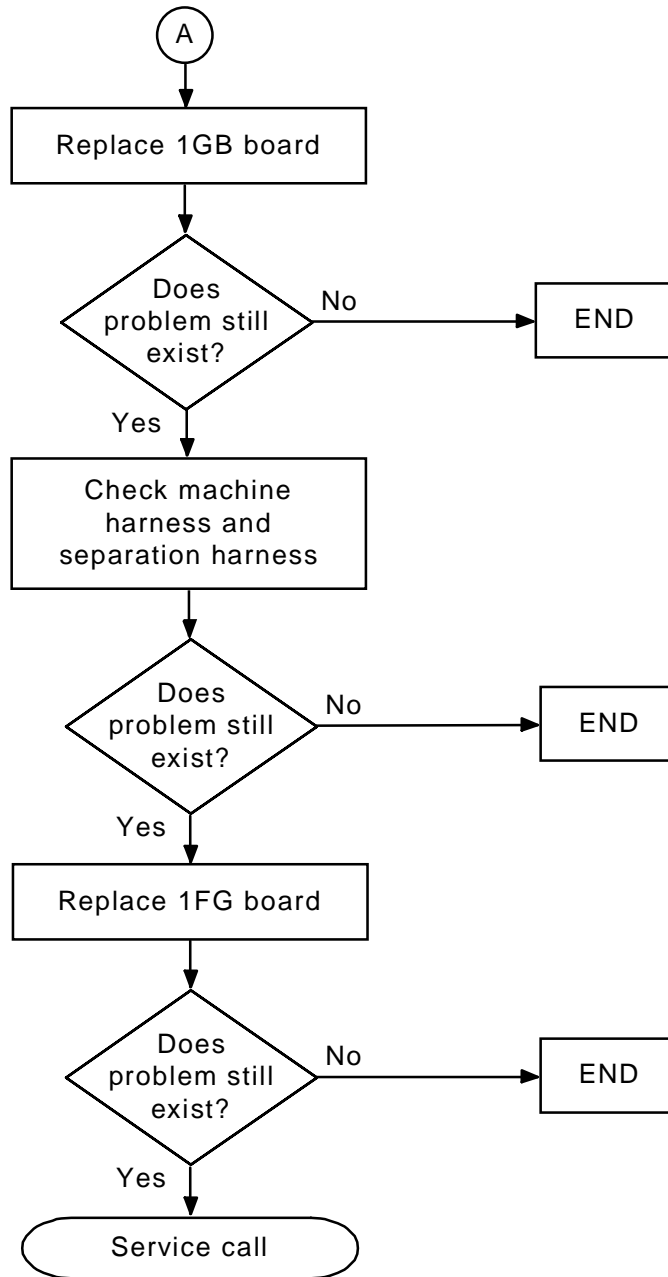


Figure 10-10 Error 1553 Flowchart (2)

### ERROR CODES/TROUBLESHOOTING

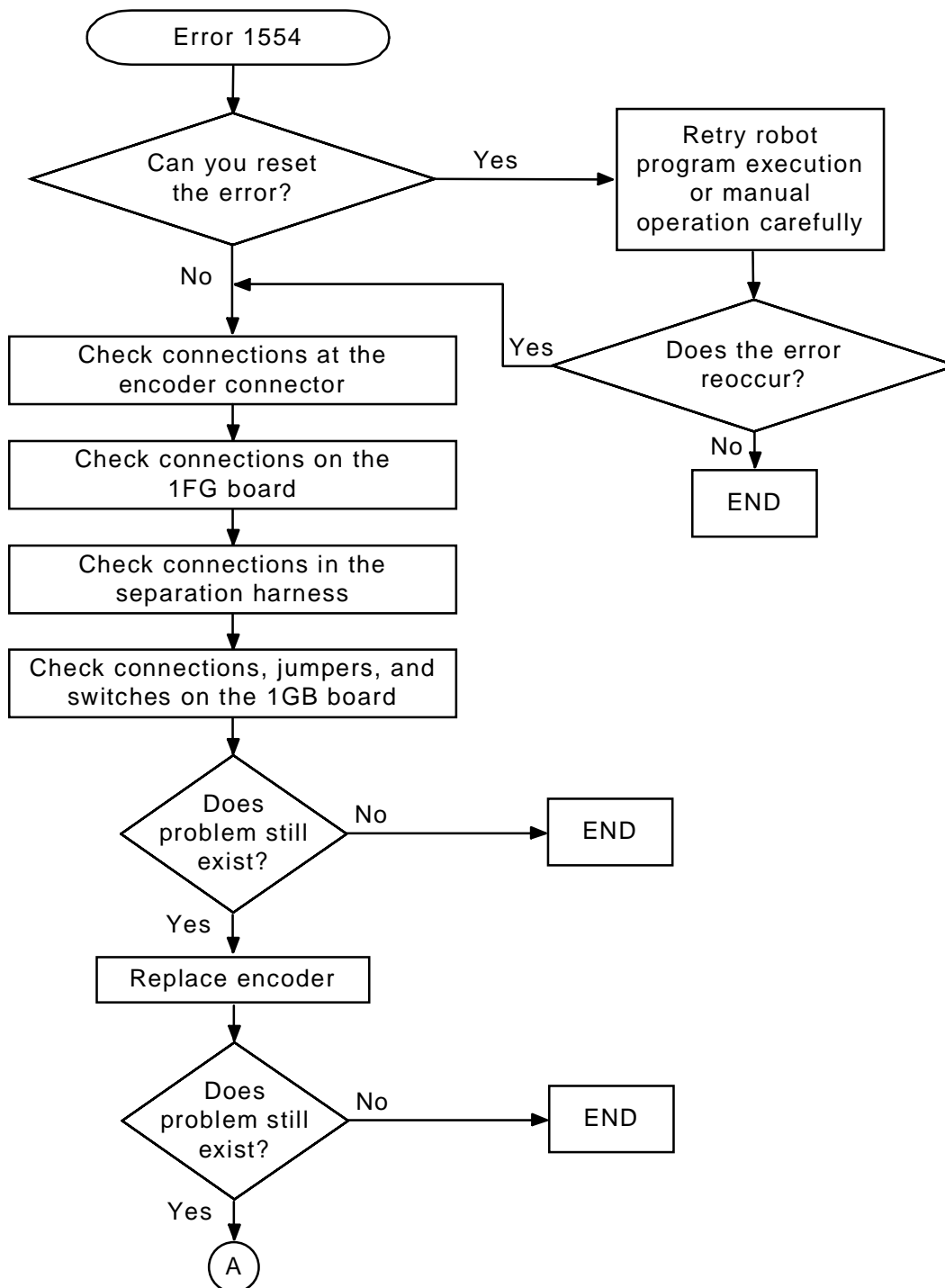


Figure 10-11 Error 1554 Flowchart (1)

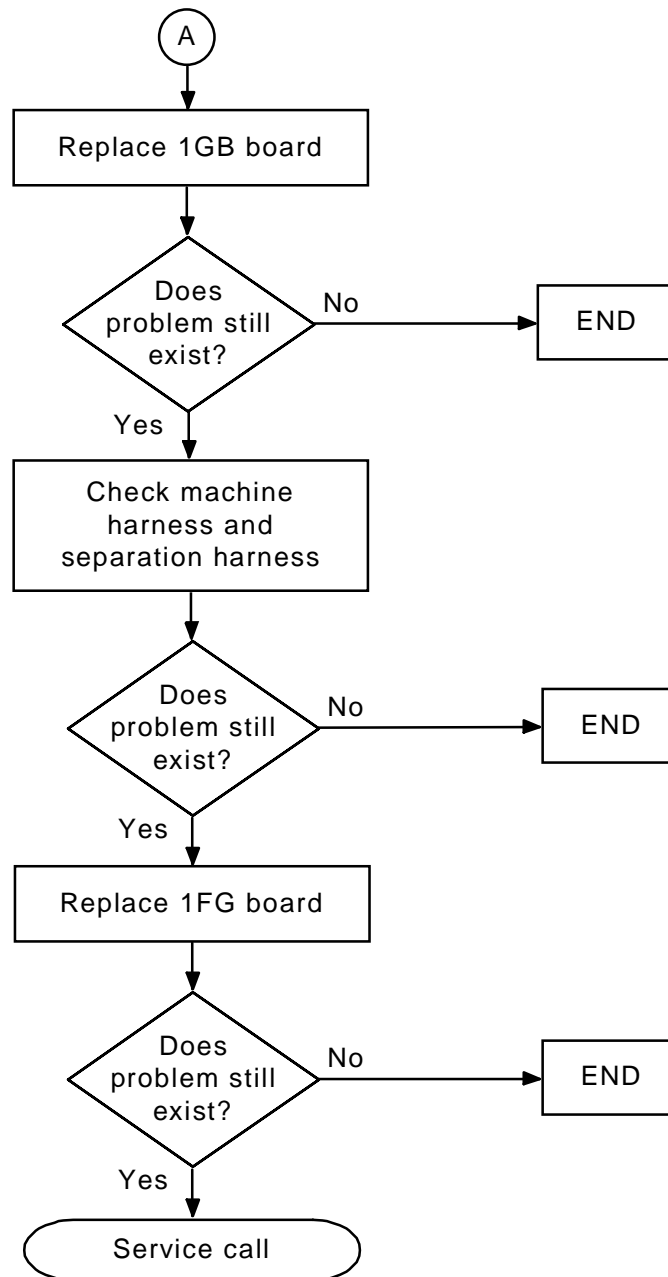
**ERROR CODES/TROUBLESHOOTING**

Figure 10-12 Error 1554 Flowchart (2)



### ERROR CODES/TROUBLESHOOTING

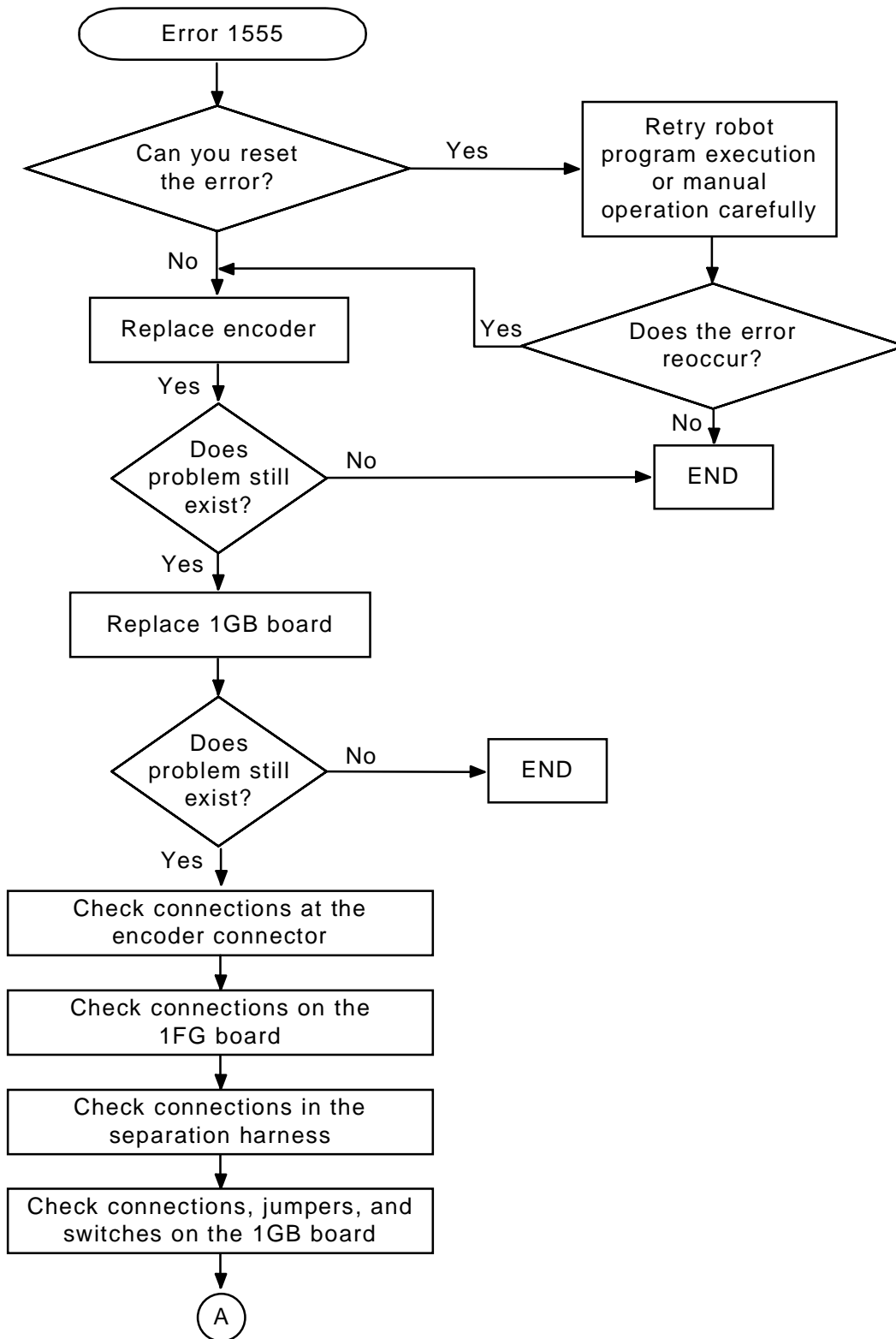


Figure 10-13 Error 1555 Flowchart (1)

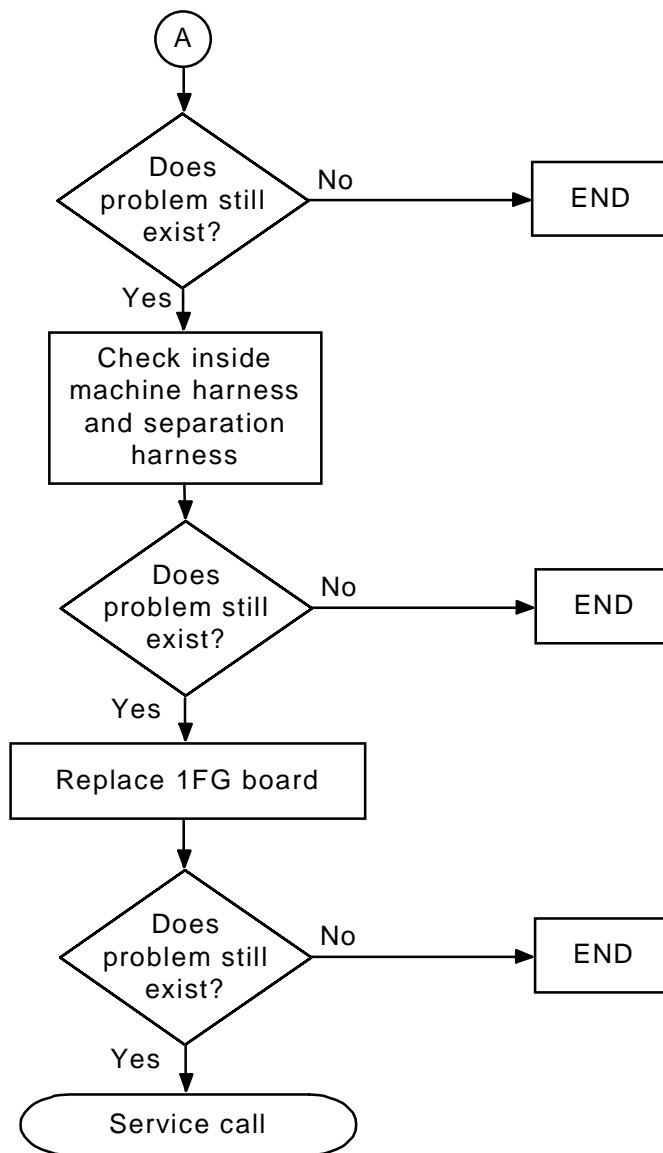
**ERROR CODES/TROUBLESHOOTING**

Figure 10-14 Error 1555 Flowchart (2)

**ERROR CODES/TROUBLESHOOTING**

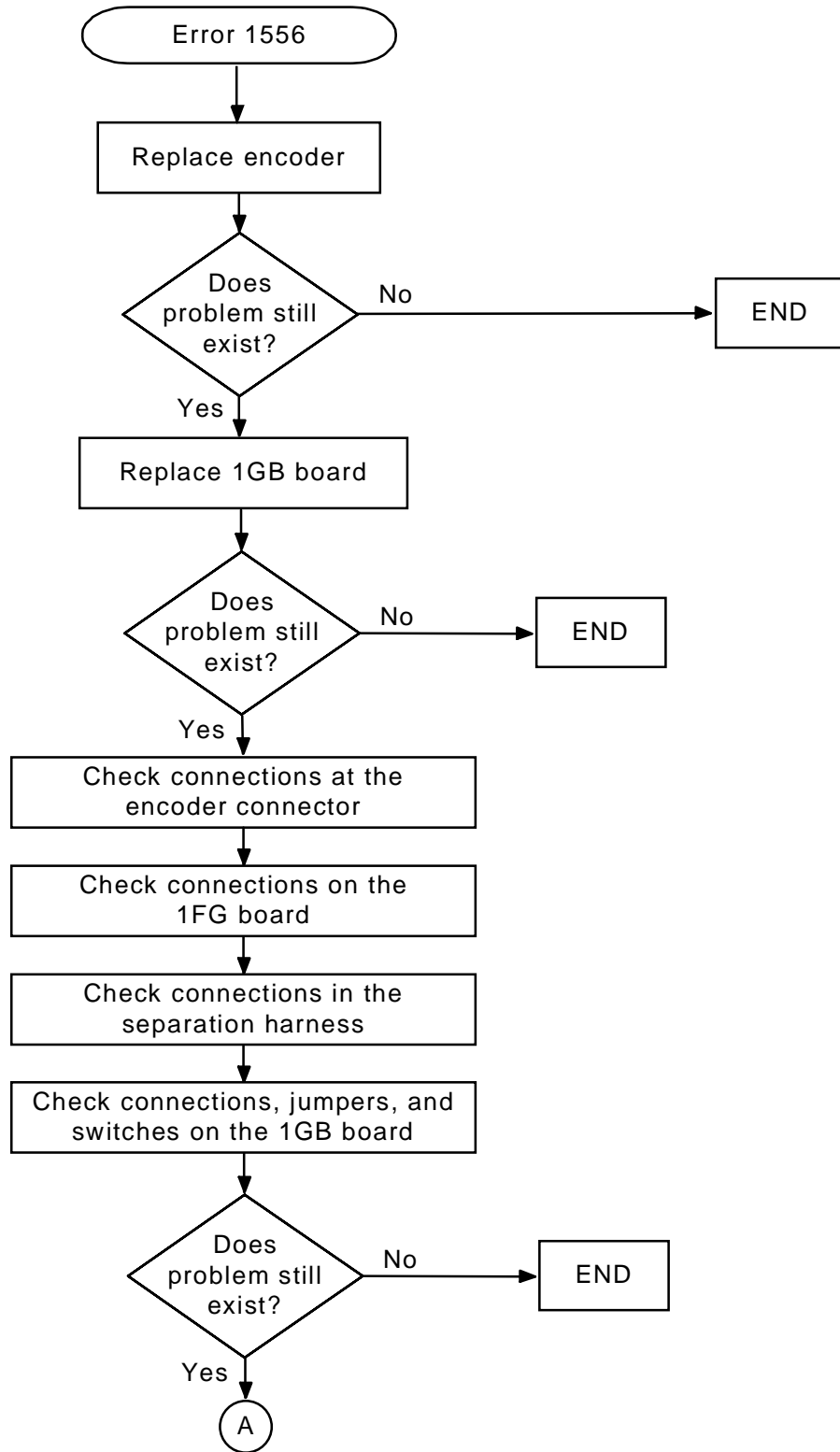


Figure 10-15 Error 1556 Flowchart (1)

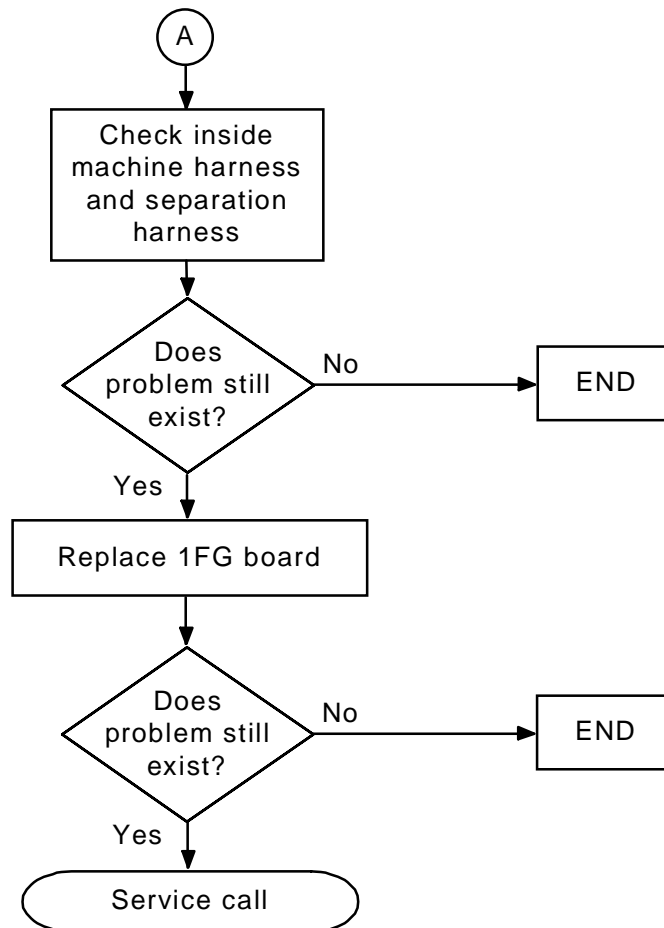
**ERROR CODES/TROUBLESHOOTING**

Figure 10-16 Error 1556 Flowchart (2)

## ERROR CODES/TROUBLESHOOTING

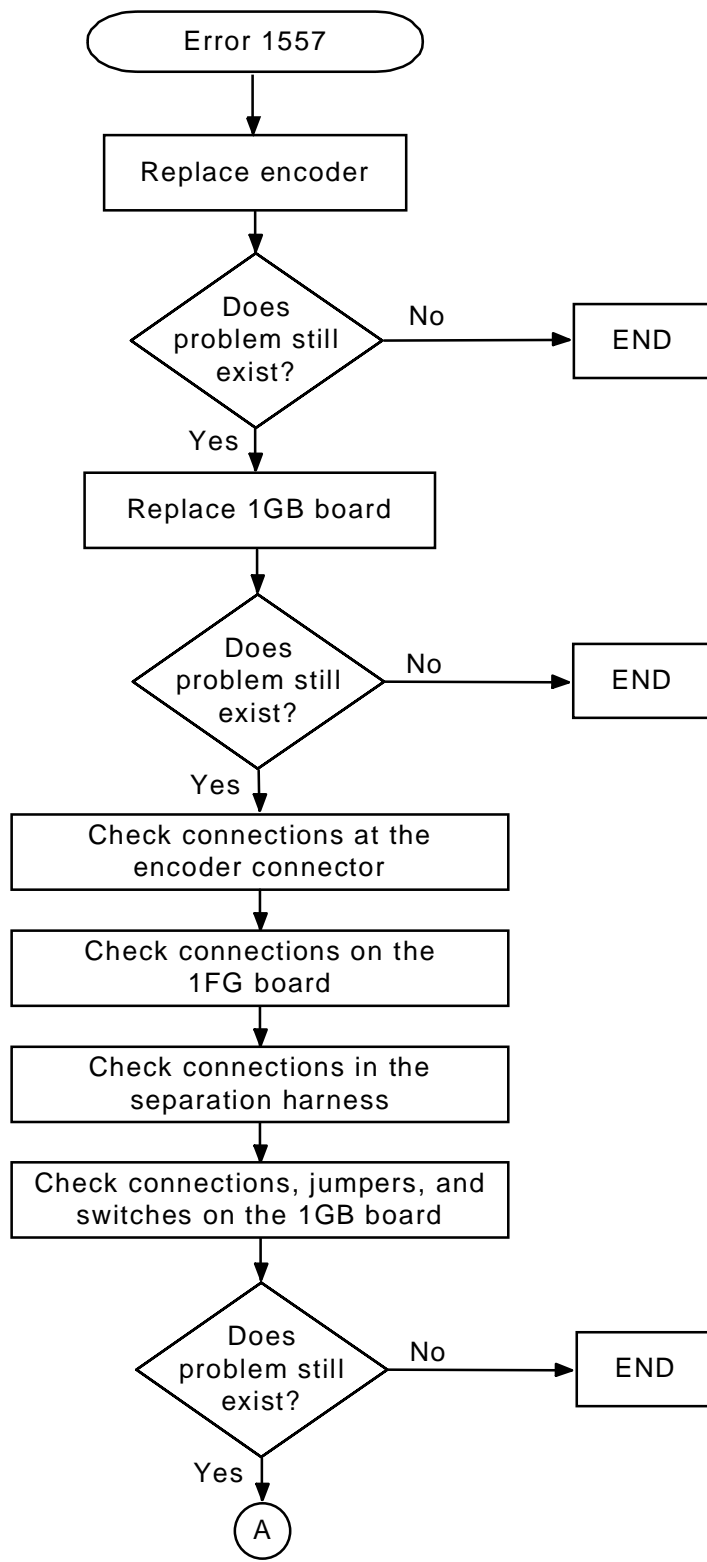


Figure 10-17 Error 1557 Flowchart (1)

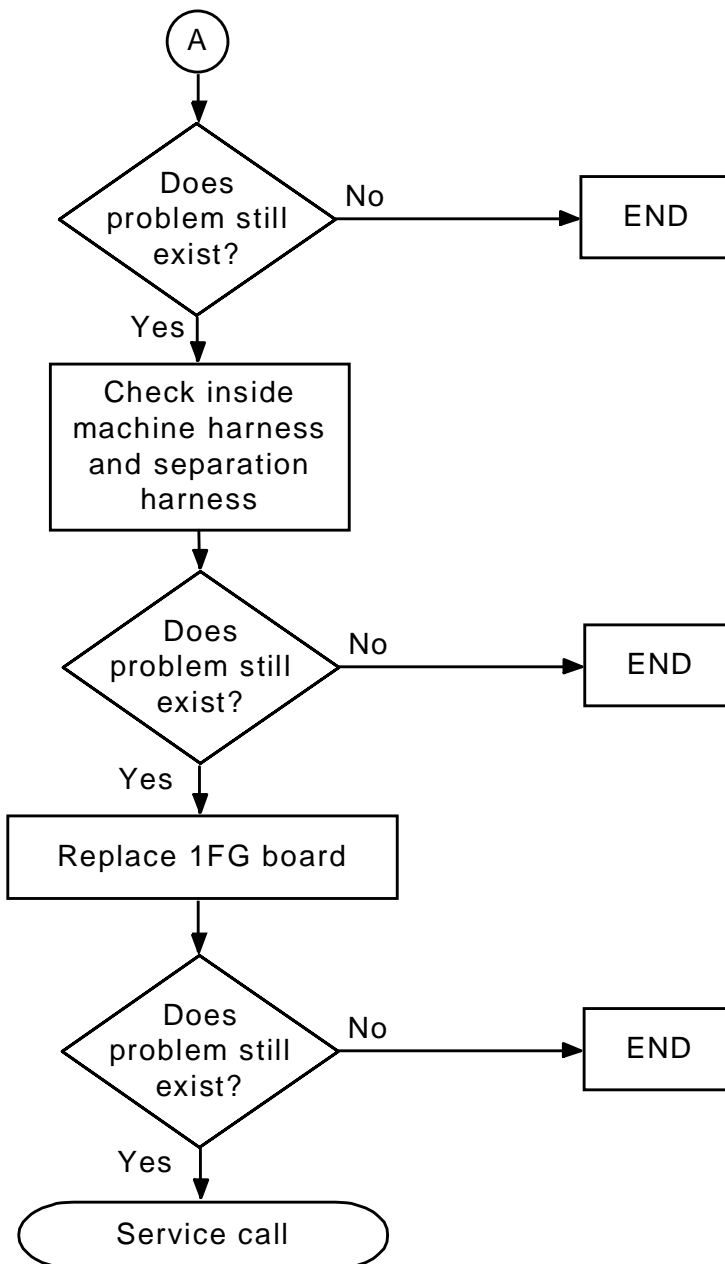
**ERROR CODES/TROUBLESHOOTING**

Figure 10-18 Error 1557 Flowchart (2)

---

**APPENDIX**

<b>A.0</b>	<b>APPENDIX</b> .....	A-2
A.1	Remote I/O and ControlNet .....	A-2
A.1.1	SLOGIC .....	A-2
A.1.1.1	Slogic Program .....	A-2
A.1.1.2	SCNT Command .....	A-6
A.1.1.3	SFLK Command .....	A-6
A.1.1.4	SFLP Command .....	A-6
A.1.1.5	STIM Command .....	A-7
A.1.1.6	RI/O Monitor, Aux 180 .....	A-7
A.1.1.7	RI/O PLC (NAC) Setting, Aux 181 .....	A-12
A.1.2	ControlNet .....	A-13
A.2	Motion Control and Location Definition .....	A-16
A.2.1	C1MOVE and C2MOVE Commands .....	A-16
A.2.2	FMOVE command .....	A-18
A.2.3	MVWAIT Command .....	A-21
A.2.4	BSPEED Command .....	A-22
A.2.5	TRHERE Command .....	A-22
A.3	System Parameters and Switches .....	A-23
A.3.1	NCHON and NCHOFF Commands .....	A-23
A.3.2	WEIGHT Command .....	A-23
A.3.3	MC Command .....	A-23
A.3.4	AUTOSTART.PC Switches .....	A-23
A.3.5	ERRSTART.PC Switch .....	A-23
A.3.6	AFTER.WAIT.TMR Switch .....	A-24
A.4	Function Commands .....	A-24
A.4.1	DEXT Function .....	A-24
A.4.2	PRIORITY Function .....	A-24
A.4.3	PCSCAN Command .....	A-24
A.5	Process Control Commands .....	A-25
A.5.1	PCSTATUS Command .....	A-25
A.5.2	PC Program Control .....	A-26
A.6	American Standard Code for Information Interchange Set .....	A-28
A.7	KeyWords List .....	A-31

---

## APPENDIX

### A.0 APPENDIX

#### A.1 REMOTE I/O AND CONTROLNET

The C-series controller provides the capability to control remote input/output functions using SLOGIC programming with two configurations of the 1FS circuit board. These configurations are the remote input/output (R I/O) and RS-485, and ControlNet. The C controller 1FS board performs functions that are similar to the AD controller 1AS board, but does not provide parallel I/O (PI/O) or analog outputs. These functions are performed by the 1GW board.

##### A.1.1 SLOGIC

Relay circuitry and wiring for the controller interface with peripheral devices is reduced by utilizing the SLOGIC software programming function. SLOGIC programming is similar in theory to the ladder logic programming used with PLCs. Table A-1 shows a comparison of SLOGIC program instructions and ladder logic. Making signal assignments for remote inputs and outputs using SLOGIC is faster and easier than making changes to discrete hard-wired systems. With discrete hard-wired systems, signal assignment changes are made by physically changing the wiring connections.

Using SLOGIC programming, signal assignment changes are made by simply editing the program. The SOUT command is used to assign a signal expression to a 1FS board output signal number.

**SOUT signal\_number = signal\_expression.**

Example:               SOUT 1001 = 1 AND 2

SLOGIC output signal 1001 (1GA board input signal WX1) is turned on when 1FS board inputs 1 and 2 (1GA board output signals 1 and 2) are on.

Figure A-1 shows the signal numbers available for communication with the 1FS board.

SLOGIC program instructions are used to specify outputs, timers, and counters. SLOGIC program instructions are edited on the 1GA board (main CPU) via the multi function panel and then transferred to the 1FS (R I/O) board, which executes the SLOGIC program.

##### A.1.1.1 SLOGIC PROGRAM

Naming an SLOGIC program is different from other AS Language programs; the only name that is assigned to an SLOGIC program is spg.

To edit an SLOGIC program use the procedure on the next page.



---

**APPENDIX**

From the status screen press the MENU key and select the keyboard screen.

Press the ENTER key on the keyboard screen to display the \$ prompt. (↵= ENTER)

**\$\$\$ ↵** SLOGIC stop, the SLOGIC program must be stopped to download the program after editing.

**\$\$SU ↵** SLOGIC upload, upload the Slogic program from the 1FS board to the 1GA board to edit the program SPG.

**\$ED SPG ↵** Enters the AS Language editor mode and selects the program.

1 SOUT 301 = 101 ↵

1?

2 SOUT 302 = 101 and 102 ↵

2?

3 SOUT 303 = 101 or 102 ↵

3?

4 SOUT 304 = (-101 and -102) or -103 ↵

4?

5 SOUT 305 = (101 or 103) and (102 or -104) ↵

5?

6 STIM 2601 = 101,5.0 ↵

6?

7 SOUT 306 = 2601 ↵

7?

8 SOUT 2701 = 101,110,102,10 ↵

8?

9 SOUT 307 = 2701 ↵

9?E

EXIT the AS Language program editor mode

**\$ SD ↵** SLOGIC download, download the SLOGIC program from the 1GA board to the 1FS board to execute the program.

Enter Password  
**4989 ↵** Enter the password 4989 (the same password is used for all Kawasaki controllers).

**\$\$\$SR** SLOGIC run, the program must be running to process signals.

Edit commands used for SPG program editing:

**S STEP ↵** Selects the step number to edit.

**I ↵** Inserts line(s) before the current step.

**D ↵** Deletes the current program step.

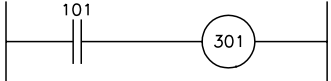
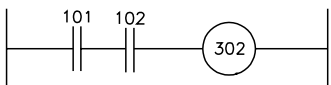
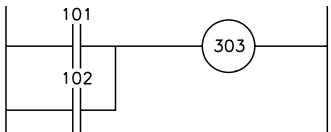
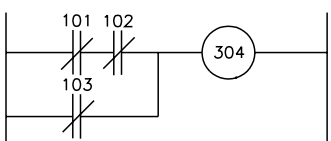
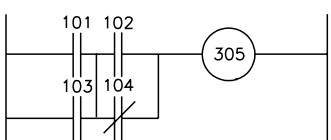
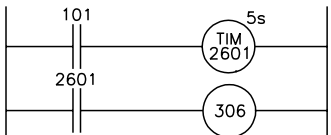
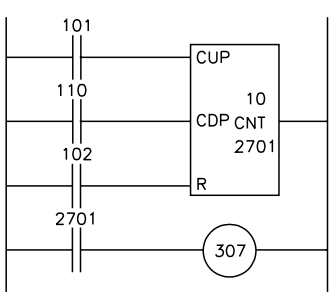
**O ↵** Places the cursor on the current step for editing (overwrite).

**E ↵** Exits the AS Language program editor mode.

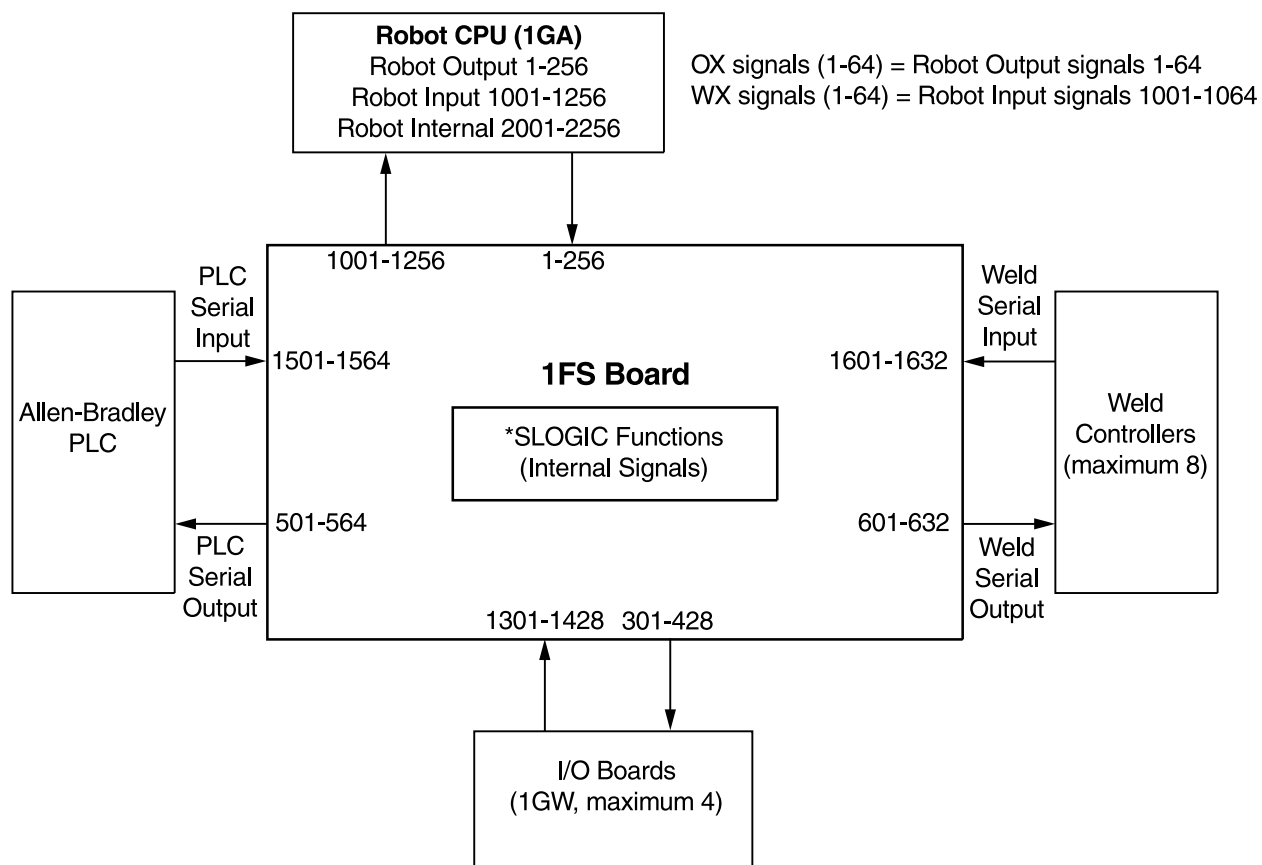
**NOTE:** Output may be used only once.

APPENDIX

Table A-1 Slogic

	<p>SOUT 301 = 101</p>
	<p>SOUT 302 = 101 AND 102</p>
	<p>SOUT 303 = 101 OR 102</p>
	<p>SOUT 304 = (-101 AND -102) OR -103</p>
	<p>SOUT 305 = (101 OR 103) AND (102 OR -104)</p>
	<p>STIM 2601 = 101, 5.0 SOUT 306 = 2601</p>
	<p>SCNT 2701 = 101, 110, 102, 10 SOUT 307 = 2701</p>

## APPENDIX



**\*SLOGIC Functions (Internal Signals)**

Relays: Non-Retentive	2301-2428
Relays: Retentive	2501-2516
Relays: Timers	2601-2616
Relays: Counters	2701-2716
SLOGIC Status Signals	2801-2816
W/C #1 Not Connected	2801
W/C #2 Communication Error	2802
NAC Communication Error	2803
Battery Error	2804
W/C #2 Not Connected	2805
W/C #2 Communication Error	2806
Spare	2807-2816
Message Display	2901-2964
Signal for Last Weld Current	2965

SLOGIC messages must be named \$w2901-\$w2964.

Figure A-1 1FS Board Signal Numbers

---

## APPENDIX

### A.1.1.2 SCNT COMMAND

**SCNT counter\_signal\_number =  
count\_up\_signal,count\_down\_signal,count\_reset\_signal,count\_value.**

The SCNT command is used to count signal pulses. When the count\_value is reached the specified counter signal number turns ON.

The counter signal number (2701 - 2716) to the left of the = sign is the output signal from the 1FS board. When the counter reaches the count value, the specified counter signal number is turned on. The count up signal to the right of the = sign is the input signal to the 1FS board that records an increase to the count. The count down signal to the right of the = sign is the input signal to the 1FS board that records a decrease to the count. The counter reset signal is the output signal to the 1FS board to reset the counter value to 0. The count value is the number of times the count signal must change states before the counter signal number is generated as an output from the 1FS board. The following is an example of the SCNT command:

```
SCNT 2701 = 101,110,102,10  
SOUT 307 = 2701
```

In this example, assuming the count down signal 110 does not change state, when the count up signal 101 changes state 10 (count\_value) times, signal 2701 (counter\_signal\_number) is output from the 1FS board. When signal 102 (counter\_reset\_signal) turns on the counter is reset to 0. In the second line of code, the 1FS board outputs signal 307 (possibly an indicator light) when the counter reaches the count value.

### A.1.1.3 SFLK COMMAND

**SFLK flicker\_signal\_number = time**

The SFLK command is used to turn a signal ON and OFF at the frequency specified by the time argument.

When the signal number specified with the SFLK command is use in the Slogic program, the signal turns on and off at the frequency designated by the time argument.

### A.1.1.4 SFLP COMMAND

**SFLP output\_signal\_number = set\_signal,reset\_signal**

The SFLP command is used to control an output signal using a set and a reset signal.

## APPENDIX

When the SLOGIC program is executing and the set signal specified with the SFLP command is on, the output signal number turns on. When the reset signal specified with the SFLP command is on, the output signal number turns off. When the set signal and the reset signal specified with the SFLP command are both on, the output signal number is turned off.

### A.1.1.5 STIM COMMAND

**STIM timer\_signal\_number = input\_signal,time**

The STIM command is used to turn ON an output signal after an input signal is ON for a specified time.

The timer signal number to the left of the = sign is turned on when the timer reaches the time value on the right. The input signal on the right of the = sign is the input signal to the 1FS board to start the timer. The time value is the length of time in seconds. When the input signal turns on timing starts. When the specified time elapses, the timer signal number turns on. The following is an example of the STIM command:

```
STIM 2601 = 36,2.4
SOUT 360 = 2601
```

In this example, 2.4 seconds after input signal 36 turns on, signal 2601 (timer signal number) turns on. In the second line of code, the 1FS board outputs signal 360 (possibly an indicator light) when the timer has reached the time value.

### A.1.1.6 R/I/O MONITOR, AUX 180

This function allows the user to view various functions regarding the remote I/O functions (Figure A-2).

AUX.180 R/I/O MONITOR			
1 SIGNAL STATUS			
2 TIMER AND COUNTER STATUS			
3 SLOGIC MONITOR			
4 LAST WELD DATA			
5 SLOGIC STATUS			
FUNCTION NUMBER:			
F1	F2	F3	F4

Figure A-2 R/I/O Monitor

### APPENDIX

The SIGNAL STATUS screens shown in figure A-3 allows the user to view the status of all remote I/O and robot signals. Figure A-3 shows two of five signal status screens. Use the NEXT PG buttons to scroll to the remaining screens.

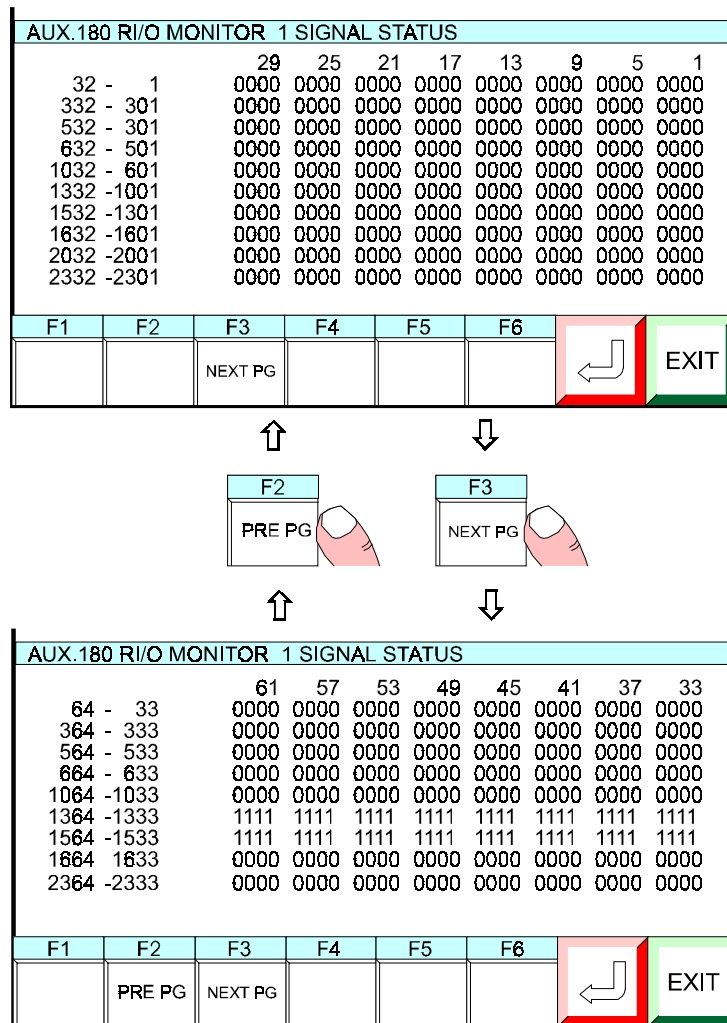


Figure A-3 RI/O Signal Status Screens

## APPENDIX

The TIMER AND COUNTER STATUS function allows the user to view the timer or counter values (Figure A-4). The elapsed time or number of counts is also displayed. The first screen shows the status of timers and the second screen shows the status of counters. Navigate between the screens using the PRE PG and NEXT PG buttons.

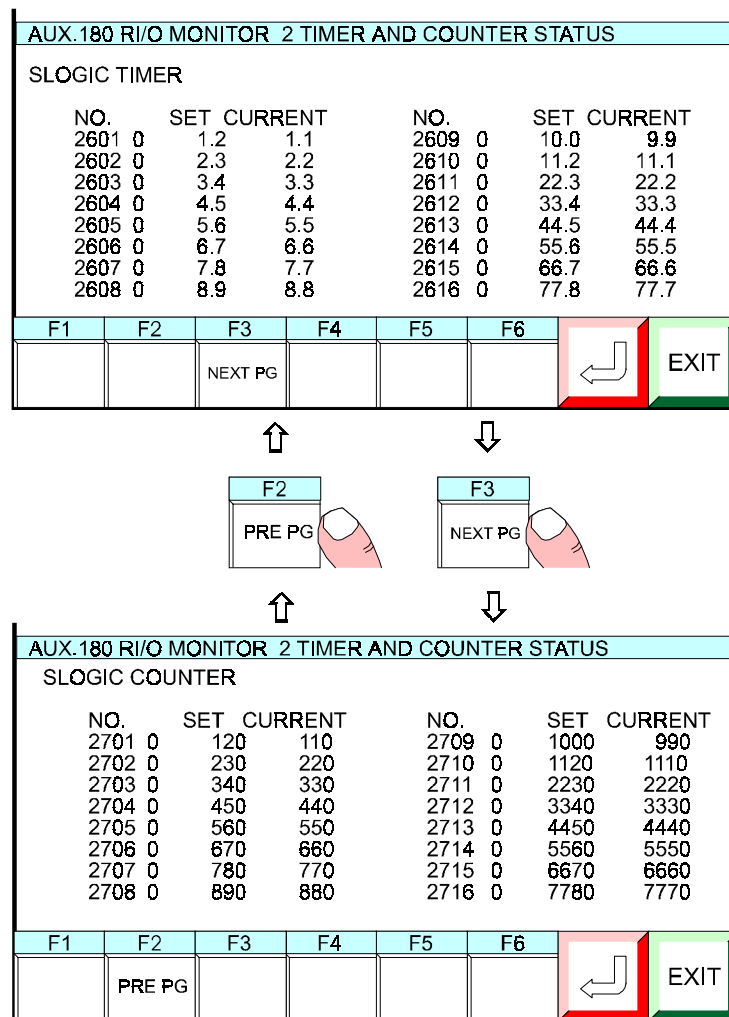


Figure A-4 Timer and Counter Status Screens

## APPENDIX

The SLOGIC MONITOR screen (Figure A-5) allows the user to monitor output signals which are used by the Slogic program. It also displays the status of the signal. When this function is accessed, key in the desired output signal at the prompt and press the ENTER key on the multi function panel.

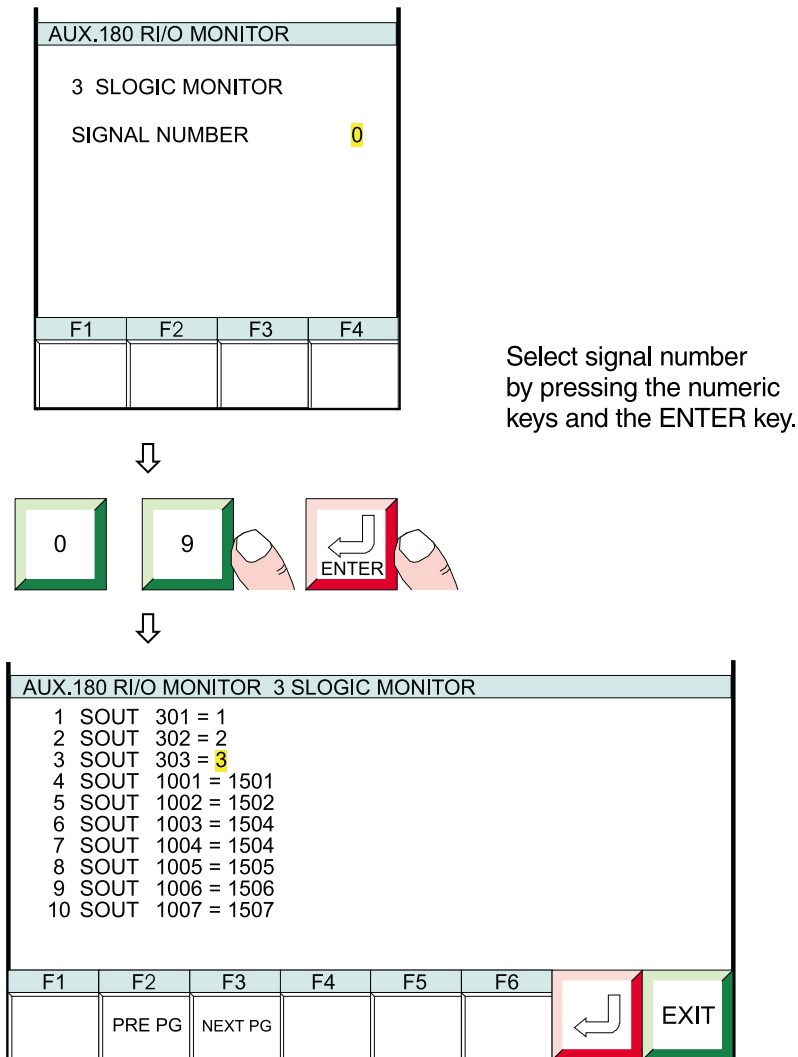


Figure A-5 Slogic Monitor Screen



## APPENDIX

The LAST WELD DATA screen displays the weld sequence parameters. Use the NEXT PG and PRE PG buttons to scroll between the three available screens. Figure A-6 shows LAST WELD DATA screens.

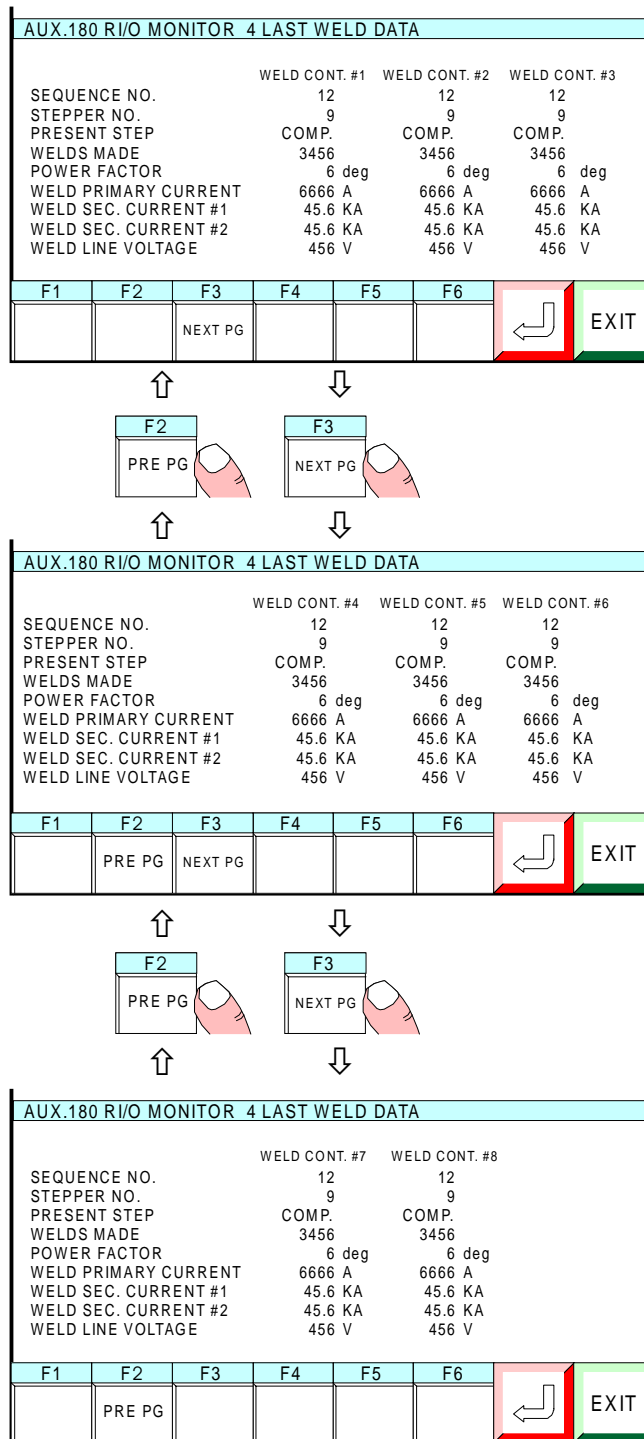


Figure A-6 Last Weld Data Screens

## APPENDIX

The SLOGIC STATUS screen allows the user to view the status of the Slogic program (Figure A-7). It displays the status of Slogic programs as active (RUN) or inactive (STOP). It also indicates how many steps are used in the Slogic program, the number of bytes free in the memory, and the date and time the Slogic program was downloaded.

AUX.180 RI/O MONITOR			
SLOGIC STATUS			
SLOGIC STATUS	RUN/	STOP	
SLOGIC STEPS	0		
SLOGIC MEMORY	0 BYTES	FREE	
DOWNLOADED DATA	98-04-09	16:50:31	
F1	F2	F3	F4

Figure A-7 Slogic Status Screen

### A.1.1.7 RI/O PLC (NAC) SETTING, AUX 181

AUX function 181 allows the user to view or change the PLC Node Adapter Chip (NAC) settings (Figure A-8).

AUX.181 RI/O PLC(NAC) SETTING			
*NAC CONTROL			
USED		YES/	NO
BAUD RATE(bps)	57600/	115200/	230400
RACK ADDRESS	0		
LAST RACK		YES/	NO
STARTING QUARTER	1		
RACK SIZE	1/4/	1/2/	3/4/
SIGNAL BITS	32/	64	
F1	F2	F3	F4

Figure A-8 RI/O PLC (NAC) Setting

---

## APPENDIX

### A.1.2 CONTROLNET

The ControlNet network (Figure A-9) is an open control network that provides real-time, high-throughput applications by linking PLC processors, I/O, computers, operator interfaces, and other intelligent devices. The ControlNet network combines the functionality of an I/O network and a peer-to-peer network, while providing high-speed performance for both functions.

The ControlNet network transfers data at 5M bit/s, and provides repeatable transfers of critical control data (e.g., I/O updates and processor interlocking).

Due to its fast update times, the network can support highly distributed systems, especially those with high-speed digital I/O or heavy analog I/O content. I/O chassis and other devices can be located hundreds of meters from PLC processors; or, for distributed processing, a processor can be located at the I/O chassis where it can monitor its own resident I/O while communicating via the network to a supervisory controller.

The ControlNet network can process the following automation and control data on a single coaxial cable: peer-to-peer messaging, remote programming, troubleshooting, and I/O updates and PLC processor interlocking.

I/O and PLC processor interlocking update times that match application requirements can be selected. The network's media access method places a higher priority on I/O updates and processor interlocking; these messages always take precedence over transfers of non-time-critical data (e.g., program uploads/ downloads and messaging). The network simplifies PLC programming by eliminating the need to program block transfers.

APPENDIX

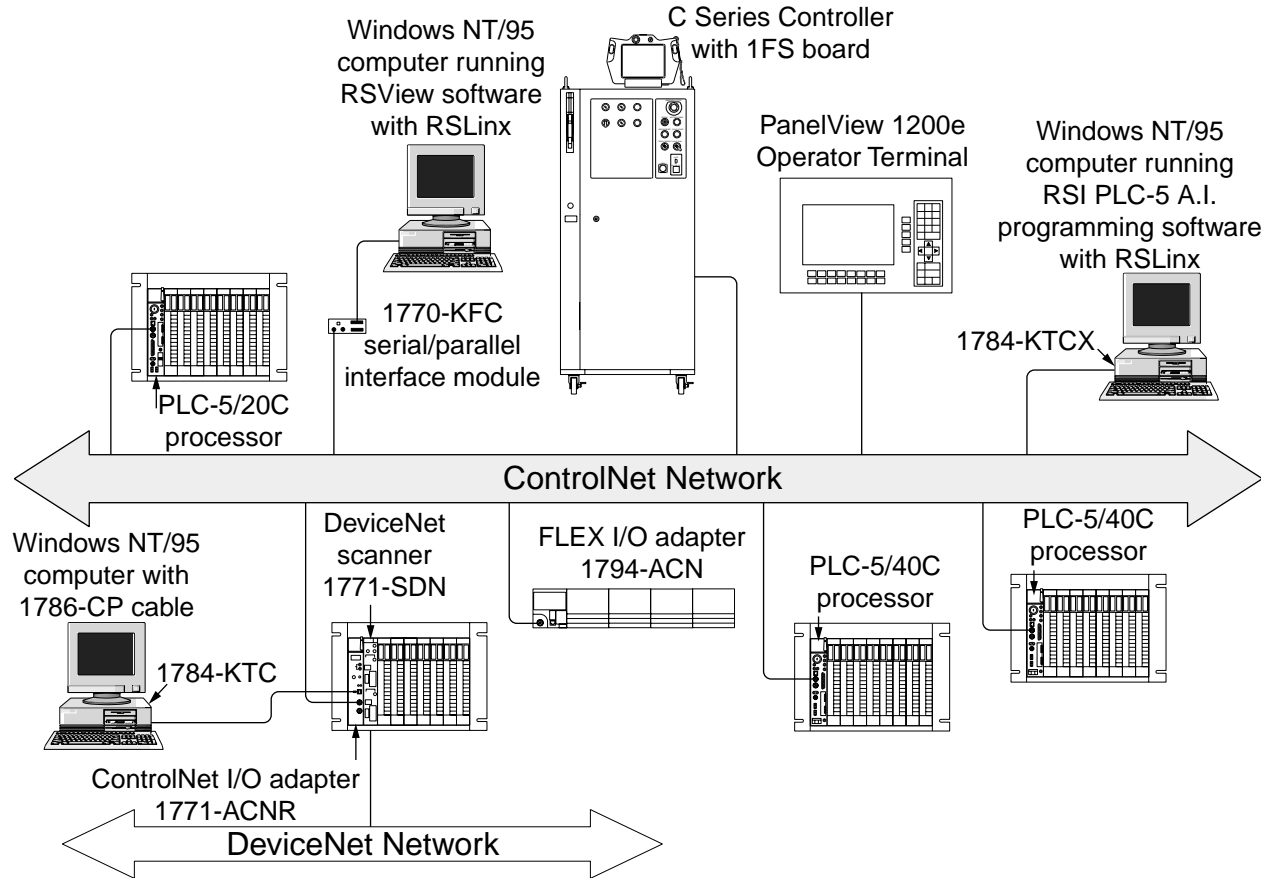


Figure A-9 ControlNet Network

A ControlNet network is simple and cost-effective to install, and offers flexible installation options. Allen-Bradley offers PLC processors with built-in ControlNet communication, ControlNet adapters for I/O chassis, and ControlNet interface cards for personal computers. Although the ControlNet network is based on bus technology, repeaters can be used to implement tree and star topologies. For added reliability, a redundant set of cables may be added between nodes. The additional cables provide a backup path if a primary cable fails. Each node compares the quality of the signals from each cable and uses the cable with the better signal. The 1FS board provides a redundant connection to the network via two BNC connectors (channel A and B).

Windows NT/95 computers can access real-time data over the network using RSLinx software. This software functions as a communications engine for software products such as Rockwell Software's man machine interface software RSVIEW, and programming software such as PLC-5 A.I. series programming software. RSLinx can also provide plant floor data to commercially available applications (e.g., Microsoft Excel and Access) for display, logging, or trending.

---

## APPENDIX

The ControlNet network provides real-time communication between PLC processors, I/O chassis, personal computers, third-party hardware and software, and related I/O devices. The network also allows the user to perform other functions such as programming the processors on the network and configuring network devices from a single terminal.

The network uses standard RG-6 coaxial cable and BNC connectors. ControlNet also supports fiber-optic media. Fiber-optic repeaters allow increased network distances and accommodate changes in topology.

The network uses taps with integrated drop cables for node connections. Up to 99 addressable nodes (with taps) can be installed anywhere along the network's trunk cable, with a maximum length per segment ranging from 250 meters (820 ft) to 1,000 meters (3,280 ft), depending on how many taps are used. There is no minimum tap separation. The network can be accessed from every node, including ControlNet I/O adapters, using the network access port.

The ControlNet network is compatible with a range of products, and can be added to existing systems. The PLC-5 processor includes standard ports for communication with devices on Universal Remote I/O and Data Highway Plus networks.

The ControlNet network provides a time-critical link between PLC processors and I/O devices, and the architectural link between information and device layers of a communication architecture.

---

## APPENDIX

### A.2 MOTION CONTROL AND LOCATION DEFINITION

#### A.2.1 C1MOVE AND C2MOVE COMMANDS

**C1MOVE**            **location, clamp number**  
**C2MOVE**            **location, clamp number.**

C1MOVE and C2MOVE commands are used to create a circular interpolation path.

The commands C1MOVE and C2MOVE specify the arc of the robot path to reach the taught location. Three locations are needed for the controller to calculate a circular path trajectory.

The first location is the beginning of the arc and can be specified with any of the following commands: Align, C1MOVE, C2MOVE, DELAY, DRAW, TDRAW, DRIVE, HOME, JMOVE, JAPPRO, JDEPART, LMOVE, LAPPRO, LDEPART, STABLE, or XMOVE.

The second location is defined by the C1MOVE command and is a location that is on the arc that connects the arc beginning and arc ending points.

The C2MOVE command defines the arc end point.

**location:**            Specifies the destination of the C1MOVE or C2MOVE.

The location specified must be the name of a location variable and can be a precision point, transformation, or compound transformation location.

**clamp number:**    Specifies the clamp number to open or close when the robot reaches the destination location.

A positive number opens the clamp.

A negative number closes the clamp.

If the clamp number is omitted, the clamp signal does not change.

APPENDIX

Example:

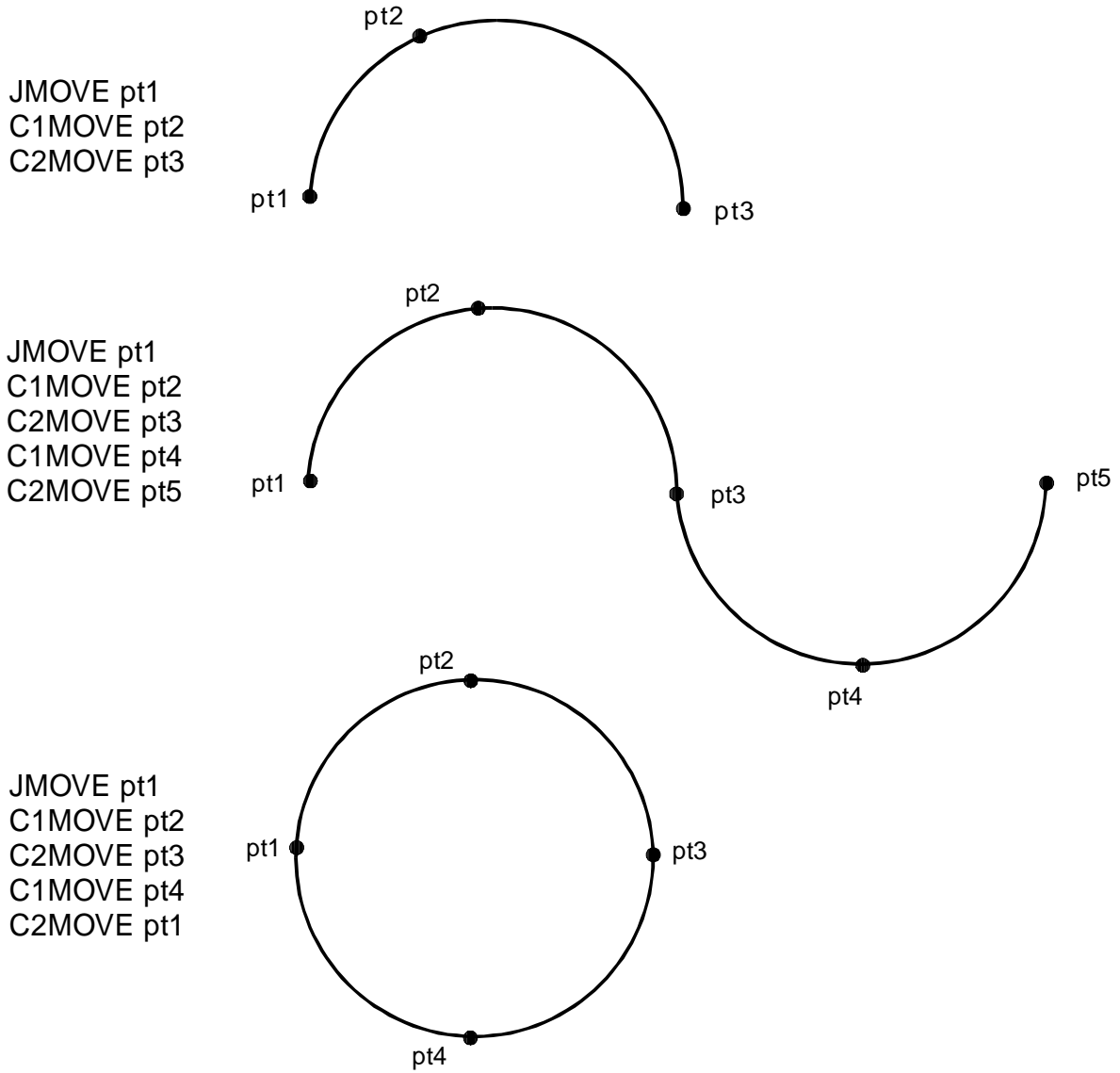


Figure A-10 Circular Interpolation

---

## APPENDIX

### A.2.2 FMOVE COMMAND

#### **FMOVE**                    **location, clamp number**

The FMOVE command is used in conjunction with the FIXED TOOL DIMENSIONS function and allows the user to program moves relative to an external fixed point in the work envelope.

This type of interpolation is called fixed linear interpolation (FLIN).

For a FMOVE, the robot path is calculated to maintain a set relationship to a fixed point.

Applications for FLIN moves include moving a part around a fixed sealing dispenser or a fixed stud welding gun.

Figure A-11 shows the auxiliary function screen used to enter the dimensions for a fixed tool used with FLIN moves.

Figure A-12 shows how the path of a windshield moved by a robot is different with a LMOVE and FMOVE.

The FMOVE path enables the programmer to reduce the number of programmed points needed to maintain a fixed distance from the sealing dispenser.

**location:**                    Specifies the destination of the FMOVE.

The location specified must be the name of a location variable and can be a precision point, transformation, or compound transformation location.

**clamp number:**            Specifies the clamp number to open or close when the robot reaches the destination location.

A positive number opens the clamp.

A negative number closes the clamp.

If the clamp number is omitted, the clamp signal does not change.



## APPENDIX

AUX 45 FIXED TOOL DIMENSIONS			
FIXED TOOL1			
X DIRECTION	0.0mm		
Y DIRECTION	0.0mm		
Z DIRECTION	0.0mm		
O ROTATION	0.0deg		
A ROTATION	0.0deg		
T ROTATION	0.0deg		
F1	F2	F3	F4
PREV.DATA		NEXT PG	

Figure A-11 FIXED TOOL DIMENSIONS Screen

**APPENDIX**

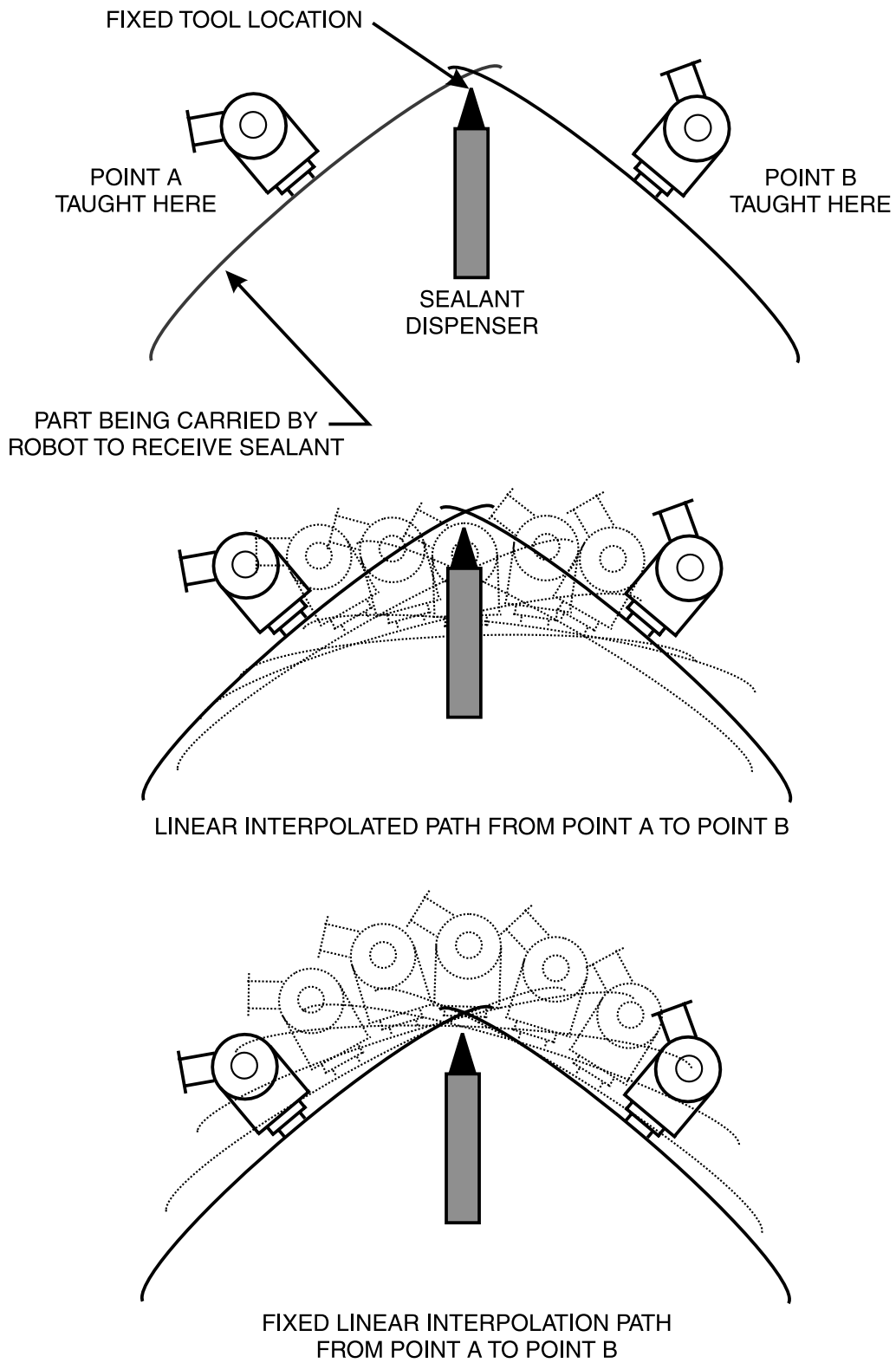


Figure A-12 FMOVE Command

**APPENDIX****A.2.3 MVWAIT COMMAND****MVWAIT distance in mm or time in seconds**

The MVWAIT command is used to determine when the next program instruction is executed. Figure A-10 shows two examples of the MVWAIT command with the PREFETCH.SIGINS switch in the on position.

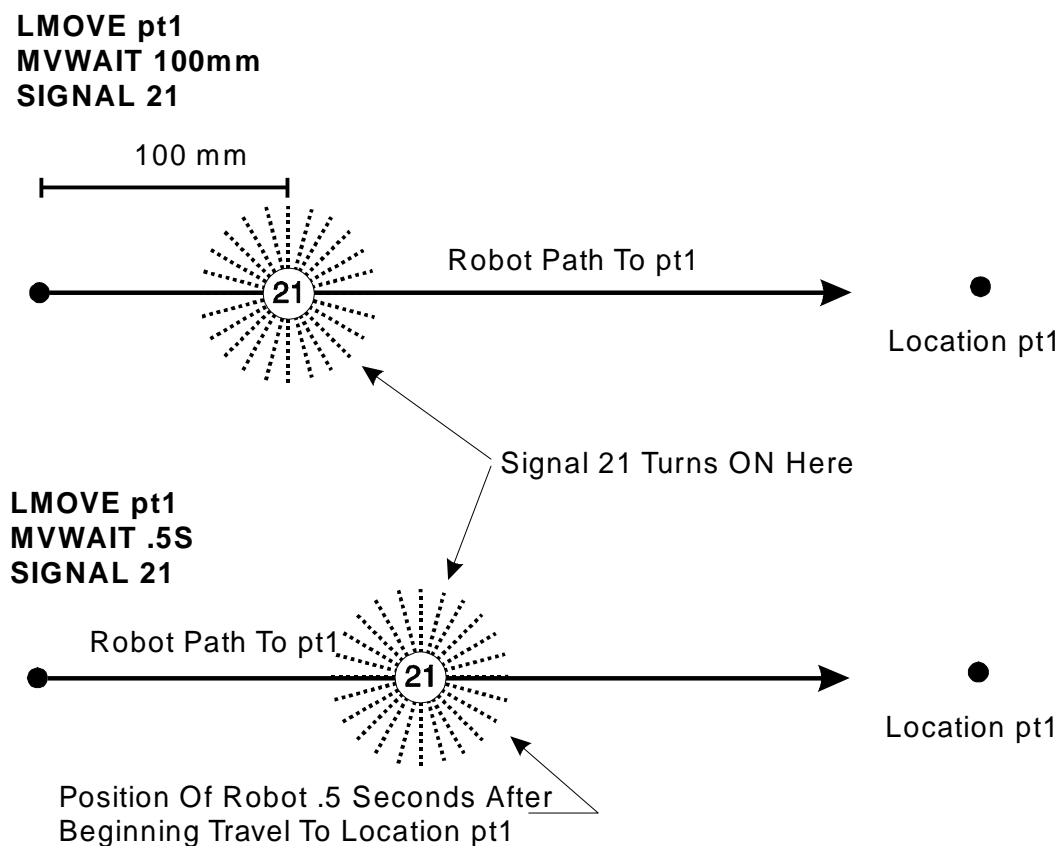


Figure A-13 MVWAIT Command

## APPENDIX

### A.2.4 BSPEED COMMAND

#### **BSPEED speed**

The BSPEED command is used to change the playback speed of block step programs. The actual playback speed of the program is the product of the specified speed in the BSPEED command combined with the repeat condition speed and the program step speed. The playback speed cannot exceed 100%. The robot continues to operate at the specified BSPEED until the next BSPEED command is processed.

### A.2.5 TRHERE COMMAND

#### **TRHERE location variable**

The TRHERE command is used to add the traverse axis component to the compound transformation. The specified location variable can be either a transformation location or a compound transformation.

---

## APPENDIX

### A.3 SYSTEM PARAMETERS AND SWITCHES

#### A.3.1 NCHON AND NCHOFF COMMANDS

The NTCHON and NCHOFF commands are monitor or program instruction commands that turn the notch filter on and off. The notch filter affects the processing parameters of the PWM for servo motor speed control. The new setting of the notch filter, as specified with the NTCHON or NTCHOFF command is effective after the execution of the following commands: PRIME, STEP, MSTEP, and EXECUTE. Before utilizing the NTCHON and NTCHOFF commands it is recommended the Kawasaki Robotics engineering group is contacted to provide details regarding program playback performance.

#### A.3.2 WEIGHT COMMAND

##### WEIGHT payload

The WEIGHT command that affects the processing parameters of the PWM for servo motor speed, acceleration, and deceleration control is a program instruction. The payload is specified in kg up to the maximum payload.

#### A.3.3 MC COMMAND

The MC command is used as a prefix to monitor commands to allow monitor commands to be processed within a program. The MC commands are: MCABORT, MCCONTINUE, MCERESET, MCEXECUTE, MCHOLD, and MCSPEED.

#### A.3.4 AUTOSTART.PC SWITCHES

The AUTOSTART.PC system switches allow execution of the specified process control program to begin as soon as controller power is applied. Three process control programs can begin execution with this process, AUTOSTART.PC, AUTOSTART2.PC, and AUTOSTART3.PC. When the process control programs to be used with the AUTOSTART.PC switches are created they must be named to correspond with the specified switch. For example, the process control program named AUTOSTART2.PC begins execution when control power is applied if AUTOSTART2.PC system switch is turned on.

#### A.3.5 ERRSTART.PC SWITCH

The ERRSTART.PC system switch allows execution of the specified process control program to begin as soon as a specified error occurs. When the process control program to use with the ERRSTART.PC switch is created, it is named ERRSTART.PC to correspond with the specified switch.

## APPENDIX

### A.3.6 AFTER.WAIT.TMR SWITCH

The AFTER.WAIT.TMR system switch determines when timers specified in block step programs begin timing. With the AFTER.WAIT.TMR switch in the off state, timing begins when the robot reaches coincidence with the taught location. With the AFTER.WAIT.TMR switch in the ON position, timing begins when the robot reaches coincidence with the taught location and all wait conditions are satisfied.

## A.4 FUNCTION COMMANDS

### A.4.1 DEXT FUNCTION

**real variable = DEXT (location name, element number)**

The DEXT command is a real value function used to assign an element of a location variable to a real variable. The following example of the DEXT function assigns the y (element 2) distance component of the transformation location loc1 to the real value loc.y.

$$\text{loc.y} = \text{DEXT} (\text{loc1}, 2)$$

In the above example, if the components of location loc1 are 1151, 1375, 950, -36, 87, 113, the value of real variable loc.y is 1375.

### A.4.2 PRIORITY FUNCTION

**real variable =PRIORITY**

The PRIORITY function is a real value function used to assign the priority level of the current program to a real variable. The following example of the PRIORITY function assigns the value of the program (as specified with the LOCK command) to the real variable stat.

$$\text{stat} = \text{PRIORITY}$$

In the above example, if the currently selected program is defined with the LOCK 2 command, the value of stat is 2.

### A.4.3 PCSCAN COMMAND

**PCSCAN time.**

The PCSCAN command is a process control program instruction used to set the scan time of the process control program. If the specified time is 1.5 seconds, the execution of the program is processed every 1.5 seconds. If the specified time is less than the normal processing time of the program, this command does not affect the program.

## APPENDIX

### A.5 PROCESS CONTROL COMMANDS

A process control (PC) program is an AS Language program executed simultaneously with a robot control program. A robot control program and up to three PC programs can run simultaneously. A PC program is used to monitor or 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 programs cannot use robot motion, BASE, or TOOL instructions. The only exception to this requirement for PC programs is the BRAKE instruction. PC programs are used to display messages on the keyboard screen using the PRINT instruction. All internal and external binary signals are available for PC programs.

#### A.5.1 PCSTATUS COMMAND

##### 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 is 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 displays the status of the specified PC program in the format shown in figure A-11.

```
$PCSTA
PC status:          <status>
Execution cycles
  Completed cycles:  (#)
  Remaining cycles:  (#)
Program name      Priority  Step number
No program is running
$
```

Figure A-14 PCSTATUS Command

---

**APPENDIX****A.5.2 PC PROGRAM CONTROL****PCEXECUTE**      **program\_name,execution\_cycles,start\_step**

Executes a PC program

**program\_name:**      Name of the program to execute as a PC program. If omitted, the program last executed with the PCEXECUTE command is selected.**execution\_cycles:**      A number specifying how many times the program executes. If omitted, it is set to 1. Entering a negative number causes the program to execute continuously.**starting\_step:**      Step number of the program to start execution.**PCABORT**      Aborts execution of the PC program. This is similar to the ABORT monitor command used for robot control programs, except it aborts the current PC program. Program execution is resumed with the PCCONTINUE command.**PCEND**      **task\_number**

Terminates the execution of the PC program when the current PC program executes a STOP instruction.

**task\_number:**      Entered as 1 or -1, if omitted, 1 is assumed.

When the current PC program executes a STOP instruction (or an equivalent instruction such as RETURN in the robot control program), execution of the PC program is stopped immediately, regardless of remaining cycles. If the current cycle is not completed, execution is resumed with the PCCONTINUE command.

This command waits until the program stops at the STOP instruction. If the PC program loops internally and the current execution cycle never ends, the PCEND command waits indefinitely. To cancel the PCEND command, enter a negative number. To terminate a PC program that loops endlessly, use the PCABORT command.



---

**APPENDIX****PCCONTINUE**      **next**

Resumes execution of an interrupted PC program. The NEXT argument is used to skip WAIT instructions.

When the PC program is interrupted by a PAUSE, PCABORT instruction, or an error, the PCCONTINUE command begins execution of the PC program at the step following the interrupted step.

**PCKILL**      Initializes the PC program stack.

The PCKILL command clears the PC program stack, but does not delete the PC program(s). If a PC program is interrupted for any reason, the program is not removed from the stack.

The current PC program must be removed from the stack with the PCKILL command before it can be deleted.

**PCSTEP**      **program\_name,execution\_cycles,start\_step.**

The PCSTEP command is used to execute a single step of a PC program.

**program\_name:**      Name of the PC program for step execution.

**execution\_cycles:**      Specifies the number of cycles for the PC program to execute with the PCSTEP command.

**start\_step:**      The step number of the program to execute first.

**PCSCAN**      **time**

The PCSCAN command is used to specify the length of time it takes 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 is processed at the CPU speed.

---

**APPENDIX****A.6 AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE SET**

The following table lists the entire American Standard Code for Information Interchange (ASCII) character set. All the characters from SP (octal 040) through “~” (octal 176) can be used as ASCII constants or within text strings. The characters with octal values from 0 to 40 and the value 177, are non-printing control characters with the meanings given in the table.

<u>ASCII Character</u>	<u>Octal Value</u>	<u>Decimal Value</u>	<u>Meaning of Control Character</u>
NUL	000	000	Null
SOH	001	001	Start of heading
STX	002	002	Start of text
EXT	003	003	End of text
EOT	004	004	End of transmission
ENQ	005	005	Enquiry
ACK	006	006	Acknowledgement
BEL	007	007	Bell
BS	010	008	Backspace
HT	011	009	Horizontal tab
LF	012	010	Line feed
VT	013	011	Vertical tab
FF	014	012	Form feed
CR	015	013	Carriage return
SO	016	014	Shift out
SI	017	015	Shift in
DLE	020	016	Data link escape
DC1	021	017	Direct control 1
DC2	022	018	Direct control 2
DC3	023	019	Direct control 3
DC4	024	020	Direct control 4
NAK	025	021	Negative acknowledge
SYN	026	022	Synchronous idle
ETB	027	023	End of transmission block
CAN	030	024	Cancel
EM	031	025	End of medium
SUB	032	026	Substitute
ESC	033	027	Escape
FS	034	028	File separator
GS	035	029	Group separator
RS	036	030	Record separator
US	037	031	Unit separator
SP	040	032	Space

## APPENDIX

<u>ASCII Character</u>	<u>Octal Value</u>	<u>Decimal Value</u>	<u>ACII Character</u>	<u>Octal Value</u>	<u>Decimal Value</u>
!	041	033	P	120	080
"	042	034	Q	121	081
#	043	035	R	122	082
\$	044	036	S	123	083
%	045	037	T	124	084
&	046	038	U	125	085
'	047	039	V	126	086
(	050	040	W	127	087
)	051	041	X	130	088
*	052	042	Y	131	089
+	053	043	Z	132	090
'	054	044	[	133	091
-	055	045	\	134	092
ù	056	046	]	135	093
/	057	047	^	136	094
0	060	048	-	137	095
1	061	049	'	140	096
2	062	050	a	141	097
3	063	051	b	142	098
4	064	052	c	143	099
5	065	053	d	144	100
6	066	054	e	145	101
7	067	055	f	146	102
8	070	056	g	147	103
9	071	057	h	150	104
:	072	058	i	151	105
;	073	059	j	152	106
<	074	060	k	153	107
=	075	061	l	154	108
>	076	062	m	155	109
?	077	063	n	156	110
@	100	064	o	157	111

**APPENDIX**

<u>ASCII Character</u>	<u>Octal Value</u>	<u>Decimal Value</u>	<u>ACII Character</u>	<u>Octal Value</u>	<u>Decimal Value</u>
A	101	065	p	160	12
B	102	066	q	161	113
C	103	067	r	162	114
D	104	068	s	163	115
E	105	069	t	164	116
F	106	070	u	165	117
G	107	071	v	166	118
H	110	072	w	167	119
I	111	073	x	170	120
J	112	074	y	171	121
K	113	075	z	172	122
L	114	076	{	173	123
M	115	077		174	124
N	116	078	}	175	125
O	117	079	~	176	126
DEL	177	127	Delete		

## APPENDIX

### A.7 KEYWORDS LIST

\*Legend

M (Monitor command)    E (Editor command)    P (Program instruction)    S (Switch)  
F (Function)            O (Operator)            K (Other keywords)

ABBRV.	NAME	FUNCTION	TYPE*	FORMAT (ARGUMENT)
AB	ABORT	Stop program execution	M	ABORT
AB	ABOVE	Change configuration	P	ABOVE
ABS	ABS	Absolute value	F	ABS (Real_value_expression)
ACCE	ACCEL	Change acceleration factor	P	ACCEL acceleration ALWAYS
ACCU	ACCURACY	Change accuracy	P	ACCURACY range ALWAYS
AF	AFTER.WAIT. TIMER	Controls when timer function begins	S	...AFTER.WAIT.TMR...
AL	ALIGN	Align tool Z axis	P	ALIGN
A	ALWAYS	Effective afterwards	K	...ALWAYS
AND	AND	Logical AND	O	...AND...
ARC	ARC	Dedicated signal to welding power supply	S	ARC ON/OFF
ARMIOSET	ARMIOSET	Arm ID board I/O signals set	M	ARMIOSET robot No.,OX sig._No. _num._of_ OX_sig.,WX_sig._No. _num._of_ WX_sig.
ASC	ASC	Get ASCII value	F	ASC (string, index)
ATAN2	ATAN2	Arctangent	F	ATAN2 (real-value, real-value)
AUTOSTART.	AUTOSTART.PC	Starts PC program execution	S	...AUTOSTART.PC...
AUTOSTART2.	AUTOSTART2.PC	Starts PC program execution	S	...AUTOSTART2.PC...
AUTOSTART3.	AUTOSTART3.PC	Starts PC program execution	S	...AUTOSTART3.PC...
AVE_TRANS	AVE_TRANS	Returns the average of two transformation locations	F	AVE_TRANS(transformation_ value, transformation_value2 ....BAND....
BAND	BAND	Binary logical AND	O	....BAND....
BA	BASE	Change base coordinate system	M/P	BASE transformation_value
BASE	BASE	Returns the origin of the base coordinate system	F	BASE
BAT	BATCHK	Battery error check on power up	M	BATCHK
BE	BELOW	Change configuration	P	BELOW
BI	BITS	Set output signals	M/P	BITS irst_signal,number_of_ signals =value
BITS	BITS	Get signal states	F	BITS (first_signal,number_of_ signals)
BOR	BOR	Binary logical OR	O	...BOR...
BRA	BRAKE	Brake robot motion	P	BRAKE
BRE	BREAK	Cause break in CP motion	P	BREAK
BSP	BSPEED	Sets speed of block programs	P	BSPEED_speed
BX	BXOR	Binary exclusive logical OR	O	...BXOR...
BY	BY		K	BY...
C	CHANGE	Change program	E	C program name, step_number
CAL	CALL	Call subroutine	P	CALL program_name
CAS	CASE	CASE structure	P	CASE number OF...VALUE... END
CH	CHECK.HOLD	Changes useage of some motion commands	S	CHECK.HOLD

**APPENDIX**

<u>ABBRV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
CH	CHSUM	Resets check sum error on power up	M	CHSUM
C1WC	C1WELD CONTINUE	Circular interpolated welding motion	P	C1WC loc, weld_condition_NO.
C2WE	C2WELD END	Circular interpolated welding motion	P	C2WE loc, weld_condition_NO.
COM	COM	Ones complement	O	COM...
CO	CONTINUE	Resume execution	M	CONTINUE (NEXT)
COLCALON	COLCALON	Collision detection autocal ON	M	COLCALON
COLCALOFF	COLCALOFF	Collision detection autocal OFF	M	COLCALOFF
COLINIT	COLINIT	Sets collision detection defaults	M	COLINIT threshold_No.
COLMVON	COLMVON	Stress remove motion ON	M/P	COLMVON
COLMVOFF	COLMVOFF	Stress remove motion OFF	M/P	COLMVOFF
COLR	COLR	Sets repeat col. detection	M/P	COLR JT1 - JT6 thresholds
COLRON	COLRON	Starts repeat col. det. monitoring	M/P	COLRON
COLROFF	COLROFF	Stops repeat col. det. monitoring	M/P	COLROFF
COLRJ	COLRJ	Sets repeat shock detection	M/P	COLRJ JT1 - JT6 thresholds
COLRJON	COLRJON	Starts rep. shock det. monitoring	M/P	COLRJON
COLRJOFF	COLRJOFF	Stops rep. shock det. monitoring	M/P	COLRJOFF
COLS	COLSTATE	Displays col. and shock det. data	M	COLSTATE threshold_No.
COLT	COLT	Sets teach col. detection	M	COLT JT1 - JT6 thresholds
COLTON	COLTON	Starts teach col. det. monitoring	M	COLTON
COLTOFF	COLTOFF	Stops teach col. det. monitoring	M	COLTOFF
COLTJ	COLTJ	Sets teach shock detection	M	COLTJ JT1 - JT6 thresholds
COLTJON	COLTJON	Starts tea. shock det. monitoring	M	COLTJON
COLTJOFF	COLTJOFF	Stops tea. shock det. monitoring	M	COLTJOFF
COP	COPY	Copies an existing program(s) to a new program name	M	COPY (new_name=old_name(s))
COS	COS	Cosine	F	COS (real_value_expression)
CP	CP	CP (continuous path)	S	CP..
CS	CS	Sets cycle start activation mode	S	CS
CY	CYCLE.STOP	Set program restart point after external hold reset	S	CYCLE.STOP
\$CHR	\$CHR	Returns value of ASCII character	F	\$CHR (real_vlaue)
D	D	Delete program steps	E	D number_of_steps
DBUSE	DEBUSE	Declare data base	M/P	DEBUSE groove_NO.,leg_length, groove_angle
DECE	DECEL	Change deceleration factor	P	DECEL (deceleration) (ALWAYS)
DECO	DECOMPOSE	Extract location components	P	DECOMPOSE array_name [index]=location
DEF	DEFSIG	Define special signals	M	DEFSIG INPUT/OUTPUT
DEL	DELAY	Stop the robot for a given time	P	DELAY time (seconds)
DEL	DELETE	Delete data in memory	M	DELETE program_name,...
DEL/P	DELETE/P	Delete programs in memory	M	DELETE/P program_name,...
DEL/L	DELETE/L	Delete locations in memory	M	DELETE/L locaton_name,...
DEL/R	DELETE/R	Delete real variables in memory	M	DELETE/R variable_name,...
DEL/S	DELETE/S	Delete char. strings in memory	M	DELETE/S string_name,...
DEST	DEST	Destination location	F	DEST
#DEST	#DEST	Destination location	F	#DEST

## APPENDIX

<u>ABBREV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
DEXT	DEXT	Returns an element of a location name	F	DEXT (location name,index)
DIR	DIRECTORY	List data in memory	M	DIRECTORY
DIR/P	DIRECTORY/P	List programs in memory	M	DIRECTORY/P (program_name,)
DIR/L	DIRECTORY/L	List locations in memory	M	DIRECTORY/L (location_name,)
DIR/R	DIRECTORY/R	List real variables in memory	M	DIRECTORY/R (variable_name,)
DIR/S	DIRECTORY/S	List strings in memory	M	DIRECTORY/S (string_name,)
DIS	DISPIO_01	Changes I/O display from x/o to 1/0	S	DISPIO_01
DIST	DISTANCE	Distance	F	DISTANCE (location, location)
DL	DLYSIG	Output delay signal	M/P	DLYSIG (signal,time)
DO	DO	Execute a program instruction	M	DO (instruction)
DO	DO	DO structure	P	DO...UNTIL...
DRA	DRAW	Move the robot by given amount (mm)	P	DRAW (x, y, z)
DRI	DRIVE	Move a single joint (degrees)	P	DRIVE (joint, degree, speed)
DW	DWRIST	Change wrist configuration	P	DWRIST
DX	DX	X component	F	DX (location)
DY	DY	Y component	F	DY (location)
DZ	DZ	Z component	F	DZ (location)
E	EXIT	Exit editor mode	E	E
ED	EDIT	Enter editor mode	M	EDIT (program_name, step)
EL	ELSE	IF structure	P	IF...ELSE...END
ENCCHK_E	ENCCHK_EMG	Checks position at E-stop	S	ENCCHK_EMG
ENCCHK_P	ENCCHK_PON	Checks current position with position at power off	S	ENCCHK_PON
END	END	End control flow structure	P	FOR...END, CASE...END
ENV	ENV_DATA	Sets MFP environment data	M	ENV_DATA
ENV2	ENV2_DATA	Sets MFP environment2 data	M	ENV2_DATA
ERE	ERESET	Reset error condition	M	ERESET
ERR	ERRLOG	Display history of errors	M	ERRLOG
ERROR	ERROR	Returns error number	F	ERROR
ERRS	ERRSTART.PC	Starts PC program at occurrence of specified error	S	...ERRSTART.PC...
EX	EXECUTE	Execute program	M	EXECUTE (program_name, cycles,step_number)
EX	EXTCALL	Call program selected by signals	P	EXTCALL
F	FIND	Search characters	E	F character_string
FALSE	FALSE	False value	F	FALSE...
FDEL	FDELETE	Delete PC card or disk files	M	FDELETE file_name
FDIR	FDIRECTORY	List PC card or disk files	M	FDIRECTORY
FO	FORMAT	Formats PC card or floppy disk	M	FORMAT
FOR	FOR	Part of FOR/TO control flow structure	P	FOR (loop_variable=start_value) TO (end_value)...END
FR	FRAME	Transformation of relative coordinate system	F	FRAME (location1, location2, location3,location4)
FR	FREE	Show free memory amount	M	FREE
G	GOTO	Branch execution	P	GOTO label IF condition
GET1WCON	GET1_WCON	Sets weld conditions DEBUSE and SETPASS	M/P	GET1WCON weld_condition_NO.,sp,a,v,wa,wf,pn

## APPENDIX

ABBRV.	NAME	FUNCTION	TYPE*	FORMAT (ARGUMENT)
GET2WCON	GET2_ WCON	Sets weld end conditions DEBUSE and SETPASS	M/P	GET2WCON weld_condition_ NO.,time,a,v
GETPASS	GETPASS	Sets number of passes (multipass option)	F	Real_variable=GETPASS
GETWPOS	GETWPOS	Set shift value	M/P	GETWPOS ΔY, ΔZ, ΔΘ
HA	HALT	Stop execution	P	HALT
HEL	HELP	Displays list of AS commands	M	HELP
HEL/DO	HELP/DO	Displays list of AS DO command arguments	M	HELP/DO
HEL/F	HELP/F	Displays list of AS function commands	M	HELP/F
HEL/M	HELP/M	Displays list of AS montior commands	M	HELP/M
HEL/MC	HELP/MC	Displays list of AS program control monitor commands	M	HELP/MC
HEL/P	HELP/P	Displays list of AS program commands	M	HELP/P
HEL/PPC	HELP/PPC	Displays list of AS process control program commands	M	HELP/PPC
HE	HERE	Record current location	M/P	HERE (location_name)
HERE	HERE	Returns current transformation location	F	HERE
#HERE	#HERE	Returns current precision location	F	#HERE
HM	HMOVE	Linear move used to prevent singulartary errors	P	HMOVE (location_name)
HO	HOLD	Stop execution	M	HOLD
HO	HOME	Moves the robot to home	P	HOME (home_position_number)
#HOME	#HOME	Moves robot to home precision point position in a joint move	F	#HOME (home_position_ number)
I2PG START	I2PG START	Starts current monitoring	P	I2PG START
I2PG END	I2PG END	Stops current monitoring	P	I2PG END
I	INSERT	Insert new steps	E	I
ID	ID	Displays current software version	M	ID
IF	IF	Cause conditional branch	P	IF Condition GOTO label
IF	IF	IF structure	P	IF...ELSE...END
IG	IGNORE	Cancel interruption	P	IGNORE (signal_number)
I	INPUT	Specify input	K	DEFSIG INPUT
INT	INT	Returns an integer value	F	INT (expression)
IO	IO	Display I/O signals	M	IO
JAPPRO	JAPPRO	Move the robot near the given location	P	JAPPRO (location_name, distance)
JAS	JARCSPT	Joint interpolated motion to tack weld	P	JAS (location_name,weld_ condition) NO.
JDEPART	JDEPART	Back the robot tool	P	JDEPART (distance)
JM	JMOVE	Start joint interpolated motion	P	JMOVE (location_name)
JWS	JWELDSTART	Joint interpolated motion to weld starting point	P	JWS (location_name,weld_ condition) NO.
KI	KILL	Initialize program stack	P	KILL



## APPENDIX

<u>ABBRV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
L	LAST	Select the previous step	E	L
LA	LAPPRO	Move the robot near the given location	P	LAPPRO (location_name, distance)
LAS	LARCSPOT	Linear interpolated motion to tack weld	P	LAS (location_name, weld_condition) NO.
LDEPART	LDEPART	Back the torch away	P	LDEPART (distance)
LE	LEFTY	Change configuration	P	LEFTY
LEN	LEN	Number of characters	F	LEN (string)
LI	LIST	Display data	M	LIST (program_name,...)
LI/L	LIST/L	Displays location listing	M	LIST/L (location_name,...)
LI/P	LIST/P	Display program listing	M	LIST/P (program_name,...)
LI/R	LIST/R	Displays real variable listing	M	LIST/R (real variable_name,...)
LI/S	LIST/S	Display string data	M	LIST/S (string_name,...)
LL	LLIMIT	Set lower software limit	M/P	LLIMIT (joint displacement_value)
LM	LMOVE	Start linear interpolated motion	P	LMOVE (location_name)
LO	LOAD	Load data from PC card or disk file	M	LOAD (file_name) LOAD/Q (file_name) for selected data load
LO	LOCK	Used to change the priority of a robot program	P	LOCK (priority_number)
LWC	LWELD_ CONTINUE	Weld continuous point	P	LWC (location_name, weld_condition)_NO.
LWE	LWELDEND	Weld end point	P	LWE (location_name, weld_condition) NO.
LWS	LWELDSTART	Weld starting point	P	LWS location_name
M	MODIFY	Replace characters	E	M/old_string/new_string
M	MESSAGE	Message switch	S	...MESSAGE
M	MC	Execute monitor commands	P	MC (monitor_command)
MM/S	MM/S	Millimeter per second	K	...MM/S
MM/MIN	MM/M	Millimeter per minute	K	...MM/MIN
MNTLOG	MNTLOG	Displays maintenance log		MNTLOG robot_number
MNTREC	MNTREC	Record maintenance log entries		MNTREC robot_number
MOD	MOD	Remainder	O	...MOD...
MS	MSTEP	Execute one motion instruction	M	MSTEP (program, count, step_number)
MVWAIT	MVWAIT	Signal timing and program processing, in time or distance	P	MVWAIT (numeric_value)
N	NEXT	Skip to the next step	K	CONTINUE NEXT
NCHOF	NCHOFF	Turns the notch filter OFF	P	NCHOFF
NCHON	NCHON	Turns notch filter ON	P	NCHON
NOT	NOT	Negation	O	...NOT...
NULL	NULL	Null transformation value	F	NULL...
O	OVER	Place cursor on current step	E	O
OF	OFF	Switch off	M/P	(switch_name),OFF
OF	OFF	False value	F	OFF..
ON	ON	Switch on	M/P	(switch_name),ON
ON	ON	Switch on	P	(switch_name),ON
ON	ON	True value	F	ON...
ON	ON	Set interruption condition	P	ON (signal_name) CALL (program_name)

**APPENDIX**

<u>ABBREV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
ON	ON	Set interruption condition	P	ON (signal_name) GOTO (label)
ONI	ONI	Set interruption condition	P	ONI (signal_number) CALL (program_name)
ONI	ONI	Set interruption condition	P	ONI (signal_number) GOTO (label)
OP	OPLOG	Display history of operations	M	OPLOG
OR	OR	Logical OR	O	...OR...
O	OUTPUT	Specify output	K	DEFSIG_OUTPUT
O	OVER	Used to overwrite program line	E	O
OX	OX	Specify OX signal number	P	OX (signal_number)
OX	OX.PREOUT	Sets OX signal timing	S	...OX.PREOUT...
P	PRINT	Display program steps	E	P (number_of_steps)
PA	PAUSE	Suspend execution	P	PAUSE
PCA	PCABORT	Stop PC program execution	M	PCABORT
PCC	PCCONTINUE	Resume PC program execution	M	PCCONTINUE NEXT
PCEN	PCEND	Terminate PC program execution at the next STOP instruction	M/P	PCEND (program,task_number)
PCEX	PCEXECUTE	EXECUTE PC program	M/P	PCEXECUTE (program, cycles, step_number)
PCK	PCKILL	Initialize PC program stack	P	PCKILL
PCSC	PCSCAN	Sets PC program scan time	P	PCSCAN (time)
PCS	PCSTATUS	Display PC program status	P	PCSTATUS (program_number)
PCSTE	PCSTEP	Executes one step of a PC program	M	PCSTEP (program,cycles,step_number)
PI	PI	Value of pi (3.14...)	F	PI...
PICKLOC	PICKLOC	Specify target location	P	PICLOCK (location_name)
PO	POINT	Define location variable	M/P	POINT (location1=location2)
PO/X	POINT/X	Set value to X component	M/P	POINT/X (location1=location2)
PO/Y	POINT/Y	Set value to Y component	M/P	POINT/Y (location1=location2)
PO/Z	POINT/Z	Set value to Z component	M/P	POINT/Z (location1=location2)
PO/O	POINT/O	Set value to O component	M/P	POINT/O (location1=location2)
PO/A	POINT/A	Set value to A component	M/P	POINT/A (location1=location2)
PO/T	POINT/T	Set value to T component	M/P	POINT/T (location1=location2)
PO/OAT	PO/OAT	Set value to O, A, T components	M/P	POINT/OAT (location1=location2)
PO/7	POINT/7	Set value to seventh axis	M/P	POINT/7 (location1=location2)
POWER	POWER	Displays status of motor power	S	SWITCH POWER (ON/OFF)
#PPOINT	#PPOINT	Joint displacement value	F	#POINT (jt1,jt2,jt3,jt4,jt5,jt6)
PR	PREFETCH. SIGINS	Sets AS Language signal timing	S	...PREFETCH.SIGINS...
PRIM	PRIME	Set up program for execution	M	PRIME (program,cycle,step)
PRIN	PRINT	Displays data on the selected screen	P	PRINT (device:output data,...)
P	PRINT	displays specified number of program steps	E	P (number_of_steps)
PRIORITY	PRIORITY	Sets program run status priority	F	PRIORITY
PROM	PROMPT	Prompts operator to enter data	P	PROMPT (device,output character string,variable)
PU	PULSE	Pulse output signal	M/P	PULSE (signal_number,time)
QTOOL	Q	Sets type of tool used, ON block step, OFF AS Language	S	QTOOL ON/OFF

## APPENDIX

<u>ABBRV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
R	R	Replace characters	E	R (characters)
RANDOM	RANDOM	Random Value	F	RANDOM...
REC	REC_ACCEPT	Inhibit or enable recording of data	S	REC ACCEPT
REN	RENAME	Change program name	M	RENAME (new_name=old_name)
REPEAT	REPEAT	Shows status of TEACH/REPEAT switch	S	SWITCH REPEAT
RES	RESET	Turn off all output signals	M/P	RESET
RET	RETURN	Return execution to caller program	P	RETURN
RI	RIGHTY	Change configuration	P	RIGHTY
RU	RUNMASK	Mask signals	P	RUNMASK (first_signal,number_of_signals)
RX	RX	Transformation representing rotation about X axis	F	RX (angle)
RY	RY	Transformation representing rotation about Y axis	F	RY (angle)
RZ	RZ	Transformation representing rotation about Z axis	F	RZ (angle)
S	S	Select program step	E	S (step_number)
SA	SAVE	Save data in a PC card or disk file	M	SAVE (file_name=program_name)
SA/P	SAVE/P	Save programs in a PC card or disk file	M	SAVE/P (file_name=program_name)
SA/L	SAVE/L	Save locations in a PC card or disk file	M	SAVE/L (file_name=location_name)
SA/R	SAVE/R	Save real variables in a PC card or disk file	M	SAVE/R (file_name=variable_name)
SA/S	SAVE/S	Save strings in a PC card or disk file	M	SAVE/S (file_name=string_name)
SA/ELOG	SAVE/ELOG	Saves error log information	M	SAVE/ELOG (file_name)
SA/SYS	SAVE/SYS	Saves system data	M	SAVE/SYS (file_name)
SCN	SCNT	Sets an output associated with a counter value	M/P	SCNT (count_up_signal,count_down_signal,reset_signal,count)
SCNTR	SCNTRESET	Resets a counter	M/P	SCNTRESET (reset_signal)
SETCOL	SETCOLTHID	Display or change col. det. dat	M	SETCOL threshold_number
SETCOL	SETCOLTHID	Apply col. det. dat in a program	P	SETCOL threshold_number
SETH	SETHOME	Set home position	M	SETHOME (accuracy),HERE
SET2	SET2HOME	Set home position 2	M	SET2HOME (accuracy),HERE
SFLK	SFLK	Sets the ON/OFF time of flicker signals	M/P	SFLK (flicker signal=time)
SFLP	SFLP	Sets an output signal based on status of a set signal and a reset signal	M/P	SFLP (output_signal=set_signal,reset_signal)
SCR	SCREEN	SCREEN system switch	S	...SCREEN...
SHIFT	SHIFT	Transformation shifted from original location by given amount	F	SHIFT (transformation_value BY dx, dy, dz)
SIG	SIGNAL	Turn output signals ON or OFF	M/P	SIGNAL_(signal_number)
SIG	SIG	Logical AND of signal states	F	SIG (signal_number)

**APPENDIX**

<u>ABBRV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
SIN	SIN	Sine	F	SIN (real-value_expression)...
SL	SLOW_REPEAT	Used to reduce robot speed	M	SLOW REPEAT
SO	SOUT	Sets 1FS board output based on conditional statement	M/P	SOUT (signal_number=signal_condition)
SP	SPEED	Set monitor speed	M	SPEED (speed)
SP	SPEED	Set program speed	P	SPEED (speed)_ALWAYS
SQRT	SQRT	Square root	F	SQRT (real-value_expression)
STA	STATUS	Display system status	M	STATUS
STA	STABLE	Stop the robot for the given time	P	STABLE (time)
STE	STEP	Execute a single program step	M	STEP (program_name, count, step_number)
STI	STIM	Sets 1FS board timer output when specified time is elapsed	M/P	STIM (timer_number=input_signal,time)
STO	STOP	Terminate execution cycle	P	STOP
STP	STPNEXT	Used to execute the next step in a program with STP_ONCE switch ON	M	STPNEXT
ST	STP_ONCE	With STP_ONCE ON, a program executes one step at a time	S	STP_ONCE
SW	SWAIT	Wait for desired signal states	P	SWAIT (signal_number)
SW	SWITCH	Set system switches	M	SWITCH (switch_name) =ON/OFF
SY	SYSINIT	Initialize system	M	SYSINIT
TDRA	TDRAW	Move robot by given amount in tool coordinates	P	TDRAW (dx, dy, dz)
TE	TEACH	Teach locations with pendant	M	TEACH (location_name)
TEACH_LOCK	TEACH_LOCK	Displays status of TEACH LOCK switch (ON/OFF)	S	SWITCH TEACH_LOCK
THEN	THEN	IF structure	P	IF (logical) THEN... ELSE... END
TIM	TIME	Set date and time	M	TIME yy:mm:dd hh:mm:ss
TIMER	TIMER	Set timer	P	TIMER (timer_number)=(time)
TI	TIMER	Timer value	F	TIMER (timer_number)
TO	TO	FOR structure	K	FOR...TO...END
TO	TOOL	Specify tool transformation	M	TOOL (transformation_value)
TOO	TOOL	Specify tool transformation	P	TOOL (transformation_value)
TRADD	TRADD	Adds component of a traverse axis to a transformation location	F	TRADD (transformation_value)
TRANS	TRANS	Transformation value	F	TRANS (X, Y, Z, O, A, T)
TRHE	TRHERE	Records transformation location including the traverse axis component	M/P	TRHERE (location_name)
TRUE	TRUE	Logical TRUE	F	TRUE
TW	TWAIT	Wait for the specified time	P	TWAIT (time)
TY	TYPE	Display data	P	TYPE output_data,...
UL	ULIMIT	Set upper software limit	M/P	ULIMIT joint_displacement_value
U	UNTIL	DO structure	P	DO...UNTIL logical
UW	UWRIST	Change wrist configuration	P	UWRIST
VA	VALUE	Case value statement	P	CASE...VALUE...ELSE...END
W1SET	W1SET	Specify weld conditions	M/P	W1SET NO. = sp,a,v,wa,wf,pn
W2SET	W2SET	Specify end of weld conditions	M/P	W2SET NO. = time,a,v
WA	WAIT	Wait until condition is satisfied	P	WAIT (condition)

## APPENDIX

<u>ABBRV.</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>TYPE*</u>	<u>FORMAT (ARGUMENT)</u>
WE	WEIGHT	Changes servo feedback calculations to compensate for moved payload	M/P	WEIGHT (payload_weight)
W	WHERE	Display current robot location status	M	WHERE (mode) (1-6)
W	WHILE	DO structure	P	WHILE...DO...END
W.INCHIN	W.INCHING	Set wire inching speed	M/P	W.INCHIN sp,time
WLIST	WLIST	Displays W1SET conditions	M	WLIST
WLIST 2	WLIST 2	Displays W2SET conditions	M	WLIST 2
XMWIRE	XMWIRE	Wire check	P	XMWIRE location_name, inching_speed, retract_speed
XF	XFER	Transfers data within a program or to another program	M	XFER (destination_program,start step=source_program,first_step, number_of_steps)
X	XMOVE	Robot moves to the next location until the specified signal is received, then moves to the following location	P	XMOVE (location) TILL (signal_number)
XOR	XOR	Exclusive logical OR	O	XOR
ZSIG	ZSIGSPEC	Set number of signals installed	M	ZSIGSPEC
ZZE	ZZERO	Set/display zeroing data	M	ZZERO joint_number
+	+	Addition	O	...+...
-	-	Subtraction	O	...-...
*	*	Multiplication	O	...*...
/	/	Division	O	.../...
^	^	Power	O	...^...
<	<	Less than	O	...<...
<=	<=	Less than or equal to	O	...<=...
==	==	Mathematically equal to	O	...==...
<>	<>	Not equal to	O	...<>...
>=	>=	Greater than or equal to	O	...>=...
>	>	Greater than	O	...>...

---

## GLOSSARY

GLOSSARY ..... G-2

---

## 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**  
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.

---

## GLOSSARY

- **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.
- **ARTICULATED**  
To join together permanently or semi-permanently 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.



---

## GLOSSARY

- **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.).

## 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.

---

## GLOSSARY

- **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.
- **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 'sums' 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.
- **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.

---

## GLOSSARY

- **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.

---

## GLOSSARY

- **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**  
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.

### 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.

---

## GLOSSARY

- **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.
- **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.



---

## GLOSSARY

- **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.
  - **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.

---

## GLOSSARY

- **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.
- **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.

---

## GLOSSARY

- **INVERTER**  
A circuit which switches a positive signal to a negative signal, and vice versa.
- **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.
- **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)

---

## GLOSSARY

- **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 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).

---

## GLOSSARY

### 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.
- **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.

---

## GLOSSARY

- **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.
- **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.
- **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.



---

## GLOSSARY

- **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 reentering 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 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.

---

## GLOSSARY

- **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.
- **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

---

## GLOSSARY

- point in time.
- **SEQUENTIAL ACCESS**  
A method used by many computers whereby data is read in the order in which it is physically stored.
- **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.
- **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.

---

## GLOSSARY

- **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.

---

## GLOSSARY

- TRAP POINT  
see pinch point
- TTL  
Acronym for transistor-transistor logic.
- 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.

---

## INDEX

INDEX .....IN-2

## INDEX

### Symbols

#DEST, 6-9, 6-15  
#HERE Function, 6-14  
1FS (RI/O) Board, A-2  
1FS Board, A-14  
1FS Circuit Board, A-2  
1GA Board, A-2

### A

Abnormal Check Sum, 4-55  
ABORT, 4-26, 4-30  
ABORT Command, 7-5  
Absolute Encoder, 4-49  
Accuracy, 1-3, 2-7  
After Wait Timer Switch, 1-5  
After.Wait.TMR. system switch, A-19  
Allen Bradley, A-14  
Analog Outputs, A-2  
ANGLES, 1-7  
ANSI/RIA, 2-2  
Arithmetic Functions, 6-16  
Arithmetic Operators, 1-18  
Arm ID Board, 4-62  
ARMIOSET, 4-62, 4-64  
Arrays, 1-13, 1-16  
AS Monitor Software, 8-3  
ASC, 6-6  
ASCII Character, 1-6, 6-2  
ASCII Value, 1-13, 6-6  
Auto Servo Timer, 4-54  
AUTOSTART.PC System Switches, A-18  
AUTOSTART.PC, 1-5  
AUTOSTART2.PC, 1-5  
AUTOSTART3.PC, 1-5  
AUX 114 Clamp Specifications, 10-12  
AUX 114-41 Max Abrasion Moving/Fixed[mm], 10-32  
Aux 149, 4-66  
AUX Function 181, A-12  
Axes, 4-51

### B

BASE, 4-31, 4-37  
BASE Function, 6-7  
Base Transformation, 4-38  
BATCHK, 4-51  
Binary Operator, 1-19  
Binary Signal, 6-2  
Bit Pattern, 6-2  
BITS, 4-56, 4-58, 6-3  
BITS Command, 7-14

BNC Connectors, A-14, A-15  
Brake Release Switches, 2-8  
Brakes, 2-5, 2-8, 2-9  
BREAK Command, 7-13

### C

CALL Command, 7-14  
CASE OF Command, 7-16  
Center of Gravity, 4-66, 4-81  
CHANGE, 4-11  
Character Strings, 1-15  
CHECK Key, 7-4  
CHSUM, 4-55  
Clear Check Sum Error, 4-55  
CMOS RAM, 1-3  
COLCAL Command, 4-81  
COLINIT Command, 4-82  
Collision Detection AS Language Commands, 4-73, 4-86  
Collision Detection Error, 4-85  
Collision Detection Function, 4-66  
Collision Detection Troubleshooting, 4-85  
COLMV Command, 4-80  
COLR Command, 4-74, 4-76  
COLRJ Command, 4-77, 4-78  
COLSTATE Command, 4-83  
COLTJ Command, 4-79  
Comment, 7-3  
Compound Transformation, 4-32, 7-10, 7-12  
Compound Transformations, 1-10  
Config.Sys, 8-5  
CONTINUE, 1-2, 4-26, 4-30  
Control Flow Structure, 7-14, 7-18  
Control-Net, A-13  
Control-Net Diagram, A-14  
Control-Net Topologies, A-14  
COPY, 4-12, 4-14, 4-21  
Counter Status, A-9  
CP Switch, 1-3  
CTL+L/CTL+N Key, 4-4  
CUT, 4-12  
Cycle Start, 7-4  
Cycle Stop Switch, 1-3

### D

Data Editing, 1-4  
Data Highway Plus, A-15  
Date, 4-47  
DECOMPOSE, 1-16  
Dedicated Signal, 4-59  
Dedicated Signals  
  Inputs, 1-5  
  Outputs, 1-5



## INDEX

DEFSIG, 4-56, 4-60  
DELETE, 4-8, 4-14  
Delete, 4-19, 4-26  
DELETE/L, 4-14, 4-20  
DELETE/P, 4-14, 4-19  
DELETE/R, 4-14, 4-20  
DELETE/S, 4-14, 4-20  
DEST, 6-9, 6-15  
Detection Level Setting, 4-67  
DEXT Format, A-19  
Dimensions, I-6  
DIRECTORY, 4-15  
DIRECTORY/L, 4-14, 4-16  
DIRECTORY/P, 4-14, 4-16  
DIRECTORY/R, 4-14  
DIRECTORY/S, 4-14, 4-17  
Discrete Hard Wired Systems, A-2  
DISPIO\_01, 1-6  
DISPLAY, 4-13  
DISTANCE, 1-7, 6-5  
Distance Calculation, 6-5  
DLYSIG, 4-57  
DO, 1-2, 4-26, 4-31  
DX, 6-5  
DY, 6-5  
DYLSIG, 4-56  
DZ, 6-5

## E

Editor Commands, 4-5  
Editor Mode, 1-2  
EMERGENCY STOP, 2-5, 2-6, 2-7, 2-9  
EMERGENCY STOP Switch, 3-2, 3-8, 7-5  
ENCCHK\_EMG, 4-51  
ENCCHK\_PON, 4-52  
Encoder, 4-49  
Encoder Abnormality Error, 4-52  
Encoder Deviation, 4-52  
Encoder Value, 4-49  
END Command, 7-14, 7-15, 7-18  
ENV\_DATA, 4-54  
ENV2\_DATA, 4-55, 8-5  
ERESET, 4-47, 4-48  
ERRLOG, 4-43  
ERROR CODES/TROUBLESHOOTING, 10-2  
Error Condition, 4-43  
Error Messages, I-5  
Error Recovery, 10-2  
Error Start PC Switch, 1-5  
ERRSTART.PC System Switch, A-18  
EXECUTE, 1-2, 4-26, 4-28, 7-4  
Execution Status, 6-7

EXIT, 4-11  
Expressions, 1-19

## F

Failure Prediction Function, 4-65  
Failure Prediction Function Setup Procedure, 4-65  
FALSE, 6-2  
FC06N/FS06N/FW06N/FS10C, Work Envelope, 2-12  
FDELETE, 4-22, 4-26  
FDIRECTORY, 4-22, 4-23  
Fiber-Optic, A-15  
Files, 4-23  
Floppy Disk, 4-23  
Flowcharting, 7-6  
FORMAT, 4-22, 4-23  
FRAME, 6-9  
FREE, 4-45  
FS02N/FS03N, Work Envelope, 2-10  
FS06L, Work Envelope, 2-11  
FS10E, Work Envelope, 2-14  
FS10L, Work Envelope, 2-15  
FS10N, Work Envelope, 2-13  
FS20C, Work Envelope, 2-16  
FS20N, Work Envelope, 2-17  
FS30L, Work Envelope, 2-18  
FS30N/FS45C, Work Envelope, 2-19  
FS45N, Work Envelope, 2-20  
Functions, 6-2

## G

GOTO Command, 5-22, 7-13

## H

HERE, 4-31, 4-32  
HERE Function, 6-14  
HOLD, 4-26, 4-30  
HOLD/RUN SWITCH, 3-8  
HOLD/RUN Switch, 3-2, 3-7, 3-8  
HOME, 7-13  
HOME position, 4-40

## I

I/O Signals, I-4, 4-49, 4-58  
I2PG Command, 4-65  
IF THEN Structure, 5-48, 7-18  
Inertia Moment, 4-66, 4-81  
Initialize, 4-47  
Input Signals, 6-3  
INSERT, 4-7  
INT Function, 6-6

---

**INDEX**

Integer Value, 6-6  
Integers, 1-12  
IO, 4-56

**J**

Joint Angle, 4-39

**K**

KCMON, 8-3  
KCWIN, 8-3  
KILL, 4-26, 4-31

**L**

Label, 7-3, 7-13  
Ladder Logic, A-2  
LAST, 4-7  
Last Weld Data, A-11  
LAST WELD DATA Screen, A-11  
LEN, 6-6  
Limit Switches, 2-9  
LIST, 4-17  
List, 4-17  
LIST/L, 4-14, 4-18  
LIST/P, 4-14, 4-17  
LIST/R, 4-14, 4-18  
LIST/S, 4-14, 4-18  
LLIMIT, 4-31, 4-39  
LOAD, 4-22, 4-25  
Load Mass, 4-81  
Location, 4-16  
Location Variables, 1-14  
Locations, 1-8  
LOCK Command, A-19  
Logical Operator, 1-18  
Logical Values, 1-12

**M**

Mainline Program, 7-13  
Maintenance Log, 4-62  
Mass, 4-66  
MC Command, A-18  
Memory, 4-45  
Memory Capacity, I-3  
MESSAGES, 1-5  
MNTLOG, 4-62, 4-64  
MNTREC, 4-62  
MODIFY, 4-9  
Monitor Commands, 1-6, A-18  
MONITOR Mode, 4-3  
Monitor Mode, 1-2  
Motion Instruction, 4-29

Motor Power Lamp, 3-7  
Motor Power Lamps, 3-2  
Motor Power Push Button, 3-7  
MSHA, 2-4  
MSTEP, 1-2, 4-26, 4-29  
Multi Function Panel, I-6, 2-9

**N**

Next, 4-30  
NIOSHA, 2-4  
Node Adapter Chip (NAC), A-12  
Notations And Conventions, 1-6  
NTCHON and NCHOFF commands, A-18  
NULL, 6-9, 6-12  
Null Base, 4-37  
Null Tool, 4-36  
NULL Tool Center Point, 7-12  
Numerical Expression, 1-17  
Numerical Information, 1-12

**O**

OFF, 4-47  
ON, 4-47  
Operator, 1-17  
OPLOG, 4-43  
OSHA, 2-2, 2-4  
Output Signals, 4-56, 6-3  
OX Preout Switch, 1-3

**P**

Parallel I/O, A-2  
PASTE, 4-12  
PASTE Reverse Order, 4-13  
PC Interface, 8-2  
PC Program, 7-2  
PC Program Instructions, 9-2, A-20  
PCABORT Command, 7-5  
PCCONTINUE Command, 7-5  
PCEND Command, 7-5  
PCEXECUTE Command, 7-5  
PCMCIA Slot, 8-5  
PCSTATUS Command, 9-2  
Play-back Mode, 1-2  
PLC, A-13  
PLC(NAC), A-12  
POINT, 4-31, 4-33  
POINT/OAT, 4-31, 4-35  
POINT/X, 4-31, 4-35  
POINT/Y, 4-31, 4-35  
POINT/Z, 4-31, 4-35  
Power On/Off Procedures, 3-2  
Power Requirements, I-6

## INDEX

- Precision Location, 1-8, 4-32, 6-14
- Precision location, 1-13, 1-14
- Precision Locations, 1-8
- PREFETCH.SIGINS, 1-4
- PRIME, 4-26, 4-27
- PRIME Command, 7-4
- PRINT, 4-6
- PRIORITY Function, A-19
- Process Control Programs, A-18
- Program and Data Control Commands, 4-14
- PROGRAM CHANGE, 4-53
- Program Control, 4-26
- Programmable Logic Controllers (PLCs), 1-2
- PULSE, 4-56
- Pulse Command, 7-18
- PWM, A-18
  
- Q**
- QTOOL Switch, 1-4, 7-10
  
- R**
- Range of Threshold, 4-67
- Real Numbers, 1-12
- Real Values, 1-12
- Real Variable, 4-16
- Real Variables, 1-14
- REC\_ACCEPT Switch, 4-53
- RECORD, 1-20
- Relational Operator, 1-18
- Remote Input/Output, A-2
- RENAME, 4-14, 4-20
- Repeat Once Switch, 1-4
- REPLACE, 4-10
- RESET, 4-59
- RETURN Command, 5-24, 7-20
- RG-6 Coaxial Cable, A-15
- RI/O Monitor, A-7
- RI/O Signal Status, A-8
- RISC CPU, 1-3
- Robot Control Program, 7-2
- Robot Controller Design Specifications, 1-3
- Robot Location, 4-41
- RPS, 1-5
- RS-485, A-2
- RS232C, 8-2
- RSLinx, A-14
- RSView, A-14
  
- S**
- S-LOGIC Monitor Screen, A-10
- S-LOGIC Status Screen, A-12
- Safety Features, 2-9
- SAVE, 4-22, 4-23
- SAVE/ARC, 4-22
- SAVE/L, 4-22, 4-24
- SAVE/P, 4-22, 4-24
- SAVE/R, 4-22, 4-24
- SAVE/S, 4-22, 4-25
- SAVE/SYS, 4-22
- Scientific Notations, 1-12
- SCREEN, 1-5
- Serial Interface, 8-2
- Servo Motor Power-on Procedures, 3-7
- SET2HOME, 4-40
- SETCLAMP, 4-47
- SETCOLTHID, 4-82
- SETHOME, 4-31, 4-40
- Setting Threshold, 4-68
- Setting Tool Weight Data, 4-66
- SHIFT, 6-9, 6-13
- SIGNAL, 4-56
- SIGNAL Command, 4-56, 7-13, 7-14
- SIGNAL NUMBER, 1-7
- Signal Number, 4-56
- Signal Numbers, 4-59
- Signal Timing, 1-3
- SLOGIC, A-2
- SLOGIC Program Name, A-2
- SLOGIC Programming, A-2
- SLOGIC Programs, 7-2
- SLOW\_REPEAT Switch, 4-52
- Small Teach Pendant, 2-6, 2-7, 2-9
- Software Features, 1-4
- Software Limit, 4-39
- Special Features, 1-5
- SPEED, 4-26
- Speed, 1-4
- SPG Program, A-2
- SRAM, 4-51
- SRAM PC Cards, 8-5
- STATUS, 4-44
- Status, 4-43
- STEP, 1-2, 4-6, 4-26, 4-28
- STEP Command, 4-28, 7-4
- Step Number, 7-3
- Step Once Switch, 1-5
- Stopping Robot Motion, 7-5
- Stopping the Robot, 3-8
- String Variable, 4-17
- String Variables, 1-15
- SWAIT Command, 5-27, 7-18
- SWITCH, 4-47, 4-48
- Switch Settings, 4-48
- SYSINIT, 4-47
- System Switch, 4-48

---

**INDEX**

System Switches, 1-2

**T**

Taps, A-15  
TEACH, 1-20, 4-31, 4-33  
Teach Lock Switch, 3-7  
Teach Pendant, I-6  
TEACH/REPEAT Switch, 3-7, 3-8  
Terminal Control, 4-4  
TIME, 4-47  
Time Delay, 10-67  
TIMER, 6-2, 6-4  
Timer and Counter Status Screens, A-9  
TIMER Command, 6-4, 7-18  
TIMER Status, A-9  
TOOL, 4-31, 4-36  
Tool, 4-36  
Tool Changer, 10-72  
Tool Dimensions, 1-4, 7-10  
TOOL Function, 6-7  
TOOL NULL Command, 6-12, 7-10  
TRANS, 6-9, 6-12  
Transformation, 1-8  
Transformation Location, 1-13, 1-14, 4-32  
Trap Points, 2-4  
Traverse Axis, A-17  
Trigger Keys, 2-6, 2-7  
Troubleshooting Flowcharts, 10-84  
TRUE, 6-2

**U**

UB150, Work Envelope, 2-21  
ULIMIT, 4-31, 4-39  
Universal Remote I/O, A-15  
UT100/150/200, Work Envelope, 2-22  
UX100/120/150, Work Envelope, 2-24  
UX200, Work Envelope, 2-25  
UX300, Work Envelope, 2-26  
UX70, Work Envelope, 2-23  
UZ100/120/150, Work Envelope, 2-27

**V**

VALUE Statements, 7-14  
Values, 1-6  
Variable Names, 1-13  
Variable Types, 1-13

**W**

WAIT Command, 5-26, 7-14  
WEIGHT Command, 4-81

Welding Clamp Subroutine, 7-18  
WHERE, 4-31, 4-41  
WHERE 1, 4-41  
WHERE 2, 4-41  
WHERE 3, 4-41  
WHERE 4, 4-41  
WHERE 5, 4-41  
WHERE 6, 4-41  
Windows™, 8-5  
Windows NT/95, A-14  
Work Cell, 2-2, 2-4  
Work Envelope, 2-4, 2-5, 2-6, 2-7, 2-8  
Work Envelope Drawings, 2-10

**X**

XD, 4-12  
XFER, 4-14, 4-21  
XP, 4-12  
XQ, 4-13  
XS, 4-13  
XY, 4-12

**Z**

ZD130, Work Envelope, 2-28  
Zeroing Data, 4-49  
ZSIGSPEC, 4-47, 4-49  
ZX130L, Work Envelope, 2-29  
ZX130U, Work Envelope, 2-30  
ZX165U, Work Envelope, 2-31  
ZX200S, Work Envelope, 2-32  
ZX200U, Work Envelope, 2-33  
ZX300S, Work Envelope, 2-34  
ZZERO, 4-47, 4-49, 4-50