

# EPSON

Reference Manual  
for SRC-3\*\*

*SPEL III Ver. 6.2*

Rev. 4

EM99NS725F

CE

Reference Manual for SRC-3\*\*

# *SPEL III Ver. 6.2*

**EPSON**

# *Reference Manual*

## *SPEL III Ver. 6.2*

Rev. 4

.....

## WARRANTY

The Robot and its optional parts are shipped to our customers only after being subjected to the strictest quality controls, tests and inspections to certify its compliance with our high performance standards.

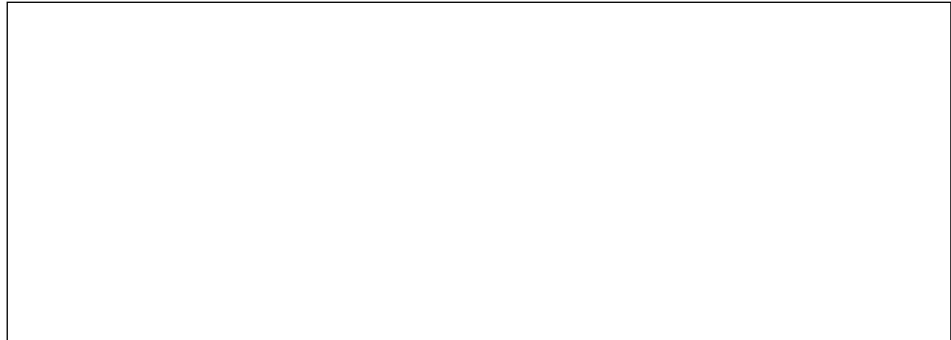
Product's malfunction(s) resulting from normal handling operation will be repaired free of charge up to 12 months after delivery.

However, customers will be charged for repairs in the following cases:

1. Damage or malfunction caused by improper use which is not described in the manual, or careless use.
2. Malfunctions caused by customers' unauthorized disassembly.
3. Damage due to improper adjustments or unauthorized repair attempts.
4. Damage caused by natural disasters such as earthquake, flood, etc.

## SERVICE CENTER

Contact the following service center for robot repairs, inspections or adjustments. Please have the model name, M. CODE, software version and a description of the problem ready when you call.



## MANUFACTURER

SEIKO EPSON CORPORATION  
Robots & FA Systems Department

Okaya Plant No. 2  
1-16-15, Daiei-cho  
Okaya-shi, Nagano-ken, 394  
Japan

TEL: 81-266-23-0020 (switchboard)  
81-266-24-2004 (direct)  
FAX: 81-266-24-2017

## NOTICE

No part of this manual may be copied or reproduced without authorization.

The content of this manual is subject to change without notice.

We ask that you please notify us if you should find any errors in this manual or if you have any comments regarding its content.

## INTRODUCTION

This reference manual contains all of the information necessary to assist you in correctly using robot programming language SPEL III. To optimize the performance of this precision assembly robot, we recommend that prior to operation you carefully read over both this manual, and the accompanying user's manual.

This manual contains the following sections:

IMPORTANT (Read this section first)

HOW THIS MANUAL IS ORGANIZED

SUMMARY OF COMMANDS, STATEMENTS, AND FUNCTIONS

EXPLANATIONS OF COMMANDS, STATEMENTS, AND FUNCTIONS  
(in alphabetical order)

ERROR CODES

### NOTE

This reference manual is for SPEL III Version 6.2. The version corresponds to the model of robot controller.

SRC-300	:	Version 6.2
SRC-310	:	Version 5.3
SRC-310A	:	Version 6.2
SRC-320	:	Version 6.2
SRC-320J	:	Version 6.2

## IMPORTANT

NOTE  
F

In upgrading SPEL III from version 3, controller significant changes that affect programming and operation have been made. If you are using version 3, prior to programming, be sure to take careful note of the following important changes:

### Motor Engagement

Motors are not engaged automatically, when the controller is turned on or reset. To engage the motors:

- Execute MOTOR ON command from Programming Unit, or
- Press MOTOR ON switch on Operating Unit, or
- Input motor power on signal to REMOTE3, or
- Execute MOTOR ON in the program.

### Calibration

SPEL III is used to both INC robot which is equipped with incremental encoder and ABS robot which is equipped with absolute encoder.

In case of INC robot, calibration by MCAL command is necessary when power is turned on.

In case of ABS robot, calibration is not necessary.

### HOME Command

The function of HOME command of this version is to move the robot to a HOME (standby) position specified with HOMESSET command. You can set a HOME position anywhere to suit a specific application. Note that at delivery, a HOME position is not defined. Executing HOME command without first defining a HOME position will thus cause an error message to be displayed.

### Motion Speed in the TEACH Mode

To assure safe operation, the robot moves at low speed in the TEACH mode, regardless of SPEED command settings.

For further details, refer to POWER or LP, and SPEED command.

HOW THIS MANUAL IS ORGANIZED

This manual describes each SPEL III command as follows:

Command Description Organization

<p><b>COMMAND, STATEMENT, or FUNCTION</b></p> <hr/> <p><b>FUNCTION</b></p> <p><b>FORMAT</b></p> <p><b>DESCRIPTION</b></p> <p><b>RELATED COMMANDS</b></p> <p><b>EXAMPLE</b></p>	<p><b>SYMBOLS</b></p>
--	-----------------------

FORMAT

This manual explains the format of each command as follows:

<p>Characters not Enclosed by [ ]</p>	<p>These characters should be entered as they are shown. For example, MOTOR ON, #endif, AOPEN "[filename]" AS #[file number]</p>
<p>[ ]</p>	<p>Type of data. Data items are enclosed by [ ], For example, ON [output bit number]</p>
<p>   </p>	<p>Any of the multiple data values enclosed by     may be selected. For example, LOAD { PRG                     PNT </p>
<p>{ }</p>	<p>Contents enclosed by { } may be omitted. For explanations of omissions, refer to DESCRIPTION comments. For example, LOAD { PRG                     PNT </p> <p>In this case, any of the following three may be described, LOAD LOAD PRG LOAD PNT</p>
<p>{ }n</p>	<p>When n is replaced by an integer, the data enclosed by { } can be written out n times. When n is not replaced by an integer, the enclosed data can be written out indefinitely.</p>
<p>~</p>	<p>Indicates continuation to the next line, or from the preceding line.</p>

## SYMBOLS

This manual uses the following symbols:



May be used as a command



May be used as a statement



Function



Dedicated for INC robot equipped with incremental encoder.

When executing for ABS robot equipped with absolute encoder Error 123 will occur.



## SUMMARY OF COMMANDS, STATEMENTS, AND FUNCTIONS

The following is a summary of SPEL III commands, statements, and functions. Pay particular attention to the commands, statements, and functions that are preceded by one or both of the following symbols:

: All first time our robot users should read this explanation

: Commands, statements, or functions that are either new, or have been changed

### Commands, Statements, and Functions Related to System Management

POWER, LP	Specification/Cancellation of low power mode in TEACH mode
CONSOLE, CNSOL	Specifies console to be used in AUTO mode
RESET	Resets controller
SETENV	Specifies, cancels and displays environment variable
FREE	Displays available memory
PRGsize	Changes and displays program area size
PNTsize, Psize	Specifies and displays number of usable position data
LIBsize, SIZE	Specifies and displays number of usable backup variables and available memory
SYSINIT	Initializes main memory
TON	Displays task status
TOFF	Stops displaying program line numbers
TSTAT	Displays task status
STAT( )	Returns status of controller or other controller connected through RS-232C
VER	Displays system management data
VERINIT	Initializes system management data
MKVER	Saves the backup data of various setting and all the data in main memory onto file memory
SETVER	Restores the data which is filed by MKVER command to corresponding memory area
DATE	Specifies and displays current date
DATE\$(0)	Returns current date
TIME	Specifies and displays current time

SUMMARY OF COMMANDS, STATEMENTS, AND FUNCTIONS

TIME\$(0)	Returns current time
HOUR	Displays controller accumulated operating time
TIME( )	Returns controller accumulated operating time
ERRHIST	Displays error history
LINEHIST	Displays line number history
CALIB	Replaces current arm orientation pulse value with current CALPLS values
CALPLS	Specifies and displays position and orientation pulse used for calibration
HOFS	Specifies and displays offset pulse used for software zero point correction
!	Executes MS-DOS Command

Commands, Statements, and Functions Related to Robot Control

MCAL	Executes machine calibration
MCORDR	Specifies and displays the moving axis order in machine calibration
MCORG	Calculates the parameter for machine calibration
MCOFS	Specifies and displays the parameter for machine calibration
HTEST	Displays HTEST values
MOTOR	Turns motor power on and off
SFREE	Cuts power to motor
SLOCK	Reengages axis (from "servo free" condition)
JUMP	Executes arch motion jump
ARCH	Specifies and displays JUMP arch parameters
LIMZ	Specifies and displays axis #3 height for JUMP
SENSE	Specifies and displays input condition that, if satisfied, complete the JUMP in progress by stopping robot above target position
JS(0)	Returns whether or not SENSE condition has been satisfied after JUMP SENSE has completed
GO	Executes simultaneous four axis PTP motion
PASS	Executes PTP motion, passing near specified points
PULSE	Executes simultaneous four axis PTP motion
TGO	Executes PTP relative motion, in the selected tool coordinate system
TMOVE	Executes linear interpolation relative motion, in the selected tool coordinate system
TILL	Specifies and displays input condition that, if satisfied, complete the JUMP, GO, or MOVE in progress by decelerating and stopping robot at an intermediate travel position
! ... !	Processes input/output statements in parallel to executing motion commands

SPEED	Specifies and displays speed for PTP motion
TSPEED	Specifies and displays maximum speed for PTP motion in TEACH mode
ACCEL	Specifies and displays acceleration for PTP motion
WEIGHT	Specifies and displays parameters for calculating PTP motion maximum acceleration
ARC	Executes arc interpolation motion in a horizontal plane
CARC	Executes arc interpolation motion in a horizontal plane, without decelerating, specified final position
MOVE	Executes simultaneous four axis linear interpolation motion, with deceleration and stop at specified position
CMOVE	Executes simultaneous four axis linear interpolation motion, without decelerating, through specified position
CURVE	Creates a file for free curve CP control motion
CVMOVE	Executes free curve CP control motion file generated by CURVE
SPEEDS	Specifies and displays speed for CP motion
ACCELS	Specifies and displays acceleration for CP motion
TSPEEDS	Specifies and displays maximum speed for CP motion in TEACH mode
HOME	Executes motion to home (standby) position
HOMESET	Specifies and displays home position and orientation pulse
HORDR	Specifies and displays HOME axis motion order
PALET	Defines and displays pallets
PALETn( )	Return position on specified pallet corresponding to specified division number
SEL	Selects and displays step jog feed travel settings
SET	Specifies step jog feed travels
FINE	Sets the allowable range for positioning completion checkout
QP	Switches quick pause mode on or off and displays current mode status
JRANGE	Defines permissible working range of specified axis in pulses
RANGE	Defines permissible working range of each axis in pulses, and displays current permissible ranges
XYLIM	Specifies and displays allowable X and Y axis coordinate motion range
CX(P )	} Returns X, Y, Z, or U axis coordinate value of specified point
CY(P )	
CZ(P )	
CU(P )	
PLS( )	Returns pulse value of specified axis
AGL( )	Returns joint angle for selected rotational axis, or position for selected linear axis
SELRB	Selects positioning device

Commands and Statements Related to Editing

NEW	Deletes source program in source program area
CLEAR	Clears position data
LIST	Displays source program
DELETE, DELET	Deletes program line(s)
RENUM	Renumbers program lines
PLIST, PLI	Displays position data
PDEL	Deletes specified position data
Pn=Position Specification	Defines point
EDIT	Switches to edit mode
FIND	Searches for string

Commands, Statements, and Functions Related to Input/Output

ON	Switches specified output bit on
OFF	Switches output bit off
OPORT( )	Returns output bit status
SW( )	Returns input bit status
IN( )	Returns input status of input port
INBCD( )	Returns input status of input port as BCD
INBIN( )	Returns the value of input port specified by starting port number and number of ports
INBIT( )	Returns the bit number which is turned on among input ports specified by starting port number and number of ports
OUT	Sends data to output port
OPBCD	Outputs BCD data to output port
ON \$	Switches robot memory I/O on
OFF \$	Switches robot memory I/O off
SW(\$ )	Returns memory I/O bit status
IN(\$)	Returns input status of memory I/O port
OUT \$	Sends data to memory I/O port
ZEROFLG(0)	Returns value of memory I/O previous to it last being switched on or off
WAIT	Specifies timer interval, stops program execution until specified condition is satisfied
TMOUT	Specifies time out interval for WAIT
TW(0)	Returns status of WAIT condition and WAIT time interval

INPUT	Inputs data from console keyboard
PRINT	Outputs data to console display
LINE INPUT	Stores one line of data from console to string variable
CONFIG, CNFG	Sets configuration parameters for RS-232C communication port
INPUT #	Inputs data from communication port
PRINT #	Outputs data to communication port
LINE INPUT #	Stores one line of data from communication port to string variable
LOF( )	Returns number of data lines stored in RS-232C buffer
OPU PRINT	Outputs characters to operating unit
DSW( )	Returns remote input status as a decimal
PEEK( )	Reads data from I/O channel
POKE	Writes data to I/O channel

#### Commands and Statements Related to Coordinate Changes

ARM	Selects and displays arm number
ARMSET	Specifies and displays auxiliary arm
TOOL	Selects and displays tool coordinate system
TLSET	Defines and displays tool coordinate system
LOCAL	Defines and displays local coordinate system
LOCAL0	Defines robot's basic coordinate system Local0
BASE	Defines and displays local coordinate system
BASE 0	Defines robot's basic coordinate system Local0
BASE( )	Returns value of local coordinate system

#### Statements and Functions Related to Program Control

FUNCTION...FEND	Defines function
END	Terminates program execution
FOR...NEXT	Executes series of statements specified number of times
GOSUB...RETURN	Branches to, executes, and returns from subroutine
GOTO	Branches to specified line number
ONGOTO	Branches to specified line number or label depending on the value of the variable
CALL	Calls function procedure as subroutine
IF..THEN..ELSE..ENDIF	Conditional statement execution

SUMMARY OF COMMANDS, STATEMENTS, AND FUNCTIONS

SELECT...SEND	Specifies branching formula and corresponding branch instruction sequences
WHILE...WEND	Executes specified statements while specified condition is satisfied
TRAP	Defines the interrupt process
ONERR...RETURN	Defines error processing
ERA(0)	Returns axis number on which error occurred
ERL(0)	Returns line number in which error occurred
ECLR	Clears error status
ERR(0)	Returns error number
ERT(0)	Returns task number at which error occurred
ERRMSG\$( )	Returns error message corresponding to specified error number

Commands and Statements Related to Program Execution

COMPILE, COM	Translates source program into an executable program
XQT	Executes program
PAUSE	Temporarily stops program execution
RUN	Compiles and executes source program
HTASK	Specifies tasks to be temporarily stopped by PAUSE command or PAUSE input
HALT	Temporarily stops execution of task in progress
QUIT	Stops tasks that are currently being executed, or have been temporarily stopped
RESUME	Resumes execution of tasks temporarily stopped by HALT
MYTASK(0)	Returns number of mytask
CHAIN	Loads into main memory and executes object program and position data files
PRGNO	Determines whether to apply up-down count or binary coding to program number selection through remote connector (REMOTE 2)

Pseudo Statements

#define	Defines identifier to be replaced by specified replacement string
#ifdef...#endif	Conditional compiling
#ifndef...#endif	Conditional compiling
#include	Pulls in specified file and compiles

Commands and Statements Related to File Management

FORMAT, FRMT	Formats file memory
FILES	Displays name and size of files in file memory

WIDTH	Specifies number of characters per line and lines per screen displayed on programing unit CRT
DIR	Displays contents of directory
CHDIR, CD	Changes and displays current directory
MKDIR, MD	Creates sub directory
RMDIR, RD	Removes (deletes) sub directory
RENDIR	Changes directory name
PATH	Specifies, cancels and displays path for executing batch file
DLOAD, DLO	Loads specified files into main memory
DSAVE, DSA	Saves main memory source program and position data files in file memory
MERGE	Transfers programing unit data into the main memory and merges them with main memory data
DMERGE	Loads specified file(s) into main memory and merges them with source program or position data
TYPE	Displays contents of file
KILL	Deletes files
DEL, ERASE	Deletes files
COPY	Copies files
RENAME,REN,NAME	Changes file name
LINK	Links two or more object files
AOPEN...CLOSE	Opens file for appending data
ROPEN...CLOSE	Opens file for readout
WOPEN...CLOSE	Opens file for writing to it
VLOAD	Loads variable data
VSAVE	Saves variable data

### Commands and Statements Related to Variables

BYTE	Defines 1-byte integer type variables
INTEGER	Defines 2-byte integer type variables
LONG	Defines 4-byte integer type variables
REAL	Defines 4-byte REAL type variables
DOUBLE	Defines 8-byte REAL type variables
STRING	Defines string variables
SYS	Declares backup variables

SUMMARY OF COMMANDS, STATEMENTS, AND FUNCTIONS



LIBRARY	Displays backup variables in memory
CLRLIB	Clears backup variables
ENTRY	Declares global variables
EXTERN	Declares reference to global variables
VARIABLE	Displays variables

Functions Related to Numeric Values

OPORT( )	Returns output bit status
SW( )	Returns input bit status
IN( )	Returns input status of input port
INBCD( )	Returns input status of input port as BCD
SW(\$ )	Returns memory I/O bit status
IN(\$ )	Returns input status of memory I/O port
ZEROFLG(0)	Returns value of memory I/O previous to it last being switched on or off
CTR( )	Counter
CTRESET	Counter reset
TMR( )	Timer function
TMRESET	Resets timer
TIME( )	Returns controller accumulated operating time
JS(0)	Returns whether or not SENSE condition has been satisfied after JUMP SENSE has completed
PALETn( )	Return position on specified pallet corresponding to specified division number } Returns X, Y, Z, or U axis coordinate value of specified point
CX(P )	
CY(P )	
CZ(P )	
CU(P )	
PLS( )	Returns pulse value of specified axis
AGL( )	Returns joint angle for selected rotational axis, or position for selected linear axis
ERA(0)	Returns axis number on which error occurred
ERL(0)	Returns line number in which error occurred
ERR(0)	Returns error number
ERT(0)	Returns task number at which error occurred



TW(0)	Returns status of WAIT condition and WAIT time interval
MYTASK(0)	Returns number of mytask
LOF( )	Returns number of data lines stored in RS-232C buffer
DSW( )	Returns remote input status as a decimal
STAT( )	Returns status of controller or other controller connected through RS-232C
SIN( )	Returns sine of specified angle
COS( )	Returns cosine of specified angle
TAN( )	Returns tangent of specified angle
ATAN( )	Returns arctangent of specified value
ATAN2( )	Returns arctangent of y/x
SQR( )	Returns square root
ABS( )	Returns absolute value
SGN( )	Returns sign of specified numeric value
INT( )	Returns largest integer that is less than or equal to specified value
NOT( )	Returns inverted value of integer bit
LSHIFT( )	Shifts numeric value data to left
RSHIFT( )	Shifts numeric value data to right

### Statements and Functions Related to Strings

ASC( )	Returns ASCII code (numeric value) for first character of specified string
CHR\$( )	Returns character that corresponds to specified ASCII code
LEFT\$( )	Returns the leftmost characters of specified string
MID\$( )	Returns specified number of characters of specified string beginning from specified start position
RIGHT\$( )	Returns the rightmost characters of specified string
LEN( )	Returns number of characters of specified string
SPACE\$( )	Returns a string consisting of specified number of spaces
STR\$( )	Returns specified numeric value as a numeric string
VAL( )	Returns numeric value of specified numeric string
ERRMSG\$( )	Returns error message corresponding to specified error number

### Commands Related to Operating Unit

OPU PRINT	Outputs characters to operating unit
CHARSIZE	Specifies character size (on operating unit)

SUMMARY OF COMMANDS, STATEMENTS, AND FUNCTIONS

CLS	Erases characters (on operating unit)
NORMAL	Specifies normal display mode for displayed characters
REVERSE	Specifies reverse display mode for displayed characters
CURSOR	Specifies cursor display on or off (on operating unit)
OPUNIT	Operation mode selection of operating unit
DSW( )	Returns remote input status as a decimal

New or Altered commands in this version

Please read this page if you have used SPEL III Ver. 4.2/5.2.

1. New commands

BASE( )	Returns value of local coordinate system.
DATE\$(0)	Returns current date.
TIME\$(0)	Returns current time.
INBIN( )	Returns the value of input port specified by starting port number and number of ports
INBIT( )	Returns the bit number which is turned on among input ports specified by starting port number and number of ports
ONGOTO	Branches to specified line number or label depending on the value of the variable
POWER	Same function with LP ON/OFF. Command changed to POWER LOW/HIGH, made the motor power mode setting easy to understand. LP ON/OFF is also available.

2. Function added/altered commands

ACCEL	} The limited value displayed when a motion command is executed in low power state changed to be easy to understand.
ACCELS	
SPEED	
SPEEDS	
STAT( )	Contents of bit 19, 22, and 23 of address 0, and bit 5 and 6 of address 1 are added.
TRAP	Interrupt process by safe guard open/close is added.

!

&gt;

!

**FUNCTION** Executes MS-DOS command

**FORMAT** ! [MS-DOS command]

**DESCRIPTION** Executes MS-DOS command while running SPEL Editor.

After MS-DOS command is executed, displays the following:

Push any key

After pressing any key, ! command completes by returning a prompt.

If COMMAND.COM processor is necessary to execute the specified MS-DOS command, but is not in the PC's current drive, an error occurs.



Parallel Processing

**FUNCTION** Processes input/output statements in parallel to executing motion commands

**FORMAT** [motion command] ! [parallel processing statement] !

**DESCRIPTION** Begins processing of statements bracketed by ! ... ! simultaneous to beginning motion command travel. The following may be used as parallel processing statements, either as single statement or as multiple statements.

Dn	Used to specify parallel processing execution timing, n is a real number from 0 to 100. Statements subsequent to Dn begin execution when n % of motion command travel has been completed. For JUMP, % travel does not include axis #3 (vertical) motion. Dn may appear a maximum of 6 times in a ! ... ! statement.
ON/OFF n	Turns output bit number n on or off
ON/OFF \$n	Turns memory I/O bit number n on or off
OUT p,d	Outputs output data d to output port p
OUT \$p,d	Outputs output data d to memory I/O port p
WAIT t	Delays execution of next statement t seconds
WAIT SW(n)=i	Delays execution of next statement until input bit n is i (1=on, 0=off) condition
WAIT SW(\$n)=i	Delays execution of next statement until memory I/O bit n is i (1=on, 0=off) condition
PRINT/ INPUT	Outputs data to (and inputs data from) console
PRINT # /INPUT #	Outputs data to (and inputs data from) communication port

In parallel processing statements, specifying time interval for ON or OFF is not allowed. For example, attempting to compile JUMP !P1 ON 5 , 3 ! would result in Error 14.



If after completing motion command travel all parallel processing statement execution has not been completed, subsequent program execution is delayed until all parallel processing statements execution has been completed.

Similarly, if TILL is used to stop at an intermediate travel position, subsequent program execution is delayed until all parallel processing statements execution has been completed.

Even for JUMP, the statement (other than Dn) at the beginning of ! ... ! is executed simultaneous to the start of (rising) motion.

To execute statements after third axis rising motion has completed, include D0 (zero) at the beginning of the statement.

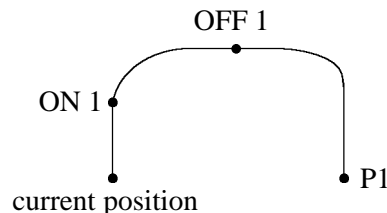
**RELATED  
COMMANDS**

JUMP, GO, MOVE, CMOVE, ARC, CARC, PULSE

**EXAMPLE**

```
JUMP P1 C0 LIMZ-45 SENSE !DO;ON 1;D50;OFF 1!
```

'Parallel to JUMP execution, turn output bit 1 on simultaneous to beginning first, second, and fourth axis JUMP travel. Turn output bit 1 off when 50 % of JUMP travel has been completed.



```
MOVE P1 !D10;ON 5;WAIT 0.5;OFF 5!
```

'Parallel to MOVE execution, turn output bit 5 on when 10 % of MOVE travel has been completed 0.5 seconds later, turn output bit 5 off.

# #define

S

<b>FUNCTION</b>	Defines identifier to be replaced by specified replacement string
<b>FORMAT</b>	<code>#define [identifier{([parameter])}] {[replacement string]}</code>
<b>DESCRIPTION</b>	<p>Searches program for specified identifier, each time it is found replaces it with replacement string, and compiles.</p> <p>At least one space must be included between the identifier and replacement string.</p> <p>The defined identifier can be used for conditional compiling with <code>#ifdef</code> or <code>#ifndef</code> commands.</p> <p>Parameter must be closed by ( ) (parenthesis). Each parameter must be punctuated by " , ", and up to 8 parameters can be specified. If parameter is specified, this command can be used as macro.</p> <p>The maximum size for identifier and replacement string which can be defined by <code>#define</code> instruction is 7 K byte, which is equivalent to 7,169 characters. When the identifier and replacement string defined by <code>#define</code> instruction exceeds 7,169 characters, error 22 occurs.</p> <p>Identifier</p> <ul style="list-style-type: none"><li>* First character must be alpha, following characters may be alphanumeric or underscore ( _ ).</li><li>* Spaces or tabs are not allowed.</li><li>* Large case and small case alpha characters may be used, they are discriminated as large or small case. Therefore the following are discriminated as different identifiers: XYZ, Xyz, xYZ</li><li>* Identifiers are also discriminated by number of characters. For example "ABC" (5 characters) is discriminated as different from ABC (3 characters) and is not replaced.</li></ul> <p>Replacement String</p> <ul style="list-style-type: none"><li>* Spaces or tabs are allowed.</li><li>* Arguments are not allowed.</li><li>* Identifiers used with other <code>#define</code> statements cannot be used as replacement strings.</li><li>* If comment symbol ( ' ) is included, characters following ( ' ) will be treated as a comment, and will not be included in the replacement string.</li><li>* Replacement string may be omitted. In this case the specified identifier is replaced by nothing, this actually deletes the identifier from the program.</li></ul>

**RELATED  
COMMANDS**

#ifdef...#endif, #ifndef...#endif

**EXAMPLE**

&lt; Example 1 &gt;

Without Pseudo Command #define	With Pseudo Command #define
>10 FUNCTION MAIN	>10 FUNCTION MAIN
>20 INTEGER O_CYLF;O_CYLF=0	>20 #define O_CYLF 0
>30 INTEGER O_CYLR;O_CYLR=1	>30 #define O_CYLR 1
>40 INTEGER I_CYLF;I_CYLF=0	>40 #define I_CYLF 0
>50 INTEGER I_CYLR;I_CYLR=1	>50 #define I_CYLR 1
>60 INTEGER RDY; RDY=0	>60 #define RDY 0
⋮	⋮

&lt; Example 2 &gt;

```

100 #define EOF -1
110 PRINT EOF           '-1 is displayed
120 PRINT "EOF"        'String of EOF is displayed
130 PRINT EOF1         'EOF1 is not identical with EOF.
                        In this case, the value of variable EOF1 is displayed.

```

&lt; Example 3 &gt;

The maximum number of characters per program line is limited to 79. However, by using #define to replace identifiers with replacement strings, the maximum number of apparent characters per program line can be increased to 255.

```

100 #define TEST 12345678901234567890123456789
110 PRINT TEST, TEST, TEST, TEST

```

After replacement, the actual number of displayed characters will exceed 80.

&lt; Example 4 &gt;

Use command to display characters on OPU-300 as macro.

```

100 #define CLS PRINT #24, CHR$( &H1B) + "E? "
110 #define LOCATE(a,b) CHR$( &H1B) + "Y" + CHR$( 31+a) + CHR$( 31+b)
120 #define BLKCLS(a,b) CHR$( &H1B) + "E" + CHR$( 31+a) + CHR$( 31+b)
130 CLS                               'Clears screen
140 PRINT #24, LOCATE(1,3) + BLKCLS(32,2) 'Clears 2 lines from the 3rd line

```

#

< NOTE >

SPEL III 's compiler checks customer's program twice.

Pseudo command, variables and labels are processed on pass 1, and each line is compiled on pass 2. Therefore, usage will be limited on the same pass.

It may cause error to use #define for variable registration and label 1, or to use identifier which is registered by #define for filename of #include.

Error examples are listed below in LIST 1 to LIST 4:

LIST 1

```
10 FUNCTION MAIN
20 #define CHAR STRING
30 CHAR A$
40 PRINT A$
50 FEND
>COM
#ERROR 2 at 40
>
```

LIST 2

```
10 FUNCTION MAIN
20 #define LBL LABEL
30 GOTO LBL
40 LBL:
50 FEND
>COM
#ERROR 8 at 30
>
```

LIST 3

```
10 FUNCTION MAIN
20 #define LMAX 10
30 REAL LBUF(LMAX)
40 FEND
>COM
#ERROR 14 at 30
>
```

LIST 4

```
10 FUNCTION MAIN
20 #define ABC "TEST.PRG"
30 #include ABC
40 FEND
>COM
#ERROR 2 at 30
>
```

However, if the identifier which has already registered is used in replacement string, there is no problem. This is called nesting of #define.

LIST 5

```
10 FUNCTION MAIN
20 #define ABC 10
30 #define DEF ABC
40 PRINT DEF
50 FEND
>RUN
10
>
```



# #ifdef...#endif

## #ifndef...#endif

S

#

if define  
if not define

**FUNCTION** Conditional compiling

**FORMAT** `#ifdef [identifier]`  
:  
`#endif`

`#ifndef [identifier]`  
:  
`#endif`

**DESCRIPTION** Used together with pseudo commands `#endif`, `#ifdef` and `#ifndef` executes conditional compiling.

`#ifdef` first checks if the specified identifier is currently defined by `#define`. If defined, statements between `#ifdef` and `#endif` are compiled. If not defined, statements between `#ifdef` and `#endif` are skipped without compiling.

`#ifndef` checks the opposite condition of `#ifdef`.

`#ifndef` first checks if the specified identifier is currently defined by `#define`. If not defined, statements between `#ifndef` and `#endif` are compiled. If defined, statements between `#ifndef` and `#endif` are skipped without compiling.

**RELATED COMMANDS** `#define`, `COMPILE`

### EXAMPLE

```

:
>100 INPUT #20,A$
>110 #ifdef DEBUG
>120 PRINT A$
>130 #endif
:

```

In the above example, printing is either executed or not basic on the presence or absence of the definition of `#define DEBUG`.

Also, the example given under the `#include` (command) explanation, by using `#ifdef` and `#ifndef`, can be changed as shown below:

VAL.PRG

```
>300 #ifdef VALUE
>310     ENTRY REAL WORK(100)
>320     ENTRY DOUBLE A_SIN,A_COS
>330 #endif
>340 #ifndef VALUE
>350     EXTERN REAL WORK(100)
>360     EXTERN DOUBLE A_SIN,A_COS
>370 #endif
```

FILE 1

```
>10 FUNCTION MAIN
>20 #define VALUE
>30 #include "VAL.PRG"
>40 EXTERN FUNCTION INIT
>50 XQT !2, INIT
>60 FEND
```

Note that FILE 2 and FILE 3, which use the #include pseudo statement, are same as those in #include explanation.

In FILE 1, VALUE is defined with the #define statement. By not defining VALUE in FILE 2 and FILE 3, it is possible to choose which statement, #ifdef VALUE or #ifndef VALUE, is to be executed in each file. The advantage of this method is that if variables are to be added or deleted, they need be changed in only one place, in the VAL.PRG program.

If using #define statement to define VALUE, because #ifdef...#endif is compiled, only that program segment is included in FILE 1. Also, if VALUE is not defined, #ifndef...#endif is compiled, that segment is included in FILE 2 and FILE 3.

# #include

S

#

<b>FUNCTION</b>	Pulls in specified file and compiles
<b>FORMAT</b>	<pre>#include "[{pathname}][filename].PRG"</pre> <p>* Nesting: up to 5 are allowed</p>
<b>DESCRIPTION</b>	<p>Pulls in contents of specified source program file. #include allows global variable declarations and #define definitions to be pulled in from other files.</p> <p>Files should be specified from files that exist in the file memory. If pathname is omitted, #include searches for file in current directory.</p> <p>An include file may contain a secondary include file. For example, FILE 2 may be included within FILE 1, and FILE 3 may be included within FILE 2. This is called nesting. The maximum number of loops (including the original one) is five.</p> <p>It is allowable that the line numbers of the file to be included match the line numbers of the original program. However, since by doing so it will become impossible to determine current program execution status, it is better not to allow the line numbers to match.</p> <p>The line number should be in ascending order after the include file is included into the source program file. When the line number is not in ascending order or overlapped, the following problems may occur.</p> <p>When the line number is not in ascending order, system may not find the destination line of the GOTO/GOSUB instruction during compiling and error "Undefined line number xxx" be issued even if the destination line of the GOTO/GOSUB instruction exists. Even when label is used as the destination, the error may occur.</p> <p>When the line number is overlapped, it is not defined which line is selected as the destination among the overlapped lines of the GOTO/GOSUB instruction. The line found first during compiling is selected as the destination. Even when label is used as the destination, the problem occurs. Be careful not to overlap line numbers because any error is not issued during compiling in this case.</p>

**EXAMPLE**

Not Using Include File	Using Include File
<pre> FILE1 (MAIN.PRG) &gt;10 FUNCTION MAIN &gt;20 EXTERN FUNCTION INIT &gt;30 ENTRY REAL WORK(100) &gt;40 ENTRY DOUBLE A_SIN,A_COS &gt;50 XQT !2 INIT &gt;60 FEND  FILE2 (INIT.PRG) &gt;100 FUNCTION INIT &gt;110 EXTERN REAL WORK(100) &gt;120 EXTERN DOUBLE A_SIN,A_COS &gt;130 EXTERN FUNCTION DISP       ⋮ &gt;180 XQT !3 DISP &gt;190 FEND  FILE3 (PRI.PRG) &gt;200 FUNCTION DISP &gt;210 EXTERN REAL WORK(100) &gt;220 EXTERN DOUBLE A_SIN,A_COS       ⋮ </pre>	<pre> INCLUDE FILE (VAL.PRG) &gt;300 EXTERN REAL WORK(100) &gt;310 EXTERN DOUBLE A_SIN,A_COS  FILE1 (MAIN.PRG) &gt;10 FUNCTION MAIN &gt;20 EXTERN FUNCTION INIT &gt;30 ENTRY REAL WORK(100) &gt;40 ENTRY DOUBLE A_SIN,A_COS &gt;50 XQT !2 INIT &gt;60 FEND  FILE2 (INIT.PRG) &gt;100 FUNCTION INIT &gt;110 #include "VAL.PRG"  &gt;400 EXTERN FUNCTION DISP       ⋮  FILE3 (PRI.PRG) &gt;200 FUNCTION DISP &gt;210 #include "VAL.PRG"  &gt;400 EXTERN FUNCTION DISP       ⋮ </pre>

If an include file is not used, then it becomes necessary to perform variable declarations, variable name changes, and adding variable names for each of the files that contain that variable.

For example, considering the previous example, by using an include file a variable definition change can be accomplished by changing the variable in only two places (FILE1.PRG and VAL.PRG). This is a useful feature for improving programming efficiency, especially when many tasks are used.

---

# ABS( )

F

Absolute

<b>FUNCTION</b>	Returns absolute value
<b>FORMAT</b>	ABS([Numeric Value])
<b>DESCRIPTION</b>	Returns absolute value of specified numeric value.
<b>RELATED COMMANDS</b>	SGN( ), INT( ), SQR( )
<b>EXAMPLE</b>	>PRINT ABS(-3.54) 3.54 >

A

# ACCEL

&gt;

S

Acceleration

**FUNCTION** Specifies and displays acceleration/deceleration for PTP motion

**FORMAT** (1) ACCEL A,B{,C,D,E,F}

A: acceleration specification value

B: deceleration specification value

C: acceleration specification value (axis #3 upward)

D: deceleration specification value (axis #3 upward)

E: acceleration specification value (axis #3 downward)

F: deceleration specification value (axis #3 downward)

\* Each value A–F to be integer from 1 to 100 (percent maximum acceleration/deceleration).

[default values: refer to the "Specifications" in the manipulator manual]

(2) ACCEL

**DESCRIPTION** (1) Specifies acceleration and deceleration values for PTP motion (GO, JUMP, PULSE, and related) commands.

Each value to be an integer from 1 to 100.

C, D, E, and F for axis #3 are effective only for JUMP command. If axis #3 values are omitted, C, D, E, and F default as follows:

C and E default to the A (acceleration) value

D and F default to the B (deceleration) value

(2) Displays currently valid ACCEL values as follows:

Acc. value

Dec. value

Acc. value (axis #3 upward)

Dec. value (axis #3 upward)

Acc. value (axis #3 downward)

Dec. value (axis #3 downward)

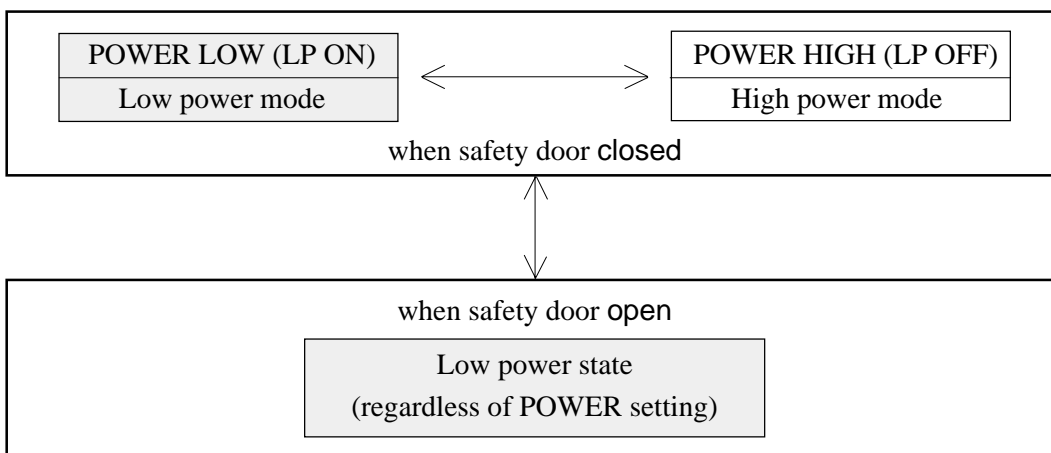
ACCEL value initializes to default value when any one of the following is performed:

- Power on
- Mode switching (TEACH/AUTO)
- Software reset
- MOTOR ON
- SFREE, SLOCK
- VERINIT
- STOP** key
- CTRL** + **C** key



While in TEACH mode, actual acceleration differs from low power state to high power state.

Motor power status	actual acceleration
Low power state	the default value of ACCEL the value of ACCEL } either lower value
High power state	the value of ACCEL



If higher acceleration is required, set high power mode using POWER HIGH or LP OFF and close safety door. If safety door is open acceleration setting will be changed to default value.

If ACCEL is executed when the robot is in low power state, the following message is displayed. The value in the message is the default value of ACCEL which varies from model to model. The following example shows that the robot will move at default acceleration (10) because it is in low power state even though the acceleration setting value by ACCEL is 100.

```
>ACCEL 100,100
Low Power State : ACCEL is limited to 10
>
>ACCEL
Low Power State : ACCEL is limited to 10
    100      100
    100      100
    100      100
>
```

**RELATED  
COMMANDS**

POWER(LP), SPEED, GO, JUMP, PASS, PULSE

**EXAMPLE**

```
>ACCEL 100,100,100,100,100,50      'Only the axis #3 downward deceleration
                                     is set at 50
>ACCEL
    100      100
    100      100
    100      50
```



# ACCELS

&gt;

S

A

**FUNCTION** Specifies and displays acceleration for CP motion

**FORMAT** (1) ACCELS { [acceleration specification value] }

\* acceleration specification value: Integer from 1 to 5000 (mm/s<sup>2</sup>)  
[default value: refer to the "Specifications" in the manipulator manual]

(2) ACCELS

**DESCRIPTION** (1) Specifies hand acceleration in mm/s<sup>2</sup> for CP motion (ARC, MOVE and CVMOVE and related) commands.

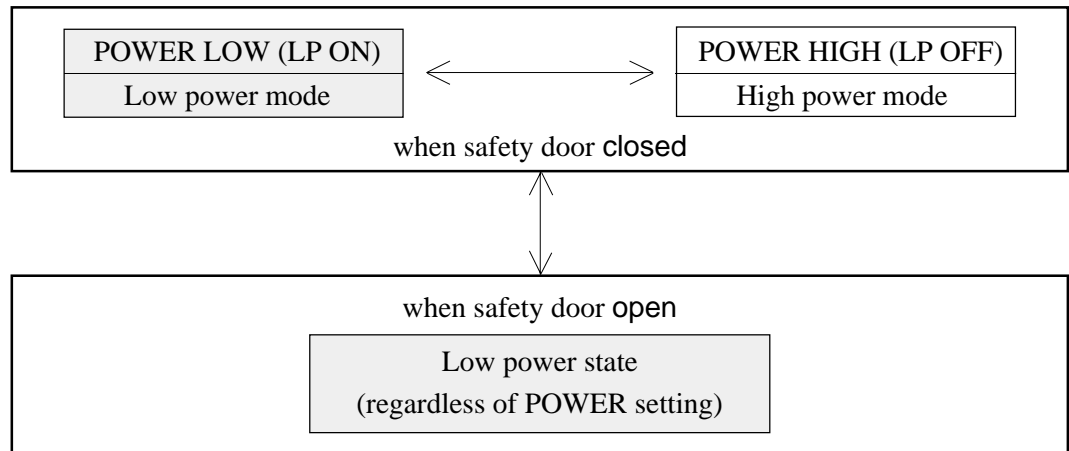
(2) Displays current ACCELS value.

ACCELS value initializes to default value when any one of the following is performed:

Power on
Mode switching (TEACH/AUTO)
Software reset
MOTOR ON
SFREE, SLOCK
VERINIT
<input type="button" value="STOP"/> key
<input type="button" value="CTRL"/> + <input type="button" value="C"/> key

While in TEACH mode, actual acceleration differs from low power state to high power state.

Motor power status	actual acceleration
Low power state	the default value of ACCELS the value of ACCELS } either lower value
High power state	the value of ACCELS



If higher acceleration is required, set high power mode using POWER HIGH or LP OFF and close safety door. If safety door is open acceleration setting will be changed to default value.

If ACCELS is executed when the robot is in low power state, the following message is displayed. The value in the message is the default value of ACCELS which varies from model to model. The following example shows that the robot will move at default acceleration (200) because it is in low power state even though the acceleration setting value by ACCELS is 1000.

```

>ACCELS 1000
Low Power State : ACCELS is limited to 200
>
>ACCELS
Low Power State : ACCELS is limited to 200
    1000
>
  
```

**RELATED COMMANDS**

POWER(LP), SPEEDS, MOVE, CMOVE, ARC, CARC, CVMOVE

**EXAMPLE**

```

>ACCELS 1000
>ACCELS
    1000
>
  
```

# AGL( )

F

Angle

**FUNCTION** Returns joint angle for selected rotational axis, or position for selected linear axis

**FORMAT** AGL([axis number])

\* axis number: integer from 1 to 4

**DESCRIPTION** If selected axis is a rotation axis, returns current angle, from selected axis 0 pulse position, in degrees. Returned value is a real number.

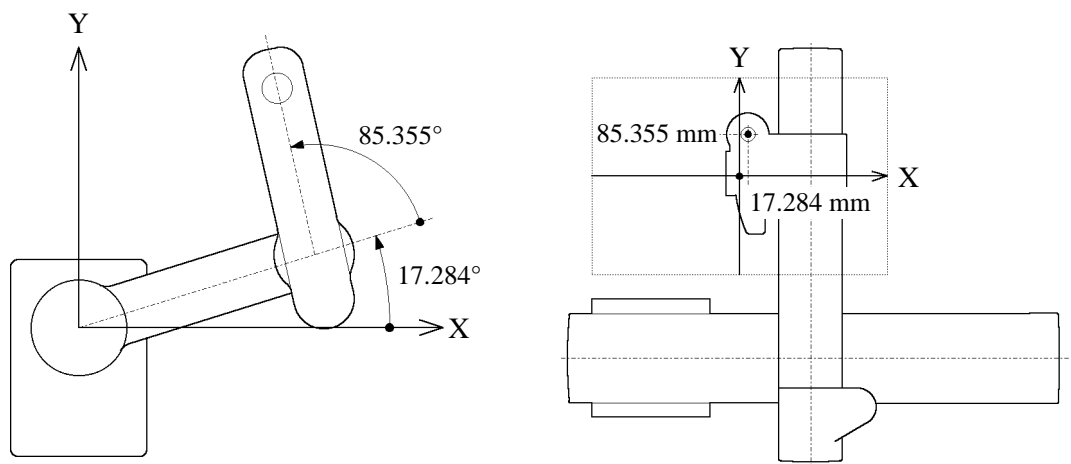
If selected axis is a linear axis, returns current position, in relation to selected axis 0 position, in mm. Returned value is a real number.

Returned value's sign (plus or minus) matches the sign for the corresponding manipulator pulse. Refer to the manipulator manual for standard arm 0 pulse position.

If an auxiliary arm is selected with ARM, AGL returns angle (or position) from standard arm 0 pulse position to selected arm.

**RELATED COMMANDS** PLS( )

**EXAMPLE** >PRINT AGL( 1 ) , AGL( 2 )  
17.234 85.355



# AOPEN...CLOSE

S

Append Open

**FUNCTION** Opens file for appending data

**FORMAT** AOPEN "[filename]" AS #[filenumber]  
 :  
 CLOSE #[filenumber]

\* Filename must include extension  
 filenumber: integer from 30 to 35

**DESCRIPTION** Opens specified file and identifies it by the specified filenumber. This statement is used for appending data to the specified file. CLOSE closes the file and releases the filenumber.

The specified file must exist on file memory. The specified filenumber identifies the file as long as the file is open, it is used by the output statement for appending (PRINT #) and the statement for closing (CLOSE #) the file. Accordingly, until the current file is closed, its file number can not be used to specify different file.

A maximum of 6 files can be open concurrently. As long as 6 files are open, however, DLOAD and DMERGE cannot be executed.

**RELATED COMMANDS** PRINT #, ROPEN, WOPEN

**EXAMPLE**

```

100 REAL DATA(200)
110 WOPEN "TEST.VAL" AS #30
120 FOR I=0 TO 100
130 PRINT #30,DATA(I)
140 NEXT
150 CLOSE #30
  :
200 AOPEN "TEST.VAL" AS #30
210 FOR I=101 TO 200
220 PRINT #30,DATA(I)
230 NEXT
240 CLOSE #30
250 '


```

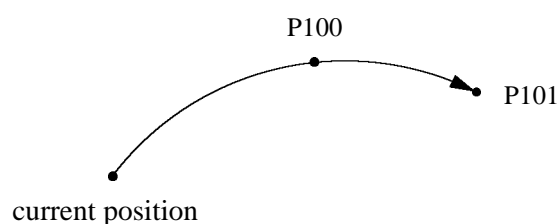
# ARC

&gt;

S

A

<b>FUNCTION</b>	Executes arc interpolator motion in the XY plane
<b>FORMAT</b>	ARC P[point number 1],P[point number 2] {![parallel processing statement!]}
<b>DESCRIPTION</b>	<p>Executes arc interpolator motion from current position, through point number 1 to point number 2. This ARC path is a true arc in a horizontal plane only.</p> <p>Arm attribute (right or left) values must be the same for current position, point number 1, and point number 2. If they are not, an error will result.</p> <p>ARC cannot execute range verification of the trajectory in advance. Therefore, even for target positions that are within an allowable range, en route the robot may attempt to traverse an invalid range, stopping with a severe shock that may damage the arm. To prevent this, be sure to perform range verifications at low speed in advance.</p> <p>ARC uses the SPEEDS speed value and the ACCELS acceleration value.</p>
<b>NOTE</b> 	<p>(1) ARC interpolates a path current position to point number 2, ignoring the Z and U coordinates of point number 1. Therefore, ARC path are suitable for horizontal work surfaces only. For non-horizontal work surfaces, use CURVE and/or CVMOVE.</p> <p>(2) Because ARC motion begins from current position, it may be necessary to use GO (or JUMP, or related) command to bring robot up to the desired current position prior to executing ARC.</p>
<b>RELATED COMMANDS</b>	SPEEDS, ACCELS, ! ... !, CARC, MOVE, CMOVE, CVMOVE
<b>EXAMPLE</b>	>ARC P100 ,P101      'Beginning at current position, execute arc interpolator motion through P100 and stop at P101



# ARCH



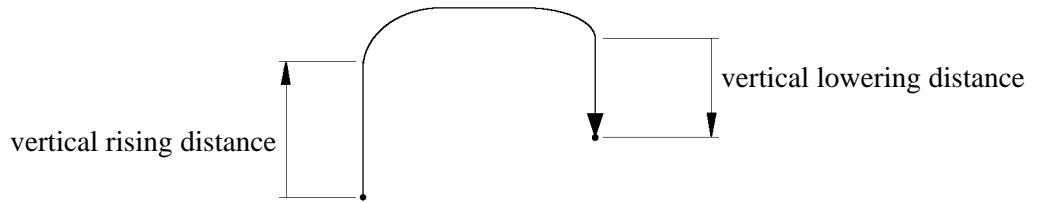
**FUNCTION** Specifies and displays JUMP arch parameters

**FORMAT** (1) ARCH [arch number],[vertical rising distance],[vertical lowering distance]

- \* arch number: integer from 0 to 6
- \* vertical rising distance: mm
- \* vertical lowering distance: mm

(2) ARCH

**DESCRIPTION** (1) Specifies JUMP arch motion vertical distance parameters.  
 ARCH can specify vertical distance parameters for arch numbers 0 - 6.  
 ARCH cannot specify parameters for arch number 7, arch number 7 is reserved for gate motion.  
 Vertical rising distance is the vertical distance above start position.  
 Vertical lowering distance is the vertical distance to target position.



ARCH values are maintained when power is off.  
 VERINIT causes ARCH values to default as shown below:

arch number	vertical rising distance	vertical lowering distance
0	30	30
1	40	40
2	50	50
3	60	60
4	70	70
5	80	80
6	90	90

If vertical rising or lowering distance specification values exceed the actual vertical motion distance, then gate motion is executed.

Arch motion is carried out per the parameters corresponding to the arch number selected in the JUMP C modifier.

(2) Displays current ARCH values.

**NOTE**

Arch motion trajectory is compounded of vertical motion and horizontal motion. It is not a continuous path control. Therefore, the actual trajectories of arch motion are not decided uniquely by ARCH parameters. The trajectory changes depending on the motion and speed.<sup>(\*)</sup> Execute JUMP with actual motion and speed to confirm the actual trajectory. When the vertical downward distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the parameter of vertical downward distance long.

<sup>(\*)</sup> In a trajectory, the vertical upward distance increases and the vertical downward distance decreases when the motion speed is set high. The change of trajectory depending on the motion is various. For general example, the above influence is large when the movement of the first arm is large in case of SCARA robot.

**EXAMPLE**

```
>ARCH 0,10,50
>JUMP P1 C0
>ARCH
  arch0=10 50
  arch1=40 40
  arch2=50 50
  arch3=60 60
  arch4=70 70
  arch5=80 80
  arch6=90 90
>
```

# ARM

&gt;

S

**FUNCTION** Selects and displays arm number

**FORMAT** (1) ARM [arm number]

\* arm number: integer from 0 to 3

(2) ARM

**DESCRIPTION** (1) Selects arm number.

ARM is used to allow each auxiliary arm to use common position data.

If no auxiliary arms are installed, the standard arm (arm number 0) operates.

Since at time of delivery the arm number is specified 0, it is not necessary to use the ARM to select an arm number.

ARM values are maintained when power is off.

VERINIT causes ARM number to initialize to 0.

Selecting auxiliary arm numbers that have not been defined by ARMSET command will result in an error.

(2) Displays current ARM number.

**RELATED  
COMMANDS** ARMSET

**EXAMPLE** >ARM 3 'Select auxiliary arm 3. If arm 3 has not been defined by ARMSET, an error will result.

>ARM 0 'Select standard arm

10 ARM 0

20 JUMP P1 'Jump to P1 with standard arm

30 ARM 1

40 JUMP P1 'Jump to P1 with auxiliary arm 1



# ARMSET

&gt;

S

A

**FUNCTION** Specifies and displays auxiliary arms

**FORMAT** (1) ARMSET [arm number],[first parameter],[second parameter],~  
[third parameter]{,[fourth parameter]{,[fifth parameter]}}

\* arm number: integer from 1 to 3

parameter: real number (refer to the chart on the next page)

(2) ARMSET

**DESCRIPTION** (1) Specifies auxiliary arm parameters, necessary if, in addition to the standard arm, an auxiliary arm (or auxiliary hand) is installed. When using an auxiliary arm, the arm is selected by the ARM.

If fourth and/or fifth parameters are omitted, the default values are the standard arm values.

< NOTE >

ARMSET values are not changed by power off. Executing VERINIT initializes ARMSET values to "unspecified."

(2) Displays current ARMSET values for all specified arms. Arm 0 (standard arm) is included in the display.

Using an auxiliary arm without first specifying its ARMSET parameters may result in the robot not being able to reach its target position. Therefore, specify ARMSET parameters prior to using an auxiliary arm, especially in the following cases:

Specifying that a single data point be operated through by two or more arms

Using PALET

Using CP motion

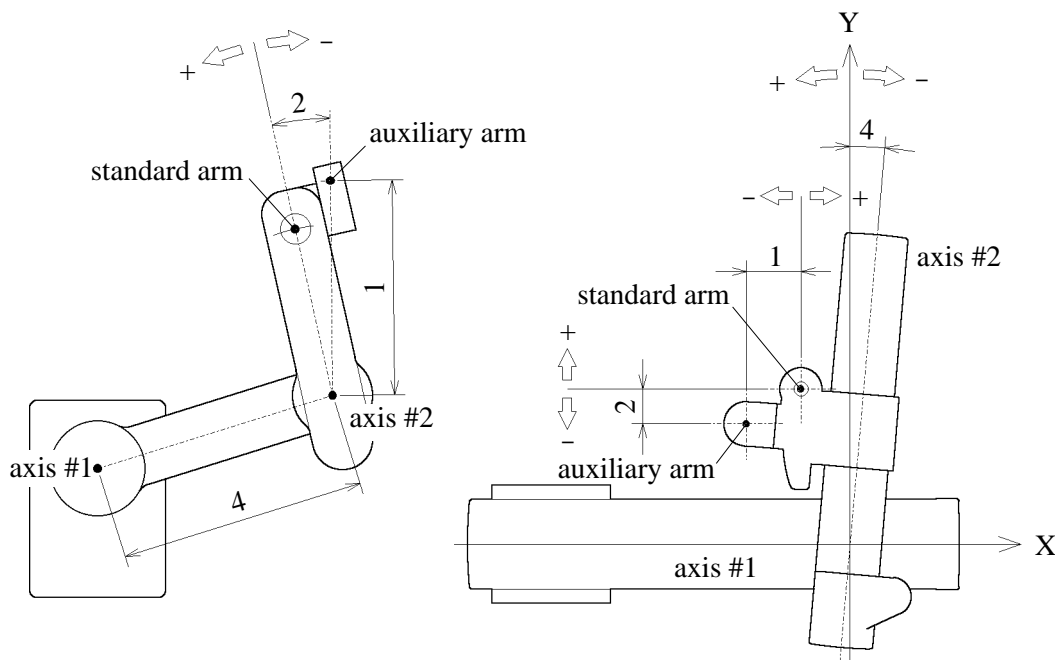
Using relative position specification

Using local coordinates

For operating SCARA robots with rotating joints or cylindrical coordinate robots in a cartesian coordinate system, joint angle calculations are based on these parameters. Therefore, specifying them with this command is very important.

		Parameter Number				
		1	2	3	4	5
SCARA TYPE	horizontal distance from axis #2 C/L to auxiliary arm C/L  (mm)	auxiliary arm offset  (degrees)	height offset  (mm)	horizontal distance from axis #1 C/L to axis #2 C/L  (mm)	orientation axis angle offset  (degrees)	
CARTESIAN TYPE	X coordinate axis direction position offset  (mm)	Y coordinate axis direction position offset  (mm)	height offset  (mm)	Angle of axis #2 from Y coordinate axis  (degrees)	orientation axis angle offset  (degrees)	

\* offset: offset to standard arm



**RELATED  
COMMANDS**

ARM

**EXAMPLE**

```
>ARMSET 1, 300, -12, -30, 300, 0
>ARMSET
  arm0 250 0 0 300 0
  arm1 300 -12 -30 300 0
>
```

# ASC( )

F

ASCII

**FUNCTION** Returns ASCII code (numeric value) for first character of specified string

**FORMAT** ASC( |[string variable name]|)  
 |" [string] "|

**DESCRIPTION** Returns ASCII code (numeric value) for first character of specified string.  
 Specified string may be constant string or variable string.

If specified variable string contains no characters (null string), an error occurs.

**EXAMPLE**

```

10 FUNCTION MAIN
20 STRING LETTERS$
30 LETTERS$="ABC"
40 PRINT ASC(LETTERS$)
50 FEND
>RUN
COMPILE END
65
>PRINT ASC("ABC")
65
>
```

'Output ASCII code for first character of the current value of variable string LETTERS\$

A

# ATAN( )

F

Arc Tangent

**FUNCTION** Returns arctangent of specified value**FORMAT** ATAN([numeric value])**DESCRIPTION** Returns arctangent, in radians, of specified value.Returned value range is from  $-\pi/2$  to  $\pi/2$ .

To convert from radians to degrees, use the following equation:

$$\text{degrees} = \text{radian} * 180 / \pi \quad (\pi = 3.141593)$$

**RELATED COMMANDS** TAN( ), SIN( ), COS( ), ATAN2( )

**EXAMPLE**

```
>PRINT ATAN(-0.55)           'Print arctangent of -0.55
-.5028432                    '-0.5028432 radians
>PRINT ATAN(0.5773503)*180/3.141593
30                            'The arctangent of 0.5773503 is 30 degrees
>PRINT TAN(30*3.141593/180)
.5773503
>
```

# ATAN2( )

F

Arc Tangent 2

**FUNCTION** Returns arctangent of y/x

**FORMAT** ATAN2([x coordinate value],[y coordinate value])

**DESCRIPTION** Returns arctangent of y/x in radians.

Returned value range is from  $-\pi/2$  to  $\pi/2$ .

To convert from radians to degrees, use the following equation:

$$\text{degrees} = \text{radian} * 180 / \pi \quad (\pi = 3.141593)$$

**RELATED COMMANDS** TAN( ), SIN( ), COS( ), ATAN( )

**EXAMPLE**

```
>PRINT ATAN2(100,100)
.7853982                                '(The angle is) 0.7853982 radians
>A=.7853982*180/3.141593                'Convert 0.7853982 radians to degrees
>PRINT A
45
>
PRINT TAN(30*3.141593/180)
.5773503
>
```

A

# BASE



**FUNCTION** Defines and displays local coordinate system

**FORMAT** (1) BASE [local coordinate system number],[position specification]

\* local coordinate system number: integer from 1 to 15

(2) BASE

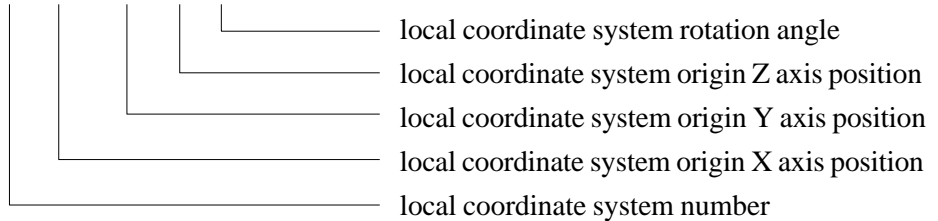
**DESCRIPTION** (1) Defines local coordinate system by specifying local coordinate system origin and rotation angle in relation to robot coordinate system.

< NOTE >

If LOCAL0 coordinate system has been defined by LOCAL0 or BASE 0, read "LOCAL0 coordinate system" instead of "robot coordinate system" in this description.

< Example >

BASE 1, 100, 200, 0, 45



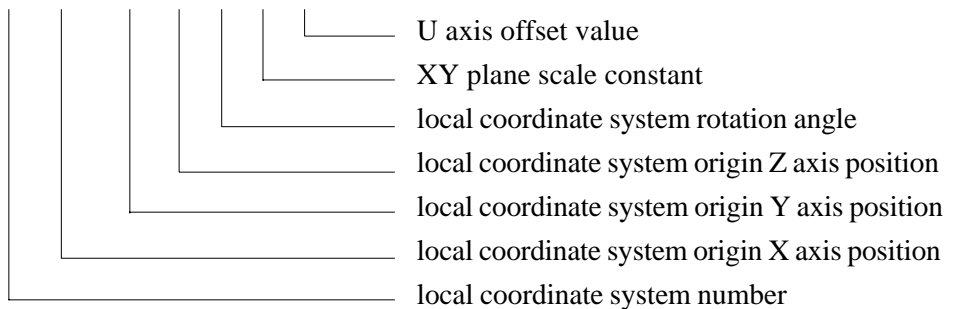
BASE 2, P10+X100

\* In this case, only the X-U axis coordinate values are referenced, arm posture and local coordinate system number of P10 are ignored.

(2) All local coordinate systems, including those defined by LOCAL, are displayed.

< Example >

base1=100 200 0 45 1 45



XY plane scale constant is usually (1). It is displayed with the coordinate system defined by SLOCAL.

U axis offset value is usually the same value as local coordinate system rotation angle. The values will differ if bit 2 of software switch SS5 is on. Refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

Local coordinate system numbers are shared between LOCAL and BASE.

Definitions of local coordinate systems #1 through #15 will be lost when the power is turned off, or when LOCAL0, BASE 0, or VERINIT are executed.

Added &n can perform reverse conversion of a local coordinate system. Remember that points determined in TEACH mode can be defined as point data in a local coordinate system.

P1=P\*&2 'Defines the current point as coordinate in the local coordinate system 2.

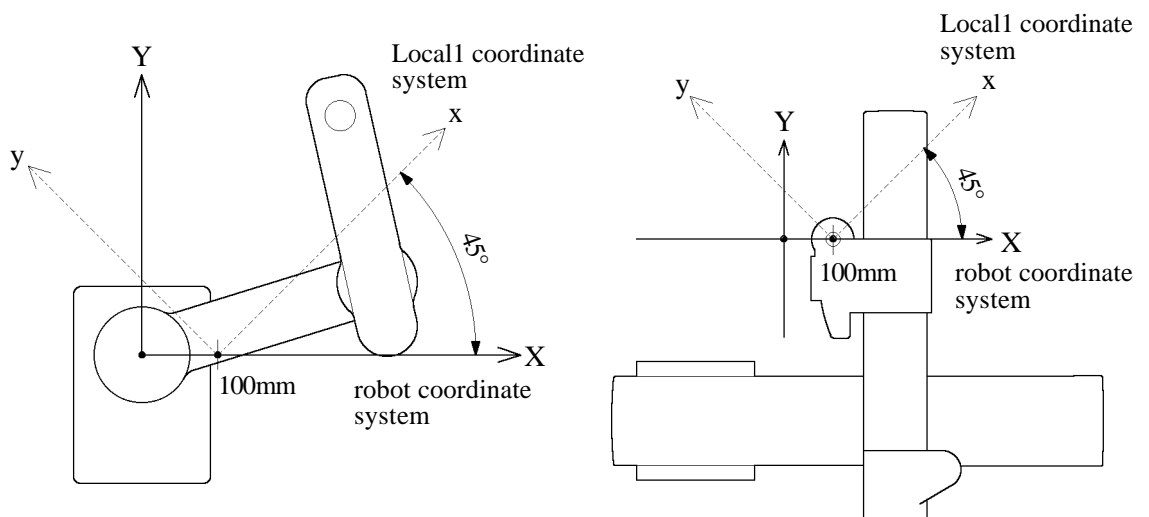
## RELATED COMMANDS

BASE 0, BASE ( ), LOCAL0, LOCAL

## EXAMPLE

```
>BASE 1,100,0,0,45 'Defines coordinate system shown below as
                    Local 1
>BASE 'Display all local coordinate systems
base0=0 0 0 0 1 0
base1=100 0 0 45 1 45
base12=100.54 11.22 0 1.554 1.554

>P5=100,200,50,0
>BASE 5,P5 'Specify local coordinate system using position
           data
>LOCAL 'Display local coordinate system per LOCAL
local 0 (p***:p***),(p***:p***) 'LOCAL0 coordinate system specified by BASE
0
local 1 (p***:p***),(p***:p***) 'Local coordinate system specified by BASE (in
this case p*** is displayed)
rlocal12 (p1 :p111),(p2 :p112) 'Local coordinate system specified by RLOCAL
```



# BASE 0

&gt;

S

Base zero

**FUNCTION** Defines robot's basic coordinate system Local0

**FORMAT** BASE 0,[position specification]

**DESCRIPTION** Each robot has an absolute coordinate system, the position of which cannot be changed, called the "robot coordinate system." This robot coordinate system becomes the reference for a local coordinate system whose position can be changed, called the Local0 coordinate system. Local0 coordinate system can be defined by either BASE 0 or LOCAL0. Defining Local0 coordinate system is the same as defining local coordinate systems with the BASE, except for the (0). Refer to BASE explanation for details.

Usually the robot coordinate system is equivalent to the Local0 coordinate system, and it is unnecessary to differentiate between them. BASE 0 is used when the local coordinate system needs to be different from the robot coordinate system, such as when executing maintenance operations.

Power off does not change BASE 0 values.

Local0 coordinate system is made equivalent to robot coordinate system by executing any one of the following:

```
BASE 0,0,0,0,0
LOCAL0 (P1:P1), (P1:P1)
VERINIT
```

Since BASE 0 asters the position of the basic coordinate system, it should be executed only when necessary.

## NOTE



Executing BASE 0 erases all local coordinate systems, including those defined by LOCAL. It is necessary to redefine them.

Executing VERINIT initializes Local0 coordinate system to (base 0 0 0 0 1 0), equivalent to the robot coordinate system.

## RELATED COMMANDS

BASE , BASE( ), LOCAL0, LOCAL



# BASE( )

F

B

**FUNCTION** Returns value of local coordinates systems

**FORMAT** BASE([local coordinate system number],[axis number])

\* local coordinate system number: integer from 1 to 15  
axis number: integer from 1 to 6

**DESCRIPTION** Returns the value of local coordinates systems.  
Returns the value of specified local coordinate and axis. Returns the value set by BASE command when the local coordinate system is set by BASE command. Returns the value displayed by BASE command when the local coordinate system is set by LOCAL command.

**RELATED COMMANDS** BASE, LOCAL

**EXAMPLE**

```
>BASE
base1=100 200 0 45 1 45
>PRINT BASE(1,1)
100
>
```

# BYTE

S

Byte

**FUNCTION** Defines 1-byte integer type variables**FORMAT** BYTE [variable name]{{(array size 1{,array size 2{,array size 3})}}~  
{,[variable name]{{(array size 1{,array size 2{,array size 3})}}})n**DESCRIPTION** This command defines 1-byte integer type variables.

If several variables of the same type are declared, use a " , " (comma) and describe several variable names. When defining an array variable, declare the name and its size enclosed with ( ). The size can be defined up to 3 dimension.

By default, the variable whose data type is not declared specifically will be treated as REAL type. Therefore, it should be declared if the integer type is sufficient or the data type because it has advantage over the REAL type in terms of performance speed and memory efficiency.

Also, there are three types to the integer type and, each handles a different size of data as follows: BYTE (1-byte integer), INTEGER (a 2-byte integer) and LONG (a 4-byte integer). The range for a BYTE type value is -128 to 127. If a value outside this range is entered, it causes an error.

For more details about the variable, refer to "2.3 Variables" in the Elementary section of the User's manual for SRC-300/320.

The following cites the restrictions on the variable names. The variable may be freely named within these restrictions.

- The usable characters are alphanumeric and underscores ( \_ ). There is no distinction between capital and small case letters.
- Must be eight characters or under.
- The first character must be an alphabet character other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. A reserved word that is followed by an underscore or numeric character is also read as a reserved word.



The variable type must be declared at the beginning of a line; otherwise, the file will not be successfully compiled. When declaring another type of variable, a new line must be created.

**RELATED  
COMMANDS**

INTEGER, LONG, REAL, DOUBLE, STRING, VARIABLE, SYS

**EXAMPLE**

```
10 FUNCTION MAIN
20 BYTE I           'Declares a 1-byte integer type variable "I."
30 BYTE ODATA(10,10) 'Declares a 2 dimensional array of 1-byte integer type
                    variable, "ODATA."
:
999 FEND
```

**B**

# CALIB

&gt;

S

Calibration

**FUNCTION** Replaces current arm posture pulse value with current CALPLS values

**FORMAT** CALIB [axis number] { ,[axis number] }3

\* axis number: integer from 1 to 4

**DESCRIPTION** Automatically calculates and specifies offset (HOFS) value. This offset is necessary to matching the origin for each robot axis motor to the corresponding robot mechanical origin.

Use CALIB when motor pulse value has changed, such as after changing a motor.

Normally, the calibration position PULSE values would match the CALPLS pulse values. However, after maintenance operations such as changing motors, these two sets of values will no longer match, and therefore calibration becomes necessary.

Calibration may be accomplished by moving the arm to a desired calibration position, and then executing CALIB. By executing CALIB, the calibration position pulse value is changed to the CALPLS value, the correct pulse value for the calibration position.

HOFS values must be determined to execute calibration. To have HOFS values automatically calculated, move the arm to desired calibration position, and execute CALIB. The controller automatically calculates HOFS values based on calibration position pulse values and on CALPLS pulse values.

If axis number is not specified, error will occur.

**NOTE**

CALIB is intended to be used for maintenance purposes only. Execute CALIB only when necessary.

Executing CALIB causes the HOFS value to be replaced. Because unintended HOFS value changes can cause unpredictable robot motion, use caution in executing CALIB only when necessary.

**RELATED  
COMMANDS**

CALPLS, HOFS

**EXAMPLE**

```
>CALPLS          'Display current CALPLS values
  65523 43320
  -1550 21351
>PULSE          'Display current position PULSE values
  65526 49358
  -1542 21299
>CALIB 2        'Execute calibration for axis 2 only
>PULSE          'Display (changed) PULSE values
  65526 43320
  -1542 21299
```

# CALL

S

**FUNCTION**            Calls a Function as a subroutine

**FORMAT**             CALL [Function name]

\* Nesting up to 10 levels is possible

**DESCRIPTION**        CALL calls (i.e., transfers program control to) a Function (defined in FUNCTION...FEND) as a subroutine. When END or FEND is encountered, program control returns to the task that called the Function procedure.

Since a Function is handled as a subroutine in a CALL statement, multiple tasks can call the same subroutine. However, should multiple tasks SIMULTANEOUSLY call the same Function, the values they assign to variables in the Function will conflict. To prevent this, write programs such that, until one task finishes processing of the Function, other tasks that follow do not call the same Function. This is called exclusive control.

To call a subroutine within a task (procedure), use a GOSUB...RETURN statement, which is described elsewhere in this manual.

**EXAMPLE**

```

100 FUNCTION MAIN
105 OFF $0                               'Memory I/O for exclusive control
110 XQT !2 SUB
    ⋮
200 CALL ERROR
    ⋮
300 FEND
310 FUNCTION SUB
    ⋮
350 CALL ERROR
    ⋮
400 FEND
410 FUNCTION ERROR
420 ON $0;IF ZEROFLG(0)=1 THEN WAIT SW($0)=0;GOTO 420
    ⋮
490 OFF $0
500 FEND

```

# CALPLS

&gt;

S

Calibration Pulse

**FUNCTION** Specifies and displays position and orientation pulse used for calibration

**FORMAT** (1) CALPLS [Axis #1 pulse value],[Axis #2 pulse value][Axis #3 pulse value], ~  
[Axis #4 pulse value]

\* pulse value: integer

(2) CALPLS

**DESCRIPTION** (1) Specifies and maintains correct position pulse value for calibration.

CALPLS is intended to be used for maintenance, such as after changing motors, when motor zero position needs to be matched to the corresponding arm mechanical zero position. This matching of motor zero position to corresponding arm mechanical zero position is called calibration.

Normally, the calibration position PULSE values would match the CALPLS pulse values. However, after performing maintenance operations such as changing motors, these two sets of values no longer match, and therefore calibration becomes necessary.

Calibration may be accomplished by moving the arm to a certain calibration position, and then executing CALIB. By executing CALIB, the calibration position pulse value is changed to the CALPLS value, the correct pulse value for the calibration position.

HOFS values must be determined to execute calibration. To have HOFS values automatically calculated, move the arm to desired calibration position, and execute CALIB. The controller automatically calculates HOFS values based on calibration position pulse values and on CALPLS values.

CALPLS values are not initialized at power off, or by executing VERINIT or SYSINIT.

(2) Displays current CALPLS values as follows:

```
[Axis #1 pulse value]  [Axis #2 pulse value]
[Axis #3 pulse value]  [Axis #4 pulse value]
```

**RELATED  
COMMANDS**

CALIB, HOFS

C

<b>EXAMPLE</b>	>CALPLS	'Display current CALPLS values
	65523 43320	
	-1550 21351	
	>PULSE	'Display current position PULSE values
	65526 49358	
	-1542 21299	
	>CALIB 4	'Execute calibration for axis #4 only
	>PULSE	'Display (changed) PULSE values
	65526 49358	
	-1542 21351	



# CARC

&gt;

S

Continuous Arc

**FUNCTION** Executes arc interpolator motion in a horizontal plane, without decelerating, through specified final position

**FORMAT** CARC P[point number 1],P[point number 2] {![parallel processing statement]!}

**DESCRIPTION** Generates and directs arc interpolator motion along a curve from current position, through point number 1 and point number 2, not decelerating as it passes through point 2. For details regarding arc interpolator motion, refer to ARC.

By immediately following CARC with CARC or CMOVE, constant speed is maintained, with no decelerations. Be sure that each such continuous motion command series completes with either a MOVE or ARC to decelerate and stop.

For CARC, CMOVE, ARC, or MOVE commands that immediately follow CARC, be sure that successive paths connect smoothly. If they do not, robot may jerk violently, or Error 152 may occur, cutting motor power and stopping robot.

To prevent this, use software switch SS5, turning bit 6 on. In this setting for CARC travel from P1 to P2, followed by MOVE, CMOVE, ARC, or CARC travel from P2 to P3, as robot approaches P2 it will generate and travel a smooth path to P3, passing near but not through P2. In this setting for CARC travel from P1 to P2, not followed by a motion command, robot will decelerate and stop at P2.

If you want to know how to set software switch, refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

If, when using MOVE or ARC to decelerate and stop subsequent to CARC, insufficient deceleration distance is provided, Error 153 will occur, cutting motor power and stopping robot.

If CARC is immediately followed by a command, statement, or function other than a motion command, the following command, statement, or function will be executed prior to the robot reaching the specified point of the preceding CARC. Therefore in principle, only SPEEDS, ACCELS, CARC, CMOVE, ARC, or MOVE may immediately follow CARC. To execute statements simultaneous to CARC motion, use parallel processing. For details, refer to parallel processing in the ! ... ! command description.

**RELATED COMMANDS** P=,! ... !, ARC, SPEEDS, ACCELS, MOVE, CMOVE

**EXAMPLE**

```
20 JUMP P1
30 CARC P2,P3 !D50;ON 0;D100;OFF 0!
40 ARC P4,P5
```

C

# CHAIN

S

**FUNCTION** Loads into main memory and executes object program and position data on file memory

**FORMAT** CHAIN "[{pathname}]{filename}"

\* Filename extensions are not allowed

**DESCRIPTION** While one program is running, loads into main memory and executes secondary object program and corresponding position data.

Currently running object program and position data is replaced by specified object program and position data, and specified program is executed from its first line. If pathname is omitted, CHAIN statement searches for file in the current directory.

Filename extension for the specified file is not allowed. From the filename specification alone, the following 3 files are automatically loaded:

```

"filename.OBJ"
"filename.SYM" } → LOAD
"filename.PNT"

```

Should any one of the above files not exist in the file memory, an error will occur. ONERR error processing cannot clear this error.

**NOTE**  


Executing CHAIN causes all variables except for backup variables to be deleted.

Because software reset does not occur when executing CHAIN, the I/O output and memory I/O conditions are maintained.

**EXAMPLE**

```

10 FUNCTION JOB1
20 IF IN(0)=1 THEN CHAIN "JOB2"           'Execute JOB2 if input port 0 is 1
30 IF IN(0)=2 THEN CHAIN "JOB3"         'Execute JOB3 if input port 0 is 2
40 FEND

```

# CHARSIZE

&gt;

F

C

Character Size

**FUNCTION** Specifies character size (on operating unit)

**FORMAT** CHARSIZE [size number]

\* size number: 2, 4, 9, or 16  
[default value: 2]

**DESCRIPTION** Specifies the character size, which are outputted to operating unit using OPU PRINT, with size number. Corresponding character size to size number is as follows. If other size number which is not in the following list is specified, it will be disregarded.

size number	character size
2	standard
4	×4
9	×9
16	×16

When power is turned on, the initial size setting is 2, standard.

**RELATED COMMANDS** CLS, CURSOR, NORMAL, REVERSE, OPUNIT, OPU PRINT

**EXAMPLE**

```
>CHARSIZE 9
>OPU PRINT 3,1,"ROBOT"
>CHARSIZE 2
>OPU PRINT 5,5,"WORLD"
```

# CHDIR, CD



Change Directory

**FUNCTION** Changes and displays current directory

**FORMAT** (1) CHDIR {[pathname]}[directory name]}

(2) CHDIR

**DESCRIPTION** (1) Changes current directory to specified directory.

If pathname and directory name are omitted, current directory is displayed. This is used to display the current directory when it is not known.

(2) Displays current directory.

At power on, root directory becomes current directory.

**RELATED  
COMMANDS** DIR

**EXAMPLE**

>CHDIR \	'Change current directory to root directory
>CHDIR . .	'Change current directory to parent directory
>CD \TEST\H55	'Change current directory to \H55 in \TEST
>CD	'Display current drive's current directory
A:\TEST\H55\	

---

# CHR\$( )

---

F

Character

**FUNCTION** Returns character that corresponds to specified ASCII code

**FORMAT** CHR\$([character code])

\* character code: integer from 0 to 255

**DESCRIPTION** Returns character that corresponds to specified ASCII code.

**EXAMPLE** >PRINT CHR\$( &H41 )+CHR\$( &H42 )+CHR\$( &H43 )

ABC

>

C

# CLEAR



**FUNCTION**            Clears position data

**FORMAT**             CLEAR

**DESCRIPTION**        Clears position data. (initializes position data area.)

Executing CLEAR in on-line mode clears position data in controller main memory.

Executing CLEAR in off-line mode clears position data in programming unit.

**RELATED  
COMMANDS**            PDEL, NEW

**EXAMPLE**            >P1=100,200,-20,0/R

>P2=0,300,0,20/L

>PLIST

P1=100,200,-20,0/R

P2=0,300,0,20/L

>CLEAR

>PLIST

>\_                    'Because all position data has been cleared, no position data is displayed

---

# CLRLIB

---

&gt;

Clear Library

<b>FUNCTION</b>	Clears backup variables
<b>FORMAT</b>	CLRLIB
<b>DESCRIPTION</b>	Clears all backup variables in memory.
<b>RELATED COMMANDS</b>	LIBRARY, SYS
<b>EXAMPLE</b>	>CLRLIB >

**C**

# CLS



Clear Screen

**FUNCTION** Erases characters (on operating unit)

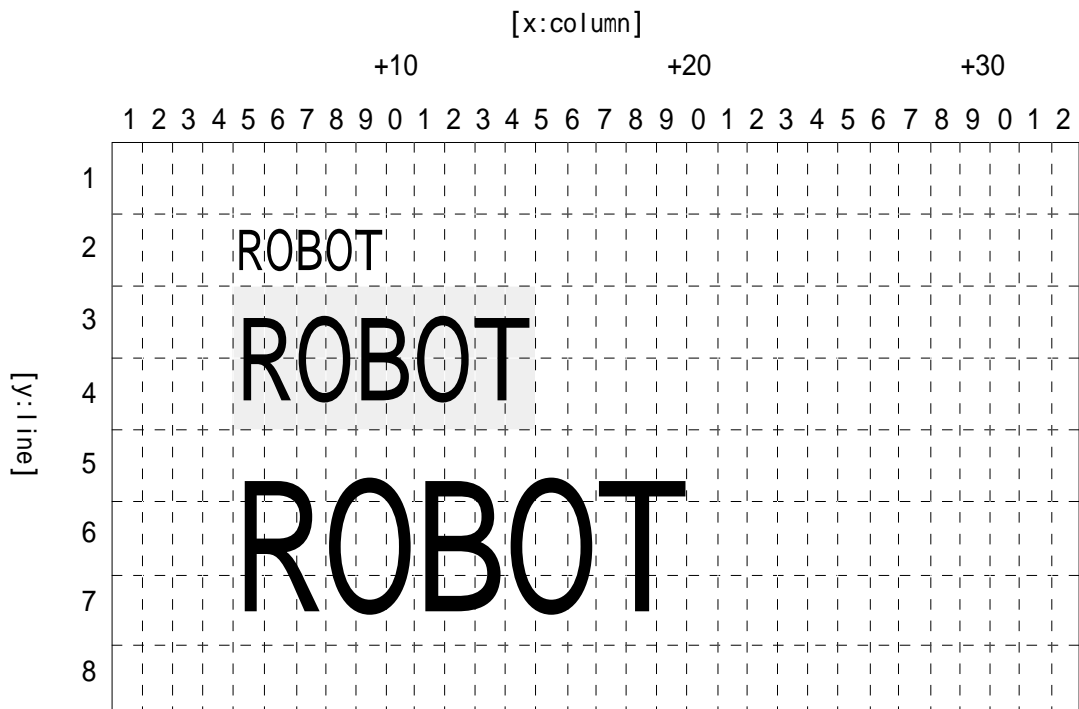
**FORMAT** CLS {[x coordinate],[y coordinate],[number of columns],[number of lines]}

**DESCRIPTION** Erases characters in the area which is specified by (x,y) and (x+number of column, y+number of line).  
After erasing characters, cursor moves to (x,y) position.

If CLS only is input, characters in all area of operating unit screen are erased. In this case, cursor moves to home position (1,1) after erasing characters.

**RELATED COMMANDS** CHARSIZE, CURSOR, NORMAL, REVERSE, OPUNIT, OPU PRINT

**EXAMPLE** >CLS 5,3,10,2 ' Erases characters in gray area shown below





# CMOVE

&gt;

S

Continuous Move

**FUNCTION** Executes simultaneous four axis linear interpolator motion, without decelerating, through specified position

**FORMAT** CMOVE [position specification] {![parallel processing statement]!}

**DESCRIPTION** Executes simultaneous four axis linear motion, from current position through specified position, not decelerating as it passes through specified position.  
For details regarding linear interpolator motion, refer to MOVE.

By immediately following CMOVE with a CARC or CMOVE command, constant speed is maintained, with no decelerations. Be sure that each such continuous motion command series completes with either a MOVE or ARC to decelerate and stop.

For CARC, CMOVE, ARC, or MOVE commands that immediately follow CMOVE, be sure that successive paths connect smoothly. If they do not, robot may jerk violently, or Error 152 may occur, cutting motor power and stopping robot.

To prevent this, use software switch SS5, switching bit 6 to 1. In this setting for CMOVE travel from P1 to P2, followed by MOVE, CMOVE, ARC, CARC travel from P2 to P3, as robot approaches P2 it will generate and travel a smooth path to P3, passing near but not through P2. In this setting for CMOVE travel from P1 to P2, not followed by a motion command, robot will decelerate and stop at P2.

If you want to know how to set software switch, refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

If, when using MOVE or ARC to decelerate and stop subsequent to CMOVE, insufficient deceleration distance is provided, Error 153 will occur, cutting motor power and stopping robot.

If CMOVE is immediately followed by a command, statement, or function other than a motion command, the following command, statement, or function will be executed prior to the robot reaching the specified position of the preceding CMOVE. Therefore in principle, only SPEEDS, ACCELS, CARC, CMOVE, ARC, or MOVE may immediately follow CMOVE .

To execute statements simultaneous to CMOVE motion, use parallel processing. For details, refer to parallel processing in the ! ... ! command description.

**RELATED COMMANDS** P=,! ... !, MOVE, SPEEDS, ACCELS, ARC, CARC

**EXAMPLE**

```
20 JUMP P1
30 CMOVE P2 !D50;ON 0;D100;OFF 0!
40 CARC P3,P4;MOVE P5
```

C

# COMPILE, COM



**FUNCTION** Compiles a source program file into an executable file

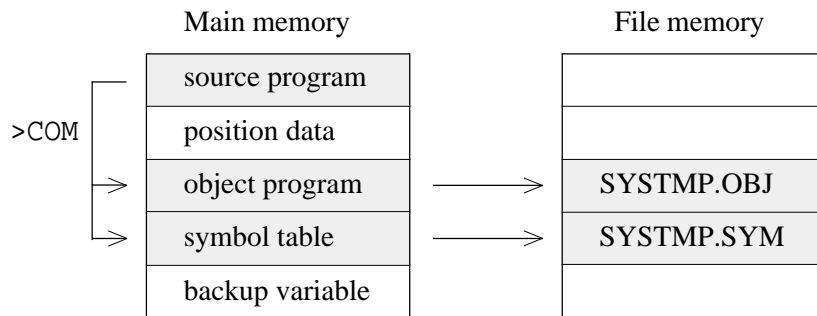
**FORMAT** COMPILE{-V}{-L}{"[pathname][filename]"}  
\* Filename extension is not allowed

**DESCRIPTION** To execute a program, the source program must be translated to machine executable form. This process is called compiling. COMPILE executes source program compiling.

In executing COMPILE, SPEL III generates an executable file in intermediate code, called the object file, and a symbol table file in which variable and Function names in source code are associated with those in intermediate code.

· If filename is omitted, SPEL III compiles the source program file existing in main memory, and generates an object file and a symbol table file in main memory. To execute the object file generated in main memory, use XQT.

After compiling, SPEL III generates in the current directory an object file named SYSTMP.OBJ and a symbol file named SYSTMP.SYM.

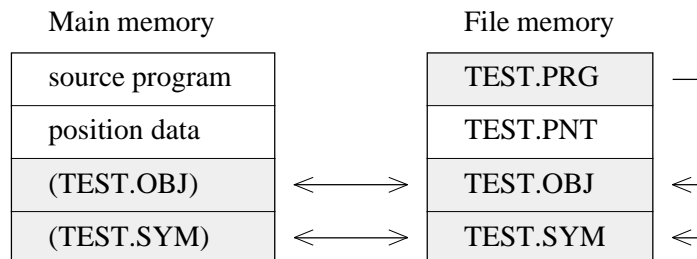


· If filename is specified, SPEL III compiles the specified source program file (filename.PRG) that resides in the specified path (pathname), and generates in the specified path of the specified drive an object file named filename.OBJ and a symbol file named filename.SYM. (Note that filename is replaced by the actual filename you specified.)

To execute the program, type XQT followed by filename. (For more information, see XQT.)

SPEL III allocates two areas in main memory for compilation: the object area and the symbol area. In executing COMPILE with the specified filename, SPEL III generates filename.OBJ and filename.SYM in the object area and symbol area as well as in the specified path.

For SPEL III to accept COMPILE with the specified filename and pathname specified, the source program file must reside in the specified path. Therefore, the source program file must be saved on file memory before executing COMPILE.



- In selecting the `-V`, SPEL III verifies whether the types of all variables are declared. If SPEL III finds any variable whose type is not declared, it generates Error 2. Thus it checks if type declarations have been made for all variables.  
If the `-V` is omitted, SPEL III regards variables with no type declared as real number type.
- In selecting the `-L` option, SPEL III compiles the specified line to be branched by GOTO/GOSUB/ONERR in 4-byte integer.  
If the `-L` is omitted, they are compiled in 2-byte integer.  
When the program is compiled without the `-L`, the following message may be displayed.

short branch [line number]

When a program is compiled without the `-L` option, specified line to be branched by GOTO/GOSUB/ONERR is expressed in 2-byte integer. It means the specified lines to be branched must be within  $\pm 32$  KB (on object area).

The above error message means that the specified lines to be branched is over  $\pm 32$  KB. In such case compile the program with the `-L` option.

In this case, object size after compiling with the `-L` option will be a little larger than the one which is compiled without the `-L` option.

#### NOTE

When executing COMPILE with the specified filename, SPEL III generates filename.OBJ and filename.SYM in the object area and symbol area as well as in the specified path. This results in loss of the consistency between the point data retained in main memory and the object file. Keep this in mind when executing the compiled program or when handling the files.

**RELATED  
COMMANDS**

RUN, XQT, QUIT, RESUME, VARIABLE

**EXAMPLE**

```

>COM                                'Compile source program file in main memory
COMPILE END
>
>XQT                                'Execute object program file in main memory
>

>COM"TEST"                          'Compile TEST.PRG in current directory
COMPILE END
>XQT"TEST"                          'Execute TEST.OBJ, the compiled file
>

>COM"TEST"
COMPILE END
>DLOAD"TEST"                        'Load into main memory the source program and position data
                                     that correspond to object file and symbol tables generated in
                                     main memory. Execute the program in response to XQT

>XQT
>
>COM-V                              'Compile while checking variable type declarations
##ERROR 2 at 50                      'Line number 50 contains a variable with no type declared
##ERROR 2 at 120                    'Line number 120 contains a variable with no type declared
>

```

# CONFIG, CNFG



Configuration

**FUNCTION** Sets configuration parameters for RS-232C communication port

**FORMAT** CONFIG {#[port number],[mode number],[protocol number],[time-out], ~  
[baud rate number]}

\* port number: 20 or 21 (With additional RS-232C: integer from 20 to 23)

mode number: integer from 0 to 47

protocol number: integer from 0 to 19

baud rate number: integer from 0 to 7

**DESCRIPTION** CONFIG sets the configuration parameters for an RS-232C communication port. The port number is 20 or 21 with standard specifications. If additional RS-232C port is installed, the port number is from 20 to 23.

For the specified mode number, protocol number, and baud rate number, see the tables on the following pages. When TTY protocol is specified, the argument time-out has no effect. The mode number 24 or later is not effective for the added port number 22 and 23.

After changing the configuration parameters for an RS-232C port by executing CONFIG, the system must be reset for the changes to take effect. This is because the system reads the configuration parameters only when it is started.

Reset the system by:

Turning the power off and on.

Switching between modes.

Executing RESET command in TEACH mode.

If an operating unit or the like is in use, pressing its reset switch in AUTO mode.

Inputting the reset characters through an RS-232C port.

If CONFIG only (without any number and time specification) is inputted, the number and time which is currently specified is displayed.

**NOTE**

Executing VERINIT initializes all port configuration parameters (mode, protocol, time-out, and baud rate) to their default values: 2, 1, 3, and 0. These numbers mean the following:

2: 7 data bits, even parity, 1 stop bit
1: basic protocol (secondary station)
3: time-out of 3 seconds
0: baud rate 9600 bps

C

**RELATED  
COMMANDS**

PRINT #, INPUT #, CON, VER, VERINIT

**EXAMPLE**

```
>CONFIG #20,0,4,0,6      'Set the configuration parameters for port #20 as follows:
                          7 data bits, even parity, 2 stop bits, TTY protocol, XON/
                          XOFF control, CR-LF terminator, baud rate 19200 bps

>CONFIG
  CONFIG #20,0,4,5,6
  CONFIG #21,2,1,3,0
```

**ARGUMENTS  
AND SETTINGS**

< mode number >

mode #	data bits	parity	stop bits	mode #	data bits	parity	stop bits
0	7	EVEN	2	24	6	EVEN	2
1	7	ODD	2	25	6	EVEN	1.5
2	7	EVEN	1	26	6	EVEN	1
3	7	ODD	1	27	-	-	-
4	8	NONE	2	28	6	ODD	2
5	8	NONE	1	29	6	ODD	1.5
6	8	EVEN	1	30	-	-	-
7	8	ODD	1	31	-	-	-
8	7	EVEN	1.5	32	6	NONE	2
9	-	-	-	33	6	NONE	1.5
10	7	ODD	1.5	34	6	NONE	1
11	-	-	-	35	-	-	-
12	7	NONE	2	36	5	EVEN	2
13	7	NONE	1	37	5	EVEN	1.5
14	7	NONE	1.5	38	5	EVEN	1
15	-	-	-	39	-	-	-
16	8	EVEN	2	40	5	ODD	2
17	8	EVEN	1.5	41	5	ODD	1.5
18	-	-	-	42	5	ODD	1
19	8	ODD	2	43	-	-	-
20	8	ODD	1.5	44	5	NONE	2
21	-	-	-	45	5	NONE	1.5
22	8	NONE	1.5	46	5	NONE	1
23	-	-	-	47	-	-	-

&lt; protocol number &gt;

protocol #	protocol	station	buffer busy control	terminator
0	TTY	-	No	CR
1	BASIC	2	-	-
2	BASIC	1	-	-
3	TTY	-	XON/XOFF	CR
4	TTY	-	XON/XOFF	CR-LF
5	TTY	-	XON/XOFF	LF
6	TTY	-	No	CR-LF
7	TTY	-	No	LF
8	BASIC2	2	-	-
9	BASIC2	1	-	-
10	TTY	-	RS/CS	CR
11	BASIC	-	RS/CS	-
12	BASIC	-	RS/CS	-
13	TTY	-	RS/CS & XON/XOFF	CR
14	TTY	-	RS/CS & XON/XOFF	CR-LF
15	TTY	-	RS/CS & XON/XOFF	LF
16	TTY	-	RS/CS	CR-LF
17	TTY	-	RS/CS	LF
18	BASIC2	2	RS/CS	-
19	BASIC2	1	RS/CS	-

\* Notes on using RS/CS

- Transmission is not controlled for RS, RS output (transmission inquiry) is always L(Low).

The controller has 20 frames and 120 byte buffer, so there is little possibility of data transmission failure for proper amount of data transmission even protocol number 10, 16, or 17 (TTY, RS/CS) is used.

However, if large amount of data is transmitted to controller, data may not be received correctly since those protocol make data transmission ready all the time. In this case, use protocol number 13, 14 or 15 (TTY, RS/CS & XON/XOFF).

- In the case of protocol number 18 or 19 (BASIC, RS/CS), if controller is in waiting status for sending data, time-out function is not effective.

&lt; baud rate number (transmission speed) &gt;

baud rate #	baud rate
0	9600 bps
1	4800
2	2400
3	1200
4	600
5	300
6	19200
7	38400

# CONSOLE, CNSOL



**FUNCTION** Specifies console to be used in AUTO mode

**FORMAT** CONSOLE { |OP | }  
 |# [port number] |  
 |BUS |

\* port number: 20 or 21  
 [default value: OP]

**DESCRIPTION** When OP is specified, the device connected to main remote is console. The robot can be controlled by using switches. The main remote is selected by the bit 1 of software switch SS1. When robot is delivered REMOTE2 (operating unit) is selected as the main remote.

When port number is specified, the specified RS-232C port is console. Through the RS-232C port specified as the console, the robot can be controlled by using SPEL III instructions in AUTO mode as you can in TEACH mode.

When BUS is specified, serial bus BUS1 (option) is console. A host computer in the serial bus network can control the robot.

With no specification, CONSOLE displays the current console.

Use VER to display the current value of port number.



When executing VERINIT, the CONSOLE value is initialized to its default value: OP.

**RELATED COMMANDS** CONFIG, VER, VERINIT

**EXAMPLE** >CONSOLE #20 'Specify RS-232C port #20  
 >CONSOLE #21 'Specify RS-232C port #21



# COPY



**FUNCTION** Copies file to another location

**FORMAT** COPY {[pathname 1]}{[filename 1]} {[pathname 2]}{[filename 2]}

\* Filename must include extension

**DESCRIPTION** Copies specified filename 1 (source) to specified filename 2 (destination).

Wildcard characters (\*, ?) are allowed in specified filenames.

The same pathname and filename may not be specified for both source and destination files.

Source Spec. (filename 1)
COPY [pathname 1]
Copies all files residing in the specified path (pathname 1).
COPY [pathname 1][filename 1]
Copies the specified file (filename 1). If pathname 1 is omitted, the current directory is assumed.

Destination Spec. (filename 2)
[pathname 2]
Copies the file to the specified directory (pathname 2). The copied file is given the same name as the original file.
[pathname 2][filename 2]
Copies the file to the specified file (filename 2) in the specified directory (pathname 2).

When copying a file within a directory, it is recommended to make that directory the current directory using CHDIR(CD). Then COPY can be executed specifying only the filenames without having to specify pathnames for the source and destination.

**RELATED COMMANDS** DIR, CHDIR, MKDIR

**EXAMPLE** COPY \BAK\\*.PRG \USR\\*.PRG 'Copy all files with ".PRG" extension from directory BAK to directory USER, preserving the original filenames

C

# COS( )

F

Cosine

**FUNCTION** Returns cosine of specified angle**FORMAT** COS([radians])**DESCRIPTION** Returns cosine of the specified angle in radians.

Angles in degrees must be converted to radians, using the following equation:

$$\text{radian} = \text{degrees} * \pi / 180 \quad (\pi = 3.141593)$$

**RELATED COMMANDS** SIN( ), TAN( ), ATAN( ), ATAN2( )

**EXAMPLE**

```
>PRINT COS(0.55)           'Display cosine of 0.55 radians
.8525245
>PRINT COS(30*3.141593/180) 'Display cosine of 30 degrees
.8660254
>A=30*3.141593/180         'Display cosine of 30 degrees, using variable
>PRINT COS(A)
.8660254
>
```

# CTR( )

F

Counter

**FUNCTION** Returns count value of counter

**FORMAT** CTR([input bit number])

\* input bit number: number of input bit set as counter  
 returned value: integer from 0 to 32767

**DESCRIPTION** Returns count value of counter specified by input bit number.

**RELATED  
 COMMANDS** CTRESET

<b>EXAMPLE</b>	100 CTRESET 3	'Reset counter 3
	110 ON 0	'Switch on
	120 TURN:	
	130 IF CTR(3)<5 THEN GOTO TURN	
	140 OFF 0	'When count value is 5 switch off

C

# CTRESET

&gt;

S

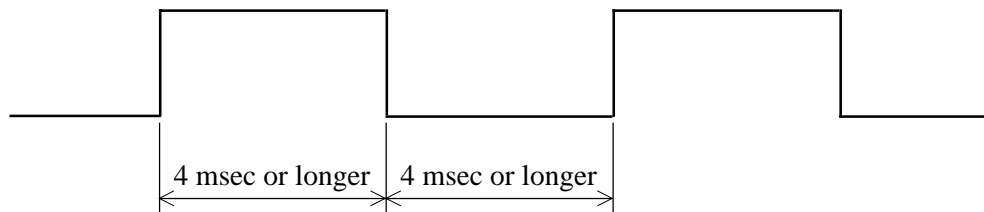
Counter Reset

**FUNCTION** Resets counter**FORMAT** CTRRESET [input bit number]

\* input bit number: integer from 0 to 127  
 number of counters: maximum 16

**DESCRIPTION** Sets specified input bit as counter and start the counter. If specified input bit is already set as a counter, it is reset and starts.

Switching input from off to on causes counter to increase 1 count. Pulse input timing chart is as shown below:



Turning power off releases all counters.

Use CTR to retrieve count value.

**RELATED COMMANDS** CTR( )

**EXAMPLE**

```

100 CTRRESET 3           'reset counter (input bit 3)
110 ON 0                 'Switch on
120 TURN:
130 IF CTR(3)<5 THEN GOTO TURN
140 OFF 0                 'When count value is 5, switch off

```

---

# CURSOR

---

&gt;

S

<b>FUNCTION</b>	Specifies cursor display on or off (on operating unit)
<b>FORMAT</b>	CURSOR  ON    OFF
<b>DESCRIPTION</b>	If ON is specified, cursor is displayed. If OFF is specified, cursor is not displayed.  When power is turned on, initial setting is OFF.
<b>RELATED COMMANDS</b>	CHARSIZE, CLS, NORMAL, REVERSE, OPUNIT, OPU PRINT

C

# CURVE



**FUNCTION** Creates a file for free curve CP control motion

**FORMAT** CURVE {[pathname]}[filename],[O],[mode setting],[number of axis],~  
|C| [curve point specifications]

- \* filename: filename extensions are not allowed
- mode setting: integer from 0 to 3
- number of axes: integer from 2 to 4

**DESCRIPTION** Creates a free curve CP (Continuous Path) motion file from specified continuous point series.

This file is used to execute CVMOVE curve motion.

It is not necessary to specify speeds and accelerations prior to executing CURVE.

Speeds and accelerations may be changed anytime prior to executing CVMOVE.

Points defined by BASE or LOCAL may be used in the series to locate the curve at the desired position.

If pathname is specified, the file is created in the specified directory. If pathname is omitted, the file is created in the current directory.

Filenames may contain a maximum of 8 alphanumeric and/or underscore ( \_ ) characters. CURVE automatically adds the extension .CRV to name of generated files.

"O" or "C" specifies open or closed curves, respectively.

For open curves, motion stops at the final specified point.

For closed curves, motion continues through the final specified point, and stops after returning to the start point.

Mode settings (decelerated stops and tangential correction) are specified as follows:

mode setting	decelerated stop	tangential correction
0	Yes	No
1	Yes	No
2	Yes	Yes
3	No	Yes

Tangential correction continuously maintains tool alignment tangent to the curve in the XY plane. It is specified when installing tools such as cutters that require continuous tangential alignment.

Because tangential correction for closed curves requires a complete 360 degree tool rotation, prior to executing CVMOVE it is necessary to extend the range with RANGE.

Number of axes is an integer from 2 to 4.

2 selects generating a curve in the XY plane with no Z axis movement or U axis rotation, 3 selects generating a curve in XYZ space with no U axis rotation, and 4 selects generating a curve in XYZ space with U axis rotation.

Curve point specifications (individual points and/or continuous increasing or decreasing point series) are separated by commas.

A continuous increasing or decreasing point series may be indicated by a dash (-) between the first and final points in the series. For example, (P1-P5) is equivalent to (P1, P2, P3, P4, P5).

Including intermediate points of continuous series, closed curves may be specified by from 3 to 50 points, open curves may be specified by from 4 to 200 points.

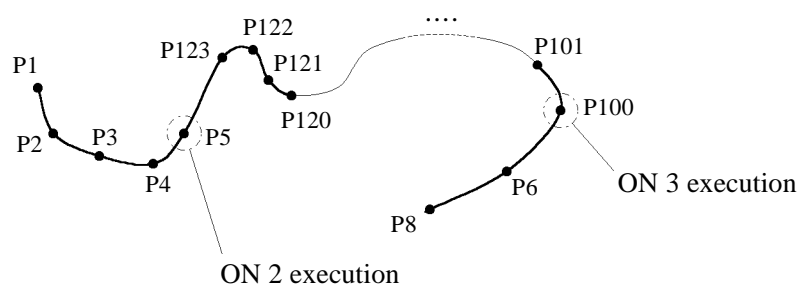
A maximum of five I/O process statements for execution during curve motion, separated by commas, may be included in CURVE. I/O processing is executed as the robot passes through the point (or final point in the series) that precedes the I/O command. (Refer to example below.)

## RELATED COMMANDS

CVMOVE, FILES

## EXAMPLE

```
>CURVE CVFIL,0,0,4,P1-P5,ON 2,P123-P100,ON 3,P6,P8 'Specifies curve
>FILES as shown below
:
CVFIL CRV 6
:
space 65
>JUMP P1
>CVMOVE CVFIL
```



# CVMOVE

&gt;

S

Curve Move

**FUNCTION** Executes free curve CP control motion file generated by CURVE

**FORMAT** CVMOVE {[pathname]}[filename]

**DESCRIPTION** Executes free curve CP (Continuous Path) motion based on CURVE files.

If pathname is specified, CVMOVE searches for the file in the specified directory.

If pathname is omitted, CVMOVE searches for the file in the current directory.

Filenames are specified without their extension(.CRV).

For free curve files created from point data that has already been assigned local attributes, movement positions can be changed with the LOCAL or BASE.

CVMOVE cannot execute range verification of the trajectory in advance. Therefore, even for target positions that are within an allowable range, en route the robot may attempt to traverse an invalid range, stopping with a severe shock that may damage the arm. To prevent this, be sure to perform range verifications at low speed in advance.

CVMOVE uses the SPEEDS speed value and the ACCELS acceleration value.

**RELATED COMMANDS** SPEEDS, ACCELS, CURVE

**EXAMPLE**

```
>CURVE CVFIL,0,0,4,P1-P5,ON 2,P123-P100,ON 3,P6,P8 'Specifies CURVE
parameters
>FILES
:
CVFIL CRV 6
:
space 65
>JUMP P1
>CVMOVE CVFIL
```



# CX(P )      CY(P ),CZ(P ),CU(P )

F

Coordinate of X-axis

**FUNCTION** Returns X, Y, Z, or U axis coordinate value of specified point

**FORMAT** (1) C|X|(P[point number])  
           |Y|  
           |Z|  
           |U|

(2) C|X|(P\*)  
       |Y|  
       |Z|  
       |U|

**DESCRIPTION** (1) Returns X, Y, Z, or U axis coordinate value of specified point.

(2) Returns each coordinate value of current position

**RELATED  
 COMMANDS** PLS( )

**EXAMPLE** >PRINT CX(P1)  
 0  
 >PRINT CY(P2)  
 200  
 >PRINT CZ(P2)  
 -50  
 >PRINT CU(P\*)  
 -56.234  
 >

P1=0, 0, 0, 0 P2=100, 200, 300, -50, 180
---

C

# DATE



<b>FUNCTION</b>	Specifies and displays current date
<b>FORMAT</b>	(1) DATE [month]-[day]-[year]  (2) DATE
<b>DESCRIPTION</b>	(1) Specifies current year, month, and date. DATE automatically corrects the current weekday. This date is used for file record keeping control.  (2) Displays current date.
<b>RELATED COMMANDS</b>	DATES\$(0), TIME, TIMES\$(0)
<b>EXAMPLE</b>	>DATE Current date is Wed 12-25-1996  >DATE 1-3-1997 >DATE Current date is Fri 1-03-1997 >

---

# DATE\$(0)

---

F

<b>FUNCTION</b>	Returns current date
<b>FORMAT</b>	DATE\$(0)  * The numeral 0 in ( )
<b>DESCRIPTION</b>	Returns current year, month, and date.
<b>RELATED COMMANDS</b>	DATE, TIME, TIME\$(0)
<b>EXAMPLE</b>	>PRINT DATE\$(0) Fri 1-03-1997

D

# DEL, ERASE



Delete

**FUNCTION** Deletes file(s)**FORMAT** DEL {[pathname]}.{[filename]}

\* At least, one of [data] must be specified  
Each file name must have an extension

**DESCRIPTION** Deletes specified file(s).  
Filename and extension may contain wildcard characters (\*, ?). If \*.\* is entered in place of the filename, the following message appears:

Are You Sure (Y/N)?

To delete all files in current directory enter  or

To delete no files enter  or

If pathname is omitted, DEL deletes file(s) in current directory.

If filename is omitted, DEL deletes all files in current directory. This is equivalent to entering \*.\* in place of the filename.

**RELATED  
COMMANDS** KILL

**EXAMPLE** DEL \BAK  
DEL \*.PNT

# DELETE, DELET


**D**

**FUNCTION** Deletes program line(s)

**FORMAT** DELETE [[line number] |  
 [[beginning line number] - |  
 [[beginning line number] -[ending line number] |  
 | -[ending line number] |

**DESCRIPTION** Deletes specified program lines as follows:

[line number]
Deletes specified line number
[beginning line number]-
Deletes all lines from beginning line number to the end of the program
[beginning line number]-[ending line number]
Deletes all lines from beginning line number up to and including ending line number. To prevent Error 2 from occurring, beginning line number must be less than ending line number.
-[ending line number]
Deletes all lines up to and including ending line number.

**RELATED COMMANDS** NEW, PDEL

**EXAMPLE**

```
>DELETE 30-40 'Delete lines from 30 up to and including 40
>LIST
 10 FUNCTION MAIN
 20 JUMP P1
 50 FEND
>DELETE 100 'Delete line 100
>DELETE 200- 'Delete all lines from 200 onward
>
```

10 FUNCTION MAIN
20 JUMP P1
30 WAIT 3
40 JUMP P5
50 FEND

# DIR



Directory

**FUNCTION** Displays contents of directory**FORMAT** DIR {[pathname]}\{[filename]}\{/P}\{/W}**DESCRIPTION** Displays filename, directory name, file size, and date and time of previous editing for specified directories and files as follows:

AUTO	BAT	12345	2-18-93	1:00
------	-----	-------	---------	------

Filename	Extension	File Size (bytes)	Date of Previous Editing	Time of Previous Editing
----------	-----------	-------------------	--------------------------	--------------------------

For subdirectories, instead of file size, <DIR> is displayed.

If pathname is omitted, DIR searches in current directory.

Filename and extension may contain wildcard characters (\*, ?). If both filename and extension are omitted, this is equivalent to entering \*.\* in place of the filename. In this case, information for all files that exist in the specified directory will be displayed.

It is possible to omit one of either the filename or the extension. Doing so is equivalent to specifying the wildcard character \* in place of either the filename or extension.

However, be aware that by omitting the extension of a file that has a filename identical to a directory name in the same path, DIR defaults to that directory. Therefore DIR information for all of the sub directories and files in that directory will be displayed.

Note that the following commands are equivalent :

Command	Equivalent Command
DIR	DIR *.*
DIR *.PRG	DIR *.PRG

**/P** Specifies page mode. One screen of information is displayed. To view the next screen, press space key.

**/W** Specifies wide display. Only filenames are displayed, five per line.

**/P/W** Specifies page mode and wide display. One screen of information is displayed. To view the next screen, press space key. Only filenames are displayed, five per line.

**RELATED  
COMMANDS**

FILES, WIDTH

**EXAMPLE**

```
>DIR  
>DIR *.PRG  
>DIR TEST.OBJ
```

# DLOAD, DLO

&gt;

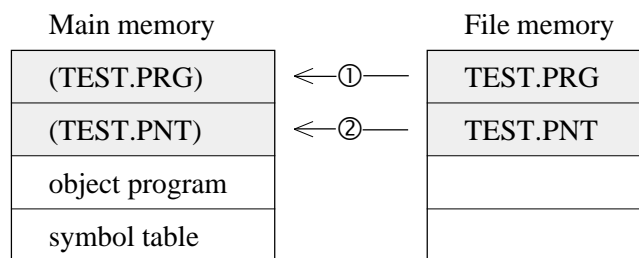
S

Disk Load

**FUNCTION** Loads specified files into main memory

**FORMAT** DLOAD "[{pathname}]{filename}{|.PRG|}"  
|.PNT|

**DESCRIPTION** Loads specified file from file memory into main memory.



If filename extension is specified, only that file is loaded.

If the extension .PRG is specified, only the source program [filename.PRG] will be loaded.

If the extension .PNT is specified, only the position data [filename.PNT] will be loaded.

If filename extension is omitted, [filename.PRG] and [filename.PNT] will be successively loaded. If either file does not exist an error will occur. If [filename.PRG] exists but [filename.PNT] does not, the error will occur after [filename.PRG] is loaded.

The statement DLOAD "[filename.PNT]" may be included in the program in order to load position data files during program execution. However, the statement DLOAD "[filename.PRG]" is not allowed.

If pathname is omitted, DLOAD loads file(s) from current directory.

## NOTE



DLOAD overwrites current source program and position data in main memory. Therefore, prior to executing DLOAD, copy current source program and position data to file memory or floppy disk, if necessary.

## RELATED COMMANDS

FILES, DIR, DSAVE, DMERGE

## EXAMPLE

```
>DLOAD "TEST"
>DLOAD "TEST.PRG"
>
```



# DMERGE

&gt;

S

Disk Merge

**FUNCTION** Loads specified file(s) into main memory and merges them with source program or position data

**FORMAT** DMERGE "[pathname][filename{[.PRG]}]"  
|.PNT|

**DESCRIPTION** Loads specified file(s) from file memory into main memory and merges them with source program or position data.

If filename and extension is specified, only that file is loaded and merged.

Valid filename extensions are .PRG and .PNT only.

If filename is specified but filename extension is omitted, [filename.PRG] and [filename.PNT] will be successively loaded and merged. If either file does not exist, an error will occur. If [filename.PRG] exists but [filename.PNT] does not, the error will occur after [filename.PRG] is loaded and merged.

If a loaded file line number matches a main memory source program line number, the loaded file line contents replace the main memory source program line contents.

In a similar manner, if a loaded file point number matches a main memory point number, the loaded file position data replaces the main memory position data.

The statement DMERGE "[filename.PNT]" may be included in the program in order to merge position data files during program execution. However, the statement DMERGE "[filename.PRG]" is not allowed.

If pathname is omitted, DMERGE loads specified file(s) from the current directory and merges them.

**RELATED COMMANDS** FILES, DIR, DSAVE

**EXAMPLE** >DMERGE "TEST.PRG"

>LIST

```
10 FUNCTION TEST
20 JUMP P2
30 WAIT 1
35 ON 2
40 FEND
45 FEND
```



Main Memory
10 FUNCTION MAIN
20 JUMP P1
30 WAIT 1
40 FEND

+

File Memory
10 FUNCTION TEST
20 JUMP P2
35 ON 2
45 FEND

# DOUBLE

S

Double

**FUNCTION** Defines 8-byte REAL type variables**FORMAT** DOUBLE [variable name]{{(array size 1{,array size 2{,array size 3})}}~  
{,[variable name]{{(array size 1{,array size 2{,array size 3})}}}}n**DESCRIPTION** This command defines 8-byte REAL type variables.

If several variables of the same type are declared, use a " , " (comma) and describe several variable names. When defining an array variable, declare the name and its size enclosed with ( ). The size can be defined up to 3 dimension.

By default, the variable whose data type is not declared specifically will be treated as REAL type. Therefore, it should be declared if the integer type is sufficient or the data type because it has advantage over the REAL type in terms of performance speed and memory efficiency.

The DOUBLE type supplies 14 valid digits. If the high accuracy is not needed, using REAL (a 4-byte REAL number, 7 valid digits) is recommended.

For more details about the variable, refer to "2.3 Variables" in the Elementary section of the User's manual for SRC-300/320.

The following cites the restrictions on the variable names. The variable may be freely named within these restrictions.

- The usable characters are alphanumeric and underscores ( \_ ). There is no distinction between capital and small case letters.
- Must be eight characters or under.
- The first character must be an alphabet character other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. A reserved word that is followed by an underscore or numeric character is also read as a reserved word.

**NOTE**

The variable type must be declared at the beginning of a line; otherwise, the file will not be successfully compiled. When declaring another type of variable, a new line must be created.

**RELATED  
COMMANDS**

BYTE, INTEGER, LONG, REAL, STRING, VARIABLE, SYS

**EXAMPLE**

```
10 FUNCTION MAIN
20 DOUBLE I           'Declares a 8-byte REAL type variable "I."
30 DOUBLE ODATA(10,10) 'Declares a 2 dimensional array of 8-byte REAL type
                        variable, "ODATA."
   :
999 FEND
```

# DSAVE, DSA

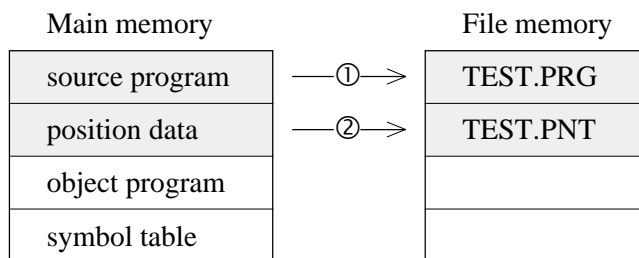


Save to Disk

**FUNCTION** Saves main memory source program and position data files in file memory or floppy disk

**FORMAT** DSAVE "[pathname][filename{[.PRG]}]  
|.PNT|"

**DESCRIPTION** Saves main memory source program and position data files, names the files as specified, and saves them in file memory or floppy disk.



The following restrictions apply to filenames: (Refer to User's manual for details.)

- Eight characters or less
- Allowed characters  
alphanumeric characters \*  
symbols such as ! # \$ % & ( ) { } - \_ @ ^

\* Either large or small case letters may be input, small case letters are converted to large case.

Valid filename extensions are .PRG and .PNT only.

If [filename.PRG] is specified, only source program is saved. If [filename.PNT] is specified, only position data file is saved.

If filename extension is omitted, both [filename.PRG] and [filename.PNT] will be saved.

If DSAVE is executed with no source program or position data in main memory, saved files will contain nothing.

The statement DSAVE "[filename.PNT]" may be included in the program in order to save position data during program execution.

In this case, it is possible to use other extensions except ".PNT". However, it is recommendable to use ".PNT" extension for position data file.

If pathname is omitted, DSAVE saves files in current directory.

If in the specified directory a file with same filename already exists, an error will occur. Should this occur, use DEL to delete unnecessary files.

**RELATED  
COMMANDS**

DIR, FILES, DEL, KILL, DLOAD

**EXAMPLE**

```
>DSAVE "TEST.PRG"  
>DSAVE "TEST.PNT"  
>KILL "TEST"  
>DSAVE "TEST"  
>
```

# DSW( )

F

**FUNCTION** Returns remote input status as decimal

**FORMAT** DSW([port number])

\* port number: integer from 2 to 6

**DESCRIPTION** Returns status of the REMOTE1 - 3 as a decimal.  
 SPEL III in this version assigns no meaning to port number 0 or 1. (DSW( ) returns 0 if port number 0 or 1 are specified.)

Emergency stop can be inputted from REMOTE1, REMOTE2 and TEACH port.  
 DSW( ) can not specify the port from which the emergency stop is inputted because of serial circuit.

The status of E.STOP indicates the emergency stop condition or not, not the condition of the emergency stop switch. In other words, even if the mechanical hold of the emergency stop switch is released DSW( ) returns 1 which means emergency stop condition until the emergency stop condition is canceled.

DSW(2)	bit0	RESET	REMOTE2 (OPU-300)	
	1	PAUSE		
	2	START		
	3	undefined		
	4	undefined		
	5	undefined		
	6	undefined		
	7	MONITOR		
DSW(3)	bit0			
	1			
	2			
	3			
	4	F1		
	5	F2		
	6	F3		
	7	F4		
DSW(4)	bit0	undefined		
	1	AUTO mode		
	2	TEACH mode		
	3	undefined		
	4	undefined		
	5	undefined		
	6	Safeguard		REMOTE1
	7	E.STOP		REMOTE1, REMOTE2, TEACH

DSW(5)	bit0	RESET	REMOTE3
	1	PAUSE	
	2	START	
	3	HOME	
	4	program No.2 <sup>0</sup>	
	5	program No.2 <sup>1</sup>	
	6	program No.2 <sup>2</sup>	
	7	program No.2 <sup>3</sup>	
DSW(6)	bit0	MCAL	
	1	Motor power on	
	2	Motor power off	
	3	undefined	
	4	undefined	
	5	undefined	
	6	undefined	
	7	undefined	

DSW(5) and DSW(6) for REMOTE3 function only for bits which set up as REMOTE3.

DSW( ) returns 0 if it is used to bits which are not set up as REMOTE3 .

## RELATED COMMANDS

PRGNO, OPUNIT

## EXAMPLE

The program can check whether the [F1] key of the OPU-300 was pressed by including the following statements:

```
10 FUNCTION CHECKSW
20 IF (DSW(3) AND &H10)=&H10 THEN...
```

# ECLR

S

Error Clear

**FUNCTION**            Clears error status

**FORMAT**             ECLR

**DESCRIPTION**       Clears error status (error number).

ECLR is use with ONERR.

By including an ONERR, should an error occur during program execution, it becomes possible to branch to an error processing subroutine.

When it is desired to return to the main program at the completion of the error processing subroutine, it is necessary to use ECLR to clear the error status. If the error status is not cleared by ECLR execution, the uncleared error status will cause the error processing subroutine to repeat.

**RELATED  
COMMANDS**            ONERR

**EXAMPLE**

```

10 ONERR ERR_SUB        'When error occurs during execution, branch to line 70
20 FOR I=0 TO 199
30 JUMP P1
40 NEXT I
50 END
60 '
70 ERR_SUB:
80 ECLR                 'Clear error status
  :
220 RETURN

```



# EDIT



**FUNCTION** Switches to edit mode

**FORMAT** EDIT {[pathname]}[filename][.{extension}]

**DESCRIPTION** Switches to edit mode, to edit text files such as batch files. Two special commands are available for exiting from edit mode: QUIT and END. Issuing either command switches back to programming mode.

In programming mode, although source program files and position data files can be edited, other text files, such as batch files, cannot be edited. Editing text files other than source program and position data files must be done in edit mode.

To edit a file with no extension, specify the filename with no extension.

```
End of input file
```

If the specified filename does not exist, the following message appears, which indicates that a new file will be created:

```
New file
```

To exit edit mode, use one of the two edit mode specific commands; QUIT (exit without saving file), or END (exit after saving file). The following messages appear after exiting:

```
>QUIT
Edit End...
>
>END
Edit End...file save
>
```

When loaded by EDIT, line numbers starting at 10 are added to each line in increments of 10. When saved by END, however, the file is stripped of line numbers.

In edit mode, only the following commands may be used:

<p>Edit commands</p> <p>LIST</p> <p>DELETE</p> <p>RENUM</p> <p>NEW</p> <p>FREE</p>
<p>Exit commands</p> <p>QUIT Exit to programing mode without saving file</p> <p>END Exit to programing mode after saving file</p>
<p>File management commands</p> <p>COPY</p> <p>DIR</p> <p>DEL(ERASE)</p> <p>RENAME</p> <p>CD(CHDIR)</p> <p>MD(MKDIR)</p> <p>RD(RMDIR)</p> <p>RENDIR</p>

SPEL III allocates for edit mode a memory area separate from that for programing mode, so that EDIT can be executed without losing the program retained in the source program area in main memory.

The number of characters in a line is limited to 79. At load time (without line numbers), the length of one line cannot exceed 74 characters. Characters exceeding this limitation will be truncated.

In edit mode, only edit-related commands are available. To use any other commands, such as XQT, return to programing mode using QUIT or END. While in edit mode, (even if switched to AUTO mode), the program cannot be executed.

**EXAMPLE**

```

EDIT CNFG.SYS
New file
>10 TASK=8
>20 ERRBUF=20
>30 LINBUF=256
>END
Edit End...file save
>

```

# END

&gt;

S

<b>FUNCTION</b>	Terminates program execution
<b>FORMAT</b>	END
<b>DESCRIPTION</b>	<p>Signifies the end of program, program execution does not proceed beyond this line number.</p> <p>For programs that contain subroutines, END at the end of the main program separates the main program from the subsequent subroutine. If END is omitted, the subroutine will be executed as part of the main program.</p> <p>In edit mode for text file by EDIT, END is used to exit to programming mode after saving file.</p>
<b>EXAMPLE</b>	<pre> 10 FUNCTION TEST 20 JUMP P5 30 IF SW(4)=0 THEN GOSUB BUZZER 40 JUMP P6 ;ON 0 ;WAIT 0.5 50 JUMP P7 ;OFF 0 ;WAIT 0.5 60 GOTO 20 70 END                                'End of program 80 ' 90 BUZZER: 100 ON 4 110 WAIT SW(4)=1 OR SW(5)=1 120 OFF 4 130 WAIT SW(4)=1 140 RETURN 150 ' 160 FEND </pre>

# ENTRY

S

**FUNCTION** Declares global variables

**FORMAT** ENTRY [variable type] [variable name]{,[variable name]}n

**DESCRIPTION** Declares specified variable(s) as global variable(s), and retains their values in a specially allocated symbol file.

With ENTRY, SPEL III provides support for global variables (i.e., variables that are shared among all files that comprise a program) in addition to local variables (i.e., variables that can be used only within a single file and have no effect outside the file).

Once a global variables has been declared, another file can refer to it by including EXTERN in that file.

Using global variables facilitates creating a program from multiple files through modular compilation and through linking.

**RELATED  
COMMANDS**

EXTERN, LINK

**EXAMPLE**

```
FILE 1(MAIN.PRG)
 10 FUNCTION MAIN
 20 INTEGER I
 30 ENTRY REAL WORK(100)
 40 EXTERN FUNCTION SUB_TASK
 50 WORK(5)=10.3
   :
1000 FEND
```

'I is local variable

'Declare global variable

```
FILE 2(INIT.PRG)
2000 FUNCTION SUB_TASK
2010 J=1
2020 EXTERN REAL WORK(100)
2030 A=J+WORK(5)
   :
3000 FEND
```

'Declare reference to global variable

'Use value of WORK(5) set in FILE 1

# ERA(0)

F

Error Axis

<b>FUNCTION</b>	Returns axis number in which error occurred
<b>FORMAT</b>	ERA(0)
<b>DESCRIPTION</b>	Returns axis number in which error occurred. Number which will be returned is 0 to 4. 0(zero) means that an axis is not causing error.
<b>RELATED COMMANDS</b>	ONERR, TRAP, ERR(0), ERL(0), ERT(0), ERD(0)

**EXAMPLE**

```

100 TRAP ERROR CALL ER_PRINT
    :
3000 FUNCTION ER_PRINT
3010 PRINT #20, "Error occurred in controller."
3020 PRINT #20, "Task number at which error occurred is ",ERT(0),"."
3030 IF ERA(0) THEN PRINT #20, "Axis which caused error is ",ERA(0),"."
3040 PRINT #20, "Error code is ",ERR(0),"."
3050 END
3060 FEND

```

E

# ERL(0)

F

Error line number

**FUNCTION** Returns line number in which error occurred

**FORMAT** ERL(0)

**DESCRIPTION** Returns line number in which error occurred.

ERL(0) is used with ONERR. If an error occurs in a program included within ONERR... RETURN, it may be difficult to locate the source of the error. ERL(0) returns the line number in which the error occurred.

Since the ECLR internal to the ONERR error processing program clears the error line number back to 0, be sure to execute ERL(0) prior to executing ECLR.

**RELATED COMMANDS** ONERR, ECLR, ERR(0)

## EXAMPLE

```
⋮
120 ONERR 1000
⋮
1000 'Error management process
1010 ELINE=ERL(0)
1020 ECODE=ERR(0)
1030 ECLR
1040 PRINT "error"
1050 PRINT "  line = ",ELINE
1060 PRINT "  code = ",ECODE
1070 RETURN
```

# ERR(0)

F

Error

**FUNCTION** Returns error number

**FORMAT** ERR(0)

\* The numeral 0 in ( )

**DESCRIPTION** ERR(0) is used with ONERR. If an error occurs in a program included within ONERR... RETURN, it may be difficult to know the error number. ERR(0) returns the error number.

Since the ECLR internal to the ONERR error processing program clears the error number back to 0, be sure to execute ERR(0) prior to executing ECLR.

**RELATED COMMANDS** ONERR, ECLR, ERL(0)

**EXAMPLE**

```

10 ONERR 60
20 FOR I=0 TO 199
30 JUMP PI
40 NEXT I
50 END
60 '
70 'ERR SUB
80 A=ERR(0)
90 PRINT A
100 ECLR
110 RETURN

```

E

# ERRHIST



## Error History

**FUNCTION** Displays error history

**FORMAT** ERRHIST {[number of errors to display]}

\* number of errors to display: integer from 1 to 50  
[default value: 20]

**DESCRIPTION** Displays a maximum of fifty of the most recently occurred errors.

The following items are displayed:

- error number
- axis number related to error
- line number in which error occurred
- task number
- date and time error occurred

If "number of errors to display" is specified, errors of specified number are displayed.

If "number of errors to display" is omitted, twenty of the most recently occurred errors are displayed.

If ERRBUF=30 is specified in the CNFG.SYS file, thirty errors are displayed.

Regarding error 2 (syntax error), if it is issued during program execution, only such errors are recorded.

If ONERR is used, such errors are not recorded.

Power off does not change ERRHIST memory or display items.

Power off is recorded as error 48.

## EXAMPLE

```
>ERRHIST 7
Error(axis)  Line  Task  Dev  Date    Time
           5                10-21  8:11:09p
           11                10-21  8:11:18p
            2                10-21  8:12:06p
          125 ( 2)          10-21  8:12:26p
            48                10-21  8:31:52p
          125 ( 2)          10-22  9:00:00a
          125 ( 2)   160    1      10-22  1:01:15p
>
```



---

# ERRMSG\$

---

F

Error Message

<b>FUNCTION</b>	Returns error message corresponding to specified error number
<b>FORMAT</b>	ERRMSG\$([error number])
<b>DESCRIPTION</b>	Returns error message which is described in ERROR CODE TABLE. Error message is displayed in the language which is specified by software switch SS1.
<b>RELATED COMMANDS</b>	ERR(0), ERL(0), ONERR, TRAP
<b>EXAMPLE</b>	>PRINT ERRMSG\$(124) Numeric value is out of allowable range.

E

# ERT(0)

&gt;

S

Error Task

**FUNCTION** Returns task number at which error occurred

**FORMAT** ERT(0)

\* The numeral 0(zero) in ( )

**DESCRIPTION** Returns task number at which error occurred.  
Number which will be returned is 1 to 16.

**RELATED  
COMMANDS** TRAP, ERR(0), ERD(0), ERA(0), ERL(0)

**EXAMPLE**

```
100 TRAP ERROR CALL ER_PRINT
   ⋮
3000 FUNCTION ER_PRINT
3010 PRINT #20, "Error occurred in controller."
3020 PRINT #20, "Task number at which error occurred is ",ERT(0),"."
3030 IF ERA(0) THEN PRINT #20, "Axis which caused error is ",ERA(0),"."
3040 PRINT #20, "Error code is ",ERR(0),"."
3050 END
3060 FEND
```

# EXTERN

S

External

**FUNCTION** Declares reference to global variables

**FORMAT** EXTERN [variable type] [variable name]{,[variable name]}n

**DESCRIPTION** Declares reference to global variable(s) declared in an ENTRY statement. Including EXTERN in a file enables the file to have access to the value of a global variable set in another file.

**RELATED  
COMMANDS** ENTRY

## EXAMPLE

```
FILE 1(MAIN.PRG)
  10 FUNCTION MAIN
  20 INTEGER I
  30 ENTRY REAL WORK(100)
  40 EXTERN FUNCTION INIT
  50 WORK(5)=10.3
    :
 1000 FEND
```

'I is local variable  
'Declare global variable

```
FILE 2(INIT.PRG)
 2000 FUNCTION INIT
 2010 J=1
 2020 EXTERN REAL WORK(100)
 2030 A=J+WORK(5)
    :
 3000 FEND
```

'Declare reference to global variable  
'Use value of WORK(5) set in FILE 1

E

# FILES



**FUNCTION** Displays name and size of files in file memory

**FORMAT** FILES {[pathname]}{[filename]}

**DESCRIPTION** Displays name and size of files in file memory. Also displays unused file memory. File size units are Kbytes.

If pathname is omitted, FILES assumes current directory.

Filename and extension may contain wildcard characters (\*, ?). If filename and extension are omitted, this is equivalent to entering \*.\* in place of the filename, and FILES displays name and size for all of the files and directories in the specified drive and directory.

**RELATED COMMANDS** DIR, WIDTH, DLOAD, DSAVE, DMERGE, KILL

**EXAMPLE**

```
>FILES
01      PRG    2
01      PNT    1
01      OBJ    2
01      SYM    1
ABC     CRV    1
02      PRG    2
02      PNT    1
EXAMPLE PRG    5
CNFG    SYS    1
AUTO    BAY    1
EXAMPLE PNT    1
EXAMPLE2 PNT    1
        space 181
>
```

# FIND



**FUNCTION** Searches for string

**FORMAT** FIND {"}[string]{"} {[drive name]}{[pathname]}{[filename]}

\* Filename extension is necessary

**DESCRIPTION** Searches for and displays line(s) of specified file that contains specified string.

For FIND to search for strings that contain small case alpha characters and spaces, the search string must be bracketed by quotation marks. If quotation marks are omitted, FIND searches for large case alpha characters when small case alpha characters are specified.

The search string may contain wildcard characters (\*, ?).

For FIND to search for wildcard characters (\*, ?) or \ as search string characters, precede each character (\*, ?, \) with a \, and specified in the search string as \\*, \?, or \\.

Filename and extension may contain wildcard characters (\*, ?).

If filename is omitted, FIND searches for specified string from main memory program, and displays the corresponding lines.

If pathname is omitted, FIND searches file(s) from current directory.

## EXAMPLE

For the filename ONOFF.PRG

```
>FIND L?D ONOFF.PRG
```

```
ONOFF.PRG
```

```
20 INTEGER LED
```

```
40 FOR LED=0 TO 31
```

```
50     ON LED
```

```
70 FOR LED=0 TO 31
```

```
80     OFF LED
```

```
>
```

```
>FIND ST*SW ONOFF.PRG
```

```
ONOFF.PRG
```

```
30 INTEGER STOPSW
```

```
100 IF SW(STOPSW)=0 THEN GOTO 40
```

```
>
```

```
>FIND "ON LED" ONOFF.PRG
```

```
50     ON LED
```

```
>
```

```
10 FUNCTION ONOFF
20 INTEGER LED
30 INTEGER STOPSW
40 FOR LED=0 TO 31
50     ON LED
60 NEXT
70 FOR LED=0 TO 31
80     OFF LED
90 NEXT
100 IF SW(STOPSW)=0 THEN GOTO 40
110 FEND
```

# FINE

&gt;

S

<b>FUNCTION</b>	Sets the allowable range for positioning completion checkout
<b>FORMAT</b>	<p>FINE {[axis #1 setting],[axis #2 setting],[axis #3 setting],[axis #4 setting]}</p> <p>* axis setting: integer from 0 to 32767</p>
<b>DESCRIPTION</b>	<p>Sets the allowable positioning range in pulses for each axis. Positioning is confirmed when all axes have arrived within the specified ranges.</p> <p>This positioning completion check begins after the sub CPU has completed sending the target position pulse to the servo system. Due to servo delay, the robot will not yet have reached the target position. This check continues to be executed every few milliseconds until each axis has arrived within the specified range setting. After positioning check has completed, control passes to the next command.</p> <p>With relatively large ranges, the positioning will be confirmed relatively early, which may cause the target positioning to be rough.</p> <p>Since the servo delay is normally less than a few thousand pulses, a range setting of a few thousand pulses or larger will result in no beneficial positioning effect.</p> <p>Depending on conditions, FINE settings, even if they are 0, do not always contribute to fine positioning.</p> <p>If settings for all four axes are omitted, FINE displays the current settings as follows :</p> <pre style="margin-left: 40px;">[axis #1 setting]    [axis #2 setting] [axis #3 setting]    [axis #4 setting]</pre> <p>When MOTOR ON, SLOCK, SFREE, or VERINIT are executed, FINE values are initialized to the default values, which vary from model to model.</p>
<b>RELATED COMMANDS</b>	GO, JUMP, MOVE, ARC, CVMOVE, PULSE
<b>EXAMPLE</b>	<pre>&gt;FINE 50,50,50,50 &gt; &gt;FINE       50    50       50    50</pre>

# FOR...NEXT

S

**FUNCTION** Executes a series of statements a specified number of times

**FORMAT** FOR [vn]=[iv] TO [final value] {STEP [increment value]}  
 :  
 NEXT {[vn]}

(replace vn with variable name)

(replace iv with initial value)

\* initial value, final value, increment value:  $\pm 0.000001$  to  $\pm 9999999$

nesting: up to 20

**DESCRIPTION** Executes a series of statements a specified number of times.

Number of loop repetitions (times for repeating the series of statements from FOR to NEXT) is determined by the initial value, final value, and increment value.

If increment value is omitted, it defaults to 1.

Increment value may be negative, but initial value must be greater than final value.

Nesting is allowable, a maximum of nineteen FOR...NEXT loops may be nested into the first FOR...NEXT loop. (For an explanation of nesting, refer to #include command.)

The variable name following NEXT may be omitted. For programs that contain nested FOR...NEXT loops, however, including the variable name following NEXT aids in quickly identifying loops.

**EXAMPLE**

```

10 FOR SP=10 TO 100 STEP 10      'Change speeds from 10 to 100 at 10 intervals,
20   SPEED SP                    jump from P1 to P5 at each speed
30   FOR I=1 TO 5
40     JUMP PI
50   NEXT I
60 NEXT SP

```

F

# FORMAT



**FUNCTION**        Formats file memory or floppy disk

**FORMAT**            FORMAT {/A}

**DESCRIPTION**     Formats file memory.

After specifying FORMAT parameters, FORMAT displays a prompt of the following form and waits for input:

```
RAM disk Format OK(Y/N)?
?_
```

To begin formatting, press Y, otherwise, press N.



Except when /A is specified, executing FORMAT erases all data in the file memory. Therefore backup data as necessary prior to executing FORMAT. If you want to know how to backup and restore, refer to "backing up and restoring a file" in chapter 3 of SPEL Editor manual or "[File] menu" of SPEL for Windows manual.

When FORMAT is executed, following message may be displayed.

```
Memory error !! address $xxxxxxxx
```

If the displayed address number with \$ is one of the below, it means the dip switch DSW1 on MPU board is set that additional RAM is installed while it is not installed actually. Turn off the corresponding bit of the software switch.

\$00080003	Bit 1 of SD2
\$00099001	Bit 1 and 2 of SD2

If other address number is displayed, please contact the authorized distributor of our robot.

/A selects, when the optional 1 Mbyte RAM modules have been added to the file memory, that file memory be formatted without erasing the current file memory contents. Install the RAMs by the steps described below.

- 1) Backup all of the data in main memory and file memory.



- 2) Turn off the controller, insert the additional RAM modules into MPU board. Refer to in "10.3 Additional RAM" of controller manual for details.
- 3) Execute FORMAT/A. The file memory is formatted without erasing the current file memory contents.

When /A is specified, you will be prompted as follows.

```
RAM disk size adjust OK(Y/N)?
?_
```

Then, if Y is entered, the file memory is formatted and contents size increases.

When FORMAT is executed, the message may be displayed "Memory error !! address \$xxxxxxx". If the address number, which starts with \$, is \$00080003 or \$00099001, it means the dip switch SD2 on MPU board is set so that additional RAM is installed while it is not installed actually. Check if RAMs are installed correctly and execute FORMAT/A again. If other address is displayed, please contact the authorized distributor of our robot.

\* When you want to increase the size of main memory (when you set the bit 1 and 2 of SD2 to the on position), initialize memory by SYSINIT command. Executing SYSINIT clears all data in main memory; object program, backup variables, point data, and source program. Restore all of backed up data to main memory.

- 4) Turn off the controller once, and turn on again.
- 5) Execute DIR command. Check if all files are displayed and free space is increased.
- 6) Execute DLOAD and DWNLD command and check if you can access a file correctly. When error 87 is occurred, it means a failure of formatting. In this case execute FORMAT without /A, restore the files that you backed up earlier to file memory.

#### EXAMPLE

```
>FORMAT
RAM disk Format OK(Y/N)?
?Y
```

# FREE



**FUNCTION** Displays available memory

**FORMAT** FREE

**DESCRIPTION** Displays available memory and unused variables.

In on-line mode, displays main memory available memory, in bytes, as follows:

PRG = Source program available memory

VAR = Unused backup variables, available memory

(Total number of variables, total memory) ← Values specified by LIBSIZE

OBJ = Object program available memory

In off-line mode, displays programing unit available memory, in bytes, as follows:

PRG = Source program available memory

PNT = Position data available memory

**EXAMPLE**

```
>FREE
```

```
PRG = 65536      'Source program available memory
```

```
VAR = 10,462    'Unused backup variables, available memory
```

```
      (10,512)  '(Total number of variables, total memory) ← Values specified by
                                     LIBSIZE
```

```
OBJ = 99660    'Object program available memory
```

```
>
```

# FUNCTION...FEND

S

Function...Function End

**FUNCTION** Defines Function

**FORMAT** FUNCTION [Function name]  
:  
FEND

**DESCRIPTION** The first line of a program must contain FUNCTION with Function name, the last line must be FEND. The program from FUNCTION to FEND is called a Function. When executing many tasks, or when using CALL statements, multiple Functions can be described in series (refer to example below.)

A Function name is used to call up each Function, the following restrictions apply to Function names:

- Eight characters or less.
- Only alphanumeric and underscore (\_).
- Either large or small case letters may be input, small case letters are converted to large case.
- First character must be an alpha character other than [P].
- Keywords (such as GO or FOR) are not allowed. Keywords followed by underscore or numerics are read as reserved words.

The first Function described in the program is called the "main Function." The main Function is executed in Task 1.

The main Function's XQT specifies task numbers and corresponding Function names, and directs execution for all subsequent Functions.

An object file can contain up to 69 functions. (A function starts with "FUNCTION" and ends with "FEND".) In order to incorporate more functions than 69 in the program, create a new object file and link to the original object file by using LINK command.

**RELATED COMMANDS** XQT, RUN, QUIT, HALT, CALL, LINK

F



```

EXAMPLE      10 FUNCTION MAIN
                20 '
                30 XQT !2 TASK2           'Execute Function TASK2 in Task 2
                40 XQT !3 TASK3           'Execute Function TASK3 in Task 3
                 :
                100 FEND
                110 '
                120 FUNCTION TASK2        'Function TASK 2
                130 ST2:
                140 WAIT SW($1)=1
                 :
                200 GOTO ST2
                210 FEND
                220 '
                230 FUNCTION TASK3        'Function TASK 3
                240 ST3:
                250 WAIT SW(1)=1
                 :
                300 GOTO ST3
                310 FEND
    
```

# GO

&gt;

S

**FUNCTION** Executes simultaneous all axes PTP motion, with deceleration and stop at specified position

**FORMAT**

(1) GO [ps] {/R} {TILL} {![parallel processing statement]!}  
 |L|

(2) GO [ps] {/R} {TILL SW(input bit number)} {= |0|} {![parallel processing statement]!}  
 |L| |1|

[ps] = position specification

**DESCRIPTION** Executes simultaneous all axes PTP motion, with deceleration and stop at specified position.

GO command uses the SPEED speed value and the ACCEL acceleration value.

In setting the arm's mode for the horizontal robot, specify either the right or left arm by declaring "/R" (for the right arm) or "/L" (for the left arm). Note that "/R" can be omitted while "/L" for the left arm cannot be omitted. Unless specified with "/L," all the set mode will apply only to the right arm.

TILL modifier is used to decelerate and stop the robot at an intermediate travel position if current TILL condition becomes satisfied. If TILL condition does not become satisfied, robot travels to the target position.

(1) GO with TILL Modifier:

Checks if current TILL condition becomes satisfied. If satisfied, this command completes by decelerating and stopping robot.

(2) GO with TILL Modifier, SW (input bit number) Modifier, and (0 or 1) Input Condition:

Checks if same line input condition becomes satisfied. If satisfied, this command completes by decelerating and stopping robot at an intermediate travel position.

GO with TILL Modifier, SW (input bit number) Modifier, but no Input Condition:

Input condition defaults to 1. If specified input bit becomes on, this command completes by decelerating and stopping robot at an intermediate travel position.

**NOTE**



GO differs from JUMP in that while GO Z axis motion is in a straight line from current position to target position, JUMP Z axis motion includes rising prior to horizontal travel, and lowering after horizontal travel. Since in using the GO there is a greater possibility of a hand colliding against a workpiece, in most cases, JUMP, and not GO, should be used.

**RELATED  
COMMANDS**

P=, ! ... !, SPEED, ACCEL, TILL, SW( ), PASS

**EXAMPLE**

100 TILL SW(1)=0 AND SW(2)=1	'Specifies TILL conditions (Input Bit 1 is off and Input Bit 2 is on)
110 GO P1 TILL	'Stop if current TILL condition (line 100) becomes satisfied
120 GO P2 TILL SW(2)=1	'Stop if Input Bit 2 is on
130 GO P3 TILL	'Stop if current TILL condition (line 100) is satisfied

# GOSUB...RETURN

S

Go to Subroutine

**FUNCTION** Branches to, executes, and returns from subroutine

**FORMAT**

```
GOSUB |line number|
      |label      |
      :
|line number|
|label      |
      :
RETURN
```

\* nesting: up to 10

**DESCRIPTION** Branches to specified line number or label, executes subroutine that follows, and return to main program.

Be sure to end each subroutine with RETURN. Doing so directs program execution to return to the line following GOSUB.

Nesting is allowable, a maximum of nine GOSUB...RETURN loops may be nested into the first GOSUB...RETURN loop. (For an explanation of nesting, refer to #include command.)

The combined total number of GOSUB and GOTO commands is limited to 447. Attempting to exceed 448 will result in an error.

To specify as a label, place ":" (colon) after the label. Other than the restrictions given below, the labels may be freely named.

Label:

- Within eight characters
- The first character must be an alphabet other than "P".
- Alphanumerics and underscores (\_)
- No distinction between capital and small case

**RELATED COMMANDS** GOTO

**EXAMPLE**

```
10 GOSUB WAIFDR
20 JUMP P1; JUMP P3
30 JUMP P*:Z0
   :
300 WAIFDR:          'label
310 WAIT SW($1)=1
320 OFF $1
330 RETURN
```

G

# GOTO

S

Go to

**FUNCTION**            Branches to specified line number

**FORMAT**             GOTO |line number |  
                              |label            |

**DESCRIPTION**        Branches unconditionally to specified line number or label.

If the specified line number or label does not exist, Error 8 will occur.

The combined total number of GOTO and GOSUB commands is limited to 447.  
Attempting to exceed 448 will result in an error.

To specify as a label, place ":" (colon) after the label. Other than the restrictions given below, the labels may be freely named.

Label:

- Within eight characters
- The first character must be an alphabet other than "P".
- Alphanumerics and underscores (\_)
- No distinction between capital and small case

**EXAMPLE**

```
10 SPEED 100
20 JUMP P50
30 WAIT 0.5
40 JUMP P80
50 WAIT 1.2
60 GOTO 20                    'Branch to line 20, continue execution
```

**EXAMPLE 2**

```
100 GOTO TIMER                'Branch to line labeled TIMER, continue execution
   :
300 TIMER:
```



# HALT

&gt;

S

**FUNCTION** Temporarily stops execution of task in progress

**FORMAT** HALT ![Task number]

\* Task number: integer from 1 to 16

**DESCRIPTION** Temporarily stops (halts) execution of task in.  
To resume execution of the halted task, use RESUME.  
To complete halted task, use QUIT.

**RELATED COMMANDS** RUN, XQT, QUIT, RESUME

**EXAMPLE**

```

10 FUNCTION MAIN
20 XQT !2 FLASH      'Execute Function FLASH as Task 2
30 LOOP:
40 WAIT 3           'Set timer interval to 3 seconds (Task 2 is executing)
50 HALT !2          'Halt Task 2 (after line 40, 3 seconds has elapsed)
60 WAIT 3           'Set timer interval to 3 seconds (Task 2 is stopped)
70 RESUME !2        'Resume Task 2 (after line 60, 3 seconds has elapsed)
80 GOTO LOOP
90 FEND
100 '
110 FUNCTION FLASH  'Flash light on/off at 0.2 seconds intervals
120 LOOP1:
130 ON 1
140 WAIT 0.2
150 OFF 1
160 WAIT 0.2
170 GOTO LOOP1
180 FEND

```

H

# HOFS



H Offset

**FUNCTION** Specifies and displays offset pulse used for software zero point correction

**FORMAT** (1) HOFS [axis #1 offset value],[axis #2 offset value],[axis #3 offset value],~  
[axis #4 offset value]

(2) HOFS

**DESCRIPTION** (1) Specifies offset from the encoder 0 point to the mechanical 0 point, this offset is used for software 0 point correction.

Although robot motion control is based on the zero point of the encoder mounted on each axis motor, the encoder zero point may not necessarily match the robot mechanical zero point. The HOFS offset pulse correction pulse is used to carry out a software correction to the mechanical 0 point based on the encoder 0 point.

HOFS values are not changed by power off, or by executing VERINIT.

HOFS values can be specified directly.

To have HOFS values automatically calculated, move the arm to a certain calibration position, and execute CALIB. The controller automatically calculates HOFS values based on CALPLS pulse values and calibration position pulse values.

(2) Displays current HOFS values as follows:

[Axis #1 value] [Axis #2 value]  
[Axis #3 value] [Axis #4 value]



The HOFS value is correctly specified prior to delivery. There is a danger that unnecessarily changing the HOFS value may result in positional errors and unpredictable motion. Therefore, we strongly recommend that HOFS values not be changed unless absolutely necessary.

**RELATED COMMANDS** CALIB, CALPLS

**EXAMPLE** HOFS 100,120,50,0  
>  
HOFS  
100 120  
50 0

---

# HOME

---

&gt;

S

<b>FUNCTION</b>	Executes motion to home (standby) position
<b>FORMAT</b>	HOME
<b>DESCRIPTION</b>	<p>Executes low speed PTP motion to home (standby) position specified by HOMESET, in (HORDR value) order.</p> <p>When robot is in home position, the controller's REMOTE 2 connector HOME output becomes active.</p>
<b>RELATED COMMANDS</b>	HOMESSET, HORDR
<b>EXAMPLE</b>	>HOME >

**H**

# HOMES

&gt;

S

<b>FUNCTION</b>	Specifies and displays home (standby) position
<b>FORMAT</b>	<p>(1) HOMES [axis #1 pulse value],[axis #2 pulse value],[axis #3 pulse value],~  <span style="float: right;">[axis #4 pulse value]</span></p> <p style="padding-left: 40px;">* pulse value: integer</p> <p>(2) HOMES</p>
<b>DESCRIPTION</b>	<p>(1) Specifies HOME position with pulse values.</p> <p style="padding-left: 40px;">HORDR specifies and displays the order of axis motion for HOME execution.</p> <p>(2) Displays current HOMES values.</p> <p>Executing VERINIT deletes HOMES values.</p>
<b>RELATED COMMANDS</b>	HOME, HORDR
<b>EXAMPLE</b>	<pre> &gt;HOME !!Error 143                'Attempting to execute HOME without HOMES values                            causes error  &gt;HOMES !!Error 143                'Attempting to display home position pulse values without                            HOMES values causes error  &gt;HOMES 0,0,0,0 &gt;HOMES       0          0       0          0  &gt;HOME &gt;_  &gt;HOMES PLS(1),PLS(2),PLS(3),PLS(4) &gt;                            'Using PLS function, specify current position as home                            position </pre>

# HORDR

&gt;

S

Home Order

**FUNCTION** Specifies and displays HOME command axis motion order

**FORMAT** (1) HORDR [specification value 1],[specification value 2],~  
[specification value 3],[specification value 4]

(2) HORDR

**DESCRIPTION** (1) Specifies HOME command axis motion order.

The axis (or group of axes) specified by value 1 execute motion first. At the completion of first motion, the axis (axes) specified by value 2 execute motion, and so on, until home position is reached.

Each axis is assigned a bit 0 to 3 as follows:

Axis	Axis #1	Axis #2	Axis #3	Axis #4
Bit Number	bit3	bit2	bit1	bit0
Binary Code	&B1000	&B0100	&B0010	&B0001

HORDR values are initialized to default values by executing VERINT.

Refer to the "Specifications" in the manipulator manual for default values, which vary from model to model.

(2) Displays current HORDR values in hexadecimal notation as follows:

```
[specification value 1]    [specification value 2]
[specification value 3]    [specification value 4]
```

## RELATED COMMANDS

HOME, HOMESET

## EXAMPLE

```
>HORDR &B0010,&B1000,&B0100,&B0001    'Specifies HORDR order (axis #3,
axis #1, axis #2, then axis #4)
>HORDR
    02    08                                'Display in hexadecimal
    04    01
```

H

# HOUR



---

**FUNCTION**        Displays controller accumulated operating

**FORMAT**         HOUR

**DESCRIPTION**    Displays, as an integer, in hours, accumulated controller time.

**RELATED  
COMMANDS**        TIME( )

**EXAMPLE**        >HOUR  
                    2560  
                    >

# HTASK

&gt;

S

Halt Task

**FUNCTION** Specifies tasks to be temporarily stopped by PAUSE command or PAUSE input

**FORMAT** HTASK {[0,]}[Task number]{,[Task number]}n

\* Task number: integer from 1 to 16  
The numeral 0(zero)

**DESCRIPTION** Specifies tasks to be temporarily stopped by PAUSE command or PAUSE input. PAUSE input is performed from 3 ports shown below.

The operating unit PAUSE switch  
REMOTE1 connector Safety door input  
REMOTE3 connector PAUSE input terminal

Tasks not defined by HTASK cannot be stopped by either PAUSE command or PAUSE input. However, Safety door input from REMOTE1 stops robot control task whether the task is defined by HTASK or not.

## NOTE



In case of controller SRC-300, restart after temporary stop is accomplished by inputting START from operating unit or REMOTE3. After temporary stop, all tasks are completely stopped by inputting RESET.

In case of controller SRC-320, how to restart is different from TEACH mode and AUTO mode. In AUTO mode, same way as SRC-300 is used. In TEACH mode, restart and reset are accomplished from programming unit (PC). Refer to the SPEL Editor or SPEL for Windows manual.

"0" parameter specifies START LED on/off status when the task specified by HTASK is temporarily stopped while other tasks are in operation.

Its status is as follows:

	START LED	PAUSE LED
Omit "0"	OFF	ON
Specify "0"	ON	ON

After turning power on, initial HTASK value causes all tasks to be temporarily stopped by PAUSE command execution or PAUSE input. In addition, 0(zero) is not specified. Therefore if after turning power on, HTASK values are not specified, executing PAUSE command or inputting PAUSE will cause all tasks to be temporarily stopped.

HTASK setting is not changed unless HTASK values are specified again, or the power is turned off. Other procedures such as RESET command input or RESET switch input does not change HTASK setting.

**NOTE**



- The task which controls robot motion must be specified to be temporarily stopped when pause is inputted.
- Specify HTASK value only once at Task 1. Do not specify it repeatedly in one program.
- If HTASK values are specified once, the setting is effective for all programs to be executed unless the power is turned off. If HTASK is used, specify HTASK values in all programs including files in file memory to stop all tasks when pause is inputted.

**EXAMPLE**

20 HTASK 1, 3, 4

'If PAUSE is inputted, Tasks 1, 3 and 4 will temporarily stop



# HTEST

INC

&gt;

**FUNCTION** Displays HTEST values

**FORMAT** HTEST

**DESCRIPTION** Displays HTEST values of all axes as follows.

```
[axis #1 HTEST value] [axis #2 HTEST value]
[axis #3 HTEST value] [axis #4 HTEST value]
```

HTEST value means the difference between the actual pulse value and the logical pulse value in calibration with MCAL where home sensor is turned on to where encoder Z phase signal is detected.

**RELATED COMMANDS** MCAL, HOFS

**EXAMPLE**

```
>HTEST
176 -75
218 59
>
```

H

# IF...THEN...ELSE...ENDIF

S

**FUNCTION** Conditional statement execution

**FORMAT** (1) IF [conditional expression] THEN

```

:
{ELSE}
:
ENDIF

```

\* nesting: up to 20 are allowed.

(2) IF [conditional expression] THEN [statement]{:[statement]}n~

{ELSE [statement]{:[statement]}n}

**DESCRIPTION** (1) Specifies IF condition.

If IF condition is satisfied, the statements between IF and ELSE are executed. If IF condition is not satisfied, the statements between ELSE and ENDIF are executed. ELSE through ENDIF may be omitted, then control passes to the line immediately following ENDIF, if IF condition is not satisfied.

Multiple IF...ENDIF statements may be nested.

IF condition may contain any operators supported by SPEL III.

(2) If IF condition specified in this line is satisfied, the statements between THEN and ELSE are executed. If not satisfied, the statements following ELSE are executed. ELSE onward may be omitted. If omitted, then control passes to the next line, if IF condition is not satisfied.

**EXAMPLE**

```

100 IF SW(0)=1 THEN PRINT "i0 ON" ELSE PRINT "i0 OFF"
110 '
120 IF SW(1)=1 THEN
130   IF SW(2)=1 THEN
140     PRINT "i1 ON i2 ON"
150   ELSE
160     PRINT "i1 ON i2 OFF"
170   ENDIF
180 ELSE
190   IF SW(2)=1 THEN
200     PRINT "i1 OFF i2 ON"
210   ELSE
220     PRINT "i1 OFF i2 OFF"
230   ENDIF
240 ENDIF

```

# IN( )

F

Input

**FUNCTION** Returns input status of (8 bit input) input port

**FORMAT** IN([port number])

\* port number: integer from 0 to 15

**DESCRIPTION** Returns, as a decimal, input status of input port specified by port number.

Each port is comprised of 8 input bits, the standard 16I/16O is made up of 2 ports.

The port number, LSB/MSB, and input bit number are related as follows:

port number	MSB							LSB
	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8

LSB : least significant bit

MSB: most significant bit

**RELATED COMMANDS** OUT, INBCD( ), OPBCD

**EXAMPLE** A=IN( 1 ) 'Store at variable A the input status of input port 1 (input bit 8 to 15), as a decimal.  
For example, if bit 9 and bit 10 are on and the remaining bits are off, returns 6 to A.

# IN(\$ )

F

Input \$

**FUNCTION** Returns input status of (8 bit unit) memory I/O port

**FORMAT** IN(\$[port number])

\* port number: integer from 0 to 63

**DESCRIPTION** Returns, as a decimal, input status of memory I/O port specified by port number.

Each port is comprised of 8 memory I/O bits. Since there are a total of 512 bits, memory I/O is made up of 64 ports.

The port number, LSB/MSB, and memory I/O bit number are related as follows:

port number	MSB 7	6	5	4	3	2	1	LSB 0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
...								
62	503	502	501	500	499	498	497	496
63	511	510	509	508	507	506	505	504

LSB : least significant bit

MSB : most significant bit

**RELATED COMMANDS** OUT\$

**EXAMPLE** A=IN(\$62) 'Store at variable A the input status of input port 62 (Bit 496 to 503), as a decimal.  
For example, if bit 500 to 502 are on and the remaining bits are off, returns 112 to A.

# INBCD( )

F

Input by Binary Coded Decimal

**FUNCTION** Returns input status of (8 bit input) input port as BCD

**FORMAT** INBCD([port number])

\* port number: integer from 0 to 15

**DESCRIPTION** Returns, as a ("upper rank 4, lower rank 4") binary coded decimal (0 to 9), input status of input port specified by port number.

Each port is comprised of 8 input bits, the standard 16I/16O is made up of 2 ports.

The port number, LSB/MSB, and input bit number are related as follows:

port number	MSB							LSB
	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8

LSB : least significant bit

MSB : most significant bit

**RELATED COMMANDS** OPBCD, IN( ), OUT

**EXAMPLE**

```
>B=INBCD(1)           'Store at variable B the input status of input port 1 returned as
                        BCD
>PRINT INBCD(1)       'Display status of input bit 8 to 15 as BCD
                        (For this example, port channel is 01010011)
53
```

# INBIN

F

**FUNCTION** Returns the value of input port specified by starting port number and number of ports

**FORMAT** INBIN([starting port number],[number of ports],[presence of parity check])

\* starting port number : integer from 0 to 127  
 number of ports : integer from 0 to 15 (0=1bit, 15=16bit)  
 presence of parity check : 0=none, 1=even, 1=odd

**DESCRIPTION** Returns the value of input port specified by starting port number and number of ports as binary data. IN( ) delimits the input port in eight bits. INBIN( ) can specify an arbitrary starting port and number of ports. The maximum number of ports is 16.

If the parity check is specified, the range of the port becomes 15 bits or less and the last bit is a parity bit. When the parity error occurs, (-1) is returned.



The value of starting port number + number of ports should not exceed the number of input ports installed in the controller.

**RELATED COMMANDS** IN( ), INBCD( ), INBIT( )

**EXAMPLE** >PRINT INBIN( 2 , 5 , 0 )  
 18

'Returns the value of input port 2 - 6.

'When the ports are in the following condition, it returns 18.

bit 6	bit 5	bit 4	bit 3	bit 2
1	0	0	1	0

# INBIT

F

**FUNCTION** Returns the bit number which is turned on among input ports specified by starting port number and number of ports

**FORMAT** INBIT([starting port number],[number of ports])

\* starting port number : integer from 0 to 127  
 number of ports : integer from 0 to 15 (0=1bit, 15=16bit)

**DESCRIPTION** Returns the bit number which is turned on among input ports specified by starting port number and number of ports.

When two or more bits are turned on, (-1) is returned.

No bit is turned on, 0 is returned.



The value of starting port number + number of ports should not exceed the number of input ports installed in the controller.

**RELATED COMMANDS** IN( ), INBCD( ), INBIN( )

**EXAMPLE** >PRINT INBIT(2,5,0) 'Returns the bit number which is turned on among input port 2 - 6.  
 2 'When the ports are in the following condition, the returning value is 2.

bit 6	bit 5	bit 4	bit 3	bit 2
0	0	0	1	0

# INPUT

&gt;

S

<b>FUNCTION</b>	Inputs data from console keyboard
<b>FORMAT</b>	INPUT [variable name 1]{,[variable name n]}n
<b>DESCRIPTION</b>	<p>Allows numeric values and strings to be input from console, and stores them at variables.</p> <p>A single INPUT can specify multiple variable names. Variable names are to be separated by commas ( , ).</p> <p>Numeric variable names and string variable names are allowed, input data type must match the variable type.</p> <p>In executing INPUT, a (?) prompt appears at the console. After inputting data press the return key.</p> <p>For multiple variables, the number of input datum must match the number of INPUT variable names. Also, data are to be separated by commas ( , ). In this case, the comma is called a delimiter.</p> <p>When inputting numerics, take note of the following:</p> <ul style="list-style-type: none"> <li>* Characters other than numerics are ignored. An error occurs if the number of datum does not match the number of INPUT numeric variable names.</li> <li>* If numerics and alpha characters are mixed, only the first numeric(s) surrounded by alpha characters are stored at the variable.</li> </ul> <p>&lt; Example &gt;</p> <pre>A123BC45      '(123) is stored</pre> <p>When inputting strings, numerics and alpha characters are allowed.</p>
<b>RELATED COMMANDS</b>	PRINT
<b>EXAMPLE</b>	<pre>&gt;INPUT A      'Specify numeric variable A ?52          'Store 52 at numeric variable A &gt;INPUT A\$,B\$  'Specify string variables A\$ and B\$ ?ROBOT, WORLD 'Store ROBOT at string variable A\$, WORLD at string               variable B\$ &gt;</pre>



# INPUT #



<b>FUNCTION</b>	Inputs data from communication port
<b>FORMAT</b>	<p>INPUT #[port number],[variable name 1]{,[variable name n]}n</p> <p>* port number: integer from 20 to 23</p>
<b>DESCRIPTION</b>	<p>Accepts numeric values and strings from communication port specified by port number, stores them at variables.</p> <p>An INPUT # can specify multiple variable names. Variable names are to be separated by commas ( , ).</p> <p>For multiple variables, the number of input datum must match the number of INPUT # variable names.</p> <p>Numeric variable names and string variable names are allowed, input data type must match the variable type.</p> <p>When inputting numerics, take note of the following:</p> <ul style="list-style-type: none"> <li>* Characters other than numerics are ignored. An error occurs if the number of datum does not match the number of INPUT numeric variable names.</li> <li>* If numerics and alpha characters are mixed, only the first numeric(s) surrounded by alpha characters are stored at the variable.</li> </ul> <p>&lt; Example &gt;</p> <p style="padding-left: 40px;">A123BC45          '(123) is stored</p> <p>When inputting strings, numerics and alpha characters are allowed.</p>
<b>RELATED COMMANDS</b>	LOF( ), PRINT #
<b>EXAMPLE</b>	<p>&gt;INPUT #20 , A          'Specify accepting one numeric datum from port 20, storing it at numeric variable A</p> <p>&gt;INPUT #21 , B , C      'Specify accepting two numeric datum from port 21, storing one at numeric variable B and the other at numeric variable C</p> <p>&gt;INPUT #20 , A\$          'Specify accepting string from port 20, storing at string variable A\$</p>

# INT( )

F

Integer

**FUNCTION** Returns the largest integer that is less than or equal to specified value

**FORMAT** INT([numeric value])

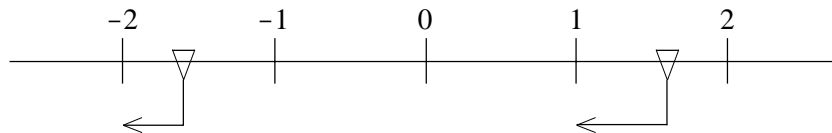
**DESCRIPTION** Returns the largest integer that is less than or equal to specified value.

Note that for negative specified values, the returned integer will have a larger absolute value than the specified value. Refer to the following example:

< Example >

INT(1.55)            1

INT(-1.55)          -2



**RELATED COMMANDS** ABS( ), SGN( ), SQR( )

**EXAMPLE**

```
>PRINT INT(111.55)
111
>PRINT SGN(-111.55)
-112
>
```

# INTEGER

S

Integer

**FUNCTION** Defines 2-byte integer type variables

**FORMAT** INTEGER [variable name]{{(array size 1{,array size 2{,array size 3})}} ~  
 {,[variable name]{{(array size 1{,array size 2{,array size 3})}}}}n

**DESCRIPTION** This command defines 2-byte integer type variables.

If several variables of the same type are declared, use a " , " (comma) and describe several variable names. When defining an array variable, declare the name and its size enclose with ( ). The size can be defined up to 3 dimension.

By default, the variable whose data type is not declared specifically will be treated as REAL type. Therefore, it should be declared if the integer type is sufficient or the data type because it has advantage over the REAL type in terms of performance speed and memory efficiency.

Also, there are three types to the integer type and, each handles a different size of data as follows: BYTE (1-byte integer), INTEGER (a 2-byte integer) and LONG (a 4-byte integer). The range for an INTEGER type value is -32768 to 32767. If a value outside this range is entered, it causes an error.

For more details about the variable, refer to "2.3 Variables" in the Elementary section of the Use's manual for SRC-300/320.

The following cites the restrictions on the variable names. The variable may be freely named within these restrictions.

- The usable characters are alphanumeric and underscores ( \_ ). There is no distinction between capital and small case letters.
- Must be eight characters or under.
- The first character must be an alphabet character other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. A reserved word that is followed by an underscore or numeric character is also read as a reserved word.

**NOTE**



The variable type must be declared at the beginning of a line; otherwise, the file will not be successfully compiled. When declaring another type of variable, a new line must be created.

**RELATED  
COMMANDS**

BYTE, LONG, REAL, DOUBLE, STRING, VARIABLE, SYS

**EXAMPLE**

```
10 FUNCTION MAIN
20 INTEGER I           'Declares a 2-byte integer type variable "I."
30 REAL ODATA(10,10)  'Declares a 2 dimensional array of 2-byte integer
                       type variable, "ODATA."
   :
999 FEND
```

# JRANGE

&gt;

S

Joint Range

<b>FUNCTION</b>	Defines permissible working range of specified axis in pulses	
<b>FORMAT</b>	JRANGE [axis number],[lower limit pulse value],[upper limit pulse value]	
	* axis number: integer from 1 to 4	
<b>DESCRIPTION</b>	<p>Defines permissible working range of specified axis with upper and lower limit in pulses. For RANGE command, all parameters for upper and lower limit of all axes must be specified.</p> <p>However, number of necessary parameters for JRANGE is 3, it is useful to define working range of one axis only.</p> <p>Lower limit must not exceed upper limit. A lower limit in excess of upper limit will cause an error, making it impossible to execute motion command.</p> <p>To confirm defined working range, use RANGE command.</p> <p>Refer to the "Specifications" in the manipulator manual for maximum working ranges, which vary from model to model.</p> <p>Neither power off nor executing VERINIT changes JRANGE setting values.</p>	
<b>RELATED COMMANDS</b>	RANGE, LIMPLS	
<b>EXAMPLE</b>	>JRANGE 2, -60000, 70000	'Define the 2nd axis range as -6000 to 70000

# JS(0)

F

Jump Sense

**FUNCTION** Returns whether or not SENSE condition has been satisfied after JUMP SENSE execution has completed

**FORMAT** JS(0)

**DESCRIPTION** Returns whether or not SENSE condition has been satisfied after JUMP with SENSE modifier execution has completed.

1: SENSE condition has been satisfied, currently stopped above target position.

0: SENSE condition has not been satisfied, jump is completed at target position.

**RELATED COMMANDS** SENSE, JUMP, STAT(1)

## EXAMPLE

```

:
100 SENSE SW(0)=1
:
200 JUMP P1 C2 SENSE
210 IF JS(0)=1 THEN GOSUB ERRPRC;GOTO 200
220 ON 1
:
1000 ERRPRC:
:

```

The above program directs motion to P1, assembles a part, and then turns output bit 1 on. When a "material supply failure" signal from a peripheral device has been received at input bit 0, program branches to subroutine ERRPRC to reestablish material supply. The input bit condition is checked as the third (Z) axis downward motion begins.

# JUMP

&gt;

S

**FUNCTION** Directs PTP movement with gate motion

**FORMAT** JUMP [ps] {/R} {C[arch number]} {LIMZ[Z coordinate value]}~  
 |L|  
 {|SENSE|} {![parallel processing statement]!}  
 | TILL |

\* arch number: integer from 0 to 7

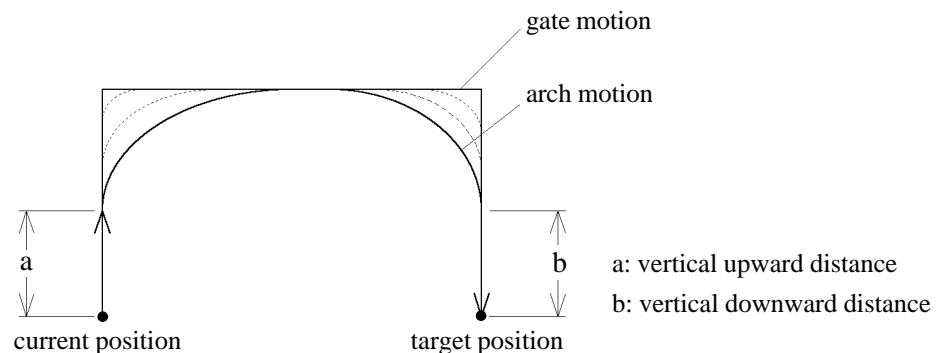
[ps] = position specification

**DESCRIPTION** Directs PTP movement of gate shape to target position.

In gate motion, axis #3 rises from current position to Z coordinate highest position ( $z=0$ ), travels horizontally to above target position while rotating to specified axis #4 angle, then axis #3 goes down to target position.

In setting the arm's mode for the horizontal robot, specify either the right or left arm by declaring "/R" (for the right arm) or "/L" (for the left arm). Note that "/R" can be omitted while "/L" for the left arm cannot be omitted. Unless specified with "/L," all the set mode will apply only to the right arm.

C [arch number] determines the arch trajectory while rising from current position, traveling horizontally, and lowering to target position. This is called arch motion.



Default value of each arch number is shown below.

To change the values, use ARCH command.

arch number	0	1	2	3	4	5	6	7
a	30	40	50	60	70	80	90	gate motion
b	30	40	50	60	70	80	90	

(mm)



Arch motion trajectory is compounded of vertical motion and horizontal motion. It is not a continuous path control. Therefore, the actual trajectories of arch motion are not decided uniquely by ARCH parameters. The trajectory changes depending on the motion and speed. Execute JUMP with actual motion and speed to confirm the actual trajectory.

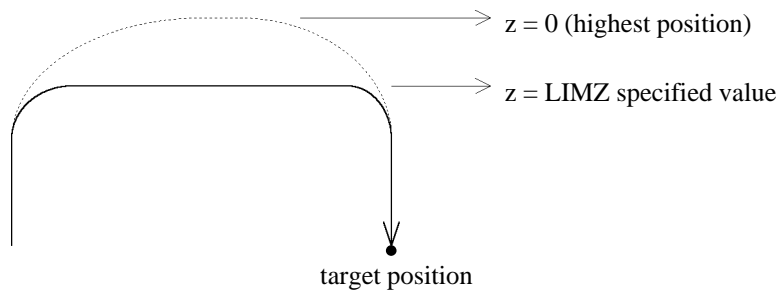
- Even if JUMP commands with same arch number are executed at an same position trajectories are different by motion speed. Lower trajectory with lower motion speed. If JUMP is executed with high speed to confirm an arch motion trajectory hand may crash into obstacle with lower speed.
- In a trajectory, the vertical upward distance increases and the vertical downward distance decreases when the motion speed is set high. When the vertical downward distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the parameter of vertical downward distance long.
- Even if JUMP commands with same distance are executed trajectories are different by motion. The change of trajectory depending on the motion is various. For general example, in case of SCARA robot, the vertical upward distance increases and the vertical downward distance decreases when the movement of the first arm is large. When the vertical downward distance of the trajectory is shorter than the expected, lower the speed and/or the deceleration, or change the parameter of vertical downward distance long.

If LIMZ [Z coordinate value] is specified, 3rd axis rises to the specified height limit, and travels horizontally.

When LIMZ modifier is omitted, robot rises to the Z axis height specified by currently valid LIMZ value.

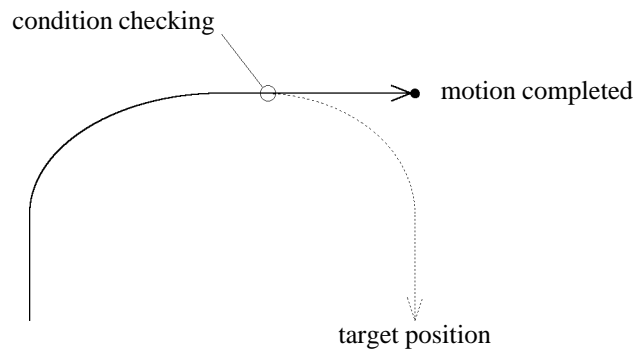
LIMZ Z axis height limit specification is the Z axis value for the robot coordinate system. It is not the Z axis value for ARM, TOOL, or LOCAL0 coordinates. Therefore take necessary precautions when using tools or hands with different operating heights.

JUMP with SENSE Modifier:





Checks if current SENSE condition is satisfied before axis #3 going down. If satisfied, this command completes with robot stopped above target position. Use JS(0) or STAT(1) to verify whether SENSE condition has been satisfied and this command has been completed, or SENSE condition has not been satisfied and robot stopped at target position.



#### JUMP with TILL Modifier:

Checks if current TILL condition is satisfied during robot motion until axis #3 starts going down. TILL condition is not checked after axis #3 starts going down. Therefore take necessary precautions when using arch motion.

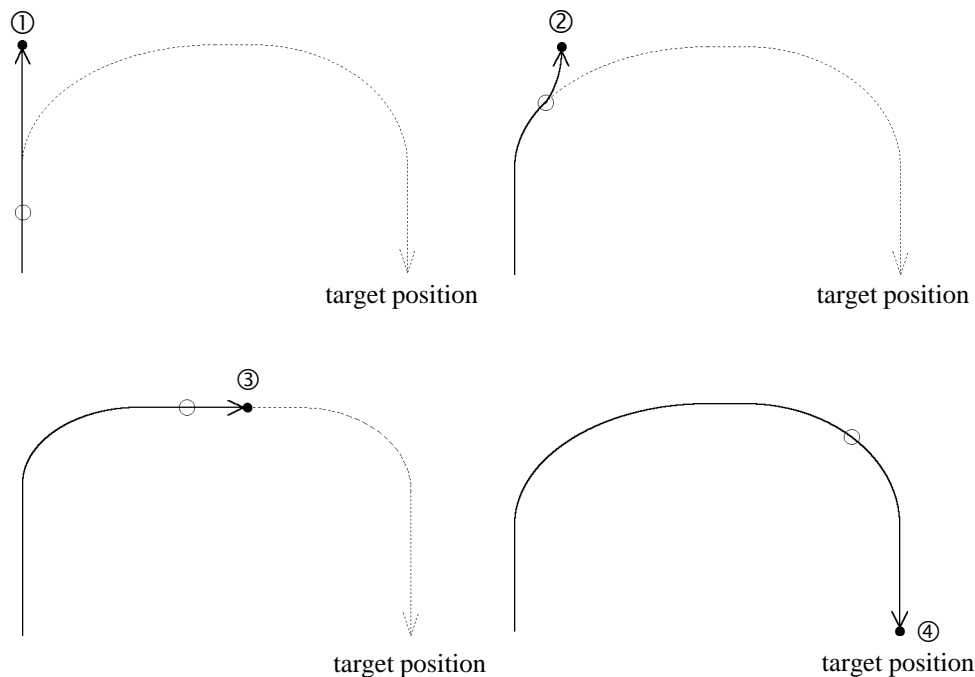
If satisfied, this command completes by decelerating and stopping robot. Robot stops at the specified height by LIMZ specification. If LIMZ is not specified, robot moves to  $z = 0$  height.

Use STAT(1) to verify whether TILL condition has been satisfied and this command has been completed, or TILL condition has not been satisfied and robot stopped at target position.

Position where condition satisfied and motion stop condition :

①	during vertical rising motion	goes up and stops
②	during combined motion (horizontal and vertical)	horizontal motion - decelerates and stops axis #3 motion - goes up and stops
③	during horizontal motion	decelerates and stops
④	after start lowering	stops at target position

- position where condition satisfied
- stop position



Parallel processing statement can direct certain commands be executed during horizontal travel. For details, refer to the "! ... !" parallel processing section.

To specify speed and acceleration, use **SPEED** and **ACCEL**. For **JUMP** command, it is possible to separately specify speeds and accelerations for Z axis upward motion, horizontal travel including U axis rotation, and Z axis downward motion.

**RELATED  
COMMANDS**

P=, ! ... !, **SPEED**, **ACCEL**, **LIMZ**, **SENSE**, **TILL**, **JS(0)**, **STAT(1)**

**EXAMPLE**

```
>JUMP P10+X50 C0 LIMZ-20 SENSE !D50;ON 0;D80;ON 1!  
>
```

# KILL



**FUNCTION** Deletes file(s)

**FORMAT** KILL "[{pathname}]{filename}"

**DESCRIPTION** Deletes specified files.

Filename extensions may be present or omitted.

If filename and extension is specified, only that file is deleted.

If filename is specified but filename extension is omitted, the following files will be deleted, in order:

filename.PRG (source program file)

filename.PNT (position data file)

filename.OBJ (object file)

filename.SYM (symbol table file)

In this case, if either filename.PRG or filename.PNT does not exist an error will occur. If filename.PRG exists but filename.PNT does not, the error will occur after filename.PRG is deleted.

If pathname is omitted, KILL deletes file from current directory.

Filename and extension may contain wildcard characters (\*, ?).

**RELATED COMMANDS** DEL, FILES, DIR, DSAVE

**EXAMPLE**

```
>FILES
ABCD PRG 2
ABCD PNT 1
space 59
>KILL "ABCD.PRG"
>FILES
ABCD PNT 1
space 61
```

---

# LEFT\$( )

---

F

<b>FUNCTION</b>	Returns the leftmost characters of specified string
<b>FORMAT</b>	LEFT\$([string variable name]  ,[number of characters] )  "[string]"
<b>DESCRIPTION</b>	Returns the leftmost specified number of characters of specified string. String may be specified by a string variable.
<b>RELATED COMMANDS</b>	RIGHT\$( ), MID\$( )
<b>EXAMPLE</b>	<pre>10 FUNCTION MAIN 20 STRING LETTER\$ 30 LETTER\$="ABCDE" 40 PRINT LEFT\$(LETTER\$,2) 50 FEND &gt;RUN COMPILE END AB &gt;</pre>

---

# LEN( )

---

F

Length

**FUNCTION** Returns number of characters of specified string

**FORMAT** LEN( [[string variable name]]  
|"[string]" |

**DESCRIPTION** Returns number of characters of specified string.  
String may be specified by a string variable.

**EXAMPLE** >PRINT LEN( "ABCDE" )  
5  
>

L

---

# LIBRARY

---



<b>FUNCTION</b>	Displays backup variable names in memory
<b>FORMAT</b>	LIBRARY
<b>DESCRIPTION</b>	Displays all backup variable names in memory, also displays types of variable.
<b>RELATED COMMANDS</b>	SYS, VARIABLE
<b>EXAMPLE</b>	<pre>&gt;LIST 10 FUNCTION BACKUPV 20 SYS BYTE ER_RB(10),ER_RA(5) 30 SYS REAL V(20) 40 FEND &gt;RUN COMPILE END &gt;LIBRARY   BYTE ER_RB(10)   BYTE ER_RA(5)   REAL V(20) &gt;</pre>

# LIBSIZE, SIZE



Library Size

**FUNCTION** Specifies and displays number of usable backup variables and available memory

**FORMAT** (1) LIBSIZE [number of backup variables],[available memory]

\* number of backup variables: integer from 0 to 794 [default values: 10]  
 available memory: 14 to 32768 bytes [default values: 512]

(2) LIBSIZE

**DESCRIPTION** (1) Specifies number of usable backup variables and available memory.

Array variables, regardless of the size of their array, are counted as one variable.

After entering command, the following message prompt appears:

```
>Backup Variable, Object all clear --> OK?
```

To change current LIBSIZE value → enter  or

To not change current LIBSIZE value → enter  or

Changing the current LIBSIZE value causes all main memory backup variable area and object area (except for the source program area and position data area) to be cleared. Therefore, if backup variables are used, it will be necessary to save them again after completing LIBSIZE execution.

Because the size of the robot controller main memory is fixed, specifying a larger number of backup variables or available memory correspondingly decreases the object area. For this reason, when increasing the number of backup variables or available memory, keep the increase as small as possible.

The memory size required by a variable depends on the variable type. For example, a 2 byte integer, including its registration table (system work area), requires about 10 bytes. The registration table memory size required for each variable is basically the same, regardless of type of variable and whether it is an array variable or not. Therefore, in order to use memory efficiently, use array variables to the extent possible.



Available variable types are as follows:

BYTE	1 byte (-128 to +127)	}	integers
INTEGER	2 byte (-32768 to +32767)		
LONG	4 byte (-2147483648 to +2147483647)		
REAL	4 byte 7 digits	}	real numbers
DOUBLE	8 byte 14 digits		

(2) Displays current usable backup variables and available memory.

Neither power off nor executing VERINIT changes LIBSIZE value.

**RELATED  
COMMANDS**

PRG SIZE, PNT SIZE, FREE, SYS

**EXAMPLE**

```
>LIBSIZE 500,10000
>LIBSIZE
 500,10000
>
```



# LIMZ

&gt;

S

Limit Z Axis

**FUNCTION** Specifies and displays Z axis height for JUMP commands

**FORMAT** (1) LIMZ [Z axis specification value]  
 ⋮  
 JUMP [position specification]

\* [LIMZ default value: 0]

(2) JUMP [position specification] LIMZ [Z axis specification value]

(3) LIMZ

**DESCRIPTION** LIMZ specifies Z axis height for JUMP.  
 MOTOR ON, software resets, SLOCK, SFREE, and VERINIT all initialize LIMZ value to 0.

(1) LIMZ Command:

Specifies Z axis height.

Multiple LIMZ are permitted, the most recent LIMZ value remains current until superseded.

JUMP Command:

Executes JUMP at current LIMZ value Z axis height.

(2) JUMP with LIMZ Modifier:

Specifies Z axis height for same line JUMP only.

(3) Displays current LIMZ value.

**NOTE**  


LIMZ Z axis height limit specification is the z axis value for the robot coordinate system. It is not the Z axis value for ARM, TOOL, or local0 coordinates. Therefore take necessary precautions when using tools or hands with different operating heights. When LIMZ modifier is omitted, robot rises to the Z axis height specified by currently valid LIMZ value.

**RELATED  
 COMMANDS** JUMP

**EXAMPLE**

10 LIMZ -10	'Specifies LIMZ value (Z=-10)
20 JUMP P1	'Execute JUMP at current LIMZ value (Z=-10)
30 JUMP P2 LIMZ-30	'Execute JUMP at Z=-30
40 JUMP P3	'Execute JUMP at current LIMZ value (Z=-10)

L

# LINEHIST



## Line History

**FUNCTION** Displays line number history

**FORMAT** LINEHIST [Task number]{,Task number}7

**DESCRIPTION** Displays in order the ten most recently executed line numbers.

Line histories for up to eight tasks can be specified. Multiple tasks are displayed in order to execution, with each task display containing its ten most recently executed line numbers.

Line history capacity can be increased to twenty by specifying LINEBUF=20 in the CNFG.SYS file.

Line histories are erased when power is turned off.

### EXAMPLE

```
>LINEHIST 1,2
```

Task 1	Task 2	Time
40		1:26:37p
30		1:26:37p
40		1:26:37p
30		1:26:37p
	140	1:26:37p
	110	1:26:37p
40		1:26:38p
30		1:26:38p
40		1:26:38p
30		1:26:38p
40		1:26:38p
30		1:26:38p
	120	1:26:38p
	130	1:26:38p
	140	1:26:38p
	110	1:26:38p
	120	1:26:38p
	130	1:26:38p
	140	1:26:38p
	110	1:26:38p

```

Program
10 FUNCTION MAIN
20 XQT !2,SUB
30 WAIT 0.1
40 GOTO 30
50 FEND
100 FUNCTION SUB
110 WAIT 0.2
120 '
130 '
140 GOTO 110
150 FEND

```

# LINE INPUT

&gt;

S

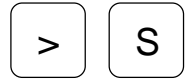
<b>FUNCTION</b>	Stores one line of data from console to string variable	
<b>FORMAT</b>	LINE INPUT [string variable name]	
<b>DESCRIPTION</b>	Stores one line of character string data from console to string variable.	
	In executing LINE INPUT, "?" prompt will appear at the console. After inputting the line of data, press enter (return) key.	
<b>RELATED COMMANDS</b>	INPUT	
<b>EXAMPLE</b>	<pre> 100 FUNCTION MAIN 110 STRING A\$ 120 LINE INPUT A\$ 130 PRINT A\$ 140 FEND &gt;RUN COMPILE END ?A,B,C A,B,C &gt; </pre>	<pre> 'Read one line of data, store at string variable A\$ 'Display string as read </pre>

L

---

# LINE INPUT #

---



**FUNCTION**        Stores one line of data from communication port to string variable

**FORMAT**         LINE INPUT #[port number],[string variable name]

\* port number: integer from 20 to 23

**DESCRIPTION**    Stores one line of character string data from communication port to string variable.

**EXAMPLE**        LINE INPUT #20 ,A\$        'Receive string from port 20, store at string variable A\$

# LINK



<b>FUNCTION</b>	Links two or more object files
<b>FORMAT</b>	LINK [filename 1]+[filename 2]{+[filename n]}n{,[linked-filename]} * filename extension is not allowed
<b>DESCRIPTION</b>	<p>Links two or more object files that are in file memory into a single object file, and saves the file as linked filename (with filename extension .OBJ). At the same time, links the symbol files into a single symbol file, and saves the file as linked-filename (with filename extension .SYM).</p> <p>Doing so a single executable object file and a single symbol file can be created, through modular compilation, from multiple object files and symbol files.</p> <p>The linked filename may be omitted. If omitted, the object file and the symbol file are saved as SYSTMP.OBJ and SYSTMP.SYM, respectively.</p> <p>In a linked object file (i.e., an object file linked from multiple object files), the first Function procedure (FUNCTION...FEND) of filename 1 becomes the main Function and is executed as Task 1.</p> <p>&lt; NOTE &gt; To execute a linked object file, a position data file with the same filename, but with the extension .PNT is necessary. For example, for linked object file SYSTMP.OBJ, position data file SYSTMP.PNT must be created.</p>
<b>RELATED COMMANDS</b>	COMPILE, ENTRY, EXTERN, FILES, DIR
<b>EXAMPLE</b>	<pre>&gt;COM"MAIN" COMPILE END &gt;COM"SUB" COMPILE END &gt;LINK MAIN+SUB,TEST           'Link MAIN.OBJ and SUB.OBJ into TEST.OBJ &gt;</pre>



# LIST



**FUNCTION** Displays source program

**FORMAT**

```
(1) LIST {[[line number]                               ]}
          [[beginning line number] -                   |
          [[beginning line number] -[ending line number] |
          |                                               |
          |                                               |
          | *                                             |
          | * -                                           |
```

(2) LIST

**DESCRIPTION** (1) Displays program lines as specified by their line numbers. Line numbers are specified as follows:

[line number]	Displays specified line
[beginning line number]-	Displays all lines from beginning line number to end of program
[beginning line number]-[ending line number]	Displays all lines from beginning line number up to and including ending line number. To prevent Error 2 from occurring, beginning line number must be less than ending line number.
-[ending line number]	Displays all lines up to and including ending line number
*	Display line where STOP or BREAK key interrupted display
* -	Display all lines from where STOP or BREAK key interrupted display to end of program

(2) Displays all lines.

**EXAMPLE**

```
>LIST 50-      'Displays all lines from 50 to end of program
>LIST -100     'Displays all lines up to and including 100
>LIST 20       'Displays 20
>LIST 100-200  'Displays all lines from 100 up to and including 200
>LIST          'Display all lines
>LIST *       'Display line where display was interrupted
>LIST * -     'Display all lines from where display was interrupted to end of program
```

# LOCAL (RLOCAL, LLOCAL, SLOCAL)

&gt;

S

- FUNCTION** Defines and displays local coordinate systems
- FORMAT** (1) LOCAL[local coordinate system number] (P[point number 1]:P[point number 2]),~  
(P[point number 3]:P[point number 4])
- \* local coordinate system number: integer from 1 to 15
- (2) LOCAL
- DESCRIPTION** (1) Defines a local coordinate system by specifying two points, P[point number 1] and P[point number 3], contained in it that coincide with two points, P[point number 2] and P[point number 4], contained in the robot coordinate system.

&lt; Example &gt;

LOCAL1 ( P1 : P11 ) , ( P2 : P12 )

local coordinate system point data

robot coordinate system point data

If the distance between the two specified points in the local coordinate system is not equal to that between the two specified points in the robot coordinate system, the XY plane of the local coordinate system is defined in the position where the midpoint between the two specified points in the local coordinate system coincides with that between the two specified points in the robot coordinate system.

Similarly, the Z axis of the local coordinate system is defined in the position where the midpoints coincide with each other.

The U axis of the local coordinate system is automatically corrected in accordance with the X and Y coordinate values of the specified 4 points. Therefore, the two points in the robot coordinate system may initially have any U coordinate values.

If may be desired to correct the U axis of the local coordinate system based on the U coordinate values of the two points in the robot coordinate system, rather than having it automatically corrected (e.g., correct the rotation axis through teaching) To do so, turn on bit 2 of software switch SS5, and then use LOCAL. Refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

L



(2) Displays all defined local coordinate systems, including those defined by BASE.

< Example >

Local coordinate systems defined by LOCAL

```

local 0 (p5 :p55 ), (p6 :p56 ) 'Local0 coordinate system
local 1 (p1 :p11 ), (p2 :p12 ) 'Local coordinate system defined by LOCAL
llocal 5 (p50 :p60 ), (p51 :p61 ) 'Local coordinate system defined by LLOCAL
rlocal 10(p70 :p80 ), (p71 :p81 ) 'Local coordinate system defined by RLOCAL
slocal 12(p90 :p100), (p91 :p101) 'Local coordinate system defined by SLOCAL
    
```

Local coordinate systems defined by BASE

```

local 0 (p***:p***), (p***:p***) 'Local coordinate system 0
local 2 (p***:p***), (p***:p***)
    
```

To move the arm to a point with Z=0 in the robot coordinate system (the uppermost point of the third axis for an ordinary robot) using the point data of a local coordinate system whose Z axis is offset, use an "@" symbol:

< Example >

```

JUMP P1:Z@
      |
      |_____ local coordinate system point data
    
```

< NOTE >

If Local0 coordinate system is defined with LOCAL0 or BASE 0, read every occurrence of "robot coordinate system" in the description above as "Local0 coordinate system."

While LOCAL basically uses midpoints for positioning the axes of your local coordinate system as described above, SPEL III provides three variants of LOCAL that use alternative positioning methods, as may be required for your specific application. These variants include LLOCAL, RLOCAL, and SLOCAL. (However, they correct the U axis by the same method as LOCAL.)

**LLOCAL** (Left Local)

LLOCAL defines a local coordinate system by specifying [point number 1] corresponding to [point number 2] in the robot coordinate system (Z axis direction is included.)

**RLOCAL** (Right Local)

RLocal defines a local coordinate system by specifying [point number 3] corresponding to [point number 4] in the robot coordinate system (Z axis direction is included.)



**SLOCAL** (Scale Local)

When defining the XY plane of a local coordinate system, SLOCAL specifies [point number 1] and [point number 3] in the local coordinate system corresponding to [point number 2] and [point number 4] in the robot coordinate system. Thus, it allows definition of the coordinate scale factor; the locus can be scaled up or down. For the Z axis, SLOCAL specifies [point number 1] in the local coordinate system corresponding to [point number 2] in the robot coordinate system.

Local coordinate system numbers are shared between BASE and LOCAL.

Definitions of local coordinate systems #1 through #15 will be lost when the power is turned off, or when LOCAL0, BASE 0, or VERINIT are executed.

Added &n can perform reverse conversion of a local coordinate system. Remember that points determined in TEACH mode can be defined as point data in a local coordinate system.

P1=P\*&2           'Defines the current point as coordinate in the local coordinate system 2.

**RELATED  
COMMANDS**

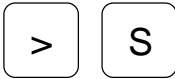
BASE, BASE 0, BASE( ), LOCAL0

**EXAMPLE**

```
>PLIST
P1=0,0,0,0/1           'Points in local coordinate system #1
P2=100,0,0,0/1        'Points in robot coordinate system
P11=0,100,0,0
P12=100,100,0,0
>LOCAL1(P1:P11),(P2:P12) 'Define local coordinate system #1
>
>LOCAL                 'Display all local coordinate systems
local0(p***:p***),(p***:p***) 'Local0 coordinate system defined by BASE 0
local 1(p1 :p11 ),(p2 :p12 ) 'Local coordinate system #1 defined by LOCAL
>GO P1                 'Move to P1 on the LOCAL System 1 via GO
>JUMP P2:Z@           'Move to P1(Z=0) on the LOCAL System 1 via
JUMP
```



# LOCAL0 (RLOCAL0, LLOCAL0)



Local Zero

**FUNCTION** Defines robot's basic coordinate system Local 0

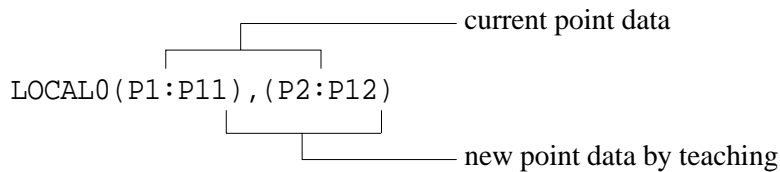
**FORMAT** LOCAL0(P[point number 1]:P[point number 2]),(P[point number 3]:P[point number 4])

**DESCRIPTION** Defined local coordinate systems are based on a single basic local coordinate system, called "Local0 coordinate system." Usually the Local0 coordinate system is equivalent to the robot coordinate system, the absolute coordinate system of the robot.

Although the position of the robot coordinate system cannot be changed, the position of Local0 coordinate system can be changed by using LOCAL0 (or BASE 0). Note that, as long as Local0 coordinate system is equivalent to the robot coordinate system, it is not necessary to redefine the Local0 coordinate system.

After disassembling and reassembling the robot from or to a table for servicing, or some other purpose, the point data no longer matches the points on the system, thereby making the current point data invalid. In such a case, the current point data can be made valid once again by teaching the robot two points in the current point data and then defining the Local0 coordinate system with LOCAL0.

< Example >



Move the robot arm to the position that corresponds to P1 and P2 in the current point data, and teach the robot to recognize P1 and P2 as P11 and P12, respectively. Then, execute LOCAL0 as shown above.

LOCAL0 and its variants, LLOCAL0 and RLOCAL0, define the XY plane, Z axis, and U axis the same way as LOCAL and its variants. Therefore, refer to the explanation of LOCAL and its variants for more information.

Power off does not change Local0 coordinate system values.

To ensure that Local0 coordinate system is equivalent to the robot coordinate system, use one of the following:

```
LOCAL0 (P1:P1) , (P1:P1)
BASE 0,0,0,0,0
VERINIT
```

**NOTE**

Since LOCAL0 alters the position of the basic coordinate system, it should be used only when necessary.

Executing LOCAL0 erases all local coordinate systems, including those defined by BASE. It is necessary to redefine them.

Executing VERINIT initializes Local0 coordinate system, making it identical to the robot coordinate system.

**RELATED  
COMMANDS**

BASE, BASE 0, BASE( ), LOCAL

# LOF( )

F

Line of File

**FUNCTION** Returns number of data lines stored in RS-232C buffer

**FORMAT** LOF([port number])

\* port number: integer from 20 to 23

**DESCRIPTION** Data sent to an RS-232C port is stored in a buffer, and is subsequently read from the buffer with INPUT # command. LOF returns the number of data lines stored in the buffer, and returns 0 if no data lines are stored.

**RELATED  
COMMANDS** INPUT #

**EXAMPLE** >PRINT LOF( 20) 'Display number of data lines stored in port 20 buffer  
5  
>

# LONG

S

Long

**FUNCTION** Defines 4-byte integer type variables

**FORMAT** LONG [variable name]{{(array size 1{,array size 2{,array size 3}})}~  
 {,[variable name]{{(array size 1{,array size 2{,array size 3}})}}n

**DESCRIPTION** This command defines 4-byte integer type variables.

If several variables of the same type are declared, use a " , " (comma) and describe several variable names. When defining an array variable, declare the name and its size enclosed with ( ). The size can be defined up to 3 dimension.

By default, the variable whose data type is not declared specifically will be treated as REAL type. Therefore, it should be declared if the integer type is sufficient or the data type because it has advantage over the REAL type in terms of performance speed and memory efficiency.

Also, there are three types to the integer type and, each handles a different size of data as follows: BYTE (1-byte integer), INTEGER (a 2-byte integer) and LONG (a 4-byte integer). The range for a LONG type value is -2147483648 to 2147483647. If a value outside this range is entered, it causes an error.

For more details about the variable, refer to "2.3 Variables" in the Elementary section of the User's manual for SRC-300/320.

The following cites the restrictions on the variable names. The variable may be freely named within these restrictions.

- The usable characters are alphanumeric and underscores ( \_ ). There is no distinction between capital and small case letters.
- Must be eight characters or under.
- The first character must be an alphabet character other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. A reserved word that is followed by an underscore or numeric character is also read as a reserved word.



**NOTE** The variable type must be declared at the beginning of a line; otherwise, the file will not be successfully compiled. When declaring another type of variable, a new line must be created.

L



**RELATED  
COMMANDS**

BYTE, INTEGER, REAL, DOUBLE, STRING, VARIABLE, SYS

**EXAMPLE**

```
10 FUNCTION MAIN
20 LONG I           'Declares a 4-byte integer type variable "I."
30 REAL ODATA(10,10) 'Declares a 2 dimensional array of 4-byte integer type
                    variable, "ODATA."
   :
999 FEND
```

# LP, POWER

&gt;

S

Low Power, Power

**FUNCTION** Switches motor power mode and displays current status

**FORMAT** LP { |ON |OFF| } | POWER { |LOW |HIGH| }

\* [default: ON] \* [default: LOW]

**DESCRIPTION** Switches motor power mode and displays current status.

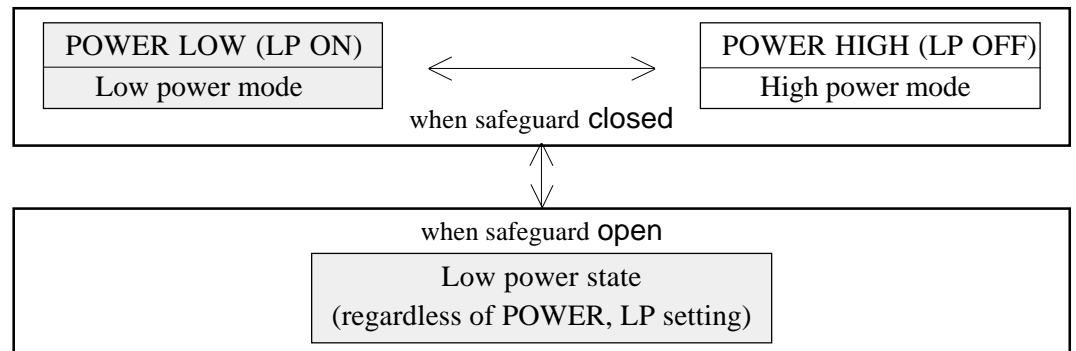
**LP** With ON specified, mode switches to low power mode.  
With OFF specified, mode switches to high power mode.  
With no ON/OFF specification, LP displays the current motor power status.

**POWER** With LOW specified, mode switches to low power mode.  
With HIGH specified, mode switches to high power mode.  
With no LOW/HIGH specification, POWER displays the current power mode and the actual power mode status.

&gt;POWER

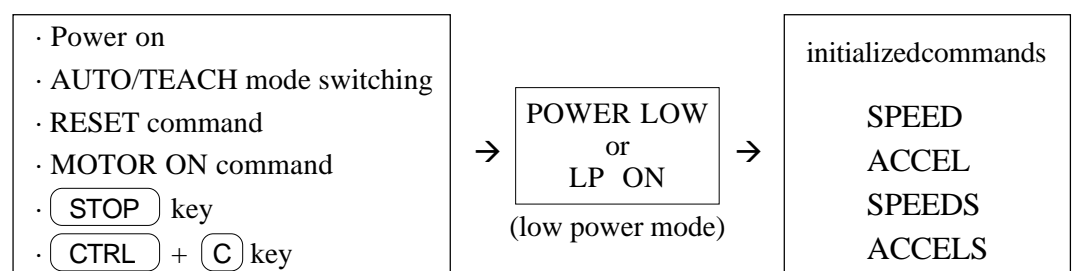
Mode :Low            current mode  
State:Low            actual status

The function of LP and POWER is same.



For the safety reasons, low power state is basic in TEACH mode for the controller.

Therefore, following operations (Reset operations) will turn the motor power mode to low as default setting. In this case, speed and acceleration setting also will be initialized to default value. Refer to "Specifications" in the manipulator manual for default values of speed and acceleration, which vary from model to model.



In low power state, motor power is limited, and effective motion speed setting is lower than the default value. If higher speed is specified directly or in a program, the speed is set to the default value.

Motor power status	Actual motion speed
Low power state	the default value of speed setting command the specified value of speed setting command <span style="font-size: 2em; vertical-align: middle;">}</span> either lower value
High power state	the specified value of speed setting command

If higher speed motion is required in AUTO mode for motion commands described in a program, specify POWER HIGH or LP OFF in the program.

In low power state, error 173 may occur if robot arm is pushed by hand or operation with down force is executed because of the limited motor power.

When speed setting command is inputted in low power state, the following message will be displayed.

It shows the robot is in low power state, and will move in low speed.

Low Power State : xxxxx is limited to xxx

**RELATED COMMANDS**

POWER, SPEED, ACCEL, SPEEDS, ACCELS

**EXAMPLE**

```

>SPEED 50                                'Specifies higher speed
  Low Power State : SPEED is limited to 5
>ACCEL 100,100
  Low Power State : ACCEL is limited to 10
>JUMP P1                                  'Moves in low speed
>SPEED; ACCEL                             'To display speed setting status
  Low Power State : SPEED is limited to 5
    50
    50      50
  Low Power State : ACCEL is limited to 10
    100     100
    100     100
    100     100
>XQT
>LP OFF                                  'Set high power mode (=POWER
HIGH)
>JUMP P2                                  'Moves in high speed
    
```



---

# LSHIFT( )

---

F

Left Shift

<b>FUNCTION</b>	Shifts numeric value data to left
<b>FORMAT</b>	LSHIFT([numeric value data],[number of bits to be shifted])
<b>DESCRIPTION</b>	Shifts specified numeric value data specified number of bits to the left (toward most significant bit). For each place shifted to the left, a 0 is inserted into the vacated space on the right.
<b>RELATED COMMANDS</b>	RSHIFT( ), NOT( )
<b>EXAMPLE</b>	<pre>&gt;PRINT LSHIFT(1,2) 4 &gt;I=5 &gt;PRINT LSHIFT(I,1) 10 &gt;</pre>

L

# MCAL



## Machine Calibration

**FUNCTION** Executes machine calibration

**FORMAT** MCAL

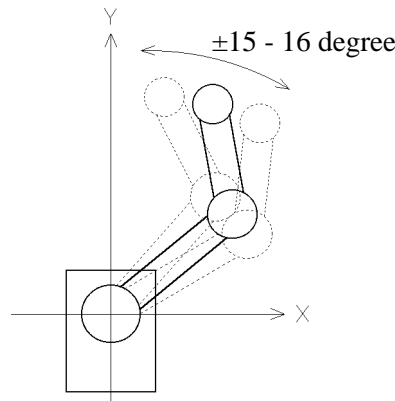
**DESCRIPTION** It is necessary to execute machine calibration with this command for INC robot on which the incremental encoder is installed. Machine calibration must be executed after turning on the power. If you attempt motion command execution or PLIST\* command (they need the current position data) without executing machine calibration, it will cause an error.



The moving distance at machine calibration differs from model to model. Refer to the specifications of corresponding manipulator manual for details.

Machine calibration is executed according to the moving axis order which is specified with MCORDR command. The default value of MCORDR at the shipment differs from model to model. Refer to the specifications of corresponding manipulator manual for details.

On the multi-calibration-point system robot, the axis #1 and #2 move  $\pm 15-16$  degree to clockwise/counter-clockwise direction. If MCAL is executed near the limit of motion envelope, the arm goes over the range. In this case, error may be issued or the arm hits the mechanical stopper, and cannot execute calibration. To avoid it, execute MCAL in the center of motion envelope.



Regarding the single-calibration-point robot such as cartesian robot(XM3000), when MCAL command is executed, the robot arm is calibrated at the origin of robot coordinate.



Do not rotate the axis #4 more than  $180^\circ$  while the controller power is off. After you rotate the axis, if you turn on the power and execute MCAL, error 237 will appear depending on a model. In this case turn off the power, and move the axis #4 to the approximate previous position, then turn on the power again.

**RELATED COMMANDS** MCORDR, MCORG, MCOFS

**EXAMPLE** >MOTOR ON  
>MCAL

# MCOFS

INC

&gt;

Machine Calibration Offset

**FUNCTION** Specifies and displays the parameter for machine calibration

**FORMAT** MCOFS { [specification value 1], [specification value 2], ~  
[specification value 3], [specification value 4], ~  
[specification value 5], [specification value 6], ~  
[specification value 7]}

**DESCRIPTION** Specifies and displays the parameter for machine calibration.  
This parameter is necessary data for executing machine calibration with MCAL.  
Meaning of each specification value is as follows:

specification value 1	Logic of sensor when MCORG is executed	hexadecimal
specification value 2-5	Pulse values between where sensor is on to where Z phase is detected at specified position of 1st to axis #4	decimal
specification value 6,7 (for multi-calibration-point robot)	Difference between logical value and sensor edge width at the specified position of axis #1 and axis #2	decimal

If you omit all the specification values, parameters for calibration which are currently set are displayed as follows:

```
[specification value 1]
[specification value 2]      [specification value 3]
[specification value 4]      [specification value 5]
[specification value 6]      [specification value 7]
```

If you specify all specification values, they are set as parameters for machine calibration. Enter zero to specification value 6,7 for single-calibration-point robot such as cartesian robot.

Parameters for calibration are calculated by moving the robot arm with MCORG. However, if those values are clear in advance, you can input the values using this MCOFS command.

MCOFS values are maintained when the power is off and even VERINIT is executed.

When machine calibration is executed with MCAL, current position is recognized based on this parameter, therefore, these data must be specified. If wrong data are specified, the robot coordinates is incorrect, and it causes improper robot motion.



Specifying the parameters for calibration using this command is used for maintenance purposes only after replacing MPU board for example. Do not use this command for other purposes.

For your reference, outline of replacing MPU board procedure is described below. Please refer to the maintenance manual for details.

1. Before replacing MPU board, display the parameters for calibration currently specified with MCOFS command.
2. Write down all those displayed 7 parameters.
3. Insert a new MPU board into controller.
4. Input the written down parameters into MCOFS in the correct order and execute it.

The parameters for calibration can be also set by using MKVER and SETVER commands, or function keys **V-BKUP** and **V-RSTR** on SPEL Editor, or <MKVER> and <SETVER> button on [Maintenance] dialog box in [Tools] menu of SPEL for Windows.

**RELATED  
COMMANDS**

MCORG, MCAL, VER, MKVER

**EXAMPLE**

```
>MCOFS
04
1364 50
4506 6304
-479 -469
>MCOFS &H04,1364,50,4506,6304,-479,-469
```

# MCORDR

INC

&gt;

S

Machine Calibration Order

**FUNCTION** Specifies and displays the moving axis order in machine calibration

**FORMAT** (1) MCORDR {[specification value 1], [specification value 2], ~  
[specification value 3], [specification value 4]}

\* [default value: Differs from model to model]

(2) MCORDR

**DESCRIPTION** (1) Specifies the order of moving axis in machine calibration with MCAL. First the axis (or axes) specified as specification value 1 executes calibration, after it finished, the axis specified as specification value 2 executes calibration. In this way the axis of specification value 3 and specification value 4 executes calibration in this order.

Each axis is assigned to bit0 to 3 as shown below:

Axis name	1st axis	2nd axis	3rd axis	4th axis
Bit number	bit3	bit2	bit1	bit0
Binary code	&B1000	&B0100	&B0010	&B0001

At the shipment, default value is specified. Refer to the specifications of manipulator manual for details.

MCORDR values are maintained when power is off. Performing VERINIT initializes to default value. The default values differ from model to model. Refer to the specifications of corresponding manipulator manual for details.

In the case of robot that has a ballscrew spline, specify the previous order for axis #3 to axis #4, or specify the same order.

Otherwise error is issued when MCAL is executed.

(2) Displays current MCORDR values in hexadecimal as follows:

```
[specification value 1] [specification value 2]
[specification value 3] [specification value 4]
```

**RELATED COMMANDS** MCAL, MCOFS

**EXAMPLE**

```
>MCORDR &B0010 , &B1000 , &B0100 , &B0001 'Specifies the order as follows:
>MCORDR axis #3 #1 #2 #4
02 08
04 01 'Values are displayed in hexadecimal.
```

M

# MCORG

INC

&gt;

Machine Calibration Origin

**FUNCTION**      Calculates the parameter for machine calibration

**FORMAT**        MCORG [axis number] { ,[axis number] }<sub>3</sub>

\* axis number: Integer from 1 to 4

**DESCRIPTION**    Calculates the parameter which is necessary for machine calibration with MCAL.



This command is intended to be used for maintenance of a multi-calibration-point system robot, it is used after replacing a motor for example. Use this command only when it is necessary.

If axis number is specified, the parameter for the specified axis is calculated.

In the case of robot that has a ballscrew spline, and need to calculate the parameter for axis #3 or #4, specify both axes together. It causes an error if one of them is specified.

If axis number is not specified, error will occur.

Calculated parameter for calibration is maintained when the power is off, and is not changed by initialization such as VERINIT.

The parameters maintained in the controller are displayed by MCOFS.

When MCORG is executed, all the specified axis start moving. Before executing it, make sure that there is no obstacles around the robot. Especially be sure that axis #3 is at higher position.

For the multi-calibration-point system robot, arm position where MCORG is allowed to be executed is limited.

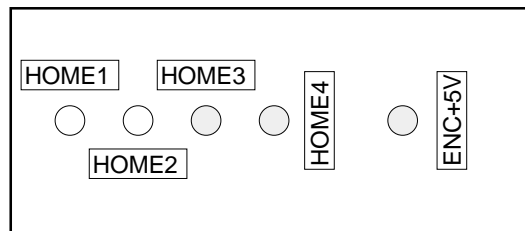
Do not execute MCORG in other positions.

If it is not multi-calibration-point system robot, it is allowed to execute this command in any positions.

How to calculate the parameter for machine calibration (execution of MCORG) in the case of multi-calibration-point system robot is described below.

Move the axis #3 and #4 of the robot within the motion range.

Decide the arm position by using LEDs on sensor monitor on the rear side of manipulator base. LEDs are shown next page. The number of 1 to 4 with printed "HOME" character on the board corresponds to the axis number of manipulator.



If you move the axis #1 or #2, LED corresponds to the axis turns on or off in accordance with the arm movement.

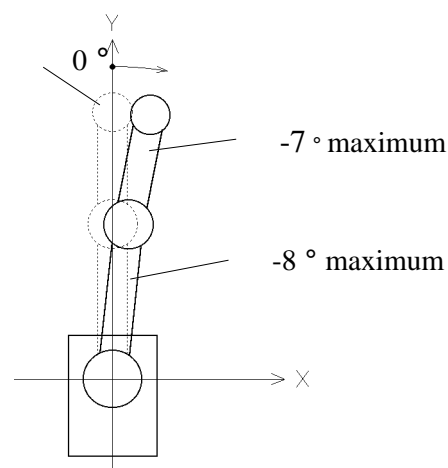
Stretch the arm as drawn with dotted line in the figure below, so that the arm is parallel to Y axis of the robot coordinates. In this case, HOME1 and HOME2 LED turn on. When the LED does not turned on, please move the arm from present posture slightly counterclockwise manually until HOME1 and HOME2 LED turns on. (The position where LED turns on is slightly different because of the mechanical difference of the position where the sensor is installed. The difference gives no influence in accuracy.)

While watching HOME2 LED, move the arm #2 to clockwise direction by the hand. Hold the arm #1 so that it does not move at that time. Stop moving the arm #2 in the approximate center of the range where the LED turn off at the first time.

Next while watching HOME1 LED, move the arm #1 to clockwise direction by the hand. Stop moving the arm #1 in the approximate center of the range where the LED turns off at the first time.

For both arm #1 and #2, the approximate position is less than the degree shown in the drawing below. The degree differs from model to model. Refer to the specifications of manipulator manual.

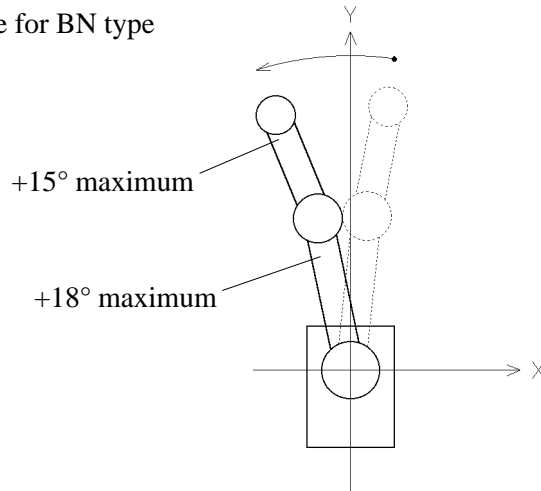
example for BN type



Execute MOTOR ON, and engage motors.

When you execute MCORG at this position arm #1 and #2 move to counterclockwise direction. The maximum movement degree is as shown below, be sure to remove any obstacles in the range before executing MCORG.

example for BN type

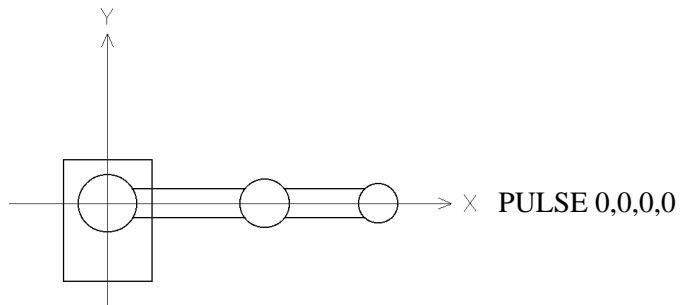


Execute MCAL to conduct machine calibration.

Execute PULSE command at the 0 pulse position of the axis #1 and #2.

>PULSE 0,0,0,0

Confirm that arm #1 and #2 are straight and are on the X axis of the robot coordinates. If they are not, it means the arm position when MCORG is executed is not correct.



In the case of multi-calibration-point-system robot, if MCORG is not executed at the right position, correct parameters for calibration is not calculated. If the parameters for calibration is not correct, the robot cannot move properly.

If you execute MCORG by mistake, do not execute any motion commands.

In this case execute MCORG at the correct position again, or find the correct data and input it with MCOFS.

**RELATED COMMANDS**

MCAL, MCOFS

**EXAMPLE**

>MCORG 1, 2 'Calculates parameters for machine calibration of axes #1 and #2.



# MID\$( )

F

Middle \$

**FUNCTION** Returns specified number of characters of specified string beginning from specified start position

**FORMAT** MID\$([string variable name],[start position],[number of characters])  
|"[string]" |

**DESCRIPTION** Beginning from specified start position, returns specified number of characters of specified string.

Specified string may be specified as a string variable.

Start position is counted from the leftmost string character.

**RELATED COMMANDS** LEFT\$( ), RIGHT\$( )

**EXAMPLE**

```
>PRINT MID$( "ABCDE" , 3 , 2)
CD
>A$="1234567"
>PRINT MID$(A$, 2 , 3)
234
>
```

M

# MKDIR, MD



Make Directory

**FUNCTION**           Creates subdirectory

**FORMAT**             MKDIR {[pathname]}[directory name]

**DESCRIPTION**       Creates subdirectory in specified path.

If pathname is omitted, creates subdirectory in current directory.

Only one subdirectory can be created at a time.

**RELATED  
COMMANDS**           DIR

**EXAMPLE**            >MD \USR  
                      >MD \USR\PNT

# MKVER



Make VER

**FUNCTION** Makes backup of various setting data on file memory

**FORMAT** MKVER {/A}

**DESCRIPTION** Backup data of various setting and all the data on main memory will be saved on current directory with filenames. Filenames are added automatically.

Various setting data means the data which are specified by the following commands:

CALPLS	HOFS	MCORDR	MCOFS
SEL	SET	ARCH	
TSPEED	TSPEEDS	HOMESSET	HORDR
ARM	ARMSET	TOOL	TLSET
RANGE	XYLIM	BASE 0	
CONFIG	CONSOLE	WIDTH	
PRGNO	OPUNIT	WEIGHT	MAXDEV
PRGSIZE	PNTSIZE	LIBSIZE	
Software switch settings		REMOTE3 setting	

If /A is not specified, various setting data and backup variables will be saved with the following filenames.

data	filename
various setting data	MK#0.BAT
backup variable	LIB#0.BIN

Batch file "MK#0.BAT" includes commands for specifying above various setting data and command for loading backup variable file "LIB#0.BIN" onto main memory.

If /A is specified, various setting data and all data on main memory will be saved with the following filenames.

data	filename
various setting data	MK#0.BAT
source program	PRG#0.PRG
position data	PNT#0.PNT
object program	OBJ#0.BIN
symbol table	SYM#0.BIN
backup variable	LIB#0.BIN



Batch file "MK#0.BAT" includes commands for specifying various setting data and command for loading five files such as source program etc. in the previous page onto main memory.

Backup variables will be saved by MKVER, however, usual variables will not be saved.

In order to load each file data which was saved by MKVER to original memory area, there are two methods below:

- ① To execute SETVER.
- ② To execute the batch file MK#0.BAT.

Various setting data includes important basic data such as HOFS etc..

It is recommendable to execute MKVER and keep those data as a file when a robot is installed for the first time.

If you want to know how to backup and restore, refer to "backing up and restoring a file" in chapter 3 of SPEL Editor manual or "[File] menu" of SPEL for Windows manual.

**RELATED  
COMMANDS**

SETVER, VER

**EXAMPLE**

This command will be used when MPU or SPU board in controller is replaced. It is especially useful for loading various setting data onto a new board.

- ① Save data on the board to be replaced (old board) as a file.

```

>MKVER                                'Following files will be made
                                         MK#0.BAT
                                         LIB#0.BIN
CPU Data Backup
Backup Variable Backup
    
```

- ② Make backup of all files including above files on file memory onto a disk of programing unit.
- ③ Install a new board into a controller.
- ④ Restore all files on the disk of programing unit to controller file memory.
- ⑤ Execute SETVER.

```
>SETVER
```

# MOTOR

&gt;

S

<b>FUNCTION</b>	Turns motor power on and off
<b>FORMAT</b>	MOTOR  ON    OFF
<b>DESCRIPTION</b>	<p>At motor on, all axes are engaged and brakes are released.</p> <p>At motor off, power to all motors is cut, and brakes are engaged.</p> <p>After an emergency stop, or after an error has occurred that requires resetting with the RESET, execute RESET, then execute MOTOR ON.</p>
<b>RELATED COMMANDS</b>	SFREE, SLOCK, RESET
<b>EXAMPLE</b>	<pre>&gt;MOTOR ON &gt;MOTOR OFF &gt; &gt;SFREE 1,2,4 &gt; &gt;MOTOR ON</pre>

# MOVE

&gt;

S

**FUNCTION** Moves all four axes simultaneously by linear interpolation

**FORMAT** (1) MOVE [ps] {TILL}

(2) MOVE [ps] {TILL SW([input bit number]){= |0|}}{![parallel processing statement]!}  
|1|

\* [ps] = position specification

**DESCRIPTION** Moves all four axes simultaneously by linear interpolation.

MOVE uses the SPEEDS speed value and the ACCELS acceleration value. Should the SPEEDS speed value exceed the allowable speed for any axis, power to all four axis motors will be cut, and robot will stop.

MOVE cannot execute U axis-only movement.

MOVE cannot execute range verification of the trajectory in advance. Therefore, even for target positions that are within an allowable range, en route the robot may attempt to traverse an invalid range, stopping with a severe shock that may damage the arm. To prevent this, be sure to perform range verifications at low speed in advance.

TILL modifier is used to decelerate and stop the robot at an intermediate travel position if currently valid TILL condition is satisfied. If TILL condition is not satisfied, robot travels to the target position.

(1) MOVE with TILL Modifier:

Checks if current TILL condition is satisfied. If satisfied, this command completes by decelerating and stopping robot.

(2) MOVE with TILL Modifier, SW(input bit number) Modifier, and (0 or 1) Input Condition:

Checks if same line input condition is satisfied. If satisfied, this command completes by decelerating and stopping robot at an intermediate travel position.

MOVE with TILL Modifier, SW(input bit number) Modifier, but no Input Condition: Input condition defaults to 1. If specified input bit is on, this command completes by decelerating and stopping robot at an intermediate travel position.

**RELATED  
COMMANDS**

P=, !...!, SPEEDS, ACCELS, TILL, SW( ), CMOVE, ARC, CARC, CVMOVE

**EXAMPLE**

100 TILL SW(1)=0 AND SW(2)=1	'Specifies TILL condition (input bit 1 is off and input bit 2 is on)
110 MOVE P1 TILL	'Stop if current TILL condition (line 100) is satisfied
120 MOVE P2 TILL SW(2)=1	'Stop if input bit 2 is on
130 MOVE P3 TILL	'Stop if current TILL condition (line 100) is satisfied

# MYTASK(0)

F

**FUNCTION** Returns number of mytask

**FORMAT** MYTASK(0)

\* The numeral 0 in ( )

**DESCRIPTION** Returns number of mytask as numeric value.

**EXAMPLE** In the following program, output bits 1 through 8 are switched on and off.

```
>LIST
10  FUNCTION MAIN
20  XQT !2, TASK      'Execute Function TASK in Task 2
30  XQT !3, TASK      'Execute Function TASK in Task 3
40  XQT !4, TASK      'Execute Function TASK in Task 4
50  XQT !5, TASK      'Execute Function TASK in Task 5
60  XQT !6, TASK      'Execute Function TASK in Task 6
70  XQT !7, TASK      'Execute Function TASK in Task 7
80  XQT !8, TASK      'Execute Function TASK in Task 8
90  '
100 '
110 ON MYTASK(0)      'Turn on mytask number output bit
120 OFF MYTASK(0)     'Turn off mytask number output bit
130 GOTO 110
150 '
200 FUNCTION TASK
210 ON MYTASK(0)      'Turn on mytask number output bit
220 OFF MYTASK(0)     'Turn off mytask number output bit
230 GOTO 210
240 FEND
```



# NEW



**FUNCTION** Deletes source program in source program area

**FORMAT** NEW

**DESCRIPTION** Deletes source program in source program area.

When executed in on-line mode, deletes source program in controller main memory (initializes source program area).

When executed in off-line mode, deletes source program in programming unit (PC).

NEW is executed to clear memory prior to entering a new program.

**RELATED  
COMMANDS** CLEAR

**EXAMPLE**

>LIST

```
10 'NEW PROGRAM
20 HOME
30 JUMP P1
40 JUMP P2
50 JUMP P3
60 END
```

>

>NEW

>LIST

>\_

```
10 'NEW PROGRAM
20 HOME
30 JUMP P1
40 JUMP P2
50 JUMP P3
60 END
```

'Because the previously existing program was deleted by executing NEW, nothing is displayed

**N**

---

# NORMAL

---

&gt;

S

<b>FUNCTION</b>	Cancels reverse display mode (on operating unit)
<b>FORMAT</b>	NORMAL [x column],[y line],[number of characters]  * x: Integer from 1 to 32 y: Integer from 1 to 8 number of character: Integer from 1 to 32
<b>DESCRIPTION</b>	Cancels reverse display mode for specified number of characters from (x,y) position, and displays characters in normal display mode.  If specified area is longer than the end of one line, extra area does not slide to the next line, but returns to the beginning of the line, and displays characters in normal mode.
<b>RELATED COMMANDS</b>	REVERSE, CHARSIZE, CLS, CURSOR, OPUNIT, OPU PRINT
<b>EXAMPLE</b>	>NORMAL 10,2,3

---

# NOT( )

---


F

<b>FUNCTION</b>	Returns inverted value of integer bit
<b>FORMAT</b>	NOT([integer])
<b>DESCRIPTION</b>	Returns inverted value of specified integer bit.
<b>RELATED COMMANDS</b>	LSHIFT( ), RSHIFT( )

# OFF

&gt;

S

<b>FUNCTION</b>	Switches output bit off, and, after a specified time, switches it back on
<b>FORMAT</b>	<p>OFF [output bit number]{,[time interval]{,[non-synchronous setting]}}</p> <p>* output bit number: integer from 0 to 127  time interval: seconds, (minimum unit is 0.01)  non-synchronous setting: 0 or 1 [default value: 1]</p>
<b>DESCRIPTION</b>	<p>If only output bit number is specified, specified bit number output is switched off.</p> <p>If time interval is specified, output bit is switched off, and switched back on after time interval elapses. If prior to executing OFF the output bit was already off, then it is switched on after the time interval elapses.</p> <p>Non-synchronous settings are applicable when time interval is specified as follows:</p> <ul style="list-style-type: none"> <li>* 1: Switches output off, switches back on after specified interval elapses, then executes next command.</li> <li>* 0: Switches output off, and simultaneously executes next command.</li> <li>* setting omitted: Same as setting 1.</li> </ul> <p>If an output bit which is set up as a REMOTE3 is specified, error will occur. REMOTE3 output bits are turned on or off automatically according to the status of robot controller.</p> <p>RESET resets all bits to off.</p>
<b>NOTE</b> 	<p>All of output bits will be turned off when emergency stop occurs.</p> <p>To maintain the current status, turn on bit 6 of software switch SS1. Refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.</p>
<b>RELATED COMMANDS</b>	ON, OUT, OPBCD, SW( )
<b>EXAMPLE</b>	<pre>&gt;OFF 1           'Switches output bit 1 off &gt;OFF 1,3         'Switches output bit 1 off, 3 seconds elapses, switches on &gt;OFF 1,3,0 ;GO P1 'Switches output bit 1 off and at the same time, travel to P1                     begins. 3 seconds elapses, and switches output bit 1 on</pre>

# OFF \$




<b>FUNCTION</b>	Switches memory I/O off
<b>FORMAT</b>	OFF \$[bit number] * bit number: integer from 0 to 511
<b>DESCRIPTION</b>	Switches specified memory I/O bit off.
<b>RELATED COMMANDS</b>	ON \$, SW(\$ ), IN(\$ )
<b>EXAMPLE</b>	>OFF \$9      'Switch memory I/O bit 9 off

# ON

&gt;

S

<b>FUNCTION</b>	Switches specified output bit on, and, after a specified time, switches it back off
<b>FORMAT</b>	<p>ON [output bit number]{,[time interval]{,[non-synchronous setting]}}</p> <p>* output bit number: integer from 0 to 127  time interval: seconds, (minimum unit is 0.01)  non-synchronous setting: 0 or 1 [default value: 1]</p>
<b>DESCRIPTION</b>	<p>When only output bit number is specified, that output bit is switched on.</p> <p>When time interval is specified, output bit is switched off when the time interval elapses after the output bit was switched on.</p> <p>Non-synchronous settings are applicable when time interval is specified as follows:</p> <ul style="list-style-type: none"> <li>* 1: Switches on and executes next command when the time interval since switching to on elapses.</li> <li>* 0: Switches on as next command executes.</li> <li>* setting omitted: Same as setting 1.</li> </ul> <p>If an output bit which is set up as a REMOTE3 is specified, error will occur. REMOTE3 output bits are turned on or off automatically according to the status of robot controller.</p> <p>RESET resets all bits to off.</p>
<b>NOTE</b> 	<p>All of output bits will be turned off when emergency stop occurs.</p> <p>To maintain the current stats, turn on bit 6 of software switch SS1. Refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.</p>
<b>RELATED COMMANDS</b>	OFF, OUT, OPBCD, SW( )
<b>EXAMPLE</b>	<pre>&gt;ON 1                'Switches output bit 1 on &gt;ON 1,3              'Switches output bit 1 on for 3 seconds, then switches off &gt;ON 1,3,0 ;GO P1    'Switches output bit 1 on for 3 seconds, then switches off. As                     output bit 1 is switched on, begins moving to P1</pre>

# ON \$

&gt;

S

<b>FUNCTION</b>	Switches memory I/O on
<b>FORMAT</b>	ON \$[bit number] * bit number: integer from 0 to 511
<b>DESCRIPTION</b>	Switches specified memory I/O bit on.
<b>RELATED COMMANDS</b>	OFF \$, SW(\$), IN(\$)
<b>EXAMPLE</b>	>ON \$5      'Switch memory I/O bit 5 on

O

# ONERR...RETURN

S

On Error...Return

**FUNCTION** Defines error processing

**FORMAT**

```
(1) ONERR |line number|
          |label      |
          :
          |line number|
          |label      |
          :
          ECLR
          :
          RETURN

(2) ONERR 0
```

**DESCRIPTION** (1) After an error occurs, by either line number or label, branches program to error processing subroutine. RETURN returns program control to line following error.

Typically when an error is generated, error number is displayed and program execution stops. By including an ONERR, should an error occur during program execution, it becomes possible to branch to an error processing subroutine, thereby allowing program execution to continue.

The error processing subroutine must include an ECLR to clear error status. Error subroutines may not contain nested error subroutines. (For details regarding nesting, refer to #include command description.)

ONERR...RETURN may be used in any task in which error processing is desired. It may be included many times in each task.

(2) Clears ONERR.

**RELATED COMMANDS** ECLR, ERR(0), ERL(0)

**EXAMPLE**

```
10 ONERR 60
20 FOR I=0 TO 199
30 JUMP PI
40 NEXT I
50 END
60 '
70 'ERR SUB           'Subroutine for error processing
80 A=ERR(0)
90 PRINT A
100 ECLR             'Clears error status
110 RETURN
```





# OPBCD

&gt;

S

Output by Binary Coded Decimal

**FUNCTION** Outputs BCD data to output port (8 bit output)

**FORMAT** OPBCD [port number],[output data]

\* port number: integer from 0 to 15  
output data: integer from 0 to 99

**DESCRIPTION** Outputs BCD data to output port specified by port number.

Each port is comprised of 8 bits, the standard 16I/16O is made up of 2 ports.  
The port number, LSB/MSB, and bit number are related as follows:

port number	MSB 7	6	5	4	3	2	1	LSB 0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8

LSB: least significant bit

MSB: most significant bit

Output data is represented by upper 4 bit and lower 4 bit BCD. Since BCD 9 is the greatest value able to be represented by 4 bit BCD, it is possible to output data from BCD 00 to 99, but not possible to output BCD that contain A or greater.

If an output bit which is set up as a REMOTE3 is specified, error will occur. REMOTE3 output bits are turned on or off automatically according to the status of robot controller.

RESET resets all bits to off.

## NOTE



All of output bits will be turned off when emergency stop occurs.

To maintain the current status, turn on bit 6 of software switch SS1. Refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

**RELATED COMMANDS** INBCD, OUT

**EXAMPLE** OPBCD 0,17 'Turn output bits 0, 1, 2, and 4 on, 3, 5, 6, and 7 off

7	6	5	4	3	2	1	0	bit
0	0	0	1	0	1	1	1	→ &H17

---

# OPORT( )

---

F

Output Port

**FUNCTION** Returns output bit status**FORMAT** OPORT([bit number])

\* bit number: integer from 0 to 127

**DESCRIPTION** Returns specified output bit status as either 0 or 1.

0: Off status

1: On status

OPORT( ) does not function for bits which set up as REMOTE3 and returns 0.

**EXAMPLE**  
>ON 5  
>PRINT OPORT(5)  
1  
>

O

# OPUNIT

&gt;

S

## Operating Unit

**FUNCTION** Operation mode selection of operating unit

**FORMAT** OPUNIT {[mode number]}

\* mode number: integer from 0 to 3

**DESCRIPTION** Selects the operation mode of operating unit.

If mode number is omitted, currently selected mode number will be displayed.

In Mode 0 (system mode), various functions are assigned to each key on operating unit for the use of monitor function, file select function, etc. Also necessary messages for such various functions are displayed.

It is also possible for customers to assign their own functions to each key in Mode 0. However, careful attention is necessary in this case since the specified function by customers' program and original function in Mode 0 may be doubly assigned to one key.

In Mode 3 (user mode), most of originally assigned system functions are ineffective so that customers can use keys as they like. Screen is for customer use only, it is not used for system in this mode. Free keys are (F1) to (F4) (↑) (↓) (←) (→) (MENU).

(START) (PAUSE) (RESET) keys are always reserved for system functions, they are not free.

In Mode 1 and 2, monitor function is provided. In Mode 2, error message display function is provided too, so error message will be displayed when error is issued. In Mode 1 and 2, free keys are (F1) to (F4) (↑) (↓) (←) (→). (They are called (Key) onward.)

When monitor function is not used, (Key) are free, however, when monitor function is used, necessary function is assigned to each key. In this case, specified function by customers' program and function for monitoring may be doubly assigned to one key, careful attention is necessary.

In Mode 2, if error is issued, screen will be totally erased once even while customers' message is displayed, and error message will be displayed.

mode number	mode name	free / reserved	free keys
0	system mode	reserved (with all functions)	—
1	user mode 1	free with monitor function	<b>Key</b> (reserved during monitor)
2	user mode 2	free with monitor function error message display	<b>Key</b> (reserved during monitor)
3	user mode 3	all free	<b>Key</b> <b>MENU</b>

Input from operating unit keys are read by DSW( ) function.

When the power is turned on, initial setting is 0, system mode.

## RELATED COMMANDS

DSW( ), OPU PRINT, CHARSIZE, CLS, CURSOR, NORMAL, REVERSE

## EXAMPLE

```

1000 OPUNIT 2                                'Select mode
1010 OPU PRINT 1,3,"Select operation mode."  'Message display
1020 OPU PRINT 1,4,"F1:Normal operation"
1030 OPU PRINT 1,5,"F2:Dummy operation"
1040 OPU PRINT 1,6,"F4:Operation End"
1050 WAIT ( DSW(3) AND &B10110000 ) <> 0  'Wait for function
                                                key input

1060 IF DSW(3) AND &B10000000 THEN GOTO F4KEY
1070 IF DSW(3) AND &B00100000 THEN GOTO F2KEY
1080 IF DSW(3) AND &B00010000 THEN GOTO F1KEY

```

# OPU PRINT



Operating Unit Print

**FUNCTION** Outputs characters to operating unit

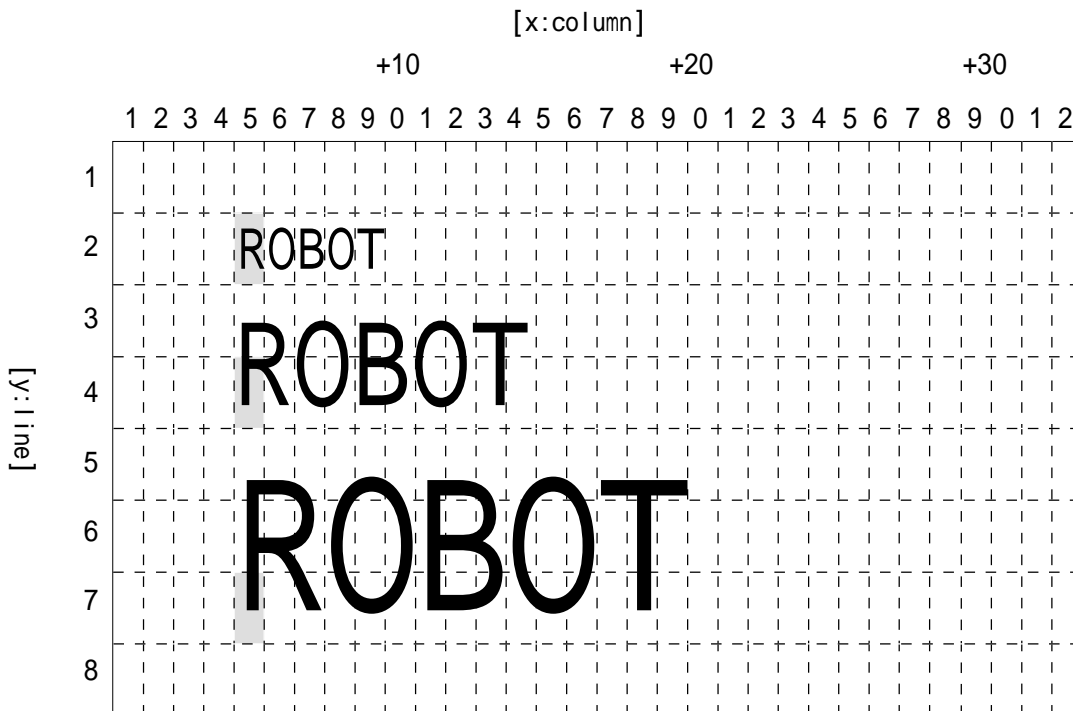
**FORMAT** OPU PRINT [x column],[y line],|"[string]" |{,|"[string]" |}n  
[numeric value]		[numeric value]
[variable name]		[variable name]
[function name]		[function name]


**DESCRIPTION** Outputs specified string to operating unit, and displays it at the specified place by (x,y) coordinate.

**RELATED COMMANDS** CHARSIZE, CLS, CURSOR, NORMAL, REVERSE, OPUNIT

**EXAMPLE**

```
>CHARSIZE 2
>OPU PRINT 5,2,"ROBOT"
>CHARSIZE 4
>OPU PRINT 5,4,"ROBOT"
>CHARSIZE 9
>OPU PRINT 5,7,"ROBOT"
```



 specified place by (x,y)

# OUT

&gt;

S

O

**FUNCTION** Sends data to output port (in 8 bit units)

**FORMAT** OUT [port number],[output data]

\* port number: integer from 0 to 15

output data: integer from 0 to 255

**DESCRIPTION** Sends data to output specified by port number.

Each port is comprised of 8 I/O bits. Using a standard 16I/16O, there are a total of 2 ports.

The port number, LSB/MSB, and I/O bit number are related as follows:

port number	MSB	7	6	5	4	3	2	1	LSB
0	7	6	5	4	3	2	1	0	0
1	15	14	13	12	11	10	9	8	8

LSB: least significant bit

MSB: most significant bit

Output data can be specified as either decimal or hexadecimal (using &H).

If an output bit which is set up as a REMOTE3 is specified, error will occur. REMOTE3 output bits are turned on or off automatically according to the status of robot controller.

RESET resets all bits to off.

**NOTE**



All of output bits will be turned off when emergency stop occurs.

To maintain the current stats, turn on bit 6 of software switch SS1. Refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

**RELATED COMMANDS**

IN, OPBCD, INBCD

**EXAMPLE**

>OUT 0,28 'Output port bits 2, 3, and 4 are on, the others are off (on port 0)

7	6	5	4	3	2	1	0	bit
0	0	0	1	1	1	0	0	→ 28

>OUT 1, &H66 'Output port bits 9, 10, 13, and 14 are on, the others are off (on port 1)

7	6	5	4	3	2	1	0	bit
0	1	1	0	0	1	1	0	→ &H66

# OUT \$



**FUNCTION** Sends data to memory I/O port (in 8 bit units)

**FORMAT** OUT \$[port number],[output data]

\* port number: integer from 0 to 63  
 output data: integer from 0 to 255

**DESCRIPTION** Sends data to memory I/O port specified by port number.

Each port is comprised of 8 memory I/O bits. Since there are a total of 512 bits, memory I/O is made up of 64 ports.

The port number, LSB/MSB, and memory I/O bit number are related as follows:

port number	MSB 7	6	5	4	3	2	1	LSB 0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
62	503	502	501	500	499	498	497	496
63	511	510	509	508	507	506	505	504

LSB: least significant bit  
 MSB: most significant bit

**RELATED COMMANDS** IN(\$)

**EXAMPLE** >OUT \$3,5



# PALET



Pallet

**FUNCTION** Defines and displays pallets

**FORMAT** (1) PALET[pallet number] P[point 1],P[point 2],P[point 3]{,P[point 4]},~  
[divisions 1],[divisions 2]



\* pallet number: integer from 0 to 15

[divisions 1]: number of points from P[point 1] to P[point 2]

[divisions 2]: number of points from P[point 1] to P[point 3]

The product of [divisions 1] times [divisions 2] must be less than 32767.

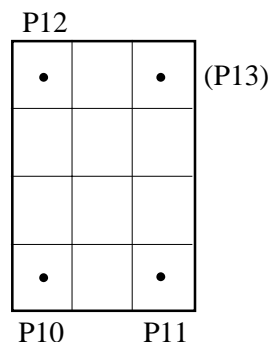
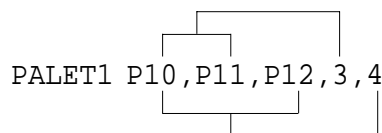
(2) PALET

**DESCRIPTION** (1) Defines a pallet by teaching the robot, as a minimum, points 1, 2, 3, and by specifying the number of points from point 1 to point 2 and from point 1 to point 3.

If the pallet is a well ordered rectangular shape, only three of the four corner points need to be specified. A less than well ordered pallet must be defined with all four corner points.

To define a pallet, first teach the robot either 3 or 4 corner points, then define the pallet as follows:

A pallet with 3 points from point 1 to point 2, and 4 points from point 1 to point 3 is shown at the lower left on this page. P10 corresponds to point 1, P11 corresponds to point 2, and P12 corresponds to point 3; there are three points from P10 to P11, and four points from P10 to P12. Hence, to define this pallet, specify:



10	11	12
7	8	9
4	5	6
1	2	3



As shown at the lower right on the previous page, points that represent divisions of a pallet are automatically assigned division numbers, which, in this example, begin at P10. These division numbers are also required by the PALETn( ).

Be aware that mistaking the order of points or the number of divisions between points will result in an incorrect pallet shape definition.

The pallet plane is defined by the Z axis coordinate values of the three corner points, P10, P11, and P12. Therefore, an upright pallet can also be defined.

(2) Displays all defined pallets.

**RELATED  
COMMANDS**

PALETn( )

**EXAMPLE**

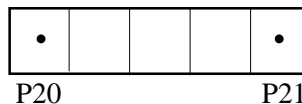
```
100 PALET1 P1 ,P2 ,P3 ,P4 , 4 ,5
110 FOR I=1 TO 20
120   JUMP PALET1(I)
130 NEXT
```

'Define a pallet with four points, P1 through P4, specified

```
>PALET
palet 1 P1 ,P2 ,P3 ,P4 , 4 , 5
>
```

A single-row pallet can be defined with a three-point PALET statement or command. To do so, teach point at each end, and define as follows. Specify 1 as the number of divisions between the same point.

```
PALET2 P20 ,P21 ,P20 , 5 , 1
```



# PALETn( )

F

Pallet

**FUNCTION** Returns position on specified pallet corresponding to specified division number

**FORMAT** PALET[pallet number]([division number])

\* pallet number: integer from 0 to 15  
 division number: integer from 1 to 32767

**DESCRIPTION** Returns position on a specified pallet (pallet number) that corresponds to a specified division number (division number).

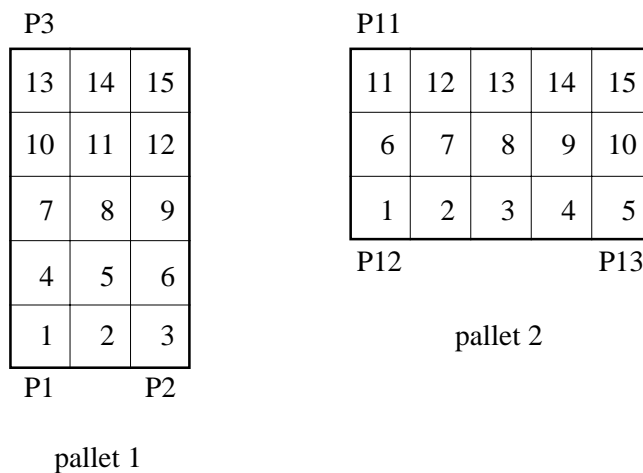
In positioning with PALETn( ), operators +, -, and : are allowed.

**RELATED COMMANDS** PALET

**EXAMPLE** This example program causes the robot to move parts from pallet 1 to pallet 2.

```

10 PALET1 P1 , P2 , P3 , 3 , 5      'Define pallet 1
20 PALET2 P12 , P13 , P11 , 5 , 3  'Define pallet 2
30 FOR I=1 TO 15
40 JUMP PALET1(I)                  'Move to pallet 1 division number I
50 ON 1                             'Grip part
60 WAIT 0.5
70 JUMP PALET2(I)                  'Move to pallet 2 division number I
80 OFF 1                             'Release part
90 WAIT 0.5
100 NEXT I
110 END
  
```



# PASS

&gt;

S

**FUNCTION** Executes simultaneous four axis PTP motion, passing near but not through specified points

**FORMAT** PASS [point series]{, |ON |[bit number],[point series]}n  
|OFF|

**DESCRIPTION** Executes simultaneous four axis PTP motion, passing in order near, but not through, specified point series points.

To specify point series points, use points (P0,P1, ...) with commas ( , ) between points.

To specify a continuous series, either in ascending or descending order, use a hyphen ( - ), as in (Pi-Pj).

To turn output bits on or off while executing motion, insert an ON or OFF delimited by commas ( , ) between points. The ON or OFF is executed before the robot reaches the point immediately preceding the ON or OFF.

If PASS is immediately followed by another PASS, control passes to the following PASS without the robot stopping at the preceding PASS final specified point.

If PASS is immediately followed by a motion command other than another PASS, the robot stops at the preceding PASS final specified point, but FINE positioning will not be executed.

If PASS is immediately followed by a command, statement, or function other than a motion command, the immediately following command, statement, or function will be executed prior to the robot reaching the final point of the preceding PASS.

If FINE positioning at the target position is desired, follow the PASS with a GO, specifying the target position as shown in each of the following examples:

```
PASS P5;GO P5;ON 1;MOVE P10
```

PASS is useful for avoiding obstacles or for rough painting with a Cartesian robot.

**RELATED COMMANDS** SPEED, ACCEL, GO

**EXAMPLE** >PASS P0 ,P2-P5 ,ON 2 ,P6 ,P120-P110  
>\_

# PATH



**FUNCTION** Specifies, cancels and displays path for executing batch file

**FORMAT** (1) PATH={[path name]};[path name]}n}  
(2) PATH

**DESCRIPTION** (1) Specifies and cancels path for executing batch file. Also displays current path setting. When batch filename is inputted, searches specified path, and specified file, then execute it. Batch file cannot be executed if path is not specified. Before executing batch file, set proper path by PATH. Several paths can be described by punctuating with " ; ". If several paths are specified, file is searched in order from the beginning, then is executed when it is found. If "PATH=" only is specified without any path name, PATH specification is canceled. Batch file cannot be executed in this condition.

(2) Displays current PATH setting.

PATH specification is effective only for executing batch file.

When the power is turned on, no path is specified.

**RELATED COMMANDS** SETENV

**EXAMPLE** >PATH=\ ; \BIN 'Two paths are specified  
>PATH 'Displays the current PATH setting  
PATH=\ ; \BIN  
>PATH=\ 'Route directory is specified as path  
>PATH  
PATH=\  
>

P

---

# PAUSE

---

S

**FUNCTION** Temporarily stops program execution

**FORMAT** PAUSE

**DESCRIPTION** Temporarily stops program execution.

Usually PAUSE is used to temporarily stop all task execution.

However, if any task(s) have been specified with the HTASK, only the HTASK task(s) are temporarily stopped, execution of all other tasks continue.

If a motion command is in a task which is not specified with HTASK the PAUSE temporarily stops the task at the motion command.

**EXAMPLE**

```
10 JUMP P1
20 ON 1
30 WAIT 0.5
40 PAUSE
50 JUMP P2
60 OFF 1
70 WAIT 0.5
80 GOTO 10
```

'Temporary stop. If subsequent action is to press START switch then execute from line 50, if subsequent action is to press RESET switch then execute reset

# PDEL



Point Delete

**FUNCTION** Deletes specified position data

**FORMAT** PDEL [[point number] |  
 [[beginning point number] - |  
 [[beginning point number] - [ending point number] |  
 | - [ending point number] |

\* Point number must be an integer. Allowable range is from 0 to 1 less than the PNTSIZE value.

Because the initial PNTSIZE value is 200, initial allowable range is from 0 to 199.

**DESCRIPTION** Deletes position data of specified position data.

Point numbers are specified as follows:

PDEL [point number]
Deletes position data of specified point number.
PDEL [beginning point number]-
Deletes all position data from beginning point number to last point number.
PDEL [beginning point number]-[ending point number]
Deletes all position data from beginning point number up to and including ending point number. To prevent Error 2 from occurring, beginning point number must be less than ending point number.
PDEL -[ending point number]
Deletes all position data up to and including ending point number.

## RELATED COMMANDS

PLIST, CLEAR, PNTSIZE

## EXAMPLE

```
P1=10,300,-20,0/L
P2=0,300,-40,0
P10=-50,350,0,0
>PDEL 1-2           'Delete Point 1 and Point 2
>PLIST
P10=-50,350,0,0
>PDEL 50           'Delete Point 50
>PDEL 100-        'Delete from Point 100 to last point
```

P

# PEEK( )

F

**FUNCTION** Reads data from I/O channel

**FORMAT** PEEK([address number])

\* address number: integer from 0 to 4095

**DESCRIPTION** Reads data from optional VME-I/O channel board.  
Address number is the offset value from the I/O channel base address (FFF000H).

< Example >

address number	address
0	FFF000H
1	FFF001H
2	FFF002H
3	FFF003H

Executing PEEK with a non-existent address causes a system error (bus error).

**RELATED  
COMMANDS** POKE

**EXAMPLE** >PRINT PEEK(0) 'Read data from address FFF000H  
5  
>



# PLIST



Point List

**FUNCTION** Displays position data

**FORMAT** PLIST {[beginning point number]-[ending point number]}{/W}  
 |-[ending point number] |  
 |\* |  
 |[point number] |

**DESCRIPTION** Displays specified position data. Displayed data are different depend on the usage of PLIST.

PLIST
If neither point numbers or asterisk * is specified, all position data in main memory will be displayed.
PLIST [point number]
To display position data for one point, specify only the point number.
PLIST [beginning point number]-
If ending point number specification is omitted, position data from beginning point number to last point in memory is displayed.
PLIST [beginning point number]-[ending point number]
Display position data from beginning point number to ending point number. In this case, the beginning point number must be smaller than the ending point number. If point number specification is not correct, error 2 will be issued.
PLIST -[ending point number]
If beginning point number specification is omitted, position data from initial point in memory to ending point number is displayed.
PLIST *
To display position data for current position, specify with asterisk *.

If /W is specified, position data will be displayed in fixed format (see example).

**RELATED  
COMMANDS**

PDEL, CLEAR, PNTSIZE

P

**EXAMPLE**

```

>PLIST
P0=450,400,0,0
P1=1.325,330.17,-58.2,-8/L
P11=-200,400,-100,150/1/R
P13=-450.456,200,0,0/2
P14=450.000,-200.000,0,0/2
>
>PLIST /W
P000= 450.000, 400.000, 0.000, 0.000
P001= 1.325, 330.170, -58.200, -8.000 /L
P011= -200.000, 400.000, -100.000, 150.000/1/R
P013= -450.000, 200.000, 0.000, 0.000/2
P014= 450.000, -200.000, 0.000, 0.000/2
>
>PLIST *
      254.3      365.256      '[X coordinate value] [Y coordinate value]
      -82.963      147      '[Z coordinate value] [U coordinate value]
PLIST 8
PLIST 100-/W
PLIST -150
PLIST 50-100

```

# PLS( )

F

Pulse

**FUNCTION** Returns pulse value of specified axis

**FORMAT** PLS([axis number])

\* axis number: integer from 1 to 4

**DESCRIPTION** Returns current pulse value (4-byte integer) of specified axis.

PLS( ) is intended for monitoring the robot orientation.

To use the pulse value of an axis while the robot is in motion, PLS( ) must be used in a task (procedure) other than the task for robot control.

PLS( ) use axis numbers 1 through 4, which correspond to axis names. (Axis numbers 0 through 3 were used in SPEL III Ver. 2.2 and earlier versions.)

**RELATED  
COMMANDS** PULSE, AGL( )

**EXAMPLE**

```

10 FUNCTION MAIN      'Display the pulse values of axis 1 and axis 2 every 2 seconds
20 LONG J1P,J2P
25 XQT !2,RB
30 J1P=PLS(1)
40 J2P=PLS(2)
50 PRINT J1P,J2P
60 WAIT 2
70 FEND
100 FUNCTION RB
110 SELRB 1
120 GO P1;WAIT 0.3
130 GO P2;WAIT 0.3
140 GOTO 120
150 FEND

```

P

# Pn=Position Specification



Point Number

**FUNCTION** Defines a point

**FORMAT** P[point number]=[position specification]

**DESCRIPTION** Defines a point and includes it in point data.

The position specification can be in any one of the following three forms. These position specification formats may be used in any command or statement [position specification].

<p>① [X coordinate],[Y coordinate],[Z coordinate],[U coordinate]~</p> <p style="text-align: right;">{/[local coordinate system number]}{/L   }  R </p>
<p>② P [point number]  {/local coordinate system number}  {/L   } +  X [coordinate value] 4</p> <p style="text-align: right;"> *    R    -   Y    :   Z   U </p>
<p>③ PALET [pallet number]  ([division number]) +  X [coordinate value] 4</p> <p style="text-align: right;">  -   Y    :   Z   U </p>

If / [local coordinate system number] is specified, SPEL III includes the defined point in the point data for that local coordinate system.

The /L and /R options are effective for SCARA robots only.

Specify /L to define the point as left arm posture point data.

Specify /R to define the point as right arm posture point data.

If omitted, right arm posture point data is assumed (except in some special configuration).

P\* refers to the point at which the arm is currently stopped.

The X, Y, Z, or U coordinates of position data can be changed by following a position data item, such as P[point number], with operators +, -, or :.

X, Y, Z, and U refer to coordinate axes.

Use + to add the specified value [coordinate value] to the specified axis coordinate value.

Use - to subtract the specified value [coordinate value] from the specified axis coordinate value.

Use : to substitute the specified value [coordinate value], for the specified axis coordinate value.

P

## RELATED COMMANDS

PLIST, PDEL, LOCAL, PALET

## EXAMPLE

```
>P1=300,200,-50,100
>P2=-400,200,-80,100/L      'Specify left arm posture
>
>P3=P2+X20                  'Add 20 to X coordinate of P2, and define resulting
                             point as P3
>PLIST 3
P3=-380,200,-80,100/L
>
>P4=P2-Y50:Z-30/R          'Subtract 50 from Y coordinate of P2, substitute -30
                             for Z coordinate, and define the resulting point P4
                             as right arm posture
>PLIST 4
P4=-400,150,-30,100/R
>
>P5=P*                      'Define current point as P5
>
>P6=PALET3(5)+U90          'Add 90 to U coordinate of PALET3(5), and define
                             resulting point as P6
>
```

# PNTSIZE, PSIZE



Point Size

**FUNCTION** Specifies and displays the usable number of position data

**FORMAT** (1) PNTSIZE [number of position data]

\* number of position data: integer from 1 to 1000 [default value: 200]

(2) PNTSIZE

**DESCRIPTION** (1) Specifies usable number of position data stored in the main memory of the controller.

(2) Displays current usable number of position data.

Changing the number of position data does not clear program area, but it does clear main memory point area, backup variable area, and object area. Therefore changing the number of position data necessitates the following:

Prior to changing → Perform backup of the main memory position data

After changing → Restore backup variables

After entering PNTSIZE [number of position data], the following prompt appears:

```
>Point, Backup Variable, Object all clear --> OK?
```

To specify a new number of data → enter  or

To maintain the current number of data → enter  or

Because the robot controller main memory size is fixed, increasing the usable number of position data correspondingly decreases the object area size. For this reason, usable data number increases should be minimized.

Usable point numbers are from 0 to one less than the usable number of position data.

Be aware that position data with position data numbers larger than the specified number of usable points cannot be sent to the robot controller main memory. For example, with the main memory usable number of position data specification still at its initial value of 200, attempting to send a position data file from programming unit that contains 1000 points will result in point number 0 to 199 being sent, point number 200 to 999 not being sent, and an error message being displayed.

Neither power off nor executing VERINIT changes PNTSIZE value.

**RELATED COMMANDS** PRGFSIZE, LIBSIZE, FREE, SYS

**EXAMPLE**

```
>PNTSIZE 500
>PNTSIZE
      500
>
```

# POKE

&gt;

S

P

**FUNCTION** Writes data to I/O channel

**FORMAT** POKE [address number],[data]

\* address number: integer from 0 to 4095

**DESCRIPTION** Writes data to optional VME-I/O channel board.  
Address number is the offset value from the I/O channel base address (FFF000H).

< Example >

address number	address
0	FFF000H
1	FFF001H
2	FFF002H
3	FFF003H

Executing POKE with a non-existent address causes a system error (bus error).

**RELATED  
COMMANDS** PEEK( )

**EXAMPLE** >POKE 3 ,&H10           'Write 10H to address FFF003H  
>

# POWER



**FUNCTION** Switches motor power mode and displays current mode and actual status

**FORMAT** POWER { |LOW|  
                  |HIGH|

\* [default: LOW]

**DESCRIPTION** Switches motor power mode and displays current status.

With LOW specified, mode switches to low power mode.

With HIGH specified, mode switches to high power mode.

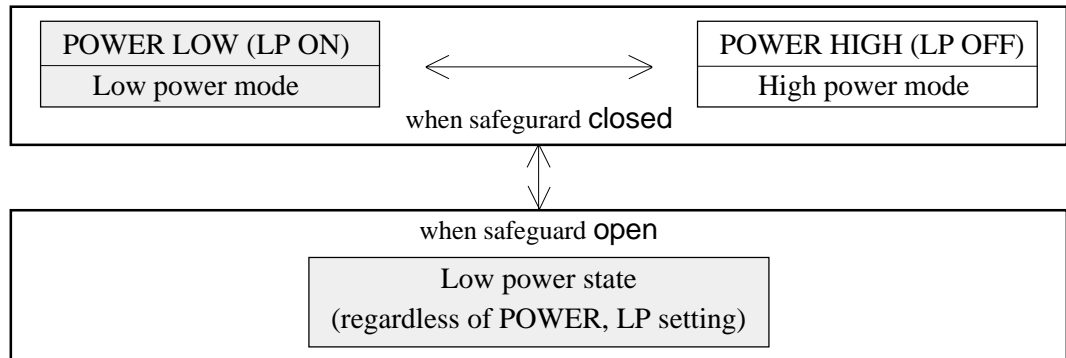
With no LOW/HIGH specification, POWER displays the current mode and the actual power status.

>POWER

Mode :Low      current mode

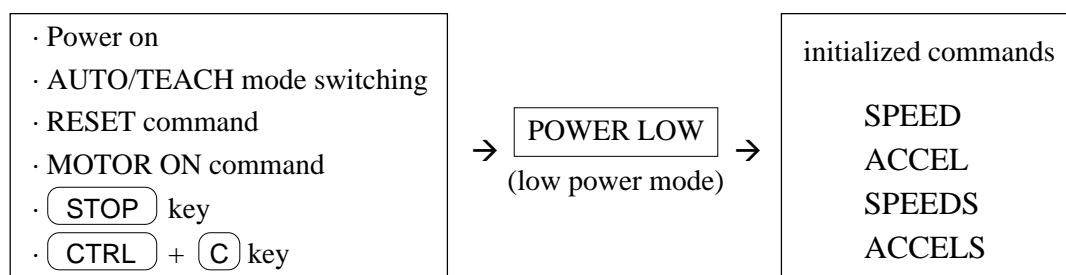
State:Low      actual status

The function of POWER and LP is same.



For the safety reasons, low power state is basic in TEACH mode for the controller.

Therefore, following operations (Reset operations) will turn the power mode to low as default setting. In this case, speed and acceleration setting also will be initialized to default value. Refer to "Specifications" in the manipulator manual for default values of speed and acceleration, which vary from model to model.





In low power state, motor power is limited, and effective motion speed setting is lower than the default value. If higher speed is specified directly or in a program, the speed is set to the default value.

motor power status	actual motion speed
Low power state	the default value of speed setting command the specified value of speed setting command
High power state	the specified value of speed setting command



If higher speed motion is required in AUTO mode for motion commands described in a program, specify POWER HIGH in the program.

In low power state, error 173 may occur if robot arm is pushed by hand or operation with down force is executed because of the limited motor power.

When speed setting command is inputted in low power state, the following message will be displayed.

It shows the robot is in low power state, and will move in low speed.

Low Power State : xxxxxx is limited to xxx

**RELATED COMMANDS**

LP, SPEED, ACCEL, SPEEDS, ACCELS

**EXAMPLE**

```

>SPEED 50                                     'Specifies higher speed
  Low Power State : SPEED is limited to 5
>ACCEL 100,100
  Low Power State : ACCEL is limited to 10
>JUMP P1                                       'Moves in low speed
>SPEED; ACCEL                                 'To display speed setting status
  Low Power State : SPEED is limited to 5
    50
    50      50
  Low Power State : ACCEL is limited to 10
    100     100
    100     100
    100     100
>XQT
>POWER HIGH                                  'Set high power mode
>JUMP P2                                       'Moves in high speed
    
```

# PRGNO



Program No.

**FUNCTION** Determines whether to apply up down count or binary coding to program number selection through remote connector (REMOTE3)

**FORMAT** PRGNO |0|  
|1|

\* 0: up down count, 1: binary coding [default value: 1]

**DESCRIPTION** If robot controller is connected through its remote connector (REMOTE3) with a sequencer, various remote control capabilities of the robot controller can be utilized by transmitting specific sequences of signals from the sequencer to the remote connector REMOTE3. One of the remote control capabilities is program number selection.

Refer to "7. I/O Remote Set Up" of robot controller manual for details.

Program number selection is executed either through up-down count or through binary coding.

To select up-down count, specify 0. Program numbers 00 through 64 are available for up-down count. Program number selection input  $2^0$  and  $2^1$  of REMOTE3 are used to input up-down count signals, and they are assigned the following functions:

Pin No.	Signal	I/O No.	Function
6	Program number selection input $2^0$	Input 4	UP
7	Program number selection input $2^1$	Input 5	DOWN

To select binary coding, specify 1. Program numbers 00 through 15 are available for binary coding. Program number selection input  $2^0$  -  $2^3$  of REMOTE3 are used to input binary code signals, and they are assigned the functions shown on the following page.

Power off does not change PRGNO setting. Executing VERINIT causes the setting to initialize to the default, binary coding.

program number	program number selection input			
	$2^3$	$2^2$	$2^1$	$2^0$
00	0	0	0	0
01	0	0	0	1
02	0	0	1	0
03	0	0	1	1
04	0	1	0	0
05	0	1	0	1
06	0	1	1	0
07	0	1	1	1
08	1	0	0	0
09	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

**RELATED  
COMMANDS**

DSW( )

# PRGSIZE



Program Size

**FUNCTION** Specifies and displays the program area size

**FORMAT** PRGSIZE {[area size]}

**DESCRIPTION** PRGSIZE changes the program area in main memory to specified size (area size).

If area size is omitted, PRGSIZE displays the current program area size.

When PRGSIZE is executed, the following prompt is displayed:

```
>Program, Point, Backup Variable, Object all clear --> OK?
```

To execute the command → enter  or

To cancel the command → enter  or

Completing PRGSIZE clears all of the program area, point area, backup variable area, and object area in main memory.

Therefore, perform the following steps before and after executing the command.

Before execution → Make backup copies of program and point data files that are in main memory

After execution → Reenter backup variables

Increasing the program area size correspondingly reduces the object area size because in most cases the main memory size is fixed. Therefore, keep program size increases to a minimum.

Neither power off nor executing VERINIT changes PRGSIZE value.

## RELATED COMMANDS

PNTSIZE, LIBSIZE, FREE, SYS

**EXAMPLE**

```
>PRGSIZE &H10000      'Set the program area size to 64 KB (&H: hexadecimal prefix)
>PRGSIZE
65536
>
```

P

SPEL III maps main memory as diagrammed below. The commands for changing the size of each area are shown at right.

area name	default settings		
	No.1	No.2	
source program area	64 k	128 k	↑ PRGSIZE
point area	200 point data		↓ PNTSIZE
backup variables area	10 variables, 0.5 k		↓ LIBSIZE
object program area	104 k	240 k	↓ Depends on the above settings
total	174 k	374 k	↓

default settings    No.2    Applies if controller contains optional expansion RAM, and the sizes of main memory and file memory have been set with dipswitch SD2 on the MPU board

                                 No.1    All other cases

# PRINT

&gt;

S

P

<b>FUNCTION</b>	Outputs data to console display	
<b>FORMAT</b>	<pre>PRINT [[numeric variable name]]{, [numeric variable name]]}n       [[string variable name]   [[string variable name]          "[string]"    "[string]"          [function]    [function]  </pre>	
<b>DESCRIPTION</b>	Outputs specified data to console display.	
<b>RELATED COMMANDS</b>	INPUT	
<b>EXAMPLE</b>	>PRINT A	'Display value of variable A
	>PRINT "LCD"	'Display "LCD"
	>PRINT "B=" ,B	'On the same line, display "B=", then display value of variable B
	B= 9	
	>	

# PRINT #



**FUNCTION** Outputs data to communication port

**FORMAT** PRINT #[port number],[[numeric value] |{,[[numeric value] |}n  
[[numeric variable name]		[[numeric variable name]
"[string]"		"[string]"
[string variable name]		[[string variable name]

\* port number: integer from 20 to 24

**DESCRIPTION** Outputs numeric values and strings to communication port specified by port number.

Port numbers and communication ports are related as follows:

port number	communication port
#20, #21	standard RS-232C port
#22, #23	auxiliary RS-232C port
#24	REMOTE1 (OPU-300)

**RELATED COMMANDS** INPUT #, CONFIG

**EXAMPLE**

```
>PRINT #20,5           'Send numeric value 5 to port 20
>PRINT #20,A           'Send contents of variable A to port 20
>PRINT #21,"PORT"     'Send string "PORT" to port 21
>PRINT #21,A,5         'Send contents of variable A and numeric value 5 to port 21
>PRINT #20,NAME$      'Send contents of string variable NAME$ to port 20
>
```

# PULSE

&gt;

S

P

**FUNCTION** Moves all four axes simultaneously by PTP motion according to specified pulse values, and displays current position pulse values

**FORMAT** (1) PULSE [axis #1 pulse value],[axis #2 pulse value],[axis #3 pulse value],~  
[axis #4 pulse value] {[!parallel processing statement!]}

\* pulse range for axis #1 to #4 : range (integer) defined by RANGE

(2) PULSE

**DESCRIPTION** (1) Moves all four axes simultaneously by PTP motion according to specified pulse values, rather than specified with orthogonal coordinate values.

(2) Displays current position pulse values for each axis as follows:

```
[axis #1 pulse value]    [axis #3 pulse value]
[axis #3 pulse value]    [axis #4 pulse value]
```

Refer to manipulator manual for the 0 pulse position and +/- orientation of each axis. These vary from model to model.

PULSE is intended primarily for use in maintenance.

**NOTE**



Unlike JUMP, PULSE moves all axes simultaneously, including axis #3 raising and lowering in traveling to target position. Therefore, when using PULSE, take extreme care so that the hand can move through an obstacle free path.

**RELATED  
COMMANDS**

SPEED, ACCEL, ! ... !, RANGE

**EXAMPLE**

```
>PULSE 16000,10000,-100,10
```

'Move axes #1, #2, #3, and #4 to the positions that correspond to pulse values 16000, 10000, -100, and 10, respectively

```
>PULSE
```

```
16000    10000
```

'Display pulse values that correspond to current position of axes #1 to #4

```
-100     -100
```

```
>
```

# QP

&gt;

S

## Quick Pause

**FUNCTION** Switches quick pause mode on or off and displays current mode status

**FORMAT** QP { |ON | }  
          |OFF|

\* [default: ON]

**DESCRIPTION** If during motion command execution either the operating unit PAUSE switch is pressed, or a pause signal is input to the controller REMOTE3 connector, quick pause mode determines whether the robot will pause immediately, or will pause after having executed the motion command.

Immediately decelerating and stopping is referred to as a "quick pause."

With ON specified, QP switches quick pause mode on.

With OFF specified, QP turns quick pause mode off.

With no specification, QP displays current quick pause mode status.

Software reset does not change quick pause mode status.

When the power is turned off and back on, quick pause mode defaults to ON.

Even if OFF is specified, the robot will pause immediately by safety door input.

In teach mode, pressing the [Esc] key on the programming unit (PC) also causes the robot to pause. However, the robot does not pause immediately, whether quick pause mode is on or off.

**EXAMPLE**

```
>QP ON      'Turn quick pause mode on
:
>QP
QP ON      'Quick pause mode is currently on
>
```



# QUIT

&gt;

S

<b>FUNCTION</b>	Stops tasks that are currently being executed, or have been temporarily stopped	
<b>FORMAT</b>	(1) QUIT ![task number]	
	* task number: integer from 1 to 16	
	(2) QUIT	
<b>DESCRIPTION</b>	(1) Stops tasks that are currently being executed, or that have been temporarily stopped.	
	Use the END to stop the task in which it is included.	
	(2) In edit mode for text file by EDIT, QUIT is used to exit to programming mode without saving file.	
<b>RELATED COMMANDS</b>	RUN, XQT, HALT, RESUME	
<b>EXAMPLE</b>	<pre> 10 FUNCTION MAIN 20 XQT !2 FLASH1 30 XQT !3 FLASH2 40 WAIT 10 50 QUIT !2;QUIT !3 60 FEND 70 ' 80 FUNCTION FLASH1 90 LOOP1: 100 ON 1;WAIT 0.2 110 OFF 1;WAIT 0.2 120 GOTO LOOP1 130 FEND 140 FUNCTION FLASH2 150 LOOP2: 160 ON 2;WAIT 0.5 170 OFF 2;WAIT 0.5 180 GOTO LOOP2 190 FEND </pre>	<pre> 'Execute FLASH1 at Task 2 'Execute FLASH2 at Task 3 'Stop Task 2 and Task 3 'This task flashes at 0.2 second interval 'This task flashes at 0.5 second interval </pre>

Q

# RANGE

&gt;

S

**FUNCTION** Defines permissible working range of each axis in pulses, and displays current permissible ranges

**FORMAT** (1) RANGE [axis #1 lower limit pulse value], [axis #1 upper limit pulse value], ~  
[axis #2 lower limit pulse value], [axis #2 upper limit pulse value], ~  
[axis #3 lower limit pulse value], [axis #3 upper limit pulse value], ~  
[axis #4 lower limit pulse value], [axis #4 upper limit pulse value]

(2) RANGE

**DESCRIPTION** (1) Defines the permissible working range for each axis by specifying the upper and lower limit in pulses.

The lower limit must not exceed the upper limit. A lower limit in excess of the upper limit will cause an error, making it impossible to execute motion.

Neither power off nor executing VERINIT changes RANGE working ranges.

(2) Displays current ranges as follows:

[axis #1 lower limit ]	[axis #1 upper limit ]
[axis #2 lower limit ]	[axis #2 upper limit ]
[axis #3 lower limit ]	[axis #3 upper limit ]
[axis #4 lower limit ]	[axis #4 upper limit ]

Refer to the "Specifications" in the manipulator manual for maximum working ranges, which vary from model to model.

**RELATED COMMANDS** XYLIM, VER

**EXAMPLE**

```
>RANGE 0,32000,0,32224,-10000,0,-40000,40000
>RANGE
      0   32000           'axis #1 range 0 to 32000
      0   32224           'axis #2 range 0 to 32224
-10000     0           'axis #3 range -10000 to 0
-40000  40000           'axis #4 range -40000 to 40000
```

# REAL

S

Real

**FUNCTION** Defines 4-byte REAL type variables

**FORMAT** REAL [variable name]{{array size 1{,array size 2{,array size 3}}}}~  
 {,[variable name]{{array size 1{,array size 2{,array size 3}}}}n

**DESCRIPTION** This command defines 4-byte REAL type variables.

If several variables of the same type are declared, use a " , " (comma) and describe several variable names. When defining an array variable, declare the name and its size enclosed with ( ). The size can be defined up to 3 dimensions.

By default, the variable whose data type is not declared specifically will be treated as REAL type. Therefore, it should be declared if the integer type is sufficient or the data type because it has an advantage over the REAL type in terms of performance speed and memory efficiency.

The REAL type supplies 7 valid digits. If the high accuracy is needed, using DOUBLE (an 8-byte REAL number, 14 valid digits) is recommended.

For more details about the variable, refer to "2.3 Variables" in the Elementary section of the User's manual for SRC-300/320.

The following lists the restrictions on the variable names. The variable may be freely named within these restrictions.

- The usable characters are alphanumeric and underscores ( \_ ). There is no distinction between capital and small case letters.
- Must be eight characters or under.
- The first character must be an alphabet character other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. A reserved word that is followed by an underscore or numeric character is also read as a reserved word.



The variable type must be declared at the beginning of a line; otherwise, the file will not be successfully compiled. When declaring another type of variable, a new line must be created.

R

**RELATED  
COMMANDS**

BYTE, INTEGER, LONG, DOUBLE, STRING, VARIABLE, SYS

**EXAMPLE**

```
10 FUNCTION MAIN
20 REAL I           'Declares a 4-byte REAL type variable "I."
30 REAL ODATA(10,10) 'Declares a 2 dimensional array of 4-byte REAL type
                    variable, "ODATA."
    :
999 FEND
```

---

# RENAME, REN, NAME

---



**FUNCTION** Changes file name

**FORMAT** RENAME {[pathname]}[filename 1] [filename 2]

\* filename must include extension

**DESCRIPTION** Changes name of specified file (filename 1) to filename 2.

If pathname is omitted, RENAME searches for filename 1 in current directory.

Pathname can not be specified for filename 2. Therefore, the renamed file exists in the same path as prior to its being renamed.

Neither filename 1 or filename 2 may be omitted.

A file may not be renamed to a filename that already exists in the same path.

RENAME allows the use of wildcard characters.

Wildcard characters used in filename 1 cause RENAME to rename all corresponding files. For example, to change names of all files named A to B, without changing their filename extensions, use the following:

```
RENAME A.* B.*
```

Wildcard characters used in filename 2 cause RENAME to maintain filename 1 characters in their respective positions in filename 2. For example, to rename all files corresponding to TEST.\* to TEXT.\*, use the following:

```
RENAME TEST.* ??X?.*
```

To make file with filename 2 into a different directory, use the COPY.

**RELATED  
COMMANDS**

COPY

**EXAMPLE**

```
RENAME A.* B.*  
RENAME TEST.* ??X?.*  
RENAME A.PRG B.PRG
```

**R**

# RENDIR



Rename Directory

**FUNCTION** Changes directory name

**FORMAT** RENDIR {[pathname]}[directory name 1] [directory name 2]

**DESCRIPTION** Changes directory name 1 to directory name 2.  
If pathname is omitted, RENDIR searches for directory name 1 in current directory.  
Pathname specification is not allowed for directory name 2. Therefore, the renamed directory exists in the same path as prior to its being renamed.  
Neither directory name 1 or directory name 2 may be omitted.  
A directory may not be renamed to a directory name that already exists in the same path.  
Wildcard characters are not allowed in either directory name 1 or directory name 2.

**RELATED  
COMMANDS** DIR

**EXAMPLE** RENDIR \BAK USR2

# RENUM



Re-number

**FUNCTION** Renumbers program lines

**FORMAT** RENUM {[first line number]}{,[increment]}

\* (first line number)+(total number of program lines)×(increment)≤32767

**DESCRIPTION** Renumbers program lines. First program line is renumbered to first line number, all subsequent lines are renumbered at specified increment.

In first line number specification is omitted, first program line is renumbered to 10. If increment specification is omitted, increment is 10.

Specify increment such that (first line number) plus (total number of program lines times increment) does not exceed 32767.

RENUM also automatically renumbers line references within GOTO, GUSUB, and ONERR.

**RELATED  
COMMANDS** LIST

**EXAMPLE** >RENUM 100,20 'Renumber first line 100, renumber at 20 increment

>

>LIST

100 HOME  
120 JUMP P10  
140 JUMP P20  
160 GOTO 120



10 HOME
20 JUMP P10
30 JUMP P20
40 GOTO 20

>\_

>RENUM 'Renumber first line 10, renumber at 10 increment

>

**R**

# RESET



**FUNCTION**        Resets controller

**FORMAT**         RESET

**DESCRIPTION**    RESET resets the following:

emergency stop status	
error status	(except for special case errors)
output bit	(all bits off)
SPEED, SPEEDS	(initialized to default value)
ACCEL, ACCELS	(initialized to default value)
LIMZ parameter	(initialized to 0)
FINE	(initialized to default value)
POWER HIGH, LP OFF	(POWER LOW, LP ON)

For servo-related errors, emergency stop status, and any other conditions requiring a RESET reset, no command other than RESET will be accepted. In this case, first execute RESET, then execute other processing as necessary.

For example, after an emergency stop, first verify safe operating conditions, execute RESET, and then execute MOTOR ON.

**RELATED  
COMMANDS**        ON, OFF, SPEED, ACCEL, LIMZ, SPEEDS, ACCELS, MOTOR ON

**EXAMPLE**        >RESET



# RESUME

&gt;

S

**FUNCTION** Resumes execution of tasks temporarily stopped by HALT

**FORMAT** RESUME ![task number]

\* task number: integer from 1 to 16

**RELATED  
COMMANDS** RUN, XQT, HALT, QUIT

**EXAMPLE**

```

10 FUNCTION MAIN
20 XQT !2 FLASH           'Execute FLASH at Task2
30 LOOP:
40 WAIT 3                 'Set timer interval to 3 seconds (Task 2 is executing)
50 HALT !2                'Halt Task 2 (after line 40 3 seconds has elapsed)
60 WAIT 3                 'Set timer interval to 3 seconds (Task 2 is stopped)
70 RESUME !2              'Resume Task 2 (after line 60 3 seconds has elapsed)
80 GOTO LOOP
90 FEND
100 '
110 FUNCTION FLASH        'Flash light on/off at 0.2 seconds intervals
120 LOOP1:
130 ON 1
140 WAIT 0.2
150 OFF 1
160 WAIT 0.2
170 GOTO LOOP1
180 FEND

```

R

---

# REVERSE

---

&gt;

S

**FUNCTION** Specifies reverse display mode for specified range (on operating unit)

**FORMAT** REVERSE [x column],[y line],[number of characters]

\* x: integer from 1 to 32

y: integer from 1 to 8

number of characters: integer from 1 to 32

**DESCRIPTION** Specifies reverse display mode for specified number of characters from (x,y) position.

If specified area is longer than the end of one line, extra area does not slide to the next line, but returns to the beginning of the line, and displays characters in reverse mode.

Characters in specified range are displayed in reverse mode.

If characters are outputted into specified range, those characters are also displayed in reverse display mode.

**RELATED COMMANDS** NORMAL, CHARSIZE, CLS, CURSOR, OPUNIT, OPU PRINT

**EXAMPLE** >REVERSE 10,2,3

---

# RIGHT\$( )

---

F

**FUNCTION** Returns the rightmost characters of specified string

**FORMAT** RIGHT\$([string variable],[number of characters])  
|"[string]" |

**DESCRIPTION** Returns the rightmost specified number of characters of specified string.  
String may be specified by a string variable.

**RELATED  
COMMANDS** LEFT\$( ), MID\$( )

**EXAMPLE** >PRINT RIGHT\$( "ABCDE" , 2)  
DE  
>

R

# RMDIR, RD



Remove Directory

**FUNCTION** Removes (deletes) an empty subdirectory

**FORMAT** RMDIR {[pathname]}[directory name]

**DESCRIPTION** Removes (deletes) specified subdirectory. Prior to executing RMDIR all of the subdirectory's files except for ( . ) and ( .. ) must be deleted.

If pathname is omitted, searches for subdirectory in current directory.

If directory name is omitted, an error occurs.

Current directory or parent directory cannot be removed.

**EXAMPLE** RD \USR2

# ROPEN...CLOSE

S

Read Open

**FUNCTION** Opens file for read-out

**FORMAT** ROPEN "[file name]" AS #[file number]  
 :  
 CLOSE #[file number]

\* filename must include extension  
 file number: integer from 30 to 35

**DESCRIPTION** Opens specified filename for read-out and identifies it by the specified file number. This statement is used to open and read data from the specified file. CLOSE closes the file and releases the file number.

The specified filename must be the name of a file existing on disk.

The file number identifies the file as long as the file is open, it is used by the input statement for reading (by INPUT #) and for closing (by CLOSE #) the file. Accordingly, until the current file is closed, its file number can not be used to specify a different file.

A maximum of six files can be open concurrently. As long as 6 files are open, however, DLOAD and DMERGE cannot be executed.

**RELATED COMMANDS** INPUT #, AOPEN, WOPEN

**EXAMPLE**

```

100 REAL DATA(200)
110 WOPEN "TEST.VAL" AS #30
120 FOR I=0 TO 100
130 PRINT #30,DATA(I)
140 NEXT
150 CLOSE #30
160 '
170 ROPEN "TEST.VAL" AS #30
180 FOR I=0 TO 100
190 INPUT #30,DATA(I)
200 NEXT
210 CLOSE #30
220 '

```

R

---

# RSHIFT( )

---

F

Right Shift

**FUNCTION** Shifts numeric value data to right**FORMAT** RSHIFT([numeric value data],[number of bits to be shifted])**DESCRIPTION** Shifts specified numeric value data specified number of bits to the right (toward least significant bit). For each place shifted to the right, a 0 is inserted into the vacated space on the left.**RELATED  
COMMANDS** LSHIFT( ), NOT( )**EXAMPLE**

```
>PRINT RSHIFT(4,2)
1
>I=10
>PRINT RSHIFT(I,1)
5
>
```

# RUN



**FUNCTION** Compiles and executes source program

**FORMAT** RUN {"[[pathname]][filename]}"

\* filename extension is not allowed

**DESCRIPTION** Compiles and executes specified file.

If pathname and filename specifications are omitted, source program in the main memory is compiled and executed.

If filename is specified, the specified file in file memory is compiled and executed. During compiling, the following files will be created in file memory:

filename.OBJ  
filename.SYM

Just before the program execution, the following files will be loaded into main memory:

filename.OBJ  
filename.SYM  
filename.PNT

Existing main memory position data will be replaced when the above files are loaded. Therefore, save main memory files as necessary prior to executing RUN.

If pathname is omitted, RUN searches for specified files in current directory.

Executing RUN is equivalent to executing COMPILE, and then XQT continuously.

**NOTE**



Existing main memory position data will be replaced when RUN is executed. Therefore, save main memory files as necessary prior to executing RUN.

**RELATED  
COMMANDS**

COMPILE, XQT, HALT, RESUME, QUIT

**EXAMPLE**

```
>RUN "TEST"  
>
```

# SEL



Select

**FUNCTION**      Selects and displays step jog feed travel settings

**FORMAT**        SEL [bank number]

\* bank number : integer from 0 to 3  
 [default value: 0]

**DESCRIPTION**      Selects a bank (array) that contains the step jog feed travel settings for each axis which are applied each time a jog key is pressed to operate the robot in step jog mode.

SEL then displays the feed travel settings contained in the specified bank (bank number) in the following form:

[setting 1]   [setting 2]  
 [setting 3]   [setting 4]

Settings contained in each bank can be changed using the SET.

The following table shows, for each of the four settings, the direction and axis of arm motion, and the unit of feed.

Plus and minus signs refer to the direction of motion.

< feed travel settings for Base mode and Tool mode >

	setting 1		setting 2		setting 3		setting 4	
Base	X axis		Y axis		Z axis		U axis	
Tool	x axis		y axis		z axis		u axis	
direction	-	+	+	-	+	-	+	-
unit	mm						° (degrees)	

X through Z axes for Base mode correspond to X through Z axes in the robot coordinate system.

x through z axes for Tool mode correspond to x through z axes in the tool coordinate system.



< feed travel settings for Joint mode >

	setting 1		setting 2		setting 3		setting 4	
Axis	1st axis		2nd axis		3rd axis		4th axis	
direction	-	+	+	-	+	-	+	-
unit	mm for linear axis ° (degrees) for rotary axis						° (degrees)	

For more information on each jog feed coordinate mode, refer to "Jog Feeding Coordinate Modes" in User's manual.

**S**

## RELATED COMMANDS

SET

## EXAMPLE

```
*SEL 1
  0.1  1
  10   5
```

# SELECT...SEND

S

Select...Select End

**FUNCTION** Specifies branching formula and corresponding branch instruction sequences

**FORMAT**

```
SELECT [formula]
      CASE [item];[statement]
      :
      CASE [item];[statement]
      {DEFAULT;[statement]}
SEND
```

\* nesting: up to 20 are allowed

**DESCRIPTION** If any one CASE item is equivalent to SELECT formula result, that CASE item statement is executed. After execution, program control transfers to command following SEND.

If no CASE item is equivalent to SELECT formula result, DEFAULT statement is executed, and program control transfers to command following SEND.

If DEFAULT is omitted, nothing is executed and program control transfers to command immediately following SEND.

SELECT formula may include constants, variables, variable formulas, and logical operators that use AND, OR, or XOR.

CASE item may include constants and variables.

CASE item statements may also be multistatements or multiple line statements.

**EXAMPLE**

```
110 FUNCTION MAIN
120 INTEGER I
130 FOR I=0 TO 10
140   SELECT I
150     CASE 0;OFF 1;ON 2;JUMP P1
160     CASE 3;ON 1;OFF 2
170           JUMP P2;MOVE P3;ON 3
180     CASE 7;ON 4
190     DEFAULT;ON 7
200   SEND
210 NEXT
220 FEND
```

# SELRB

&gt;

S

Select Robot

**FUNCTION**      Selects positioning device

**FORMAT**        SELRB [device number]

\* device number: integer from 1 to 3  
[default value: 1]

**DESCRIPTION**    Used in a Function (FUNCTION...FEND) that controls a positioning device such as a manipulator or RAIOC, specifies which device the Function controls.

Device numbers correspond to the addresses of the respective devices.

1	manipulator
2	RAIOC at address #2
3	RAIOC at address #3

In the main task and Task 1, device number defaults to 1 ("SELRB 1"). Therefore, when only one manipulator is controlled by Task 1, a SELRB statement is not necessary. If more than one manipulator is controlled, however, the proper positioning device(s) must be specified with SELRB statement(s).

Once the positioning device has been specified by placing a SELRB statement in a Function, all position control statements following the SELRB are applied to that device till you place another SELRB statement.

**EXAMPLE**

```

10 FUNCTION MAIN
20 XQT !2 ROBOT
30 FEND
40 '
50 FUNCTION ROBOT
60 SELRB 1
70 JUMP P1
   ⋮
500 FEND

```

S

# SENSE

&gt;

S

**FUNCTION** Specifies and displays input condition that, if satisfied, complete the JUMP in progress by stopping robot above target position

**FORMAT** (1) SENSE [input condition]  
 :  
 JUMP [position specification] SENSE

\* The following functions and operators may be used in input conditions:  
 functions: SW, IN (either I/O or memory I/O may be used)  
 operators: AND, OR, XOR, +, \*  
 other : parenthesis for prioritizing operations, and variables

(2) SENSE

**DESCRIPTION** (1) SENSE command:  
 Specifies SENSE condition.  
 SENSE condition must include at least one input or memory I/O input function.  
 Include in SENSE condition only the operators noted above. (Using any other operator will result, not in error processing, but in unpredictable motion.)  
 When variables are included, their values are computed during SENSE execution.

Multiple SENSE statements are permitted, the most recent SENSE condition remains current until superseded.

At power on, SENSE condition is:

```
SENSE SW(0)=1      'Input Bit 0 is on
```

Use JS(0) or STAT(1) function to verify if SENSE condition is satisfied.

JUMP With SENSE Modifier:  
 Checks if current SENSE condition is satisfied. If satisfied, this command completes with robot stopped above target position.

(2) Displays current SENSE Condition.  
 Because the SENSE condition display format is altered to clearly indicate operation order, it may differ from the originally input format.

**RELATED  
COMMANDS**

JUMP, SW( ), STAT(1), JS(0)

**EXAMPLE**

```

1000 SENSE SW(1)=0           'Specifies SENSE condition (Input Bit 1
                             is off)
1010 JUMP P1 SENSE           'Stop above target if current SENSE
                             condition (line 1000) is satisfied
1020 SENSE SW(1)=1 AND SW($1)=1 'Specifies SENSE condition (Input Bit 1
                             is on and Memory I/O bit 1 is on)
1030 JUMP P2 SENSE           'Stop above target if current SENSE
                             condition (line 1020) is satisfied
1040 FOR I=1 TO 2
1050   SENSE SW(I)=1         '(Iteration 1) Specifies SENSE condition
                             (Input Bit 1 is on)
                             (Iteration 2) Specifies SENSE condition
                             (Input Bit 2 is on)
1060   JUMP PI SENSE
1070 NEXT
1080 I=5
1090 JUMP P4 SENSE           'Stop above target if current SENSE
                             condition (line 1050 Iteration 2) is
                             satisfied
1110 JUMP P6 SENSE           'Stop above target if current SENSE
                             condition (line 1050 Iteration 2) is
                             satisfied

>SENSE SW(1)=1 AND SW($1)=1
>SENSE
    SW(1)=1 AND SW($1)=1
>
>SENSE SW(0) OR SW(1) AND SW($1)
>SENSE
    (SW(0) OR SW(1)) AND SW($1)
>

```

# SET

&gt;

S

**FUNCTION** Sets step jog feed travels

**FORMAT** SET [bank number],[setting 1],[setting 2],[setting 3],[setting 4]

\* bank number: integer from 0 to 3

Feed travels can be specified in units of 0.001

**DESCRIPTION** Specifies step jog feed travels for each axis. These are applied each time a jog key is pressed to operate the robot in step jog mode. These settings are stored in an array called the bank. When specifying the step jog feed travels for each of the four axes, the bank number must be specified.

To reverse the feed direction of an axis, use a minus sign.

The following table shows the default settings in each bank.

	setting 1	setting 2	setting 3	setting 4
SEL 0	0.03	0.03	0.03	0.03
SEL 1	0.1	0.1	0.1	0.1
SEL 2	1	1	1	1
SEL 3	10	10	10	10

To select a bank of settings, specify the bank number with SEL.

For the direction and axis of arm motion that are applied when each jog key is pressed and the unit of feed travel, refer to SEL.

SEL values are not changed by power off, but they are initialized to the default values shown in the table above when the VERINIT is executed.

**RELATED  
COMMANDS** SEL

**EXAMPLE** >SET 1,10,5,3,3 'Sets feed travels in bank 1  
>SET 2,0.1,0.5,1,0 'Sets feed travels in bank 2

# SETENV



Set Environment

**FUNCTION** Specifies, cancels and displays environment variable

**FORMAT** SETENV { |COM={-V}{-L} | }  
 |PLI={/W} |  
 |XQT={ [path name] ; [path name] }n |  
 |PATH={ [path name] ; [path name] }n |

**DESCRIPTION** Specifies and cancels environment variable. Also displays the setting values. In the format above, the left side of = is called environment variable name, and contents of variable is specified by character strings on the right side.

Environment variable is effective when COMPILE and PLIST is executed, or file is executed.

If environment variable is specified, various settings, which have to be specified each time when those commands are executed, can be omitted.

If batch file is used, path must be specified in environment variable.

If SETENV only is inputted, the current environment variable contents are displayed. When the power is turned on, environment variable is not specified at all.

SETENV COM={-V}{-L}

Specifies compiling condition.

If -V only, -L only, or both are specified in environment variable, COMPILE is executed in specified condition when COMPILE is inputted.

If "COM=" only is specified without -V and -L, it cancels environment variable specification.

Environment variable name is COM, and it is impossible to describe as COMPILE.

SETENV PLI={/W}

Specifies display format for PLIST.

If /W is specified in environment variable, PLIST/W is executed when PLIST is inputted. If "PLI=" only is specified without /W, it cancels environment variable specification.

Environment variable name is PLI, and it is impossible to describe as PLIST.



<p><b>SETENV XQT={ [path name] ; [path name] }n</b></p> <p>Specifies path for executing file.</p> <p>When XQT "[filename]" is inputted, file is searched in the specified path in order, then is executed when it is found. If paths are not specified, file is searched in the current directory.</p> <p>If "XQT=" only is specified, it cancels environment variable specification.</p> <p>&lt; NOTE &gt; When path is defined in environment variable, but different path is specified with filename when XQT "filename" is executed, path setting in environment variable is invalid.</p>
<p><b>SETENV PATH={ [path name] ; [path name] }n</b></p> <p>Specifies path for execution batch file.</p> <p>If it is specified in environment variable, searches specified path, and specified file, then executes the file when batch filename is inputted.</p> <p>If "PATH=" only is specified, it cancels environment variable specification.</p> <p>This condition can be specified not only by SETENV but also by PATH.</p> <p>If PATH condition is not specified, batch file cannot be executed.</p>

**RELATED COMMANDS**

PATH, COMPILE, PLIST, XQT

**EXAMPLE**

```

>SETENV PLI=/W
>SETENV XQT=\
>SETENV
PLI=/W
XQT=\
>
>PLIST                                     'Display in PLIST/W format
P000=  450.000,  400.000,   0.000,   0.000
P001=   1.325,  330.170, - 58.200, - 8.000 /L
P011= -200.000,  400.000, -100.000, 150.000/1/R
P013= -450.000,  200.000,   0.000,   0.000/2
P014=  450.000, -200.000,   0.000,   0.000/2
>

```



# SETVER



Set VER

**FUNCTION** Restores the various setting data, which was saved as a file by MKVER, to the corresponding memory area

**FORMAT** SETVER

**DESCRIPTION** Executes batch file "MK#0.BAT" which is on current directory.

This batch file was made by MKVER. Following commands are described in the file, however, file contents will be different whether "/A" is specified or not when MKVER is executed.

Refer to MKVER for details.

CALPLS	HOFS	MCORDR	MCOFS
SEL	SET	ARCH	
TSPEED	TSPEEDS	HOMESSET	HORDR
ARM	ARMSET	TOOL	TLSET
RANGE	XYLIM	BASE 0	
CONFIG	CONSOLE	WIDTH	
PRGNO	OPUNIT	WEIGHT	MAXDEV
PRGSIZE	PNTSIZE	LIBSIZE	
Software switch settings		REMOTE3 setting	
PRG#0.PRG	} Commands for loading these files		
PNT#0.PNT			
LIB#0.BIN			
SYM#0.BIN			
OBJ#0.BIN			

When SETVER is executed, various setting data will be specified again and backup variables will be loaded onto main memory.

If commands for loading other files are described, those files also will be loaded onto main memory.

Various setting data includes important basic data such as HOFs etc..

It is recommendable to execute MKVER and keep those data as a file when a robot is installed for the first time.

If you want to know how to backup and restore, refer to "backing up and restoring a file" in chapter 3 of SPEL Editor manual or "[File] menu" of SPEL for Windows manual.

**RELATED  
COMMANDS**

MKVER, VER

**EXAMPLE**

This command will be used when MPU or SPU board in controller is replaced. It is especially useful for loading various setting data onto a new board.

- ① Save data on the board to be replaced (old board) as a file.

```
>MKVER                                'Following files will be made
                                         MK#0.BAT
                                         LIB#0.BIN
CPU Data Backup
Backup Variable Backup
```

- ② Make backup of all files including above files on file memory onto a disk of programming unit (PC).
- ③ Install a new board into a controller.
- ④ Restore all files on the disk of programming unit to controller file memory.
- ⑤ Execute SETVER.

```
>SETVER
```

# SFREE

&gt;

S

Servo Free

**FUNCTION** Disengages motor

**FORMAT** SFREE {[axis number]{,[axis number]}3}

**DESCRIPTION** Disengages motor. This condition is called "servo free."  
It is used when in direct teaching, and for other times when its desirable to cut power to a specific axis.

If axis number is omitted, frees all axes.

If axis numbers are specified, frees specified axes only.

For the third axis, because the electric brake activates, it cannot be moved by hand, even if SFREE is specified. Continuously pressing the third axis unit brake release switch allows manual manipulation.

Executing SFREE initializes the following:

SPEED	default value which varies from model to model
ACCEL	default value which varies from model to model
LIMZ	0

To engage axis, execute SLOCK or MOTOR ON.

Attempting to execute a motion command while in SFREE condition will cause an error. Turn on bit 5 of software switch SS6 to avoid this error when executing a motion command. If you want to know how to set software switch, refer to "9.2 Setting switches" of SPEL Editor manual or "[Setup] menu" of SPEL for Windows manual.

## RELATED COMMANDS

SLOCK, MOTOR

## EXAMPLE

< Example 1 >	< Example 2 >	
>SFREE	100 JUMP P1:Z0	'Perform assembly with axes #1 and #2 disengaged, using only axes #3 and #4
>SLOCK	110 SWITCH 1,&H10	
>SFREE 1,2	120 SFREE 1,2	
>SLOCK 1,2	130 GO P1	
	140 SLOCK 1,2	

# SGN( )

F

Sign

**FUNCTION** Returns sign of specified numeric value

**FORMAT** SGN([numeric value])

**DESCRIPTION** Returns sign of specified numeric value.

If numeric value is positive, returns	1
If numeric value is 0, returns	0
If numeric value is negative, returns	-1

**RELATED COMMANDS** ABS( ), INT( ), SQR( )

**EXAMPLE**

```
>PRINT SGN(0.55)
1
>PRINT SGN(-0.55)
-1
>
```

# SIN( )

F

Sine

**FUNCTION** Returns sine of specified angle

**FORMAT** SIN([radians])

**DESCRIPTION** Returns sine of the specified angle when specified in radians.

Angles in degrees must be converted to radians, using the following equation:

$$\text{radian} = \text{degrees} * \pi / 180 \quad (\pi = 3.141593)$$

**RELATED COMMANDS** COS( ), TAN( ), ATAN( ), ATAN2( )

**EXAMPLE**

```
>PRINT SIN(0.55)           'Display sine of 0.55 radians
.5226872
>PRINTSIN(30*3.141593/180) 'Display sine of 30 degrees
.5
>A=30*3.141593/180        'Display sine of 30 degrees using variable
>PRINT SIN(A)
.5
>
```

S

# SLOCK

&gt;

S

Servo Lock

**FUNCTION** Re-engages axis (from "servo free" condition)

**FORMAT** SLOCK {[axis number]{,[axis number]}3}

\* axis number: integer from 1 to 4

**DESCRIPTION** Re-engages axis from "servo free" caused by executing SFREE.

If axis number is omitted, engages all axes.

If axis number are specified, engages specified axis only.

SLOCK initializes the following:

SPEED	default value which varies from model to model
ACCEL	default value which varies from model to model
LIMZ	0

Engaging the third axis causes the electric brake to release.

To engage all axes, MOTOR ON may be used instead of SLOCK.

Executing SLOCK while in MOTOR OFF mode will cause an error.

**RELATED COMMANDS** SFREE, MOTOR

**EXAMPLE** < Example 1 >

```
>SFREE
```

```
>SLOCK
```

```
>SFREE 1,2
```

```
>SLOCK 1,2
```

< Example 2 >

```
100 JUMP P1:Z0          'Disengage axes #1 and #2, operate axes #3 and #4 only
```

```
110 SWITCH 1,&H10
```

```
120 SFREE 1,2
```

```
130 GO P1
```

```
140 SLOCK 1,2
```

---

# SPACE\$( )

---

F

**FUNCTION** Returns a string consisting of specified number of spaces

**FORMAT** SPACE\$([number])

**DESCRIPTION** Returns a string consisting of specified number of spaces.

**EXAMPLE**

```
> PRINT AB" +SPACE$( 5) + "CD"
AB-----CD
  |
  |----- 5 spaces
```

S

# SPEED



**FUNCTION** Specifies and displays speed for PTP motion

**FORMAT** (1) SPEED [speed specification value]{, [axis #3 upward speed specification value],~  
[axis #3 downward speed specification value]}




\* each speed specification value: integer from 1 to 100 (%)  
[default values: refer to the "Specifications" in the manipulator manual]

(2) SPEED

**DESCRIPTION** (1) Specifies speed for PTP motion (GO, JUMP, PULSE, and related) commands. Speed specification values to be integers from 1 to 100 (percent maximum speed). Axis #3 upward and downward speed values apply only to JUMP command. If omitted, each defaults to speed value.

(2) Displays current SPEED values as follows:  
[speed]  
[axis #3 upward speed] [axis #3 downward speed]

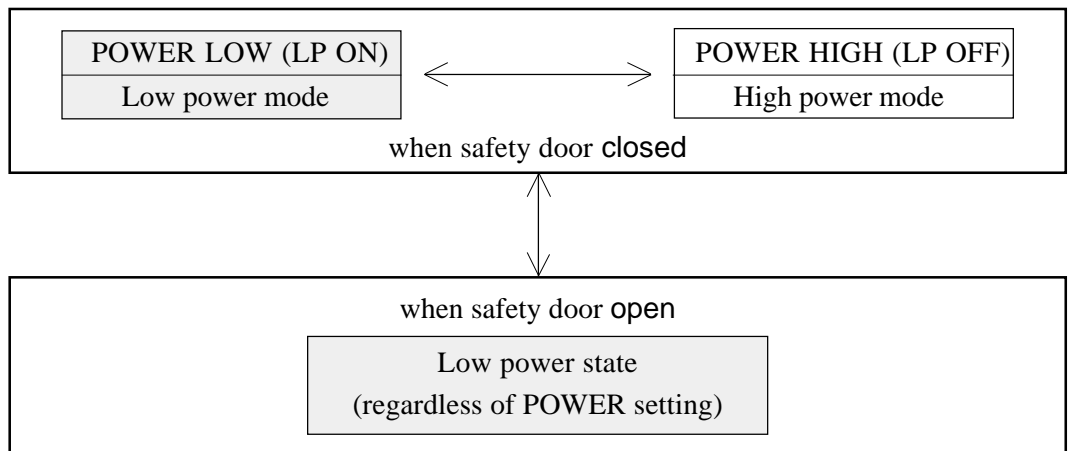
SPEED value initializes to default value when any one of the following is performed:

- Power on
- Mode switching (TEACH/AUTO)
- Software reset
- MOTOR ON
- SFREE, SLOCK
- VERINIT
-  key
-  +  key

While in TEACH mode, actual motion speed differs from low power state to high power state.

Motor power status	actual motion speed
Low power state	the default value of SPEED } either lower value
	the value of SPEED
High power state	the value of SPEED





If higher speed motion is required, set high power mode using POWER HIGH or LP OFF and close safety door. If safety door is open speed settings will be changed to default value.

If SPEED is executed when the robot is in low power state, the following message is displayed. The value in the message is the default value of SPEED which varies from model to model. The following example shows that the robot will move at default speed (5) because it is in low power state even though the speed setting value by SPEED is 80.

```
>SPEED 80
Low Power State : SPEED is limited to 5
>
>SPEED
Low Power State : SPEED is limited to 5
      80
      80      80
>
```

## RELATED COMMANDS

POWER(LP), TSPEED, ACCEL, GO, JUMP, PASS, PULSE

## EXAMPLE

```
>SPEED 100 , 100 , 50    'Only the axis #3 downward speed is specified 50
>SPEED
  100
  100    50
```

**S**

# SPEEDS



**FUNCTION** Specifies and displays speed for CP motion

**FORMAT** (1) SPEEDS [speed specification value]

\* speed specification value: integer from 1 to 1120 (mm/s)  
 [default value: refer to "Specifications" in the manipulator manual]

(2) SPEEDS

**DESCRIPTION** (1) Specifies hand speed in mm/s for CP motion (ARC, MOVE, and CVMOVE and related) commands.

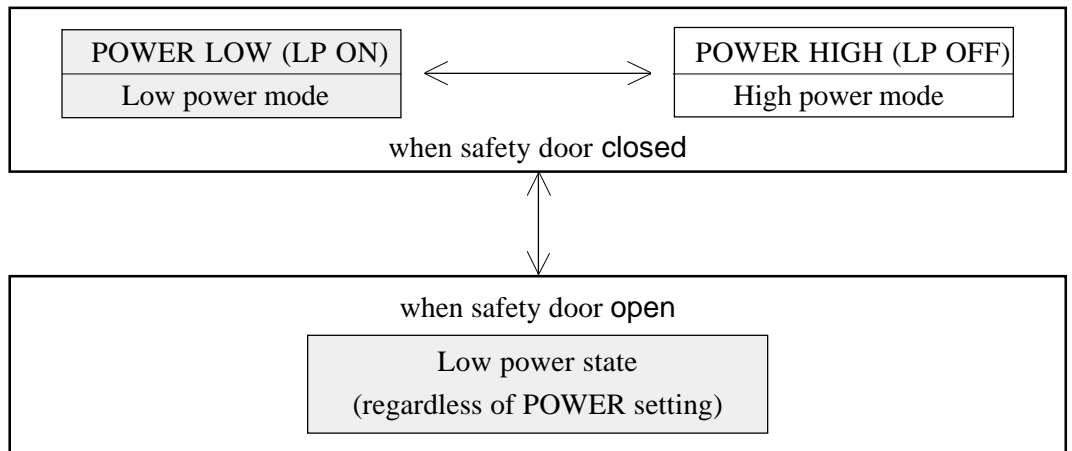
(2) Displays current SPEEDS value.

SPEEDS value initializes to default value when any one of the following is performed:

- Power on
- Mode switching (TEACH/AUTO)
- Software reset
- MOTOR ON
- SFREE, SLOCK
- VERINIT
- STOP key
- CTRL + C key

While in TEACH mode, actual motion speed differs from low power state to high power state.

Motor power status	actual motion speed
Low power state	the default value of SPEEDS the value of SPEEDS
High power state	the value of SPEEDS

**S**

If higher speed motion is required, set high power mode using POWER HIGH or LP OFF and close safety door. If safety door is open speed settings will be changed to default value.

If SPEEDS is executed when the robot is in low power state, the following message is displayed. The value in the message is the default value of SPEEDS which varies from model to model. The following example shows that the robot will move at default speed (50) because it is in low power state even though the speed setting value by SPEEDS is 800.

```
>SPEEDS 800
Low Power State : SPEEDS is limited to 50
>
>SPEEDS
Low Power State : SPEEDS is limited to 50
      800
>
```

## RELATED COMMANDS

POWER(LP), TSPEEDS, ACCELS, MOVE, CMOVE, ARC, CARC, CVMOVE

## EXAMPLE

```
100 SPEEDS 200      'Specify hand speed (200 mm/s)
110 MOVE P1
120 MOVE P20
130 SPEEDS 100     'Specify hand speed (100 mm/s)
140 MOVE P30
```

# SQR( )

F

Square Root

<b>FUNCTION</b>	Returns square root
<b>FORMAT</b>	SQR([numeric value])
<b>DESCRIPTION</b>	Returns square root of specified numeric value.
<b>RELATED COMMANDS</b>	ABS( ), SGN( ), INT( )
<b>EXAMPLE</b>	>PRINT SQR( 2) 1.414214 >

# STAT( )

F

Status

**FUNCTION** Returns status of the controller itself, or of another controller connected through RS-232C

**FORMAT** STAT([address])

\* address: 0 or 1 for the controller itself,  
integer (port number) from 20 to 23 for a controller connected through RS-232C

**DESCRIPTION** If 0 or 1 is specified for address, STAT( ) returns a 4 byte integer that presents the status of the controller itself. For detail, refer to Table 1 on the next page.

If an integer from 20 to 23 (number of an RS-232C port) is specified for address, STAT( ) returns a 3 byte integer that represents the status of the controller which is connected to the RS-232C port. For details refer to Table 2 on the next page.

Undefined bits shown in Table 1 and Table 2 must be masked with an AND, or similar operator.

<b>EXAMPLE</b>	<pre>&gt;PRINT (STAT(0) AND &amp;HOFF) 6 &gt;PRINT (STAT(20) AND &amp;H030000)/65536 1</pre>	<pre>'Display the task execution status of the controller itself 'Only Task 2 and 3 are being executed  'Display the execution/pause/error status of the controller connected to RS-232C port</pre>
----------------	--	---

S

address	bit	controller status indicated by ON bit
0	0 to 15	Task 1 is being executed (XQT) or in a halt (HALT) to Task 16 is being executed (XQT) or in a halt (HALT)
	16	Task(s) is being executed
	17	pause condition
	18	error condition
	19	TEACH mode
	20	emergency stop condition
	21	low power mode (POWER LOW or LP ON)
	22	Safeguard is open
	23	SRC-300: undefined SRC-320: Enable switch is ON
	24 to 31	undefined
1	0	Log of stop above target position upon satisfaction of condition in JUMP...SENSE statement. (This log is erased when another JUMP statement is executed.)
	1	Log of stop at intermediate travel position upon satisfaction of condition in GO/JUMP/MOVE...TILL statement. (This log is erased when another GO/JUMP/MOVE...TILL statement is executed.)
	2	Log of execution of MCAL command / statement.
	3	Log of stop at intermediate travel position upon satisfaction of condition in TRAP statement.
	4	MOTOR ON mode
	5	home position
	6	low power state
	7	undefined
	8	Axis #4 motor excited
	9	Axis #3 motor excited
	10	Axis #2 motor excited
	11	Axis #1 motor excited
12 to 31	undefined	

Table 1. Results of STAT( ) with 0 or 1 specified for address

bit	controller status indicated by ON bit
0 to 15	Memory I/O bit 0 is on to Memory I/O bit 15 is on
16	Task(s) is being executed
17	pause condition
18	error condition
19 to 23	undefined

Table 2. Results of STAT( ) with RS-232C port number specified for address

# STR\$( )

F

String\$

**FUNCTION** Returns specified numeric value as a numeric string**FORMAT** STR\$([numeric value])**DESCRIPTION** Returns specified numeric value as a numeric string.

For positive numeric values, the numeric string is returned with a space inserted prior to the first numeral, and another space inserted after the last numeral.

For negative numeric values, the numeric string is returned with minus sign (-) inserted prior to the first numeral, and a space inserted after the last numeral.

**EXAMPLE**

```

>PRINTSTR$( 5 )+" "+STR$( 6 )
_5__6_                                '_ represents a space
>
10 FUNCTION MAIN
20 STRING LETTER$;LETTER$=" "
30 INTEGER I
40 FOR I=1 TO 5
50LETTER$=LETTER$+STR$( I)            'Adding numeric string to LETTER$
60 NEXT I
70 PRINT LETTER$
80 FEND
>RUN
COMPILE END
_1__2__3__4__5_
>

```



# STRING

S

**FUNCTION** Defines string variable

**FORMAT** STRING [variable name]{{(array size 1{,array size 2{,array size 3}})}~  
 {,[variable name]{{(array size 1{,array size 2{,array size 3}})}}n

**DESCRIPTION** Defines string variable.

The last character of the string variable must be a \$, to distinguish it from a numeric variable. String variable name must be 8 characters or less, including \$.

If several variables of the same type are declared, use a " , " (comma) and describe several variable names. When defining an array variable, declare the name and its size enclose with ( ). The size can be defined up to 3 dimension. However, specifying a large array may cause the shortage of the object area because a string variable requires about 80 bytes per each array element. The use of the array of string variable must be kept at minimum.

Into a string variable, up to 79 characters can be set as data.

The following operations can be performed on string variables:

- + combining string variables
- = comparing string variables, is true only if the two string variables are identical
- <> comparing string variables, is true if at least one character of the two string variables are different

For more details about the variable, refer to "2.3 Variables" in the Elementary section of the User's manual for SRC-300/320.

The following cites the restrictions on the variable names. The variable may be freely named within these restrictions.

- The usable characters are alphanumeric and underscores ( \_ ). There is no distinction between capital and small case letters.
- Must be eight characters or under.
- The first character must be an alphabet character other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. A reserved word that is followed by an underscore or numeric character is also read as a reserved word.

S



The variable type must be declared at the beginning of a line; otherwise, the file will not be successfully compiled. When declaring another type of variable, a new line must be created.

**RELATED  
COMMANDS**

BYTE, INTEGER, LONG, REAL, DOUBLE, VARIABLE, SYS

**EXAMPLE**

```
10 FUNCTION LETTER
20 STRING LETTER$
30 INTEGER LETTER
40 LETTER$="Letter Count="
50 LETTER=LEN(LETTER$)
60 PRINT LETTER$,LETTER
70 FEND

RUN
COMPILE END
Letter Count=13

10 FUNCTION LETTER
20 STRING LETTER$
30 INPUT LETTER$
40 IF LETTER$="A" THEN GOSUB JOB1
50 IF LETTER$="B" THEN GOSUB JOB2
60 GOTO 30
70 JOB1:
   :
```

# SW( )

F

Switch

**FUNCTION** Returns input bit status

**FORMAT** SW([bit number])

\* bit number: integer from 0 to 127

**DESCRIPTION** Returns specified input bit status as either 0 or 1.

0: Off status

1: On status

**RELATED  
COMMANDS** WAIT, IF...THEN

**EXAMPLE**

```
80 A=SW(0)
```

'Store input bit 0 status at A

```
90 IF A=0 THEN GOTO 200
```

```
⋮
```

```
200 WAIT SW(5)
```

'Same as specifying WAIT SW(5)=1, waits until input bit 5 turns on

S

# SW(\$ )

F

Switch\$

**FUNCTION** Returns memory I/O bit status

**FORMAT** SW(\$[bit number])

\* bit number: integer from 0 to 511

**DESCRIPTION** Returns specified memory I/O bit status as either 0 or 1.

0: Off status

1: On status

**RELATED  
COMMANDS** ON \$, OFF \$, WAIT, IF...THEN

**EXAMPLE** >A=SW(\$18) 'Store 1 at A if memory I/O bit 18 is on, store 0 at A if memory I/O bit 18 is off

# SYS

S

System

**FUNCTION** Declares backup variables**FORMAT** SYS [type declaration] [variable name]{,[variable name]}n**DESCRIPTION** Declares backup variables and stores them in the backup variable area in main memory. Backup variables are so called because they are battery backed up when the controller is turned off. Once a backup variable has been declared it can be used until the backup variable area is cleared by CLRLIB command.

Variable types include the following:

BYTE	1 byte (-128 to +127)	}	integer
INTEGER	2 bytes (-32768 to +32767)		
LONG	4 bytes (-2147483648 to +2147483647)		
REAL	4 bytes, 7 digits	}	real number
DOUBLE	8 bytes, 14 digits		

Multiple variable names can be included in a single SYS (type declaration) statement. However, a new line must be started every time you declare a different variable type.

The SYS instruction must be issued from a SPEL program. This means that a program must be written and executed by RUN command (or by XQT command after COMPILE) in order to register (define) backup variables.

The same variable names are not allowed to be registered twice.

Backup variables should be registered in the program, which is not an application program. If the backup variables are registered in the application program, when the program is executed twice, error occurs.

To compile a source program in which particular variables are treated as backup variables, those variables must be declared in advance; otherwise, the variables will be treated as non-backup, real number type.

Backup variables are erased when CLRLIB is executed. Keep in mind that CLRLIB clears the entire backup variable area without allowing you to specify particular variables.

**RELATED COMMANDS** LIBRARY, CLRLIB, LIBSIZE, BYTE, INTEGER, LONG, REAL, DOUBLE

**EXAMPLE**

```
>10 FUNCTION B_UP
>20 SYS INTEGER V(50)
>30 SYS REAL A(10)
>40 FEND
>RUN
```

S

# SYSINIT



System Initialize

**FUNCTION**      Initializes main memory**FORMAT**        SYSINIT

**DESCRIPTION**      Initializes the sizes in main memory to default value as the following table.  
If SYSINIT is executed all data in the main memory will be erased.

area name	default settings	
	No.1	No.2
source program area	64 k	128 k
point area	200 point data	
backup variables area	10 variables, 0.5 k	
object program area	104 k	240 k
total	174 k	374 k

default settings    No.2: Applies if controller contains optional expansion RAM, and the sizes of main memory and file memory have been set with dipswitch SD2 on the MPU board

No.1: All other cases

SYSINIT is used to initialize the main memory when main memory check error occurs. All data in the main memory will be erased by the command. Make backup copies of program and point data in the main memory before the execution of SYSINIT, if necessary.

After the execution, set area sizes if necessary. Backup variables also have to be registered again.

**RELATED COMMANDS**      PRGSIZE, PNTSIZE, LIBSIZE, SYS

# TAN( )

F

Tangent

**FUNCTION** Returns tangent of specified angle

**FORMAT** TAN([radians])

**DESCRIPTION** Returns tangent of the specified angle when specified in radians.

Angles in degrees must be converted to radians, using the following equation:

$$\text{radian} = \text{degrees} * \pi / 180 \quad (\pi = 3.141593)$$

**RELATED COMMANDS** SIN( ), COS( ), ATAN( ), ATAN2( )

**EXAMPLE**

```
>PRINT TAN(0.55)           'Display tangent of 0.55 radians
.6131052
>PRINT TAN(30*3.141593/180) 'Display tangent of 30 degrees
.5773503
>A=30*3.141593/180        'Display tangent of 30 degrees using variable
>PRINT TAN(A)
.5773503
>
```

T

# TGO



Tool-coordinate Go

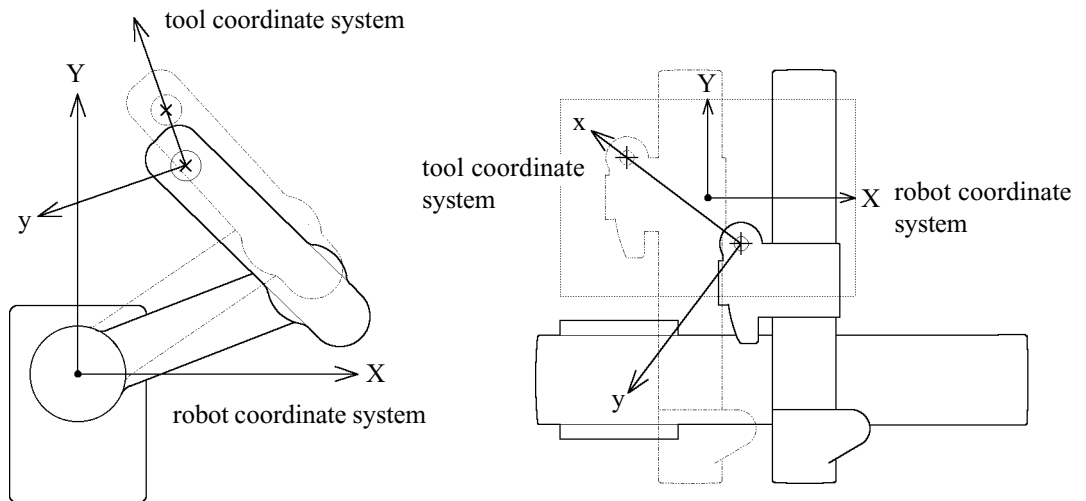
**FUNCTION** Executes PTP relative motion, in the selected tool coordinate system

**FORMAT** TGO [position specification] {/R} {TILL} {![parallel processing statement]!}  
|/L|

**DESCRIPTION** Executes PTP relative motion, in the selected tool coordinate system.  
In setting the arm's mode for the horizontal robot, specify either the right or left arm by declaring "/R" (for the right arm) or "/L" (for the left arm). Note that "/R" can be omitted while "/L" for the left arm cannot be omitted. Unless specified with "/L," all the set mode will apply only to the right arm.  
TILL modifier is used to complete the TGO by decelerating and stopping robot at an intermediate travel position if current TILL condition is satisfied.

**RELATED COMMANDS** P=, ! ... !, TOOL, SPEED, ACCEL, TILL, TMOVE

**EXAMPLE** >TGO 100,0,0,0 'Move currently selected tool 100 mm in the x direction (in tool coordinate system)  
>\_  
TGO P1 x  
>





# TILL

&gt;

S

**FUNCTION** Specifies and displays input condition that, if satisfied, complete the (JUMP, GO, or MOVE) command in progress by decelerating and stopping robot at an intermediate travel position

**FORMAT**

(1) TILL [input condition(s)]  
 ∴  
 |JUMP| [position specification] TILL  
 |GO |  
 |MOVE|

\* The following functions and operators may be used in input condition:  
 functions: SW, IN (either I/O or memory I/O may be used)  
 operators: AND, OR, XOR, +, \*  
 other: parenthesis for prioritizing operations, and variables

(2) |GO | [position specification] TILL SW([input bit number]){= |0|}  
 |MOVE| |1|

(3) TILL

**DESCRIPTION**

(1) TILL Command:  
 Specifies TILL condition.  
 TILL condition must include at least one input or memory I/O input function. Include in TILL condition only the operators noted above. (Using any other operator will result, not in error processing, but in unpredictable motion.)  
 When variables are included, their values are computed during TILL execution.  
 Multiple TILL statements are permitted, the most recent TILL condition remains current until superseded.

JUMP, GO, or MOVE With TILL Modifier:

Checks if current TILL condition is satisfied. If satisfied, this command completes by decelerating and stopping robot at an intermediate travel position. Does not check TILL condition during JUMP Z axis downward motion. Refer to JUMP for details.

At power on, TILL condition is:

TILL SW(0)=1 'Input Bit 0 is on

Use STAT(1) function to verify if TILL condition is satisfied.

T



(2) GO or MOVE with TILL Modifier, SW(input bit number) Modifier, and (0 or 1) Input Condition:

Checks if same line input condition is satisfied. If satisfied, this command completes by decelerating and stopping robot at an intermediate travel position.

GO or MOVE with TILL Modifier and SW(input bit number) Modifier, but no Input Condition:

Input condition defaults to 1. If specified input bit is on, this command completes by decelerating and stopping robot at an intermediate travel position.

(3) Displays current TILL Condition.

Because the TILL condition display format is altered to clearly indicate operation order, it may differ from the originally input format.

**RELATED COMMANDS**

JUMP, GO, MOVE, SW( ), STAT(1)

**EXAMPLE**

```

1000 TILL SW(1)=0           'Specifies TILL condition (Input Bit 1 is off)
1010 GO P1 TILL             'Stop if current TILL condition (line 1000) is
                             satisfied
1020 TILL SW(1)=1 AND SW($1)=1 'Specifies TILL condition (Input Bit 1 is on and
                             memory I/O bit 1 is on)
1030 MOVE P2 TILL           'Stop if current TILL condition (line 1020) is
                             satisfied
1040 MOVE P3 TILL           'Stop if current TILL condition (line 1020) is
                             satisfied
1050 FOR I=1 TO 2
1060   TILL SW(I)=1         '(Iteration 1) Specifies TILL condition (Input
                             Bit 1 is on)
                             (Iteration 2) Specifies TILL condition (Input
                             Bit 2 is on)
1070   MOVE PI TILL        '(Iteration 1) Stop if current TILL condition
                             (line 1060 Iteration 1) is satisfied
                             (Iteration 2) Stop if current TILL condition
                             (line 1060 Iteration 2) is satisfied
1080 NEXT
1090 I=5
1100 GO P4 TILL             'Stop if current TILL condition (line 1060
                             Iteration 2) is satisfied
1110 MOVE P5 TILL SW(10)=1 'Stop if Input Bit 10 is on
1020 MOVE P6 TILL           'Stop if current TILL condition (line 1060
                             Iteration 2) is satisfied
>TILL SW(1)=1 AND SW($1)=1
>TILL
    SW(1)=1 AND SW($1)=1
>TILL SW(0) OR SW(1) AND SW($1)
>TILL
    (SW(0) OR SW(1)) AND SW($1)
    
```

# TIME



**FUNCTION** Specifies and displays current time

**FORMAT** (1) TIME [hour]:[minute]:[second]|A|  
|P|

(2) TIME

**DESCRIPTION** (1) Specifies current time.

This date is used in internal file record keeping.

Time specification format may be either twelve hour (with A or P) or twenty four.  
If neither A or P is specified, time specification 11:59:59 or less will be input as A.

(2) Displays current time in twelve hour (with a or p) format.

**RELATED  
COMMANDS** DATE, TIMES\$(0)

**EXAMPLE** >TIME  
The current time is 10:15:32a  
  
>TIME 1:05:00P  
>TIME  
The current time is 1:05:15p  
>

**T**

# TIME( )

F

**FUNCTION** Returns controller accumulated operating time

**FORMAT** TIME([unit selection])

\* unit selection: 0, 1, or 2

**DESCRIPTION** Returns controller accumulated operating time as an integer.

0: hours

1: minutes

2: seconds

If seconds are selected by entering 2, the returned integer value will be very large. Prior to storing TIME( ) value at a variable, declare the variable type as 4 byte integer (LONG).

**RELATED COMMANDS** HOUR, TIME, TIMES\$(0)

```

EXAMPLE
10 FUNCTION MAIN
20 LONG A,B,C   'Declare A, B, and C as 4 byte integer
30 A=TIME(0)   'Store time in hours at variable A
40 B=TIME(1)   'Store time in hours at variable B
50 C=TIME(2)   'Store time in hours at variable C
60 PRINT A
70 PRINT B
80 PRINT C
90 FEND
>COM
COMPILE END
>XQT
    100          '100 hours
    6000         '6,000 minutes
    360000       '360,000 seconds
>

```

---

# TIME\$(0)

---

F

<b>FUNCTION</b>	Returns current time
<b>FORMAT</b>	TIME\$(0)  * The numeral 0 in ( )
<b>DESCRIPTION</b>	Returns current time as an integer.
<b>RELATED COMMANDS</b>	TIME, TIME( ), DATES\$(0)
<b>EXAMPLE</b>	>PRINT TIME\$(0) 5:25:04p

T

# TLSET



Tool Set

**FUNCTION** Defines and displays tool coordinate system

**FORMAT** (1) TLSET [tool coordinate system number],[position specification]

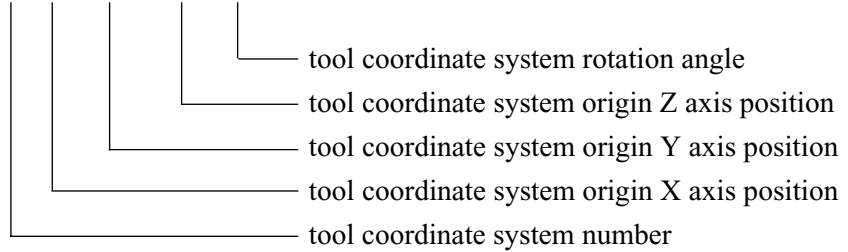
\* tool coordinate system number: integer from 1 to 3

(2) TLSET

**DESCRIPTION** (1) Defines tool coordinate systems Tool 1, Tool 2, or Tool 3 by specifying tool coordinate system origin and rotation angle in relation to Tool 0 coordinate system (hand coordinate system.)

< Example >

```
TLSET 1, 50, 100, -20, 30
```



```
TLSET 2, P10+X20
```

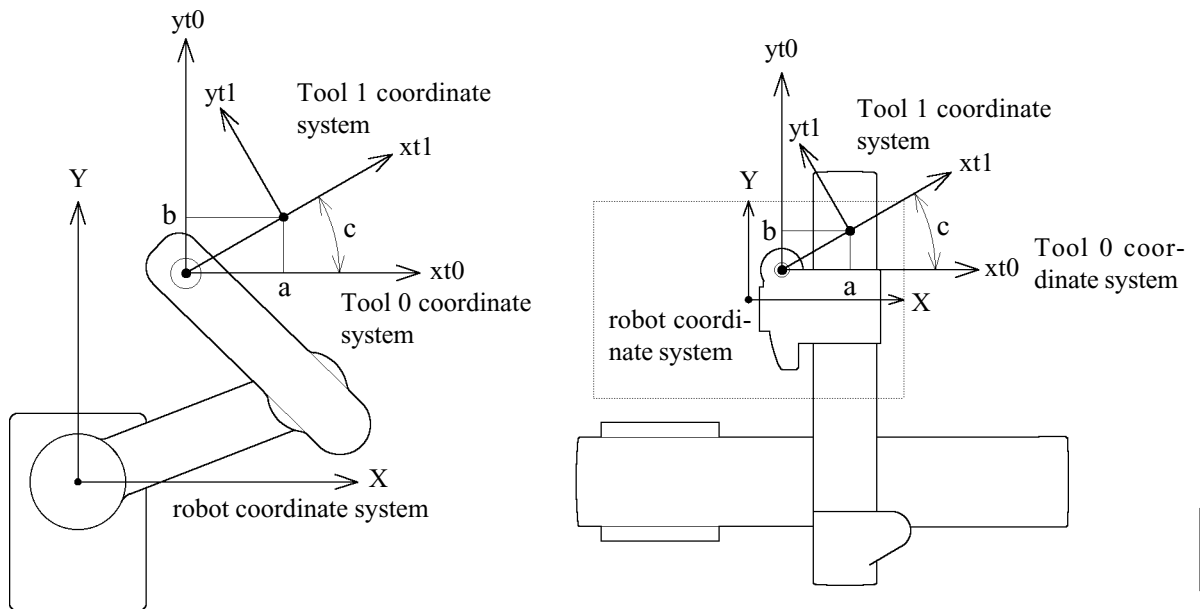
\* In this case, only the X-U axis coordinate values of P10 are referenced, arm attribute and local coordinate system numbers are ignored.

(2) Displays all current tool coordinate systems, including Tool 0.

To use a defined tool coordinate system, select it with the TOOL.

Power off does not change TLSET values.

Tool 0 origin is the robot center of rotation. Rotating about this point causes Tool 0 coordinate system to rotate, and with it coordinate systems Tool 1, Tool 2, and Tool 3.



Positioning the robot's axis #4 at 0 degrees, the Tool 0 coordinate system ( $xt_0 - yt_0$ ) becomes parallel to the robot coordinate system. The above figure illustrates this. At this time, with the tool installed on the axis #4 as shown, it is convenient to define coordinate system Tool 1 ( $xt_1 - yt_1$ ) using rotation angle "c" from coordinate system Tool 0.

#### NOTE

Executing VERINIT deletes tool coordinate systems Tool 1, Tool 2, and Tool 3. However, tool coordinate system Tool 0 cannot be changed.

#### RELATED COMMANDS

TOOL, TGO, TMOVE, VERINIT

#### EXAMPLE

```
>TLSET 1,100,0,0,0      'Define tool coordinate system Tool 1 (plus 100 mm in x
                        direction from Tool 0 coordinate system)
>TOOL 1                 'Select tool coordinate system Tool 1
>TGO P5
>
```

T

# TMOUT

&gt;

S

Time Out

**FUNCTION** Specifies time out interval for WAIT

**FORMAT** TMOUT [time out]

\* time out (seconds): integer from 0 to 32767

**DESCRIPTION** Specifies time interval, in seconds, that the WAIT allows to elapse for WAIT condition to be satisfied. If after time out interval has elapsed WAIT condition is not satisfied, time out error occurs.

If time out interval is specified as 0, time out function does not operate, and therefore WAIT will continue to wait indefinitely for WAIT condition to be satisfied.

**RELATED  
COMMANDS** SW( ), SW(\$ ), IN( ), IN(\$ )

**EXAMPLE** 10 TMOUT 60 'Time out interval specified 60 seconds  
:  
100 WAIT SW(0)=1



# TMOVE

&gt;

S

Tool-coordinate Move

**FUNCTION** Executes linear interpolator relative motion, in the selected tool coordinate system

**FORMAT** TMOVE [position specification] {TILL} {![parallel processing statement]!}

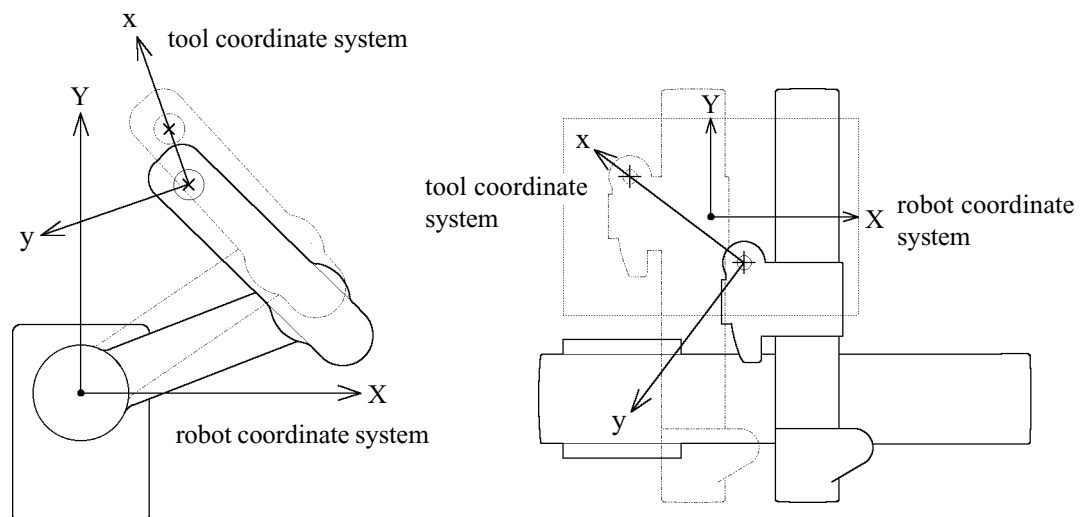
**DESCRIPTION** Executes linear interpolator relative motion, in the selected tool coordinate system.

TILL modifier is used to complete the TMOVE by decelerating and stopping robot at an intermediate travel position if current TILL condition is satisfied.

**RELATED COMMANDS** P=, ! ... !, TOOL, SPEEDS, ACCELS, TILL, TGO

**EXAMPLE** >TMOVE 100,0,0,0 'Move currently selected tool 100 mm in the x direction (in tool coordinate system)

```
>
TMOVE P1
>
```



T

# TMR( )

F

Timer

**FUNCTION**           Timer function

**FORMAT**             TMR([timer number])

\* timer number: integer from 0 to 15

returned value: 0 to 21,474,836.47 (&H7FFFFFFF/100)

**DESCRIPTION**       Returns elapsed time in seconds, minimum unit is 0.01 second.  
Sixteen timers are available, numbered from 0 to 15.

Timers are reset with TMRESET.

**RELATED  
COMMANDS**           TMRESET

**EXAMPLE**

```

100 TMRESET 0                   'Reset timer 0
110 FOR I=1 TO 10               'Perform operation ten times
120     GOSUB CYCL
130 NEXT
140 PRINT TMR(0)/10             'Calculate and display cycle time

```

# TMRESET

&gt;

S

Timer Reset

**FUNCTION** Resets timer

**FORMAT** TMRESET [timer number]

\* timer number: integer from 0 to 15

**DESCRIPTION** Resets and starts timer specified by timer number.  
Sixteen timers are available, numbered from 0 to 15.  
Use TMR( ) function to retrieve timer value (elapsed time).

**RELATED  
COMMANDS** TMR( )

**EXAMPLE**

```

100 TMRESET 0           'Reset timer 0
110 FOR I=1 TO 10
120   GOSUB CYCL
130 NEXT
140 PRINT TMR(0)/10     'Displays 1/10 of elapsed time

```

T

# TOFF

S

Trace Off

**FUNCTION** Stops displaying of program line numbers

**FORMAT** TOFF {![task number]}

**DESCRIPTION** Stops displaying of program line numbers.

If task number is omitted, TOFF stops displaying of line numbers for only those tasks that have executed TOFF.

**RELATED  
COMMANDS** TON

# TON

&gt;

S

Trace On

**FUNCTION** Displays program line numbers**FORMAT** TON {![task number],}[device specification]

\* task number: integer from 1 to 16 [default value: 1]

device number: integer from 0 to 3 [default value: 1]

**DESCRIPTION** During program execution, displays specified task program line numbers at specified device.

If task number is omitted, TON displays line numbers for only a task that has executed TON.

Devices are specified as follows:

0	No display
1	Display at controller LED line number display
2	Display at console
3	Display at both controller LED line number display and at console

Because specifying multiple tasks results in displaying program line numbers of multiple tasks simultaneously, usually only one task is specified.

TON default value is TON !1,1. With default value, Task 1 program line numbers are displayed at controller LED line number display.

Stopping program causes TON value to default.

**RELATED  
COMMANDS** TOFF

**EXAMPLE**

```
>XQT !2,MAIN      'Necessary for specified task to be started
>TON !2,2
[150]
[160]
⋮
```

T

# TOOL

&gt;

S

**FUNCTION** Selects and displays tool coordinate system

**FORMAT** (1) TOOL [tool coordinate system number]

\* tool coordinate system number: integer from 0 to 3  
[default value: 0]

(2) TOOL

**DESCRIPTION** (1) Selects tool coordinate system by number.

(2) Displays currently selected tool coordinate system number.

TOOL 0 selects the standard tool coordinate system (hand coordinate system).

Power off does not change tool coordinate system selection.

Executing VERINIT initializes tool coordinate system selection to standard tool (Tool 0).

**RELATED  
COMMANDS** TLSET, TGO, TMOVE

**EXAMPLE**

>TOOL 1	'Selects tool coordinate system Tool 1. Tool 1 needs to have been already defined by TLSET'
>JUMP P1	'Positions Tool 1 at P1'
>TOOL 0	'Selects tool coordinate system Tool 0'
>JUMP P1	'Positions standard hand at P1'

# TRAP

S

**FUNCTION** Defines the interrupt process

**FORMAT**

```
(1) TRAP [trap number] {[input motion] |GOTO [[line number]] |}
                                     |[label] |
                                     |GOSUB[[line number]] |
                                     |[label] |
                                     |CALL [function mane]]
```

\* trap number: integer from 1 to 4

the following functions and operators may be used in input condition:

functions: SW, IN (either I/O memory or I/O may be used)

operators: AND, OR, XOR, +, \*

others: parenthesis for prioritizing operations, and variables

```
(2) TRAP |EMERGENCY| {CALL [function name]}
          |ERROR      |
          |PAUSE      |
          |SGOPEN     |
          |SGCLOSE    |
```

**DESCRIPTION** (1) Executes interrupt process which is specified by GOTO, GOSUB, or CALL when input condition is satisfied. If interrupt process is executed, its TRAP setting is cleared. If the same interrupt process is necessary, define TRAP again.

If input condition is satisfied while other function by CALL is in execution, interrupt process by GOTO or GOSUB in TRAP setting is not executed.

If input condition and later is omitted:
Specified TRAP setting is canceled.
If GOTO is specified:
Command being executed will be processed as follows, then branches to the specified line number or label.
· Arm motion will pause immediately.
· Waiting status by WAIT or INPUT command will discontinue.
· For other commands, wait until the command process will end.
In case of input condition is satisfied simultaneously, only the TRAP which have the biggest trap number is executed.

T

<p>If <b>GOSUB</b> is specified:</p> <p>After executing the same process as <b>GOTO</b>, branches to the specified line number or label, then executes subroutine that follows. By <b>RETURN</b> statement at the end of subroutine, program execution returns to the line following <b>GOSUB</b>.</p> <p>Even if error is issued while subroutine of interrupt process is executed, error process subroutine by <b>ONERR</b> is not executed. On the contrary, even if input condition is satisfied while error process subroutine is executed, trap process subroutine is not executed. <b>GOSUB</b> and <b>CALL</b> is not allowed to specify in subroutine of trap process.</p> <p>In case of input condition is satisfied simultaneously, only the <b>TRAP</b> which have the biggest trap number is executed.</p>
<p>If <b>CALL</b> is specified:</p> <p>Executes specified function. In this case, task which executes <b>TRAP</b> command will keep on operating.</p> <p>In case of input condition is satisfied simultaneously, the <b>TRAP</b> is executed from the smallest trap number by turns.</p>

- (2) When emergency stop, error, **PAUSE** input, or safe guard open/close exists, trap process function by **CALL** is executed as privileged task. Be sure to end each trap process function with **END**. Do not use **XQT** in trap process function.

<p>If <b>CALL</b> and later is omitted:</p> <p>Specified <b>TRAP</b> setting is canceled.</p>
<p>If <b>EMERGENCY</b> is specified:</p> <p>When emergency stop is inputted, executes trap process function after finishing system process. If I/O status is specified to be reset when emergency stop is inputted, <b>ON</b>, <b>OFF</b>, and <b>OUT</b> command for I/O in trap process function cannot be executed. If I/O is specified to be preserved its status in emergency stop, those commands are available. Turn on bit 6 of software switch <b>SS1</b> before that, if necessary.</p>
<p>If <b>ERROR</b> is specified:</p> <p>When error (including error occurred inside system) is issued, executes trap process function after finishing system error process. This is not effective when error is issued by executing command directly.</p>
<p>If <b>PAUSE</b> is specified:</p> <p>When <b>PAUSE</b> is inputted while a program is being executed in <b>AUTO</b> mode, executes trap process function after pause status is processed. However, the trap process function is not processed at pause caused by the safe guard open/close.</p>



<p>If <b>SGOPEN</b> is specified:</p> <p>In <b>TEACH</b> mode  When the safeguard is opened while a program is being executed, executes trap process function after pause status is processed.  When the safeguard is opened after a pause caused by the closed safeguard while a program is being executed, executes trap process.</p> <p>In <b>AUTO</b> mode  When the safeguard is opened while a program is being executed, executes trap process function after pause status is processed.</p>
<p>If <b>SGCLOSE</b> is specified:</p> <p>In <b>TEACH</b> mode  When the safeguard is closed while a program is being executed, executes trap process function after pause status is processed.  When the safe guard is closed after a pause caused by the opened safeguard while a program is being executed, executes trap process.</p> <p>In <b>AUTO</b> mode  When the safeguard is closed after a pause caused by the opened safeguard while a program is being executed, executes trap process.</p>

T

## RELATED COMMANDS

ONERR, GOTO, GOSUB, CALL, ERT( ), ERR( ), ERA( ), ERL( ), ERRMSG\$( )

## EXAMPLE

< Example 1 > Error process defined by customers

SW(0) input is regarded as an error input defined by customer.

```

100 FUNCTION MAIN
110 TRAP 1 SW(0)=1 GOTO ERROR      'Defines TRAP
   :
500 ERROR:
510 ON 31                          'Signal tower lights
520 PRINT #20, "Error is issued."
530 END
540 FEND

```

< Example 2 > Usage like multi-tasking

```

100 FUNCTION MAIN
110 TRAP 2 SW($0)=1 OR SW($1)=1 CALL FEEDER
   :
540 FEND
   :
700 FUNCTION FEEDER
710 IF SW($0)=1 THEN OFF $0;ON #2,$0
720 IF SW($1)=1 THEN OFF $1;ON #3,$1
730 FEND

```

< Example 3 > Dummy operation of program

If input SW(31) is turned on, WAIT status is compulsorily discontinued, and branches to trap process subroutine.

```

110 FUNCTION MAIN
120 TRAP 1 SW(31)=1 GOSUB SKP_WAIT
  ⋮
200 WAIT SW(0)=1
210 WAIT SW(1)=1
  ⋮
700 SKP_WAIT:
710 IF (STAT(1) AND &H0008)=1 THEN END 'TRAP condition is satisfied
                                         during robot motion. It does not
                                         restart

720 WAIT SW(31)=0
730 TRAP 1 SW(31)=1 GOSUB SKP_WAIT      'Defines TRAP again
740 RETURN
750 '
760 FEND

```

< Example 4 > ERROR is specified

```

10 FUNCTION MAIN
20 TRAP ERROR CALL MSGOUT
  ⋮
999 FEND
1000 FUNCTION MSGOUT
1010 PRINT #20, ERRMSG$(ERR(0))
1020 END
1030 FEND

```

# TSPEED

&gt;

S

Teach Mode Speed

**FUNCTION** Specifies and displays maximum speed for PTP motion in TEACH mode

**FORMAT** (1) TSPEED [speed specification value]

\* speed specification value: integer from 1 to 100 (%)  
[default value: 100]

(2) TSPEED

**DESCRIPTION** (1) Specifies speed limit for PTP motion (GO, JUMP, PULSE, or related) commands while in TEACH mode.

While in low power mode, PTP motion speed will not exceed the currently valid TSPEED speed value and SPEED default value, regardless of programmed or direct SPEED specifications.

While in high power mode, PTP motion speed will not exceed the currently valid TSPEED speed value.

TSPEED speed value is maintained when power is off.

Performing VERINIT initializes the value to 100.

While in the AUTO mode, TSPEED speed value is ignored.

(2) Displays the current TSPEED value.

**RELATED COMMANDS** SPEED, POWER(LP)

## EXAMPLE

```
>TSPEED 20
>COM
COMPILE END
>XQT
>
>SPEED 100
>SPEED
```

'JUMP speed is limited to 20%  
by TSPEED specification

## PROGRAM

```
10 FUNCTION MAIN
20 SPEED 100
30 JUMP P1
40 JUMP P2
50 GOTO 20
60 FEND
```

```
20 'TSPEED value has priority over specified SPEED value
20 20
```

T

# TSPEEDS

&gt;

S

Teach Mode Speed S

**FUNCTION** Specified and displays maximum speed for CP motion in TEACH mode

**FORMAT** (1) TSPEEDS {[speed specification value]}

\* speed specification value: integer from 1 to 1120 (mm/s)  
[default value: 1120]

(2) TSPEED

**DESCRIPTION** (1) Specifies speed limit for CP motion (ARC, MOVE, CVMOVE, and related) commands while in TEACH mode.

While in low power mode, CP motion speed will not exceed the currently valid TSPEEDS speed value and SPEEDS default value, regardless of programmed or direct SPEEDS specifications.

TSPEEDS speed value is maintained when power is off.  
Performing VERINIT initializes the value to 100 mm/s.

While in the AUTO mode, TSPEEDS speed value is ignored.

(2) Display the current TSPEEDS value.

**RELATED COMMANDS** SPEEDS, POWER(LP)

**EXAMPLE**

```
>TSPEEDS 150
>SPEEDS 500
>SPEEDS
150      'TSPEEDS value has priority over specified SPEEDS value
```

# TSTAT



Task Status

**FUNCTION** Displays task status

**FORMAT** TSTAT

**DESCRIPTION** Displays current status of Task 1 to 16.

QUIT	Stopped (initial condition)
RUN	Executing
HALT	Temporarily stopped
Line	Most recently executed line number (0 for tasks not executed by program)

**T**

**RELATED COMMANDS** RUN, XQT, HALT, QUIT, RESUME

```

EXAMPLE >TSTAT
Task      1      2      3      4      5      6      7      8
Status QUIT  QUIT  QUIT  QUIT  QUIT  QUIT  QUIT  QUIT
Line 1250  180   60    0     0     0     0     0

Task      9     10     11     12     13     14     15     16
Status QUIT  QUIT  QUIT  QUIT  QUIT  QUIT  QUIT  QUIT
Line    0   570    0     0     0     0     0     0
>

```

# TW(0)

F

Timer Wait

**FUNCTION** Returns status of WAIT condition and WAIT time interval

**FORMAT** TW(0)

\* numeral 0 inside ( )

**DESCRIPTION** Returns status of the preceding WAIT condition and WAIT time interval with a 0 or 1.

If WAIT condition has been satisfied	0
If time interval has elapsed	1

**RELATED COMMANDS** WAIT, TMOUT

**EXAMPLE**

```
100 WAIT SW(0)=1,5           'Wait up to 5 seconds for Input Bit 0 to
                              be on
110 IF TW(0)=1 THEN GOTO TIME_UP 'Go to TIME_UP after 5 seconds has
                              elapsed
```

# TYPE



**FUNCTION** Displays contents of file

**FORMAT** TYPE {[pathname]}[filename]

\* filename must include extension

**DESCRIPTION** Displays contents of specified file.

Since only ASCII files can be displayed, be sure to specify only ASCII files.

The purpose of TYPE is to display the contents of files, not to edit files. To edit a file, use the appropriate method for that file.

\* ASCII file

source program (.PRG)
point data file (.PNT)
files created in edit mode
files created by ROPEN, WOPEN

\* binary files

object program (.OBJ)
symbol table (.SYM)
files created by CURVE

Do not specify binary files.

TYPE does not load files into main memory. The displayed source program and point data differs from the contents in the main memory.

## CAUTION

The enter (or return) key must not be pressed with the cursor placed at any of the displayed file contents.

Pressing the enter (or return) key would cause the displayed file contents to be input, possibly causing the robot to move, or causing the main memory program or point data to be altered.

## EXAMPLE

```
>TYPE TEST.PRG
10 FUNCTION TEST
20 PRINT "TEST Program"
30 FEND
```

# VAL( )

F

Variable

**FUNCTION** Returns numeric value of specified numeric string

**FORMAT** VAL(|[string variable name]|)  
|[numeric string]|

**DESCRIPTION** Returns numeric value of specified numeric string.

**RELATED  
COMMANDS** STR\$( )

**EXAMPLE** >PRINT VAL( "1234" )  
1234  
>



# VARIABLE



- FUNCTION** Displays variables
- FORMAT** VARIABLE {-A}
- DESCRIPTION** Displays variables, along with their types, used in compiled main memory object program.
- VARIABLE -A directs that function names also be displayed.
- VARIABLE does not display backup variables. To display backup variables, use LIBRARY.

**RELATED COMMANDS** LIBRARY, BYTE, INTEGER, LONG, REAL, DOUBLE

## EXAMPLE

```
>COMPILE
COMPILE END
>
>VARIABLE
INTEGER I
INTEGER J
REAL    DATA
CHR     DISPLAY$    'CHR represents string variable
>
>VARIABLE -A
FUNCTION MAIN
INTEGER I
INTEGER J
REAL    DATA
CHR     DISPLAY$
>
```

```
10 FUNCTION MAIN
20 INTEGER I,J
30 REAL DATA
40 STRING DISPLAY$
50 FEND
```

**V**

# VER



Version

**FUNCTION** Displays system control data

**FORMAT** (1) VER  
(2) VER {/A}

**DESCRIPTION** Displays currently set system control data.

When robot is delivered or after changing system control data, save the VER data. To save system control data, use MKVER.

(1) If VER only is inputted, following data will be displayed:

(Following data is for reference since the data will vary with robot model or set condition.)

>VER	
<VERSION>	< SPEL III version >
VER 6.2	
<ROM>	< ROM version >
MPUas-62a	ROM on MPU board
DSPas-61a	ROM for DSP on MPU board
OPUaa-42	ROM on operating unit
SCCaa-41	Serial bus (SYS.BUS)
ERSaa-41	ROM on additional RS-232C board
<OPTION>	< Installed options >
OPU	
I/O 16-31	
SYS.BUS	
RS-232C #22,#23	
<CONSOLE>	< AUTO mode console >
OPERATING UNIT	
<SD>	< Dip switch setting status >
8 F0	SD1, SD2 on MPU board
00	SD3 on MPU board
<SS>	< Software switch specification status >
0 0	
0 0	
0 0	
<REMOTE>	< REMOTE3 setting status >
0 0	
0 0	
<OPUNIT>	< Mode of operating unit mode >
0	
<CONFIG>	< RS-232C port configuration setting >
#20 2 1 3 0	
#21 2 1 3 0	

<SELRB>		< Positioning device number >
1		
<MAXDEV>		< Serial bus maximum address>
1		
<MANIPULATOR>		< Manipulator model >
SSR-H554BN		
<M.CODE>		< M.CODE >
10302		
<RANGE>		< Allowable motion range (pulse) >
-36409	364089	axis #1
-159289	159289	axis #2
-92160	0	axis #3
-172032	172032	axis #4
<HORDR>		< Axis motion order of home return >
02	0D	
00	00	
<HOFS>		< Encoder offset value >
1234	4567	
0	0	
<MCORDR>		< Axis motion order of machine calibration >
02	0D	(displayed in case of INC robot)
00	00	
<MCOFS>		< Machine calibration offset values >
04		(displayed in case of INC robot)
6510	5368	
5662	3830	
-620	-295	
<HTEST>		< HTEST command values >
-174	-11	(displayed in case of INC robot)
0	10	
<WEIGHT> (kg, mm)		< WEIGHT command values >
2	225	
<ARM>		< Setting value of standard and additional arm >
arm0=225 0 0 325 0		
<TOOL>		< Setting value of standard and additional tool
tool0=0 0 0 0		coordinate >
<HOMESET>		< HOME position (pulse) >
0	0	
0	0	
<CALPLS>		< CALPLS command values >
65523	43320	
-1550	21351	
<FREE>		< Free capacity of main memory >
PRG = 65312		Source program area
VER = 9 , 458		Backup variable area
( 10 , 512)		
OBJ = 99469		Object area

<pre> &lt;MEMORY&gt;   PRGsize 65536   PNTsize 125   LIBsize 10 , 512 &lt;PRGNO&gt;       1 &lt;HOUR&gt;       100         </pre>	<pre> &lt; Capacity of each area in main memory &gt;   Source program area   Available number of position data   Object area &lt; Program number selection method from   REMOTE2 &gt; &lt; Accumulated controller operating time &gt;         </pre>
---	--

(2) If VER/A is inputted, data will be displayed as follows:

<pre> &gt;VER/A &lt;DATE &amp; TIME&gt;       1996.12.01/ 14:48:50 &lt;VERSION&gt;   Ver 6.2   .   .   . &lt;HOUR&gt;       100 &lt;SELDISK&gt;       A: &lt;WIDTH&gt;       80 , 25 &lt;ARCH&gt; arch0=30 30 arch1=40 40 arch2=50 50 arch3=60 60 arch4=70 70 arch5=80 80 arch6=90 90 &lt;TSPEED&gt;       70 &lt;TSPEEDS&gt;       1120 &lt;XYLIM&gt;       0      0       0      0 &lt;LOCAL&gt;   local0 (p***:p***), (p***:p***)         </pre>	<pre> &lt; Time and date when VER/A is   executed &gt; } Same as of VER data } &lt; Current drive &gt; &lt; Display format of FILES &gt; &lt; Arch shape &gt; &lt; TSPEED value &gt; &lt; TSPEEDS value &gt; &lt; Allowable motion range on XY   plane &gt; &lt; Local coordinate &gt;         </pre>
---	---

**RELATED  
COMMANDS**

VERINIT, MKVER, SETVER

# VERINIT



Version Data Initialization

**FUNCTION**      Initializes system control data

**FORMAT**        VERINIT

**DESCRIPTION**    Initializes system control data.  
Executing VERINIT changes each command values to predetermined initial values.

VERINIT initializes values for the commands shown on the next page.

After entering VERINIT, the following prompt appears:

```
Version data initialize --> OK?
?
```

To initialize values      → enter  or

To maintain present values → enter  or

**NOTE**



This command is intended to be used for maintenance purposes. VERINIT initializes values as described on next page. Therefore, for cases in which initial command values are not desired, be sure to revise those values after executing VERINIT. You can save those values by MKVER on file memory.

**RELATED  
COMMANDS**      VER, MKVER, SETVER



command	initial value		
PRG. No.	00		
SELDISK	A:		
CONFIG	2, 1, 3, 0 (All RS-232C ports)		
CONSOLE	OP		
MAXDEV	1		
WIDTH	20, 23		
PRGNO	1		
SEL	0		
SET	0, 0.03, 0.03, 0.03, 0.03		
	1, 0.1, 0.1, 0.1, 0.1		
	2, 1, 1, 1, 1		
	3, 10, 10, 10, 10		
SPEED	Varies according to manipulator type		
ACCEL	Varies according to manipulator type		
WEIGHT	Varies according to manipulator type		
TSPEED	100		
TSPEEDS	1120		
SPEEDS	Varies according to manipulator type		
ACCELS	Varies according to manipulator type		
FINE	Varies according to manipulator type		
LIMZ	0		
ARCH	0, 30, 30	1, 40, 40	2, 50, 50
	3, 60, 60	4, 70, 70	5, 80, 80
	6, 90, 90		
HOMESSET	Delete		
HORDR	Varies according to manipulator type		
MCORDR	Varies according to manipulator type		
XYLIM	0, 0, 0, 0		
ARM	0		
ARMSET	Delete (except for ARM 0)		
TOOL	0		
TLSET	Delete (except for TOOL 0)		
LOCAL0	Equivalent to original robot coordinate system		
LOCAL1-15	Delete		
BASE 0	Equivalent to original robot coordinate system		
BASE 1-15	Delete		

# VLOAD

&gt;

S

Variable Load

<b>FUNCTION</b>	Loads variable data
<b>FORMAT</b>	VLOAD "[pathname][filename]"
<b>DESCRIPTION</b>	Loads a variable definition file created by VSAVE, and specifies the variable to the value contained in the file.

If pathname is omitted, the current directory is assumed.

Be sure that filename is identical to the name specified when saving the variable data by using the VSAVE. If the file name contains an extension, the filename, complete with the extension, must be specified.

VLOAD requires that the variable name referenced by the specified definition file be already contained in the symbol table file in the main memory.

<b>RELATED COMMANDS</b>	VSAVE
-----------------------------	-------

<b>EXAMPLE</b>	I(0)=1 I(1)=2 I(2)=3
----------------	----------------------------

The above array is contained in a variable definition file named IDATA.DAT. In this case, the variable I can be specified as follows:

```
>PRINT I(0),I(1),I(2)
0 0 0
>VLOAD "IDATA.DAT"
>PRINT I(0),I(1),I(2)
1 2 3
>
```

V

# VSAVE



Variable Save

**FUNCTION** Saves variable data

**FORMAT** VSAVE [variable name], "[pathname][filename]"

**DESCRIPTION** Saves data to be assigned to a specified variable name under a specified filename (thus creating a variable definition file).

If pathname is omitted, the current directory is assumed.

The specified filename may include an arbitrary extension (up to 3 characters). If no extension is specified VSAVE appends no extension to the filename. In order to distinguish variable definition files from other files, however, it is recommended to use an extension, such as ".DAT", which is common to many variable definition files.

One variable definition file can store the data for only one variable.

To delete a file without any extension by using the KILL, be sure to append a period (.) to the end of the filename that follows KILL.

< Example >

```
KILL "VARIABLE. "
```

**RELATED  
COMMANDS** VLOAD

**EXAMPLE**

```
>I(0)=1;I(1)=2;I(2)=3  
>VSAVE I, "IDATA.DAT"  
>
```

Following are the resulting contents of the file "IDATA.DAT".

```
I(0)=1  
I(1)=2  
I(2)=3
```



# WAIT

&gt;

S

<b>FUNCTION</b>	Specifies timer interval, stops program execution until specified condition is satisfied
<b>FORMAT</b>	<p>(1) WAIT [time interval]</p> <p>(2) WAIT [input condition]</p> <p>(3) WAIT [input condition],[time interval]</p> <p>* time interval units: seconds, minimum unit is 0.01          The following functions and operators may be used in input condition:          functions: SW( ), IN( ), (either I/O or memory I/O may be used), DSW( )          operators: AND, OR, XOR, +, *          others: parenthesis for prioritizing operations, and variables</p>
<b>DESCRIPTION</b>	<p>(1) WAIT with time interval:          Specifies time interval. After specified time interval elapses, program control transfers to next command.</p> <p>(2) WAIT and input condition without time interval:          Specifies WAIT condition. Program execution stops until WAIT condition is satisfied, then transfers to next command.          If after the TMOUТ time interval has elapsed WAIT condition has not yet been satisfied, an error occurs.</p> <p>(3) WAIT and input condition with time interval:          Specifies WAIT condition and time interval. After either WAIT condition is satisfied, or time interval has elapsed, program control transfers to next command. Use TW(0) to verify if WAIT condition has been satisfied or if the time interval has elapsed.</p>
<b>RELATED COMMANDS</b>	TMOUТ, TW(0), SW( ), IN( ), DSW( )

W

**EXAMPLE**

```

>WAIT 1;ON 1                                'Wait one second, then turn output bit 1 on
>WAIT SW(1)=1                                'wait until input bit 1 is on
>WAIT SW(1)=1,5                              'Wait 5 seconds for input bit 1 to be on. If
                                                after 5 seconds input bit 1 has not become
                                                on, go to next command

>WAIT SW(5)=1 AND SW(6)=1                    'Wait until both input bit 5 and 6 are on
100 CHK:
110   WAIT SW(1)=1,5
120   IF TW(0)=1 THEN GOSUB ERRPRC;GOTO CHK   'If input bit 1 does not
                                                become on, execute
                                                ERRPRC, then re-check
                                                status of input bit 1

200 ERRPRC:
210   PRINT "TURN ON SW(2) IF OK"
220   WAIT SW(2)=1
230   RETURN
      ⋮

< Select operation mode by using function keys on OPU-300 >

1000 OPUNIT 2                                'Mode selection
1010 OPU PRINT 1,3,"Select operation mode."   'Message display
1020 OPU PRINT 1,4,"F1:Normal operation"
1030 OPU PRINT 1,5,"F2:Dummy operation"
1040 OPU PRINT 1,6,"F4:Operation End"
1050 WAIT ( DSW(3) AND &B10110000 ) <> 0    'Wait for function key
                                                input
1060 IF DSW(3) AND &B10000000 THEN GOTO F4KEY
1070 IF DSW(3) AND &B00100000 THEN GOTO F2KEY
1080 IF DSW(3) AND &B00010000 THEN GOTO F1KEY

```

# WEIGHT

&gt;

S

**FUNCTION** Specifies and displays parameters for calculating PTP motion maximum acceleration/deceleration.

**FORMAT** (1) WEIGHT [hand weight]{,[arm length]}

(2) WEIGHT

**DESCRIPTION** (1) Specifies parameters for calculating PTP motion maximum acceleration/deceleration. Refer to the section "The hand and operating acceleration/deceleration speed" of the manipulator manual for details.

Hand weight includes workpiece weight.

Arm length specification is necessary only for SCARA type robot, it is the distance from the second arm rotation axis center line to the hand/workpiece combined center of gravity.

If the equivalent value workpiece weight calculated from specified parameters exceeds the maximum allowable payload, an error occurs.

(2) Displays current WEIGHT parameters.

Take note that specifying a WEIGHT hand weight significantly less than the actual workpiece weight can result in excessive accelerations and decelerations. These, in turn, may cause severe damage to the manipulator.

**NOTE**



WEIGHT value are not changed by power off, but are initialized by executing VERINIT. Since initial values vary with each manipulator type, refer to "Specifications" of manipulator manual for WEIGHT initial values.

**RELATED COMMANDS**

ACCEL, VERINIT

**EXAMPLE**

```
>WEIGHT
  2, 225
>WEIGHT 5
>WEIGHT
  5, 225
```

---

# WHILE...WEND

---

S

While...While End

**FUNCTION** Executes specified statements while specified condition is satisfied

**FORMAT** WHILE [condition]  
:  
WEND

\* nesting: up to 20 is allowed

(For definition of nesting, refer to #include command.)

**DESCRIPTION** Specifies WHILE condition. If satisfied, executes statements between WHILE and WEND, then once again checks WHILE condition. Executing WHILE...WEND statements and rechecking WHILE condition continues as long as WHILE condition is satisfied.

If WHILE condition is not satisfied, program control transfers to command following WEND. For WHILE condition that is not satisfied at the first check, WHILE...WEND statements are never executed.

Every WHILE must be followed by a WEND.

Up to 20 WHILE...WEND loops may be nested into one WHILE...WEND loop. Each WEND corresponds to the preceding WHILE, a WEND without a preceding WHILE will result in an error.

Any valid operator may be included in WHILE condition.

**EXAMPLE**

```
100 I=1
110 WHILE I<60      'Execute statements between WHILE and WEND if I<60
    :
300 I=I+2
310 WEND
```

# WIDTH



**FUNCTION** Specifies number of characters per line and lines per screen displayed on programming unit CRT

**FORMAT** WIDTH [number of characters per line][[,number of lines per screen]]

\* number of characters per line: 20, 40, or 80 [default value: 20]

number of lines per screen: from 4 to 23 [default value: 23]

**DESCRIPTION** The number of characters per line above becomes the format for displaying file names with FILES.

When executing FILES, file names are displayed according to the WIDTH number of characters per line.

Number of lines per screen specifies the number of lines per page for DIR/P command.

WIDTH values are not changed by turning power off, but are initialized by executing VERINIT.

**RELATED  
COMMANDS** FILES

**EXAMPLE**

```
>WIDTH 20
>FILES
AAAA PRG 1
AAAA OBJ 1
AAAA SYM 1
    space 59
>
>WIDTH 80
>FILES
AAAA PRG  AAAA OBJ  AAAA SYM
    space 59
>
```

**W**

# WOPEN...CLOSE

S

Write Open

**FUNCTION** Opens a file for writing to it

**FORMAT** WOPEN "[filename]" AS #[file number]  
:  
CLOSE #[file number]

\* Filename must include extension  
file number: integer from 30 to 35

**DESCRIPTION** Opens specified file and identifies it by the specified file number. This statement is used to write data to the specified file. (To append data, refer to AOPEN explanation.) CLOSE closes the file and releases the file number.

If the specified file does not exist on current directory, WOPEN creates the file and writes to it.

If the specified file exists, WOPEN erases all of its data, and writes to it.

The file number identifies the file as long as the file is open, it is used by the output statement for writing (by PRINT #) and for closing (by CLOSE #) the file. Accordingly, until the current file is closed, its file number can not be used to specify different file.

A maximum of six files can be open concurrently. As long as 6 files are open, however, DLOAD and DMERGE cannot be executed.

**RELATED  
COMMANDS** PRINT #, AOPEN, ROPEN

**EXAMPLE**

```
100 REAL DATA(200)
110 WOPEN "TEST.VAL" AS #30
120 FOR I=0 TO 100
130 PRINT #30,DATA(I)
140 NEXT
150 CLOSE #30
160 '
170 ROPEN "TEST.VAL" AS #30
180 FOR I=0 TO 100
190 INPUT #30,DATA(I)
200 NEXT
210 CLOSE #30
220 '
```

# XQT

&gt;

S

Execute

**FUNCTION** Executes program**FORMAT** (1) XQT {"{[pathname]}[filename]}"

\* Filename extensions are not allowed

(2) XQT ![task number] [function name] [[beginning line number] -  
| [[beginning line number] -[ending line number]]  
| | -[ending line number]]

(3) XQT ![task number] [function name]

**DESCRIPTION** Executes object program. The object program first needs to be created by compiling the source program execution.

(1) If pathname and filename are omitted, XQT executes program in the main memory.

XQT in this format cannot be used as a statement.

If filename is specified, XQT executes program in file memory. In doing so, the following files are loaded into main memory:

filename.OBJ  
filename.SYM  
filename.PNT

Accordingly, position data previously in main memory will be erased. Therefore prior to executing XQT, save position data as necessary.

If pathname is omitted, XQT searches for file in current directory.

X

- (2) Executes specified Function (previously loaded in main memory) as specified task.

XQT in this format cannot be used as a statement.

If line number(s) are specified, executes as follows:

XQT ![task number],[function name][beginning line number]-
all lines from beginning line number to end of program
XQT ![task number],[function name][beginning line number]-[ending line number]
all lines from beginning line number up to and including ending line number
XQT ![task number],[function name]-[ending line number]
all line up to and including ending line number

- (3) Executes specified Function as specified task.

XQT in this format can be used as a statement. To do so, include it in the main Function.

When XQT is executed in low power mode, the following message will be displayed. It shows the robot is in low power mode, and will move in low speed.

Low Power Mode ( --> LP Command )

**RELATED  
COMMANDS**

COMPILE, HALT, RESUME, QUIT, TSTAT, RUN, FUNCTION...FEND




# XYLIM

&gt;

S

XY Limit

<b>FUNCTION</b>	Specifies and displays allowable X and Y axis coordinate motion range
<b>FORMAT</b>	(1) XYLIM [X axis lower limit],[X axis upper limit],[Y axis lower limit],[Y axis upper limit]  (2) XYLIM
<b>DESCRIPTION</b>	<p>(1) Specifies robot coordinate system X and Y axis coordinate lower and upper limits that establish allowable motion range.</p> <p>The motion range established with XYLIM values applies to motion command target positions only, and not to motion paths from starting position to target position. Therefore, arm may move out from XYLIM range during motion.</p> <p>XYLIM range does not affect PULSE or CVMOVE.</p> <p>(2) Displays current XYLIM values.</p> <p>XYLIM values are not changed by power off.</p>
<b>NOTE</b> 	XYLIM values are initialized by executing VERINIT, or by specifying XYLIM 0,0,0,0. Doing so deletes motion range limits.
<b>RELATED COMMANDS</b>	RANGE
<b>EXAMPLE</b>	<pre>&gt;XYLIM                                'Display current XYLIM values       0      0       0      0 &gt;XYLIM -200,300,0,500                 'Specify robot motion range       -200 X 300       0   Y 500</pre>

X

# ZEROFLG(0)

F

Zero Flag

**FUNCTION** Returns value of memory I/O previous to it last being switched on or off

**FORMAT** ZEROFLG(0)

\* The numeral 0 in ( )

**DESCRIPTION** Returns value of memory I/O previous to it last being switched on or off.

ZEROFLG(0) is for exclusive control, used for sharing use of a single resource (such as an RS-232C port) while running multiple task programs.

**RELATED  
COMMANDS** ON \$, OFF \$

**EXAMPLE**

```

40 OFF $0
50 XQT !2,SUB
  ⋮
300 ON $0                                'Requests use of RS-232C port
310 IF ZEROFLG(0)=1 THEN WAIT SW($0)=0;GOTO 300
320 PRINT #20,1
330 OFF $0                                'Releases RS-232C port
  ⋮
1000 FUNCTION SUB
1010 ON $0                                'Requests use of RS-232C port
1020 IF ZEROFLG(0)=1 THEN WAIT SW($0)=0;GOTO 1010
1030 FOR I=0 TO 100
1040 PRINT #20,I
1050 NEXT
1060 OFF $0                                'Releases RS-232C port
  ⋮

```

Z

---

# ERROR CODE TABLE

---

There are three major categories of errors. In this table, they are indicated as follows:

code      Errors that do not require RESET to be executed to recover.

code
------

      Errors that RESET command should be executed to recover.

<table border="1"><tr><td>code</td></tr></table>	code
code	

      Errors that the power should be turned off once and on again to recover.

ERROR CODE TABLE

Number	Meaning	Remedy
0	No errors	
1	① FOR statement corresponding to NEXT statement is missing. ② WHILE statement corresponding to WEND statement is missing. ③ SELECT statement corresponding to SEND statement is missing. ④ IF statement corresponding to ELSE, ENDIF statement is missing.	① Make FOR statement. ② Make WHILE statement. ③ Make SELECT statement. ④ Make IF statement.
2	① Syntax error. ② Undefined variable is used in command. ③ Undefined array is used as an array. ④ Batch file is attempted to execute without specifying path.	① Correct syntax. ② Use the variables which were already compiled and registered. ③ Use the variables which were already compiled and registered. ④ Make path for the batch file.
3	GOSUB statement is absent, while RETURN statement is present.	Make GOSUB statement or delete RETURN statement.
4	① Program contains 851 or more GOTO and GOSUB statements in total. ② Program contains 410 or more labels.	① Reduce GOTO or GOSUB statement. ② Reduce label.
5	① Too large parameter is entered. ② Numeral beyond specified range is entered. ③ Argument is abnormal. ④ Undefined parameter is used. ⑤ Undefined PALET is used. ⑥ Null string is specified.	
6	① Numeral or variable is overflowed. ② Undefined address is specified.	① Reduce numeral or variable.
7	① GOSUB...RETURN program has too many nesting levels. ② Too large source program is saved.	① Reduce nesting levels. Maximum number of GOSUB...RETURN nesting levels is 10. ② Refer to PRG SIZE.
8	Line called by GOTO or GOSUB does not exist.	Correct program.
9	Subscription of array variable is beyond specified size.	
10	① Same names of variable, label or function exist. ② Reserved words are used for variable, label or function. ③ Name of variable, label or function starts with P.	① Do not specify the same name. ② Reserved words are not allowed to use. ③ First letter should be other than P.

Number	Meaning	Remedy
11	① Division by 0 (zero) is attempted. ② Undefined PALET function is used.	Correct program.
12	Command is used as a statement, or statement is used as a command.	
13	Number of parentheses on the left "(" and on the right ")" are not equal.	Correct program.
14	Number of parameters does not match.	Correct the number of parameters of executed command.
15		
16	Program line is not within FUNCTION...FEND.	Include program lines between FUNCTION...FEND.
17	FUNCTION declaration exists between FUNCTION ... FEND.	Describe FEND so that FUNCTION corresponds to FEND.
18	Variable declaration is not at a head of statement line.	Variable declaration is not allowed after multi-statements. When variable type is different, be sure to change statement line.
19	Number of characters on one line is 80 and over.	Number of characters on one line is up to 79.
20	Structured program has too many nesting loops.	Reduce nesting loops. Maximum total number of nesting loops is 40.
21	Overflow occurred in converting variable.	Change the value to allowable variable type.
22	Parameter specification table exceeds allowable range.	Subscription of PALET is 0 to 15.
23	Too many characters are on one line. (intermediate code)	
24	① Mixed operation of character and numerical value is attempted. ② Data tag error of object program.	
25	① Numerical variable error. Entry of too many digits is attempted. ② ASCII character in INPUT statement cannot be converted to numeral.	
26	① Specified address does not exist. ② Robot operation command is executed as Task 2 to 16, without SELRB declaration.	① Maximum address is 19. ② Set robot operation task with SELRB command.

ERROR CODE TABLE

Number	Meaning	Remedy
27	① Specified I/O bit number does not exist. ② File not opened is attempted to access.	① Confirm I/O board address setting. ② Open the file. Confirm the file name.
28	① NEXT statement corresponding to FOR statement is missing. ② WEND statement corresponding to WHILE statement is missing. ③ SEND statement corresponding to SELECT statement is missing. ④ ENDIF statement corresponding to IF statement is missing. ⑤ #endif statement corresponding to #ifdef is missing.	① Make NEXT statement. ② Make WEND statement. ③ Make SEND statement. ④ Make ENDIF statement. ⑤ Make #endif statement.
29	DMERGE, DLOAD commands are executed as Task 2 to 16.	Execute DMERGE, DLOAD commands as Task 1.
30	Number of received data and that of variable for INPUT is not equal.	
31	① Communication from a personal computer connected to TEACH port is not possible. ② Devices connected to RS-232C port cannot communicate.	① Check the cable connection, and change to TEACH mode. ② Check the cable connection, and configuration of RS-232C.
32		
33	Buffer memory overflow (receiving buffer is filled up and data continues to be transferred to RS-232C.)	
34	Parity, overrun, or framing error due to RS-232C communication.	
35		
36	Data over 80 characters is transferred to RS-232C port.	Number of characters per line is up to 79.
37	Overtime error of RS-232C communication	
38		
39	The number of FUNCTION...FEND is more than 70.	Reduce the number of FUNCTION...FEND up to 70.
40	Specified task does not exist.	
41	Cannot begin a task that has already begun.	
42	Command cannot be executed due to either insufficient memory, or memory malfunction.	

Number	Meaning	Remedy
43		
44		
45	① Prohibited command is executed during task execution. ② Edit mode is selected.	① Press BREAK (STOP) key or RESET switch. ② Exit Edit mode. Refer to EDIT.
46	Improper voltage of DC 24 V for customer use.	Check +24 V output at PSU unit.
47		
48	Instantaneous power loss or power failure.	
49	Warning - Low battery voltage.	Backup all of the data immediately before they are lost. Then, replace the lithium battery on MPU board with new one.
50	Optional designation of command is invalid.	
51		
52		
53	Specified file does not exist.	① Check the file name. ② Check whether the specified file exists or not in file memory.
54		
55		
56		
57	File with same name already exists.	Change file name, or delete existed file in file memory, and save it.
58	① Filename is incorrect. ② Filename cannot be changed by NAME command. (Same name already exists.)	
59		
60	No available space in file memory.	Backup the files on disk if necessary, delete the unnecessary files. Refer to the SPEL Editor or SPEL for Windows manual for the details of backup operation.
61		

ERROR CODE TABLE

Number	Meaning	Remedy
62	①Filename is more than 8 characters. ②File cannot be opened.	
63		
64		
65	Disk reading error.	File cannot be loaded. Delete (DEL) the erroneous file. If same error occurs in other files reformat the file memory by FORMAT.
66	Disk writing error.	File cannot be saved. Reformat the file memory with FORMAT.
67	Improper disk drive is selected.	Omit the drive name or specify drive "A:".
68		
69		
70	Invalid file format.	
71		
72	Check sum error of position data.	
73	Check sum error of source program.	
74	Check sum error of object program.	
75	①Undeclared or undefined Function or variable is used. ②Specified function is not supported.	
76	Invalid commands executed.	
77	Point number is improper.	
78	Use of undefined position data is attempted.	Define point. Refer to PNTSIZE.
79		
80		
81		
82	System error of internal process.	①Eliminate external noise source. ②Check storage condition of the MPU board and looseness of connectors on



Number	Meaning	Remedy
		the board. If this error frequently occurs, replace or repair MPU board.
83	① Program contains 400 or more variables. ② Registration of backup variables is too many.	① Number of variables is up to 399. ② Refer to LIBSIZE.
84	Compiled object program is too long.	Reduce size of program.
85	Restored file by RESTORE command is too long.	
86	Memory check error in system work area.	Initialize MPU board by SYSINIT.
87	Check sum error of file.	Delete (DEL) the erroneous file. If this error frequently occurs, replace MPU board.
88	The I/O bit is assigned as REMOTE3.	
89	I/O board communication error.	Input RESET command.
90	Use of invalid statements in parallel processing is attempted.	
91	No D parameter is returned from internal process.	
92	Safeguard circuit malfunction.(SRC-320 only)	
93	Number of parameters for command is improper.	D parameters should be 5 or less.
94	In WAIT SW command, specified condition is not satisfied within specified period of time.	
95	5 V for encoder is improper.	Check +12 V output at PSU unit.
96	I/F board communication error.	
97	Memory error of system control parameter.	① Eliminate external noise source and execute VERINIT. ② Check the voltage of the lithium battery on MPU board. Replace it if the voltage is less than 3.4 V. If this error frequently occurs, replace or repair MPU board.
98	Parameter check sum error of RAIOC.	
99	Memory check error of ROM.	Replace ROMs on MPU board.
100	Device (RAIOC etc.) communication error.	Check device address, connections, or MAXDEV setting.

ERROR CODE TABLE

Number	Meaning	Remedy
101	MPU error by malfunction of hardware.	Eliminate external noise source and execute VERINIT. If this error frequently occurs, replace MPU board.
102	Incorrect voltage of DC 7 V or DC 24 V.	Check +12 V output at PSU unit.
103	Malfunction of robot control.	Eliminate external noise source. If this error frequently occurs, replace MPU board.
104	Improper voltage of DC 12 V.	Check $\pm 12$ V output at PSU unit.
105	Communication error in internal process.	Eliminate external noise source. If this error frequently occurs, replace MPU board.
106	Communication buffer overflow of internal process.	Same as error 105.
107	Check sum error of robot model data.	① Check the setting of the DIP switch SD1 on the MPU board. (Refer to the specifications table of the manipulator manual.) ② If the setting is correct, replace the MPU board.
108	Memory check error of internal process.	① If you replaced ROM, initialize MPU board. ② Check the voltage of the lithium battery on MPU board. Replace it if the voltage is less than 3.4 V, and initialize the controller. ③ If this error frequently occurs, replace or repair MPU board.
109	Parameter is abnormal, cannot display.	Turn off and on the controller, execute VERINIT. If this error still exists, replace the MPU board.
110	Improper voltage of DC 24 V.	Check +24 V output at PSU unit.
111	Coprocessor error. (code error)	Turn off and on the controller, execute VERINIT. If this error still exists, replace the MPU board.
112	Coprocessor error. (overflow)	Same as error 111.
113	Coprocessor error. (underflow)	Same as error 111.
114	Coprocessor error. (division by zero)	Same as error 111.

Number	Meaning	Remedy
115	Improper voltage or temperature of motor power supply unit.	① Clean the filter of cooling fan and inspect the cooling fan, replace it if necessary. ② If ambient temperature of the controller is over 40°C, cool the place where the controller is. ③ Check the motor power unit and replace if necessary.
116	Malfunction of Servo CPU dual port RAM.	Eliminate external noise source. If this error frequently occurs, replace or repair MPU board.
117		
118	Improper voltage of DC 5 V	Check -12 V output at PSU unit.
119	Arm is moved too much after switching off the power of controller.	① If you moved the arm manually after switching off the power, turn off and on the controller or input RESET. ② Check arm position data. Move the arms to a taught point manually and display the numbers of pulse of this point. If the values of current are different too much from the numbers of pulse of taught point, calibrate the manipulator.
120	Invalid commands is executed.	
121	Executed invalid command under emergency stop condition.	Release emergency stop status. (One of TEACH, REMOTE1 or REMOTE2.)
122	Improper type of data is used.	
123	Specified command is not supported.	
124	Numeric value is out of allowable range.	
125	Arm reached the limit of motion range.	
126	Arm reached the limit of motion range specified by XYLIM command.	
127	Specified ARM is not defined.	① Use already defined arm number. ② Define Arm by ARMSET.
128	Specified TOOL is not defined.	① Use already defined tool number. ② Define Tool by TLSET.

ERROR CODE TABLE

Number	Meaning	Remedy
129	LIMZ error.	Set lower Z axis value of current or target position than LIMZ value.
130	Different LOCAL attribute is specified.	Define LOCAL by LOCAL command.
131	Specified LOCAL is not defined.	① Specify already defined LOCAL number. ② Define LOCAL by LOCAL command.
132	HOFS value is out of allowable range.	HOFS should be in the range of -40960 to 40960.
133	HOME command is attempted even though certain axes are not engaged.	Engage all axes by SLOCK command before HOME.
134	Change of arm attribute in CP control.	Arm attribute cannot be changed in CP control.
135	SFREE is attempted for axis that cannot be disengaged by SFREE.	Check the setting of software switch SS6.
136	Improper number (many or few) of point data is specified by CURVE.	Refer to CURVE.
137	Point data specified by CURVE contain point which has different arm attribute.	Refer to CURVE.
138	Free curve cannot be made by CURVE.	Verify that no two successive points overlap one another.
139	Restart of CVMOVE motion after quick pause is attempted.	Restarting CVMOVE motion after quick pause is impossible.
140	Only axis #4 movement is attempted in CP control.	
141	Improper point data for ARC command.	In ARC command, specified points are too close, or points are on straight line. Refer to ARC.
142		
143	HOME position is not defined.	Define home position by HOMESSET.
144	Improper number of parameters for command.	
145	Malfunction of CVMOVE command file.	
146	Motion command is executed under SFREE condition.	Engage all axes by SLOCK.

Number	Meaning	Remedy
147	The value of 4th and 6th parameter in BASE 0 command is different.	Refer to BASE.
148	Function not supported for this manipulator.	
149	The command can not be used in MOTOR ON condition.	
150	Motion command is executed under MOTOR OFF condition.	
151	Positioning cannot be completed by specified FINE.	<ul style="list-style-type: none"> <li>① Check disconnection of motor power circuit, or loose connection. Then, check the motor power unit and the voltage of AC servo driver.</li> <li>② Check each axis motion moving by hand. Secure the arm locking bolts tightly if looseness of them are found. Also, check each reduction gear. Verify appropriate lubrication or replace if necessary.</li> <li>③ Eliminate binding factor such as an obstacle if arm is bound.</li> <li>④ Other than the above, replace motor, AC servo driver or MPU board.</li> </ul>
152	Arm motion exceeds maximum speed or acceleration.	
153	CP motion cannot decelerate and stop in specified distance.	<ul style="list-style-type: none"> <li>① Prepare enough deceleration distance.</li> <li>② Don't finish operation with CP motion without deceleration.</li> </ul>
154	The axis #4 movement exceeds limit of its movement.	There is limit for axis #4 movement of the robot which has ballscrew spline unit.
155	Communication error with Servo CPU at internal process.	Replace or repair MPU board.
156	Malfunction of Servo CPU.	Replace or repair MPU board.
157	Robot moved in improper speed.	<ul style="list-style-type: none"> <li>① Turn off and on the controller.</li> <li>② Check arm position data. Move the arms to a taught point manually and display the numbers of pulse of this point. If the value of current are different too much from the numbers of pulse of taught point, calibrate the manipulator.</li> <li>③ If this error frequently occurs, replace or repair MPU board.</li> </ul>

ERROR CODE TABLE

Number	Meaning	Remedy
<p>158</p> <p>159</p>	<p>Reading error of encoder revolution data.</p> <p>① External noise exists.</p> <p>② Disconnection of encoder signal line.</p> <p>③ Voltage of encoder power source is abnormal.</p> <p>④ Other than the above.</p>	<p>① Eliminate external noise source.</p> <p>② Check disconnection or miscontact of signal line between controller rear panel and mother board, and servo driver.</p> <p>If ALARM LED(red) on servo driver is on, check signal line shown below also.</p> <ul style="list-style-type: none"> <li>· in manipulator</li> <li>· signal cable</li> </ul> <p>③ Check ENC+5V voltage at signal connector of motor. If voltage is not proper, inspect PSU board and wiring. If POWER LED(green) on servo driver is off, check the connection of signal connector of servo driver.</p> <p>④ Consider the following.</p> <ul style="list-style-type: none"> <li>· Connection of MPU board</li> <li>· Replace U15 and U16 (SCC) on MPU board.</li> <li>· Replace MPU board.</li> <li>· Replace servo driver.</li> <li>· Replace motor.</li> </ul>
160	Encoder signal disconnection of A phase.	Same as error 159.
161	Encoder signal disconnection of B phase.	Same as error 159.
162	Encoder signal disconnection of Z phase.	Same as error 159.
163	Encoder signal disconnection of S phase.	Same as error 159.
164	Character error of absolute encoder.	Same as error 159.
165	< NOTICE > Encoder is initialized.	<p>① This error is displayed when encoder is initialized and controller power is switched on. Repeat turning the power off and on twice.</p> <p>② In other case than the above, check whether RES wire of encoder shorted with other line or not.</p>
166	Servo calculation overflow.	Replace or repair MPU board.
167	Malfunction of Servo CPU hardware.	Replace or repair MPU board.
168	Servo CPU's communication error with Sub CPU.	Replace or repair MPU board.

Number	Meaning	Remedy
169	Servo overspeed.	① Check whether signal cable is surely connected. ② Check disconnection or miscontact of encoder line in manipulator. ③ Check cable damage or loose connection between controller rear panel and mother board, and servo driver. ④ Eliminate external noise source. ⑤ Other than the above, replace MPU board.
170	Servo overflow. ① Motor power line has problem. ② Encoder signal line has problem. ③ Motor power unit is not outputting correctly to servo driver. ④ Mechanical load is added on axis. ⑤ Tension of timing belt is not proper. ⑥ Arm is bound. ⑦ Axis #3 brake cannot be released. ⑧ Arm hit an obstacle. ⑨ Manipulator moved unexpectedly. ⑩ External noise source exists. Other than the above.	Try the following remedies ① to . ① Check motor power line in manipulator and controller. ② If ALARM LED(red) on servo driver is on, check disconnection or miscontact of signal line in the following : · in manipulator · signal cable · between controller rear panel and mother board, and AC servo driver. ③ Inspect motor power unit. ④ Check each axis motion moving by the hand. Verify appropriate lubrication or replace reduction gear. ⑤ Check tension of timing belt. Tighten or replace if necessary. ⑥ Eliminate binding factor such as an obstacle. ⑦ If brake cannot be released with a brake release button at MOTOR OFF, check the wire of brake. If brake does not release at MOTOR ON, inspect brake control signal circuit. ⑧ Prevent arm from hitting. ⑨ Take the same remedy as ②. If there is no problem in ②, replace AC servo driver and/or MPU board. ⑩ Eliminate external noise source. Replace MPU board or AC servo driver.
171	Communication check sum error of Servo CPU.	Eliminate external noise source. If this error frequently occurs, replace MPU board.
172	Instructive motor torque is abnormal.	Same as error 170.

ERROR CODE TABLE

Number	Meaning	Remedy
173	Robot is in low power state. Motor output power is limited.	Same as error 170.
174		
175	Hardware malfunction related Servo CPU.	① Replace MPU board. ② Check disconnection or miscontact of the power and signal line in manipulator and controller.
176		
177		
178		
179	Servo adjustment mode is selected.	
180	Driver overheat/overcurrent. ● When the red LED on the front of the AC servo driver is off. ① External noise source exists.  ● When the red LED on the front of the AC servo driver is on. ② Weight of the end effector(s) and work piece(s) exceeds rated payload. ③ Motion duty of manipulator is too much. ④ Mechanical load is added on axis.  ⑤ Tension of timing belt is not proper. ⑥ Cooling fan is not functioning. ⑦ Filter is clogged. ⑧ Ambient temperature around controller is over 40°C. ⑨ Motor power line is shorted or grounded.  ⑩ Other than the above.	① Eliminate external noise source. Refer to the controller manual. If no external noise source exists, replace MPU board.  ② Check actual weight, and set proper WEIGHT. ③ Reduce operation speed of robot. Refer to ACCEL, SPEED, WEIGHT. ④ Check each axis motion moving by the hand. Verify appropriate lubrication or replace reduction gear. ⑤ Check tension of timing belt. Tighten or replace if necessary. ⑥ Inspect the cooling fan. Replace it if necessary. ⑦ Clean the filter of cooling fan. ⑧ Cool the place where the controller is. ⑨ Check motor power line in manipulator and controller. ⑩ Replace AC servo driver.
181	Driver overload. ● When the red LED on the front of the AC servo driver is off. ① External noise source exists.	① Eliminate external noise source. If no external noise source exists, replace MPU board.



Number	Meaning	Remedy
	<ul style="list-style-type: none"> <li>● When the red LED on the front of the AC servo driver is on.</li> <li>② Weight of the end effector(s) and work piece(s) exceeds rated payload.</li> <li>③ Motion duty of manipulator is too much.</li> <li>④ Mechanical load is added on axis.</li> <li>⑤ Tension of timing belt is not proper.</li> <li>⑥ Arm is bound.</li> <li>⑦ Arm hit an obstacle.</li> <li>⑧ Axis #3 brake cannot be released.</li> <li>⑨ Motor power line is shorted or grounded.</li> <li>⑩ Other than the above.</li> </ul>	<ul style="list-style-type: none"> <li>② Check actual weight, and set proper WEIGHT.</li> <li>③ Reduce operation speed of robot. Refer to ACCEL, SPEED, WEIGHT.</li> <li>④ Check each axis motion moving by the hand. Verify appropriate lubrication or replace reduction gear.</li> <li>⑤ Check tension of timing belt. Tighten or replace if necessary.</li> <li>⑥ Eliminate binding factor such as an obstacle.</li> <li>⑦ Prevent arm from hitting.</li> <li>⑧ If brake cannot be released with a brake release button at MOTOR OFF, check the wire of brake. If brake does not release at MOTOR ON, inspect brake control signal circuit.</li> <li>⑨ Check motor power line in manipulator and controller.</li> <li>⑩ Replace the AC servo driver.</li> </ul>
182	Driver detected overspeed.	<ul style="list-style-type: none"> <li>① Replace the AC servo driver.</li> <li>② Replace the motor.</li> </ul>
183	Driver detected locked motor.	
	<ul style="list-style-type: none"> <li>● When the red LED on the front of the AC servo driver is off.</li> <li>① External noise source exists.</li> <li>● When the red LED on the front of the AC servo driver is on.</li> <li>② Motor power line is shorted or grounded.</li> <li>③ Mechanical load is added on axis.</li> <li>④ Tension of timing belt is not proper.</li> <li>⑤ Arm is bound.</li> <li>⑥ Axis #3 brake cannot be released.</li> </ul>	<ul style="list-style-type: none"> <li>① Eliminate external noise source. If no external noise source exists, replace MPU board.</li> <li>② Check motor power line in manipulator and controller.</li> <li>③ Check each axis motion moving by the hand. Verify appropriate lubrication or replace reduction gear.</li> <li>④ Check tension of timing belt. Tighten or replace if necessary.</li> <li>⑤ Eliminate binding factor such as an obstacle.</li> <li>⑥ If brake cannot be released with a brake release button at MOTOR OFF, check the wire of brake. If brake does not release at MOTOR ON, inspect brake control signal circuit.</li> </ul>

ERROR CODE TABLE

Number	Meaning	Remedy
<p>184</p>	<p>⑦ Arm hit an obstacle. ⑧ Other than the above.</p> <p>Driver detected improper motion.</p> <ul style="list-style-type: none"> <li>● When the red LED on the front of the AC servo driver is off.                             <ul style="list-style-type: none"> <li>① External noise source exists.</li> </ul> </li> <li>● When the red LED on the front of the AC servo driver is on.                             <ul style="list-style-type: none"> <li>② Encoder signal line has problem.</li> </ul> </li> </ul> <p>③ Other than the above.</p>	<p>⑦ Prevent arm from hitting. ⑧ Replace AC servo driver.</p> <p>① Eliminate external noise source. If no external noise source exists, replace MPU board.</p> <p>② Check disconnection or miscontact of signal line in the following : · in manipulator · signal cable · between controller rear panel and mother board, and AC servo driver.</p> <p>③ Replace motor or AC servo driver.</p>
<p>185</p>	<p>Encoder signal disconnection.</p>	<p>Same as error 184.</p>
<p>186</p>	<p>Malfunction of driver's CPU.</p> <ul style="list-style-type: none"> <li>● When the red LED on the front of the AC servo driver is off.                             <ul style="list-style-type: none"> <li>① External noise source exists.</li> </ul> </li> <li>● When the red LED on the front of the AC servo driver is on.                             <ul style="list-style-type: none"> <li>② Cable between AC servo driver and mother board has problem.</li> </ul> </li> </ul> <p>③ +24 V output at PSU unit is improper.</p> <p>④ Over than the above.</p>	<p>① Eliminate external noise source. If no external noise source exists, replace MPU board.</p> <p>② Check disconnection of the line or looseness of the connectors between AC servo driver and mother board.</p> <p>③ Check whether POWER LED (green) on the front of AC servo driver is on or not. If the LED is off, check +24 V output at PSU unit.</p> <p>④ Replace the AC servo driver.</p>
<p>187</p>		
<p>188</p>	<p>Driver detected improper torque/speed.</p>	<p>① Input RESET. ② Replace the AC servo driver.</p>
<p>189</p>	<p>Encoder S phase signal is not transmitted within predetermined period of time.</p>	<p>Check disconnection or miscontact of the encoder S phase signal line.</p>
<p>190</p>	<p>Encoder overheat.</p>	
<p>191</p>	<p>Encoder overspeed.</p>	<p>Confirm whether any axis moved at power on. If axis #3 moved, inspect the axis #3 brake.</p>

Number	Meaning	Remedy
192	Encoder data error.	Replace the motor.
193	Encoder battery error.	① Check the voltage of the battery on signal relay board. Replace it if the voltage is less than 2.8 V. ② Check disconnection of line between motor encoder and signal relay board.
194	Encoder check sum error.	Calibrate the axis. If this problem still exists, replace the motor.
195	Encoder backup error. ① This is the error shown after replacing the motor with absolute encoder. ② The battery voltage on signal relay board is improper. ③ The voltage of encoder power (ENC+5V) is improper.  ④ Other than the above.	① Refer to the manipulator's maintenance manual. ② Replace the battery if the voltage is low. ③ Check the voltage at signal connector closest to motor. If the voltage is low or improper, check PSU board or wiring. ④ Replace motor.
196		
197	Framing error of encoder data. ① This is the error shown after replacing the motor with absolute encoder. ② Strong external noise source exists. ③ Other than the above.	① Refer to the section "Calibration" of the manipulator's maintenance manual. ② Eliminate external noise source. ③ Take the following procedures : · Check inserted condition of MPU board. · Replace MPU board. · Replace motor.
198	Overrun error of encoder data.	Same as error 197.
199	Parity error of encoder data.	Same as error 197.
200		

ERROR CODE TABLE

Number	Meaning	Remedy
230	MCORG command has not executed.	
231	MCAL command has not executed.	
232	Home sensor detection error.	Check whether home sensor LED at the rear of manipulator lights or not while moving the arm by hand. If the LED does not light, execute MCORG command. If the LED lights, check disconnection of signal cable.
233	Encoder Z phase signal detection error.	① Check signal line of Z phase. ② If the line has no problem, replace the motor.
234	Quick pause interrupted MCORG execution.	
235	Incorrect ballscrew spline axis setting.	Set the bit 7 and 8 of SD2 on MPU board to on.
236	Current position calculated by the sensor and encoder Z phase is out of the default RANGE value.	① Connect the manipulator and the controller with the same M. CODE. ② If arm hit an obstacle, enter MCORG. ③ Other than the above, contact authorized dealer with its VER data.
237	HTEST data is improper.	① Confirm the M. CODE of manipulator and controller. If they are different, connect the ones with same M. CODE. ② If axis #4 has rotated more than 180° while the power is off, turn off the controller power once, and return axis #4 around the previous position. ③ If the arm has been hit, there is a possibility of mechanical position deviation. Execute MCORG command. Refer to MCORG.
238		
239		
240		

Number	Meaning	Remedy
300	Specified directory does not exist.	
301	① Cannot delete specified directory because does not exist. ② Cannot delete specified directory because includes files.	
302	Unable to create directory.	① Specified directory already exist. Change name. ② Disk is full.
303	Specified file or directory does not exist.	
304	Date designation method is incorrect.	
305	Time designation method is incorrect.	
306	Specified drive does not exist.	
307	Memory shortage for environment string.	Maximum memory for environment string is 512 bytes.
308	Batch file is too large.	Maximum memory for batch file is 4 KB.
309	File cannot be copied onto itself.	
310		

ERROR CODE TABLE



Number	Meaning	Remedy