## EDIT    program name, step number

**Function**

Enters the editor mode that enables program creation and editing.

**Parameter**

Program name

Selects a program for editing.  If a program name is not specified, then the last program edited or held (or stopped by an error) is opened for editing.  If the specified program does not exist, a new program is created.

Step number

Selects the step number to start editing.  If no step is specified, editing starts at the last step edited.  If an error occurred during the last program executed, the step where the error occurred is selected.

**[ NOTE ]**

A program cannot be edited during execution.

A program cannot be executed or deleted while it is being edited.  If a program calls a program that is being edited, an error occurs, and the execution of that program stops.

## C program name, step number

**Function**

Changes the program currently selected in editor mode.

**Parameter**

Program name

Selects the program to be edited.

Step number

Selects the step number to start editing. If no step is specified, the first step of the program is selected.

## S step number

**Function**

Selects and displays the specified step for editing. (Step)

**Parameter**

Step number

If no step is specified, the first step of the program is selected. If the step number is greater than the number of steps in the program, a new step following the last step in the program is selected.

## P    number of steps

**Function**

Displays the specified number of steps starting with the current step.

**Parameter**

Number of steps

Sets the number of steps to display.  If the number of steps is not specified, only the current step is displayed.

**Explanation**

Displays only the specified number of steps.  The last step on the list is ready for editing.

**Function**

Displays the previous (last) step for editing.

([Current step number] –1=[step number of the step to be displayed])

## I

**Function**

Inserts lines before the current step.

**Explanation**

The steps after the inserted line are renumbered. To exit insert mode, press the ↵ key. All lines written before exiting the insert mode are inserted in the program.

**Example**

CLOSEI instruction is inserted between steps 3 and 4.

```
1?OPENI
2?JAPPRO  #PART,  500
3?LMOVE  #PART
4?LDEPART  1000
5?  S    4                          ;Display step 4 to insert a line before it.
4       LDEPART  1000
4?  I                               ;Type the I command.
4I  CLOSEI    ↵                     ;Type in the instructions for the inserted line.
5I  ↵                               ;Press enter to finish inserting the lines.
5       LDEPART  1000               ;Step 4 is now renumbered as step 5.
5?
```

**[ NOTE ]**

To insert blank line, press Spacebar or TAB, then ↵ while in the insert mode.

## D　number of steps

**Function**

Deletes the specified number of steps including the current step.

**Parameter**

Number of steps

Specifies number of steps to delete beginning with the current step.　If no number is specified, only the current step is deleted.

**Explanations**

Deletes only the specified number of steps beginning with the current step.　Once deleted, all remaining steps are automatically renumbered and displayed.

**[ NOTE ]**

If the number of steps specified is greater than the number of steps in the program, all the steps after the current step are deleted.

## F   character string

**Function**

Searches (finds) the current program for the specified string from the current step to the last, and displays the first step that includes the string.

**Parameter**

Character string

Specifies the string of characters to be searched.

**Example**

Searches for character string “abc” in steps after the current step and displays the step containing that string.

```
1?F abc ↵
3           JMOVE abc
3?
```

## M /existing characters/new characters

**Function**

Modifies the characters in the current step.

**Parameter**

Existing characters

Specifies which characters are overwritten in the current step.

New characters

Specifies the characters that replace the existing characters.

**Example**

Modifies step 4 by replacing the pose variable abc with def.

```
4       JMOVE abc ↵
4?M/abc/def↵
4       JMOVE def
4?
```

## R character string

**Function**

Replaces existing characters in the current step with the specified characters.

**Parameter**

Character string

Specifies the new characters that replace the existing characters.

**Explanation**

The procedure for using the R command is as follows:

1. Using the Spacebar, move the cursor under the first character to replace.

2. Press the R key and then the Spacebar.

3. Enter the new replacement character(s). Note that the characters entered do not replace characters above the cursor but those two spaces to the left, starting above the R. (See example below.)

4. Press ↵.

Once ↵ is pressed, the AS system checks if the line is correct. If there is an error, the entry is ignored.

**Example**

The speed is changed from 20 to 35 using the R command.

```
1       SPEED 20 ALWAYS
1?              R 35 ↵
1       SPEED  35 ALWAYS
1?
```

## O

### Function

Places the cursor on the current step for editing. ("O" for "one line", not zero).

### Example

The pose variable abc is changed to def using the O command. The cursor is moved using $\boxed{\leftarrow}$ or $\boxed{\rightarrow}$ key.

```
3        JMOVE abc
3?O ↵
3        JMOVE abc BackSpace              ; delete "abc" using Backspace
3        JMOVE def ↵                      ; Enter "def"
3        JMOVE def
3?
```

**[ NOTE ]**

This command cannot be used via teach pendant.

## E

**Function**

Exits from the editor mode and returns to monitor mode.

## XD number of steps

**Function**

Cuts the specified number of steps from a program and stores them in the paste buffer.

**Parameter**

Number of steps

Specifies number of steps to cut and store in the paste buffer beginning with the current step.
Up to ten steps can be cut. If not specified, only the current step is cut.

**Explanation**

Cuts the specified number of steps and stores them in the paste buffer.

The XY command copies and does not cut the steps, but the XD command cuts the steps. The remaining steps in the program are renumbered accordingly.

## XY number of steps

**Function**

Copies the specified number of lines and stores in the paste buffer.

**Parameter**

Number of steps

Specifies number of steps to copy and store in the paste buffer.   Up to ten steps can be copied. If the number is not specified, only the current step is copied.

**Explanations**

Copies the specified number of steps including the current step and stores them in the paste buffer.

The XD command cuts the steps, but XY command copies the steps.   The program remains the same and step count does not change after the XY command is used.

## XP

**Function**

Inserts the contents of the paste buffer before the current step.

**Explanation**

Use the XD or XY command prior to this command to store the desired contents in the paste buffer.

**Function**

Inserts the contents of the paste buffer before the current step with the contents being inserted in reverse order.

**Explanation**

Inserts the contents of the paste buffer in reverse order as it would be inserted using XP command.

## XS

**Function**

Displays the contents of the paste buffer.

**Explanation**

Displays the current contents of the paste buffer.    If the paste buffer is empty, nothing will be displayed.

## T    pose variable

Option

**Function**

Enables teaching of motion instructions (JMOVE, LMOVE, etc.) using the teach pendant while in editor mode.

**Parameter**

Pose variable

Specifies pose variable name of destination to be taught, expressed in transformation values or joint displacement values.    It is read as an array variable if specified in the form of A[ ].    In this case, variables cannot be used in the element numbers.    If omitted, the current joint displacement values are taught as constants (pose constant).

**Explanation**

Enter this command while in editor mode.    When executed, teach pendant displays a specialized teaching screen.    Motions taught here are recorded as instructions in the program, and are written on the step where the T command is entered.    When more than one step is taught, the variable is renamed by incrementing the last number in the variable name.    See Operation Manual for more details.

**Example**

With pose variable

```
        2    JAPPRO #a
        3?     T pos                          Teach using TP.    Press [R] to return to AS.
                                              (3 steps are taught here)
        3      JMOVE pos0
        4      JMOVE pos1
        5      LMOVE pos2
                    .
```

Without pose variable

```
2     JAPPRO #a
3?     T                          Teach joint values using TP. Press R to end.
                                  (2 steps are taught here)
3     JMOVE #[0,10,20,0,0,0]
4     JMOVE #[10,10,20,0,0,0]
         .
```

**WARNING**

**Teach pendant must be connected to the controller to use this command. Also, the robot has to be in Teach mode, and TEACH LOCK ON.**

## USB_FDIR folder name

**Function**

Displays the name of files on USB flash drive.

**Parameters**

Folder name

Specifies the folder which contains the list of files to display. When omitted, the files in the root folder of the USB flash drive memory are displayed. Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

**Explanation**

By using the USB_FDIR command, all the files on USB flash drive are displayed.

**Example**

| | |
|---|---|
| >USB_FDIR ↵ | Displays the names all files on the USB flash drive. |

When the switch SCREEN is ON, the display does not scroll and stops at the end of the screen. To continue display, press Spacebar. To end the display, press ↵.

**DIRECTORY program name, .........**
**DIRECTORY/P program name, .........**

| | |
|---|---|
| DIRECTORY/L | pose variable, ......... |
| DIRECTORY/R | real variable, ......... |

**DIRECTORY/S string variable, .........**
**DIRECTORY/INT integer variable, .........**

**Function**

Displays the specified program and data in a list format.

Program name (/P), pose variable (/L), real variable (/R), string variable (/S), integer variable (/INT)
Specifies the type of data to display. If not specified, all the data in memory is displayed.

**Explanation**

The DIRECTORY command displays all the program names, their subroutines and variables. On the other hand, DIRECTORY/P command displays only the contents of the specified program itself.

**Example**

| | |
|---|---|
| >DIRECTORY | Displays the contents of all programs, including the variables and their values in a list. |
| >DIRECTORY/L | Displays all the pose variable names in a list. |
| >DIRECTORY/R | Displays all the real variables names in a list. |
| >DIRECTORY test3 | Displays all the names of the variables used in program test3. |
| >DIRECTORY test* | Displays all the names of programs that start with test. |

When the SCREEN switch is ON, the display does not scroll and stops when the screen is full. To continue the display, press Spacebar. To quit the display, press ↵.

**[ NOTE ]**

Program and variable names with * or ~ in front of the name indicate that no data are set to that program or variable.

**LIST program name, .........**

**LIST/P program name, .........**

**LIST/L pose variable, .........**
**LIST/R real variable, .........**

LIST/S string variable, .........
LIST/INT integer variable, .........

**Function**

Displays the specified program and data.

**Parameters**

Program name (/P), pose variable (/L), real variable (/R), string variable (/S), integer variable (/INT)

Specifies the type of data to display. If not specified, all the data in memory is displayed. If an array variable is selected, all the elements of that array variable are displayed on the screen.

**Explanation**

The LIST command displays all the program names, their subroutines and variables. On the other hand, LIST/P command displays only the contents of the specified program itself.

**Example**

| | |
|---|---|
| >LIST ↵ | Displays the name of all programs, including the variables and their values. |
| >LIST/L ↵ | Displays all the pose variables and their values. |
| >LIST/R ↵ | Displays all the real variables and their values. |
| >LIST test* ↵ | Displays the contents of all programs that start with "test", their subroutines and variables. |

When the SCREEN switch is ON, the display does not scroll and stops when the screen is full. To continue the display, press Spacebar. To quit the display, press ↵.

**DELETE/D program name, .........**

**DELETE/P/D program name, .........**

```
DELETE/L/D   pose variable [array elements] , .........
DELETE/R/D   real variable [array elements] , .........
```

**DELETE/S/D string variable [array elements] , .........**
**DELETE/INT/D string variable [array elements] , .........**

**Function**

Deletes the specified data from the memory.

**Parameters**

Program name (/P), pose variable (/L), real variable (/R), string variable (/S), integer variable (/INT), forced delete(/D)
Specifies the type of data to delete.

**Explanation**

DELETE command deletes the specified program completely; i.e. the main program itself and, if used in the program, the following data. (However, data used in other programs are not deleted).

- All subroutines called by the program or by subroutines within that program.
- All pose variables used in the program and in the subroutines in that program.
- All real variables used in the program and in the subroutines in that program.
- All string variables used in the program and in the subroutines in that program.

DELETE/P command, unlike the DELETE command, deletes only the program itself, and not the subroutines and variables used by that program.

If the array elements are not specified with the DELETE/L, DELETE/R, DELETE/S and DELETE/INT commands, all the elements in that array variable are deleted. If the element(s) are specified, only the specified element(s) are deleted.

When forced delete (/D) is specified, all subroutine and variable including those used in other programs are deleted. Those programs that are in execution or selected cannot be deleted. Also, robot programs and PC programs in execution cannot be deleted forcibly.

**Example**

| | |
|---|---|
| >DELETE test ↵ | Deletes the program “test”, and all the subroutines and variables used in them. |
| >DELETE/P pg11, pg12 ↵ | Deletes the programs “PG11” and “PG12”. (The subroutines and the variables are not deleted.) |
| >DELETE/R a ↵ | Deletes all the elements of the array variable “a”. |
| >DELETE/R a[10] ↵ | Deletes the 10th element of array variable “a”. |
| >DELETE/D test ↵ | Deletes the program “test” even if it is the subroutine called by another program. |
| >DELETE/R/D a[10] ↵ | Deletes only the 10th element of array variable “a” even if the array variable “a[]” is used in a program. |

```
.PROGRAM aa()
  CALL bb
.END
.PROGRAM bb()
  …
.END
```

When the program “aa” mentioned before is selected,

| | |
|---|---|
| >DELETE/D aa ↵ | Cannot delete “aa” because it is selected. |
| >DELETE/D bb ↵ | Deletes “bb”. |

When the program “aa” mentioned before is executed and the program “bb” is in HOLD in execution,

| | |
|---|---|
| >DELETE/D aa ↵ | Cannot delete “aa” because it is in execution. |
| >DELETE/D bb ↵ | Cannot delete “bb” because it is in execution. |

## USB_FDEL file name, .........

**Function**

Deletes the specified file from the USB flash drive memory.

**Parameters**

File name

Specifies the file names of a program and a variable. If no file name extension is specified, the extension “.as” is automatically added to the file name. To specify the folder including the file, add “folder name¥” before the file name. Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

**Explanation**

Deletes the programs in the specified file completely.

## RENAME　new program name＝existing program name

**Function**

Changes the name of a program currently held in memory.

**Parameters**

New program name

Sets new name for the program.

Existing program name

Specifies the current name of the program.

**Explanation**

If the new program name already exists, RENAME command results in an error.

**Example**

```
> RENAME  test=test.tmp ↵
```

Changes the name of the program from “test.tmp” to “test”.

## USB_RENAME　new file name＝existing file name

**Function**

Changes the name of a file currently held in USB flash drive memory.

**Parameters**

New file name

Sets new name for the file. To specify the folder including the file, add “folder name¥” before the file name. Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

Existing file name

Specifies the current name of the file. To specify the folder including the file, add “folder name¥” before the file name. Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

**Explanation**

If the new file name already exists, USB_RENAME command results in an error.

**Example**

| | |
|---|---|
| `>USB_ RENAME file_new =file` ↵ | Changes the name of the file in USB memory, from “file” to “file_new”. |

**XFER destination program name, step number1 =**
**source program name, step number2, number of steps**

**Function**

Copies and transfers steps from one program to another program.

**Parameters**

Destination program name

Sets the program for receiving the copied data. If a program of that name does not exist, the data is transferred to a new program with that name.

Step number 1

Sets the step number before which the copied data is inserted. If no step is specified, the data is inserted at the end of the specified program.

Source program name

Sets the name of the program from where the data is copied.

Step number 2

Sets the step number in the source program where the data is copied. If no number is specified, the data is copied starting from the top of the program.

Number of steps

Sets the number of steps to copy from the source program, starting from the step number set above (parameter: step number 2). If no step count is specified, all remaining steps in the program are copied.

**Explanation**

Copies from the specified program a specified number of steps, and inserts the data before the specified step in the destination program.

[ **NOTE** ]

If the destination program is being displayed using the STATUS or PCSTATUS commands, or if it is being edited (EDIT command), XFER command cannot be used.

**COPY new program name =**
**source program name + source program name +**

**Function**

Copies the complete program to a new program.

**Parameters**

New program name

Specifies the name of the program to where the copied program is placed. This must be specified.

Source program name

Specifies the name of the program to be copied. At least one program must be specified.

**Explanation**

When two or more source programs are specified, the programs are combined into one program under the new program name. The name specified for the new program cannot be an existing program.

## USB_COPY new file name = source file name

**Function**

Copies the specified files to a new file on USB flash drive memory.

**Parameters**

New file name

Specifies the new name for the file to be created. This must be specified. To specify the folder including the file, add “folder name¥” before the file name. Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

Source file name

Specifies the name of the file(s) to be copied. This must be specified. To specify the folder including the file, add “folder name¥” before the file name. Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

**Explanation**

The name specified for the new file cannot be an existing file name.

## TRACE stepper number: ON/OFF

**Function**

Starts (ON) or ends (OFF) logging of the robot or PC program to allow program tracing.

**Parameters**

Stepper number

Specifies the program to trace using the following number selection:

| | |
|---|---|
| 1: Robot program | |
| 1001: PC program 1 | 1004: PC program 4 |
| 1002: PC program 2 | 1005: PC program 5 |
| 1003: PC program 3 | |

If the program is not specified, the program currently in execution is logged.

ON/OFF

Starts/ ends logging.

**Explanation**

If the necessary memory is not reserved using the SETTRACE command before TRACE ON, the error (P2034) “Memory undefined” occurs.  Execute SETTRACE before retrying.

## SETTRACE number of steps

**Function**

Reserves the necessary memory to log the data for program tracing.

**Parameters**

Number of steps

Specifies the number of steps to log (setting range: 1 to 9999). If not specified, memory for 100 steps will be reserved.

**Explanation**

A portion of the user memory is set aside to accommodate the specified number of steps and the current number of existing robot and PC programs.

If TRACE ON and LSTRACE commands are executed without reserving the memory for logging, the error (P2034) "Memory undefined" occurs. If the SETTRACE command is used while logging, error (P2033) "Logging is in process" occurs, and all tracing are turned OFF (ends).

## RESTRACE

### Function

Releases the memory set aside by SETTRACE command.

### Explanation

If the RESTRACE command is used while logging, the error (P2033) “Logging is in process” occurs.

## LSTRACE stepper number: logging number

**Function**

Displays the logging data of the specified robot program or PC program.

**Parameters**

Stepper number

Specifies the program to be displayed by the following number selection:

|  |  |
|---|---|
| 1: Robot program |  |
| 1001: PC program 1 | 1004: PC program 4 |
| 1002: PC program 2 | 1005: PC program5 |
| 1003: PC program 3 |  |

If no program is specified, the log for robot program is displayed.

Logging number

Specifies the line number of the logging data from which to start the display. If not specified, line 1 is selected.

**Explanation**

If the necessary memory is not reserved using the SETTRACE command before executing LSTRACE, error (P2034) "Memory undefined" occurs.

If the LSTRACE command is used while logging, error (P2033) "Logging is in process" will occur.

When the LSTRACE command is executed, the logging data is displayed. The prompt appears after the data and the following commands can be entered:

N ↵ displays the next 9 lines.

L ↵ displays the previous 9 lines.

S number ↵ displays the specified log line number, and the 4 lines logged before and after that line (total of 9 lines). If no line number is specified, lines 1 to 9 are displayed. If the number is greater than the existing lines, the highest line number is displayed.

F character ↵ displays the line that includes the specified character(s), and the 4 lines logged before and after that line (total of 9 lines). If no character(s) are specified, the characters entered previously with the F command are used.  If the characters are not found in the data, nothing is displayed.

E ↵ ends the display and returns to AS monitor mode.

↵ entered alone displays the next 9 lines.

**Example**

```
91        pg1     31  JOINT SPEED9 ACCU1 TIMER0 TOOL1 WORK0   CLAMP1 (OFF,0,0,0)   2
92        pg1     32  SIGNAL  14;sig on
93        pg1     33  JOINT SPEED9 ACCU1 TIMER0 TOOL1 WORK0   CLAMP1 (OFF,0,0,0)   2
94        pg1     34  CALL "sub1"
95        sub1     1  PRINT "SUB1"
96        sub1     2  xyz:
97        sub1     3  JMOVE a
98        sub1     4  JMOVE b
99        sub1     5  JMOVE c
----N: Next page, L: Previous page, S number: Jump to number, F character: Find character,
E: End-------
```

**Function**

Executes the data backup to the CFast in the controller.

**Explanation**

The data backup to the CFast starts when the SHUTDOWN command is executed.

If the data backup is completed normally, the message (D0906) "Data backup to CF is completed. Turn OFF the controller power." appears. If the data backup is not completed after the specified time (10[sec]), the message (D0907) "Data backup to CF is failed. Turn OFF & ON the controller power." appears.

Executing following operations becomes impossible after the SHUTDOWN command is executed, because the memory of the controller is locked. Turn OFF and ON the controller power.

The SHUTDOWN command cannot be used while a robot motion program is running.

If the SHUTDOWN command is executed while the robot motion program is running, the error (P1012) "Robot is moving now." occurs.

**SAVE/SEL  file name=program name,  ………**

**USB_SAVE/SEL  file name=program name,  ………**

**Function**

SAVE command stores programs and variable data on the computer hardware. (Use only when a PC is connected to the robot controller.)

USB_SAVE command stores programs and variable data on USB flash drive.

**Parameters**

File name

Specifies the file names of a program and a variable. If no file name extension is specified, the extension ".as" is automatically added to the file name. To specify the folder including the file, add "folder name¥" before the file name. For the USB_SAVE command, writing "USB1¥"-"USB4¥" as a folder name can specify the file on the USB memory of "USB1¥" to "USB4¥" as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing "CFast¥" before the file name can specify the file on the CFast in the controller.

Program name

Select the program to save.  If not specified, all the programs in the memory are saved.

**Explanation**

The commands SAVE/P, SAVE/L, SAVE/R, SAVE/S, SAVE/SYS store each data type (program, pose variable, real variable, string variable, and system data, respectively) in separate files. Using the SAVE command alone stores all the five data types in one file.

SAVE command (without /SEL) stores the specified program(s), including any variables and subroutines used by the program(s). SAVE/SEL command stores only the program and not the subroutines and variables used by that program.

**Example**

| | |
|---|---|
| >SAVE f3=cycle,motor ↵ | Stores under the file name "f3.as" the system data, the two programs "cycle" and "motor", the subroutines called from those programs, and the variables used in those programs. |

```
SAVE/P/SEL  file  name=program  name,  .........
SAVE/L/SEL  file  name=program  name,  .........
SAVE/R/SEL  file  name=program  name,  .........
SAVE/S/SEL  file  name=program  name,  .........
SAVE/A/SEL  file  name=program  name,  .........
SAVE/SYS  file name
SAVE/ROB   file name
SAVE/ELOG file name
SAVE/OPLOG   file name
SAVE/ALLLOG   file name
SAVE/FULL   file name
SAVE/STG   file name
```

---

```
USB_SAVE/P/SEL   file name=program name,  .........
```

**USB_SAVE/L/SEL  file  name=program  name,  ………**
**USB_SAVE/R/SEL  file  name=program  name,  ………**
**USB_SAVE/S/SEL  file  name=program  name,  ………**
**USB_SAVE/A/SEL  file  name =program  name,  ………**
**USB_SAVE/SYS file name**
**USB_SAVE/ROB file name**
**USB_SAVE/ELOG file name**
**USB_SAVE/OPLOG  file name**
**USB_SAVE/ALLLOG  file name**
**USB_SAVE/FULL  file name**
**USB_SAVE/STG  file name**

**Function**

Stores in the file the program (/P), pose variable (/L), real variable (/R), string variable (/S), auxiliary information (/A), system data (/SYS), robot data (/ROB), error log (/ELOG), operation log (/OPLOG), all logs (/ALLLOG), all savable data (/FULL) and data logging function (option) logging data (STG).

As with SAVE command, USB_SAVE/ commands are used to save files to the USB flash drive memory.  Use SAVE/ command only when a PC is connected to the robot controller.  See 2.6.2 Uploading and Downloading Data.

**Parameters**

File name

Saves the data under this file name. If no file name extension is specified, the following extensions are automatically added to the file name according to the type of data in the file:

| | | | | | |
|---|---|---|---|---|---|
| program | .PG | pose information | .LC | real variables | .RV |
| string variables | .ST | auxiliary information | .AU | system data | .SY |
| robot data | .RB | error log | .EL | operation log | .OL |
| all logs | .AS | all data | .AS | data storage | .CSV |

For the USB_SAVE command, writing "USB1¥" to "USB4¥" as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing "CFast¥" before the file name can specify the file on the CFast in the controller.

Program name

Selects the name of the program to save. If not specified, all the programs and data in the memory will be saved on the file.

**Explanation**

1. SAVE/P

   Stores in the specified file the selected program(s) and the subroutines called by those program(s) (including the subroutines called by the subroutines).

   The names of the program(s) that were saved to the file are displayed on the system terminal. Some additional program names other than those specified by the SAVE command may appear. These are the names of the subroutines the specified program calls. These subroutines are stored in the same file as the program(s). Programs are stored in the file in alphabetical order regardless of the order in which they were saved.

2. SAVE/L, SAVE/R, SAVE/S

   Stores only the variables used in the specified program(s) and the subroutine(s) called by those program(s). (/L: stores only the pose variables, /R: stores only the real variables, /S: stores only the string variables)

3. SAVE/A

   Stores the auxiliary information.

4. SAVE/SYS

   Stores the system data.

5. SAVE/ROB
   Stores the data pertaining specifically to the robot (robot data).

6. SAVE/ELOG
   Stores the error log. This command cannot be entered together with other SAVE/ commands. For example, SAVE/ELOG/P does not function.

7. SAVE/OPLOG
   Stores the operation logs. This command cannot be entered together with other SAVE/ commands. For example, SAVE/OPLOG/P does not function.

8. SAVE/ALLLOG
   Stores all logs such as the error log, operation log, etc. This command cannot be entered together with other SAVE/ commands. For example, SAVE/ALLLOG/P does not function.

9. SAVE/FULL
   Stores all data that can be saved. This command cannot be entered together with other SAVE/ commands. For example, SAVE/FULL/P does not function.

10. SAVE/STG
   Stores the logging data logged in data storage function (option). This command cannot be entered together with other SAVE/ commands. For example, SAVE/STG/P does not function.

11. If /SEL is entered with /P,/L,/R,/S, only the main program and the variables used only in the main program are stored. The subroutines and the variables used in the subroutines are not stored.

   If the specified file name already exists in memory, then the existing file is automatically renamed with a "b" in front of the file extension. (See also SAVE command.)

**Example**

| | |
|---|---|
| `>SAVE/L   file2=pg1, pg2↵` | The pose variables used in programs pg1 and pg2 are stored under the file name "file2.lc". |

## LOAD/Q file name

## USB_LOAD/Q file name

**Function**

LOAD command loads the files in the computer memory into the robot memory. (Use only when PC is connected to the robot controller.) USB_LOAD command loads the files on the USB flash drive.

**Parameters**

File name

Specifies the file names of a program and a variable. If no file name extension is specified, the extension ".as" is automatically added to the file name. To specify the folder including the file, add "folder name¥" before the file name. For the USB_LOAD command, "USB1¥" to "USB4¥" as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot. Also, writing "CFast¥" before the file name can specify the file on the CFast in the controller.

**Explanation**

This command loads the data (system data, programs, and variables) from the specified file into the robot memory. Attempting to load a program name that already exists in memory results in error, and execution of the LOAD command is aborted.

**[ NOTE ]**

When loading a pose variable, real variable, or string variable name that already exists in memory, the data in the memory is overwritten without any warning. (Programs are not overwritten.)

The original data is deleted if LOAD is canceled while overwriting the data in the memory.

For LOAD command with /Q, the following message appears before loading each system data or program:

```
Load? (1:Yes, 0:No, 2:Load all, 3:Exit)
```

The choices are as follows:

1: Loads the data.

0: Does not load the data and goes on to the next data.

2: Loads the data and the remaining data in the file without inquiry.

3: Does not load the data, and ends the LOAD command.

If there is an unreadable or incorrect step in the program, the following message appears: “The step format is incorrect (0: Continue load 1: Delete program and exit)”. If the operation is continued by entering “0”, use the editor (Edit mode) to correct the step after the program has been loaded.

**Example**

```
>LOAD    pallet↵               Loads the data in the file “pallet.as” into the memory.
Loading…
System data
Program     a1()
Program     test()
 :
Transformation values
Joint interpolation values
Real values
Loading done.

>
>LOAD    f3.pg↵                Loads all programs in file “f3.pg” into the memory.
```

## USB_MKDIR　folder name

**Function**

Creates a file folder on the USB flash drive memory by the specified name.

**Parameters**

Folder name

Creates a folder by this name.

**Explanation**

Creates a new folder on the USB flash drive memory with the specified name. If a file or folder with the same name already exists on the memory, the message “Could not create folder” is displayed and the folder is not created.

Writing “USB1¥” to “USB4¥” as a folder name can specify the file on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the controller front USB slot, USB#3 does the controller rear USB slot and USB#4 does the internal USB slot.  Also, writing “CFast¥” before the file name can specify the file on the CFast in the controller.

## 5.4 Program Control Commands

| | |
|---|---|
| SPEED | Sets the monitor speed. |
| PRIME | Prepares the program for execution. |
| EXECUTE | Executes the program. |
| STEP | Executes one step of the program. |
| MSTEP | Executes one robot motion instruction. |
| ABORT | Stops execution after the current step is completed. |
| HOLD | Stops execution. |
| CONTINUE | Resumes execution of the program. |
| STPNEXT | Executes the program in step once mode. |
| KILL | Initializes the execution stack. |
| DO | Executes a single program instruction. |

## SPEED monitor speed

**Function**

Sets the monitor speed in percentage.

**Parameter**

Monitor speed

Sets the speed in percentage. If this value is 100, then the speed will be 100% of the maximum speed. If it is 50, the speed will be the half of the maximum speed. If the speed restriction removal option is set, the speed will be specified in percentage (%) up to 99999.

**Explanation**

A product of the monitor speed (set by this command) and program speed (set in the program using the SPEED instruction) determines the robot motion speed. For example, if the monitor speed is set at 50 and the speed set in the program is 60, and then the robot's maximum speed will be 30%.

[ **NOTE** ]

The maximum speed of the robot is automatically set at 100%, if the product of the monitor speed (set by monitor command or MON SPEED instruction) and the program speed (set by SPEED instruction in program) exceeds 100%.

The default setting of the monitor speed is 10%.

This command will not affect the speed of motion currently in execution. The new speed setting takes effect after the current motion and the planned motion are completed.

**Example**

If the program speed is set at 100%:

| | |
|---|---|
| `>SPEED 30 ↵` | The robot motion speed is set at 30% of the maximum speed. |
| `>SPEED 50 ↵` | The robot motion speed is set at 50% of the maximum speed. |
| `>SPEED 100 ↵` | The robot motion speed is set at 100% of the maximum speed. |

## PRIME program name, execution cycles, step number

### Function

Prepares the system so that a program can be executed using the $\boxed{\text{A}}$+$\boxed{\text{CYCLE START}}$ key. This command alone does not execute the program.

### Parameter

Program name

Selects the program to prepare for execution. If not specified, the program last executed or used in prime command will be selected.

Execution cycles

Sets how many times the program is executed. If not specified, 1 is assumed. To execute the program continuously, enter a negative number (-1).

Step number

Selects the step from which to start execution. If not specified, the execution starts from the first step of the program.

### Explanation

This command only prepares the system for program execution. It does not execute the program. A program can be executed using the CONTINUE command after the PRIME command prepares the system. The program can also be executed using $\boxed{\text{A}}$+ $\boxed{\text{CYCLE START}}$ keys.

**[ NOTE ]**

When using this command, the execution stack in the robot memory is initialized; i.e. any information indicating a program is held (e.g. by HOLD command or by an error) will be lost. For example, if program execution is held while in a subroutine (the information is memorized in the stack), and then the subroutine is executed using this command with CYCLE START or CONTINUE command (the stack is initialized), the processing cannot return to the main program as the stack has been initialized.

## EXECUTE program name, execution cycles, step number

### Function

Executes a robot program.

### Parameter

Program name

Selects the program to execute. If not specified, the program last executed (by EXECUTE, PRIME, STEP, or MSTEP command) is selected.

Execution cycles

Sets how many times the program is executed. If not specified, 1 is assumed. To execute the program continuously, enter a negative number (-1). The maximum limit is 32767.

Step number

Selects the step from which to start execution. If not specified, execution starts from the first executable step of the program. If the program is executed more than once, from the second cycle, the program is executed from the first step.

### Explanation

Executes a specified robot program from the specified step. The execution is repeated the specified number of cycles.

**⚠ CAUTION**

**When this command is used, the following conditions are set automatically:**

**SPEED 100 ALWAYS**

**ACCURACY 1 ALWAYS**

**The STOP instruction or the last step of the program marks the end of a cycle.**

### Example

| | |
|---|---|
| >EXECUTE test,-1↵ | Executes the program named "test" continuously. (Program execution continues until stopped by commands such as HALT, or when an error occurs.) |
| >EXECUTE↵ | Executes the program last executed. (One cycle only). |

## STEP program name, execution cycles, step number
## MSTEP program name, execution cycles, step number

**Function**

Executes one step of a robot program.

**Parameter**

Program name

Selects the program to execute. If not specified, the program currently suspended or the program last executed is selected.

Execution cycles

Sets how many times the program is executed. If not specified, 1 is assumed.

Step number

Selects the step from which to start execution. If not specified, execution starts from the first executable step of the program. If no parameters are specified, the step after the last executed step is selected.

**Explanation**

This command can be executed without parameters only in the following conditions:

1. after a PAUSE instruction,
2. after the program is stopped by causes other than error,
3. when the previous program instruction was executed using the STEP command.

MSTEP command executes one motion segment (i.e. one motion instruction and the steps before the next motion instruction). STEP command executes only one step of the program (the robot does not necessarily move).

**Example**

| | |
|---|---|
| >STEP assembly,,23 ↵ | Executes only step 23 of the program “assembly”. Entering STEP again without parameter immediately after this executes step 24. |

## ABORT

**Function**

Stops execution of the robot program.

**Explanation**

Stops execution of the robot program after the current step is completed. If the robot is in motion, the execution stops after that motion is completed. Program execution is resumed using the CONTINUE command.

**[ NOTE ]**

In AS system, the motion of the robot and the step in execution may not always be the same. Therefore, if the processing of steps is faster than the motion of the robot, the robot may perform one more motion after the current motion before it stops.

## HOLD

**Function**

Stops execution of the robot program immediately.

**Explanation**

The robot motion is stopped immediately. Unlike EMERGENCY STOP switch, the motor power does not turn OFF. This command has the same effect as when HOLD/RUN state is changed from RUN to HOLD. Program execution is resumed using the CONTINUE command.

## CONTINUE NEXT

**Function**

Resumes execution of a program stopped by PAUSE instruction, ABORT or HOLD command, or as a result of an error. This command can also be used to start programs made ready to execute by PRIME, STEP or MSTEP command.

**Parameter**

NEXT

If NEXT is not entered, execution resumes from the step at which execution stopped. If it is entered, execution resumes from the step following the step at which execution stopped.

**Explanation**

The effect that keyword NEXT has on restart of the program differs depending on how the program was stopped.

1. Program stopped during execution of a step or of a motion:
   CONTINUE restarts the program and re-executes the interrupted step.
   CONTINUE NEXT restarts at the step after the step where program stopped.

2. Program execution is stopped after a step or a motion is completed:
   CONTINUE and CONTINUE NEXT restarts program from the step immediately after the completed step, regardless of NEXT.

3. Program suspended by a WAIT, SWAIT or TWAIT instruction:
   CONTINUE NEXT skips the above instructions and resumes execution from the next step.

**[ NOTE ]**

The CONTINUE command cannot resume the program execution when:

- The program ended properly
- The program was stopped using the HALT instruction
- The KILL command was used

When the program is executed by CONTINUE command, the last step is displayed when the program is completed normally. On the other hand, the first step of the program is displayed when the program completes normally by $\boxed{\text{A}}$ + $\boxed{\text{CYCLE START}}$.

## STPNEXT

**Function**

Executes the next step when the system switch STP_ONCE is ON.

**Explanation**

When the system switch STP_ONCE is ON, the program can be executed in one step increment. This command advances the execution to the next step in the program.

## KILL

**Function**

Initializes the stack of the robot program.

**Explanation**

If the program is stopped by PAUSE instruction, ABORT command, or an error, the program stack is kept at the current status.  The KILL command is used to initialize the stack.  Once the KILL command is used, the CONTINUE command is ineffective, since there is no program on the stack.

## DO program instruction

**Function**

Executes a single program instruction. (Some program instructions cannot be used with this command.)

**Parameter**

Program instruction

Executes the specified program instruction. If omitted, the program instruction last executed using the DO command is executed again.

**Explanation**

Program instructions are typically written within the programs and executed as program steps. However the DO command enables execution of a single instruction without having to create a program to run that instruction.

The robot moves in speed equivalent to monitor speed 10% when operated by DO command. When the monitor speed is set below 10%, then the robot moves at that set monitor speed.

**Example**

| | |
|---|---|
| >DO JMOVE safe ↵ | The robot moves to the pose “safe” in joint interpolation motion. |
| >DO HOME ↵ | The robot moves to the home pose in joint interpolation motion. |

## HERE pose variable

**Function**

Assigns the current pose to the pose variable with the specified variable name. The pose may be expressed in transformation values, joint displacement values or compound transformation values.

**Parameter**

Pose variable

Variable value can be specified in transformation values, joint displacement values, or compound transformation values.

**Explanation**

Assigns the current pose values to the specified variable in joint displacement values, transformation values, or compound transformation values.

**[ NOTE ]**

Only the right most variable in the compound transformation values is defined. (See example below). If the other variables used in the compound values are not defined, this command results in an error.

The values of the variable are displayed on the terminal followed by the message “Change?” The values can be changed by entering the values separating each value with a comma. The value that is not changed may be skipped. Press ↵ after the message “Change?” to finish editing the values.

If the variable is defined in joint displacement values (variable name starting with #), the joint values of the current pose are displayed. If the variable is transformation values, the XYZOAT values are displayed. The XYZ values describe the position of the origin of the tool coordinates with respect to the base coordinates. The OAT values describe the orientation of the tool coordinates.

**Example**

| | |
|---|---|
| >HERE #pick↵ | Assigns the robot’s current pose to variable “#pick” (joint displacement values) |
| >HERE place↵ | Assigns the robot’s current pose to variable “place” (transformation values) |
| HERE plate+object ↵ | Pose “object ” is defined so that its relative pose to the pose “plate” becomes the robot’s current pose. Error occurs if “plate ” is undefined. |

## POINT pose variable 1＝pose variable 2, joint displacement value variable

**Function**

Assigns the pose information on the right of "=" to the pose variable on the left side of "=".

**Parameter**

Pose variable 1

Specifies the name of pose variable to be defined by joint displacement values, transformation values, or compound transformation values.

Pose variable 2

If not specified, the "=" sign is also omitted.

Joint displacement value variable

Specifies a variable defined by joint displacement values. This parameter must be set if the pose variable values on the left are in joint displacement values and the pose variable values on the right are in transformation values (if the parameter on the left is not in joint displacement values, this parameter cannot be set). The joint displacement values specified here expresses the configuration of the robot at the pose. If not specified, the current configuration is used to define the pose variable.

**Explanation**

Assigns pose values specified by the parameter on the right to the pose variable specified as pose variable 1. When pose variable 2 is not specified, any value already defined for pose variable 1 is displayed on the terminal, and can be edited. If pose variable 1 is undefined, the values displayed will be 0, 0, 0, 0, 0, 0.

Once POINT is executed, the pose values appear followed by the message "Change?" and a prompt. The values can then be edited. Exit by pressing only ↵ at the prompt.

If pose variable 1 is defined by joint displacement values, joint values appear on the display. If the variable is specified by transformation values, the XYZOAT values are displayed. The XYZ values describe the position of the origin of the tool coordinates with respect to the base coordinates. The OAT values describe the orientation of the tool coordinates. When the variable is expressed in compound transformation values, the right most variable in the compound transformation value is defined. If the other variables used in the compound value are not defined, this command results in error.

**[ NOTE ]**

When value types on the right and the left side of ”=” differ, this command works as follows:

1. POINT transformation values=joint displacement values
   The joint displacement values on the right are transformed into transformation values and assigned to pose variable 1 on the left.
2. POINT joint displacement values=transformation values, joint displacement values
   The transformation values on the right are transformed into joint displacement values and assigned to pose variable 1 on the left. If pose variable 3 is specified, the transformation value of pose variable 2 is transformed with the robot taking the configuration determined by the specified joint displacement values. If not specified, the transformation value is transformed with the robot in its current configuration.

When specifying values, maximum of nine decimal digits can be entered. The accuracy of entries with more than nine digits cannot be guaranteed.

**Example**

```
>POINT   #park
```

Displays the values of joint displacement value variable “#park”. (0,0,0,0,0,0 is displayed if it is undefined)

```
JT1       JT2       JT3       JT4       JT5       JT6
10.000    15.000    20.000    30.000    50.000    40.000
Change?(If not, hit RETURN only)
>,,,-15
JT1       JT2       JT3       JT4       JT5       JT6
10.000    15.000    20.000   -15.000    50.000    40.000
Change?(If not, hit RETURN only)
>↵
```

```
>POINT   pick1=pick↵
```

Assigns the transformation values of “pick” as transformation values of “pick1” and displays the values for correction.

```
>POINT   pos0=#pos0↵
```

Transforms the joint displacement values of variable #pos0 into transformation values and assigns them to variable “pos0”.

```
>POINT   #pos1=pos1,#pos2↵
```

Transforms the transformation values of variable “pos1” into joint displacement values using the robot configuration given by variable #pos2 and assigns the values to variable #pos1.

POINT/ X transformation value variable 1= transformation value variable 2
POINT/ Y transformation value variable 1= transformation value variable 2
POINT/ Z transformation value variable 1= transformation value variable 2
POINT/ OAT transformation value variable 1= transformation value variable 2
POINT/ O transformation value variable 1= transformation value variable 2
POINT/ A transformation value variable 1= transformation value variable 2
POINT/ T transformation value variable 1= transformation value variable 2
POINT/ 7 transformation value variable 1= transformation value variable 2

.
.
.

**POINT/ 18 transformation value variable 1= transformation value variable 2**
**POINT/ EXT transformation value variable 1= transformation value variable 2**

**Function**

Assigns the components of the transformation values of pose variable 2 to the corresponding components of the transformation values of pose variable 1. The values will be displayed on the terminal for editing.

**Parameter**

Transformation value variable 1
Specifies the variable to be defined by transformation values. (variable defined by transformation values or compound transformation values)

Transformation value variable 2
If not specified, the "=" sign can be also omitted.

**Explanation**

Assigns only the specified components (X, Y, Z, O, A, T, 7 - 18th axes, all external axes) of the transformation values. Once this command is executed, the values of each component are displayed followed by the message "Change?" and a prompt. These values can then be edited. Exit by pressing only ↵ key at the prompt.

**Example**

The following command assigns the OAT values of a1 to a2. The transformation values of a1 and a2 are as below:

a1 = (1000, 2000, 3000, 10, 15, 30) , a2 = undefined

```
>POINT/OAT a2 = a1
X[mm]    Y[mm]    Z[mm]    O[deg]   A[deg]   T[deg]
  0.       0.       0.       10.      15.      30.
Change? (If not, hit RETURN only)
> ↵
```

## STATUS

### Function

Displays the status of the system and the current robot program.

### Explanation

The system and the robot program status are displayed in the following format:

```
1....  Robot status:
       REPEAT mode:
2....  Environment:
       Monitor Speed(%) = 10.0
       Program Speed(%)ALWAYS = 100.0        100.0
       ALWAYS Accu.[mm]=   1.0
3....  Stepper status:  Program is not running.
4....  Execution cycles
       Completed  cycles:     3
       Remaining  cycles:    Infinite
5....  Program name        Prio       Step number
       test                  0            1          WAIT   sig(1001)
```

1. Robot status

The current robot status is one of the following:

| | |
|---|---|
| Error state: | An error has occurred; try the error reset operation. |
| Motor power off: | Motor power is OFF. |
| Teach mode: | Motor power is ON; the robot is controlled using the teach pendant. |
| Repeat mode: | Motor power is ON; the robot is controlled by the robot program. |
| Repeat mode cycle start ON: | Motor power is ON; the robot program is running. |
| Program waiting: | Motor power is ON; the robot program is running and in wait condition (executing a WAIT, SWAIT, or TWAIT instruction). |

2. Environment

Displays the conditions of current set speed and accuracy as follows:

| | |
|---|---|
| Monitor speed (%) | The current monitor speed is displayed. |
| ALWAYS Program Speed (%) | The rotation speed (see SPEED instruction) is displayed if the program speed specified by ALWAYS and the direction speed option are enabled. |
| ALWAYS Accu.[mm] | The positioning accuracy range specified by ALWAYS is displayed. |

3. Stepper status

The current status of step execution.

4. Execution cycles

| | |
|---|---|
| Completed cycles: | Execution cycles already completed (0 to 32767) |
| Remaining cycles: | Remaining execution cycles. If a negative number was specified for execution cycles in the EXECUTE command, “infinite” is displayed. |

5. Program name

The name of the program or step currently being executed or in wait condition.

## WHERE display mode

**Function**

Displays the current robot pose.

**Parameter**

Display mode

Selects the mode in which the data is displayed. There are 16 modes as shown below (modes 7 to 16 are options). If the mode is not specified, transformation values of the TCP in the base coordinates and the joint angles (JT1, JT2, …, JT3) are displayed. The display mode does not change until ↵ is pressed again.

| | | |
|---|---|---|
| WHERE | …… | Displays the current robot pose in transformation values in the base coordinates and the joint angles. |
| WHERE 1 | …… | Displays the current pose by joint angles (deg). |
| WHERE 2 | …… | Displays the current pose in XYZOAT in the base coordinates (mm, deg). |
| WHERE 3 | …… | Displays the current command values (deg). |
| WHERE 4 | …… | Displays deviations from the command values (bit). |
| WHERE 5 | …… | Displays encoder values of each joint (bit). |
| WHERE 6 | …… | Displays speed of each joint (deg/s). |
| WHERE 7 | …… | Displays the current pose including the external axis. (Option) |
| WHERE 8 | …… | Displays current pose in the fixed workpiece coordinates. (Option) |
| WHERE 9 | …… | Displays the instructed value of each joint for transformation values. |
| WHERE 10 | …… | Displays the motor current. |
| WHERE 11 | …… | Displays the motor speed. |
| WHERE 12 | …… | Displays the current transformation values expressed in base coordinates of |

another robot. (Option)

WHERE 13 …… Displays the current transformation values expressed in tool coordinates of another robot. (Option)

WHERE 14 …… Displays the instructed value of the motor current.

WHERE 15 …… Displays the original data of the encoder.

WHERE 16 …… Displays the speed of TCP.

WHERE 35 …… Displays the motor load arrival rate (%) for each joint.

WHERE 43 …… Displays the encoder temperature (°C) for each joint.

**Example**

```
>WHERE↵
JT1      JT2      JT3       JT4      JT5      JT6
9.999    0.000    0.000     0.000    0.000    0.000
X[mm]    Y[mm]    Z[mm]     O[deg]   A[deg]   T[deg]
15.627   88.633   930.000   -9.999   0.000    0.000
```

## IO/E signal number

**Function**

Displays the current status of all the external and internal I/O signals.

**Parameter**

Signal number

1………Displays 1–32, 1001–1032, 2001–2032

2………Displays 33–64, 1033–1064, 2033–2064

3………Displays 65–96, 1065–1096, 2065–2096

4………Displays 97–128, 1097–1128, 2097–2128

If not specified……Displays 1–32, 1001–1032, 2001–2032

**Explanation**

If the system switch DISPIO_01 is OFF, “o” will be displayed for signals that are ON, “x” is for signals that are OFF. Dedicated signals are displayed in uppercase letters (“O” and “X”). If the system switch DISPIO_01 is ON, “1” is displayed for signals that are ON and “0” for those that are OFF. “-” is displayed for external I/O signals that are not installed.

If “/E” is entered with the command, signal numbers 3001 and above are displayed along with the signals numbered 1–, 1001–, 2001–. (Option)

The display updates continuously until the display is terminated with the ↵ key.
(See 7.0 DISPIO_01 system switch)

**Example**

When DISPIO_01 is OFF

```
>IO↵
  32 -     1    xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxo   xxxo
 1032 - 1001    xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   oxxx
 2032 - 2001    xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx   xxxx
>
```

```
>IO/E↵
   32 -      1    xxxx   xxxx   xxxx   xxXX   xxxx   XXXX  XXXO  XXXO
 1032 - 1001    xxxx   xxxx   xxxx   xxXX   xxxx   XXXX  XXXO  XXXO
 2032 - 2001    xxxx   xxxx   xxxx   xxxx   xxxx   xxxx  xxxx  xxxx
 3032 - 3001    xxxx   xxxx   xxxx   xxxx   xxxx   xxxx  xxxx  xxxx
 >
```

When DISPIO_01 is ON

```
>IO↵
   32 -      1    0000   0000   0000   0000   0000   0000   0001   0001
 1032 - 1001    0000   0000   0000   0000   0000   0000   0000   1000
 2032 - 2001    0000   0000   0000   0000   0000   0000   0000   0000

>
```

## FREE

### Function

Displays the size of the memory currently not used in percentages and bytes.

### Explanation

Displays the size of the memory which can be used for programs, etc. in percentages and bytes.

### Example

```
>FREE↵
Total memory 8192 kbytes
Available memory size 8191 kbytes (99 %)
```

## TIME year – month - day hour: minute: second

### Function

Sets and displays the current time and date.

### Parameter

year – month –day hour: minute: second

Sets the time and date in the format described below. When setting “hour: minute: second”, the parameter “year – month –day” cannot be omitted. The values set are displayed followed by the message “Change?” When all the parameters are omitted, the current time and date are displayed.

### Explanation

This command sets the calendar within the robot. The range of values for each element are as below:

| | |
|---|---|
| Year | (00 - 99) Set the last two digits of the year. |
| Month | (01 - 12) |
| Day | (01 - 31) |
| Hour | (0 - 23) |
| Minute | (0 - 59) |
| Second | (0 - 59) |

The current time or the value input is displayed followed by the message “Change?” To change the data, enter new values. Press the ↵ key to terminate the command.

### Example

```
>TIME   02-04-29 09:45:46 ↵

>TIME
            Current time 02-04-29 09:47:33
 Change? (If not , hit RETURN only)
 02-05-17
            Current time 02-05-17 09:47:50
 Change? (If not , hit RETURN only)
      > ↵
```

## LANGUAGE language number

**Function**

Sets the display language.

**Parameter**

Language number

| | | | | | |
|---|---|---|---|---|---|
| Japanese | 1 | English | 2 | Italian | 3 |
| French | 4 | German | 5 | Chinese | 6 |
| Korean | 7 | Polish | 9 | Spanish | 10 |
| Dutch | 11 | | | | |

**Explanation**

Specifying parameters and pressing ↵ key displays the language corresponding to the number. If the parameter is omitted, a list of languages which can be displayed is displayed in English and a message “LANGUAGE?” appears. Specifying parameters and pressing ↵ key displays the language corresponding to the number.

**Example**

```
>LANGUAGE 1 ↵            The display language is changed to Japanese.

>LANGUAGE↵               The list of language which can be displayed is displayed in
                         English. Specify a language number.
 1:JAPANESE
 2:ENGLISH
 3:ITALIAN
 ..
 10:SPANISH
 11:DUTCH
 >LANGUAGE ? 1 ↵         The display language is changed to Japanese.
```

## ULIMIT joint displacment value variable
## LLIMIT joint displacment value variable

**Function**

Sets and displays the upper/lower limits of the robot motion range.

**Parameter**

Joint displacement value variable

Specifies a variable defined by joint displacement values. Sets the software limit (upper or lower) in joint displacement values. If this parameter is not specified, the current values are displayed.

**Explanation**

If the parameter is specified, the values of the specified pose variable are displayed followed by the message “Change?”. Enter the desired values after this message, as done in the POINT command. To end the command, press the ↵ key.

If the parameter is not specified, the values of the limit currently set are displayed, followed by the message “Change?”.

**Example**

```
>ULIMIT                                   Displays the current setting.
                  JT1      JT2     JT3      JT4       JT5       JT6
  Maximum   120.00    60.00    60.00   190.00    115.00    270.00    (The maximum allowable limit)
  Current    30.00    15.00    25.00   -40.00     60.00     15.00    (Current setting)
  Change? (If not , hit RETURN only)
>110,50
                  JT1      JT2     JT3      JT4       JT5       JT6
  Maximum   120.00    60.00    60.00   190.00    115.00    270.00
  Current   110.00    50.00    25.00   -40.00     60.00     15.00
  Change? (If not , hit RETURN only) ↵
```

```
>ULIMIT #upper ↵                          Sets the upper software limit to the pose defined as variable
                                          “#upper”.

>LLIMIT #low ↵                            Sets the lower software limit to the pose defined as
                                          variable“#low”.
```

## BASE transformation value variable

**Function**

Defines the base transformation values, which specifies the pose relation between the base coordinates and the null base coordinates.

**Parameter**

Transformation value variable

Specifies a pose variable defined by transformation values or compound transformation values. Defines the new base coordinates. The pose variable here describe the pose of the base coordinates with respect to the null base coordinates, expressed in null base coordinates. If not specified, the current base transformation values are displayed.

**Explanation**

If “NULL” is designated for the parameter, the base transformation values are set as “null base” (XYZOAT=0, 0, 0, 0, 0, 0,). When the system is initialized, the base transformation values are set automatically as the null base.

After a new base transformation value is set, the values (XYZOAT) and the message “Change?” are displayed. To change the values, enter new values separated by commas and press ↵. If no parameter is specified, the current values are displayed.

When the robot moves to a pose defined by transformation values or is manually operated in base mode, the system automatically calculates the robot pose taking in consideration the base transformation values defined here.

When a pose variable is used as the parameter and if that pose variable is redefined, note that the base transformation must also be redefined using the BASE command and the newly defined pose as the parameter. The change made in the pose variable will then be reflected to the base transformation.

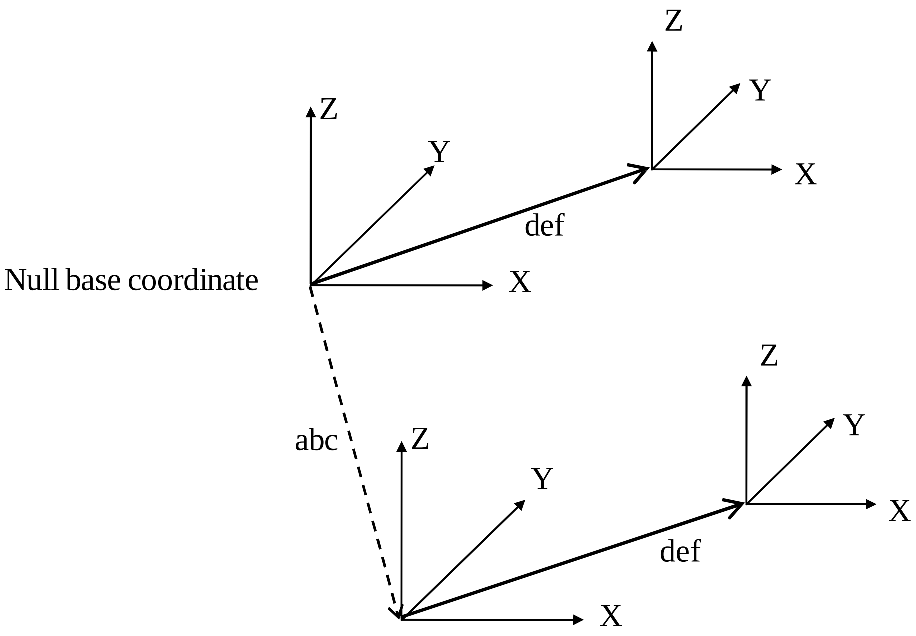
The BASE command has no effect on poses defined by joint displacement values.

**[ NOTE ]**

```
BASE  abc ↵
DO   JMOVE   def ↵
```

```
BASE  NULL ↵
DO   JMOVE   def ↵
```

Even if the pose information “def” are the same in both above examples, the destination of the robot will differ according to the base transformation. See the diagram below.



**Example**

```
>BASE ↵                         Displays the current base transformation values.
        X[mm]     Y[mm]    Z[mm]    O[deg]   A[deg]   T[deg]
        -300.        0.       0.       0.       0.       0.
Change? (If not , hit RETURN only) ↵
```

```
>BASE   NULL  ↵                 Changes the base transformation value to the null base.
                                (X, Y, Z, O, A, T)=(0, 0, 0, 0, 0, 0)
```

```
>BASE   abc  ↵                  Changes the pose of the base coordinates to the pose described
                                by the base transformation value variable “abc”.
```

## TOOL transformation value variable, tool shape number

**Function**

Defines the tool transformation values, which specify the pose relation between the tool coordinates and the null tool coordinates.

**Parameter**

Transformation value variable

Specifies a pose variable defined by transformation values or compound transformation values. Defines the new tool coordinates. The pose variable here describe the pose of the tool coordinates with respect to the null tool coordinates, expressed in null tool coordinates. If no pose variable is specified , the current tool transformation values are displayed.

Tool shape number

Specifies the tool shape to use for speed control in teach and check mode.

**Explanation**

If "NULL" is designated for the parameter, the tool transformation values are set at "null tool" (XYZOAT=0, 0, 0, 0, 0, 0,). The null tool coordinates have their origin at the center of the tool mounting flange and the axes are parallel to the axes of the robot's last joint. When the system is initialized, the tool transformation values are set automatically at the null tool.

After a new tool transformation is set, the values (XYZOAT) and the message "Change?" are displayed. To change the values, enter the new values separated by commas and press ↵. If no parameter is specified, the current values are displayed.

When the robot moves to a pose defined by transformation values or is manually operated in base mode or tool mode, the system automatically calculates the robot pose taking in consideration the tool transformation values defined here.

When a pose variable is used as the parameter and if that pose variable is redefined, note that the tool transformation must also be redefined using the TOOL command and the newly defined pose as the parameter. The change made in the pose variable will then be reflected to the tool transformation. See 11.4 Tool Transformation.

**Example**

| | |
|---|---|
| >TOOL grip↵ | Changes the pose of the tool coordinates to the pose described by the pose variable “grip”. |
| >TOOL NULL ↵ | Changes the tool transformation values to null tool. (X, Y, Z, O, A, T)=(0, 0, 0, 0, 0, 0) |
| >TOOL tool1,1 ↵ | Selects 1 for the tool shape number for the tool with tool transformation values “tool1”. |

## SET_TOOLSHAPE tool shape no. = transformation value variable 1, transformation value variable 2, …, transformation value variable 8

**Function**

Registers the tool shape used to control speed in teach mode and check mode.

**Parameter**

Tool shape no.

Specifies the number of tool shape to register. Setting range: 1 to 9.

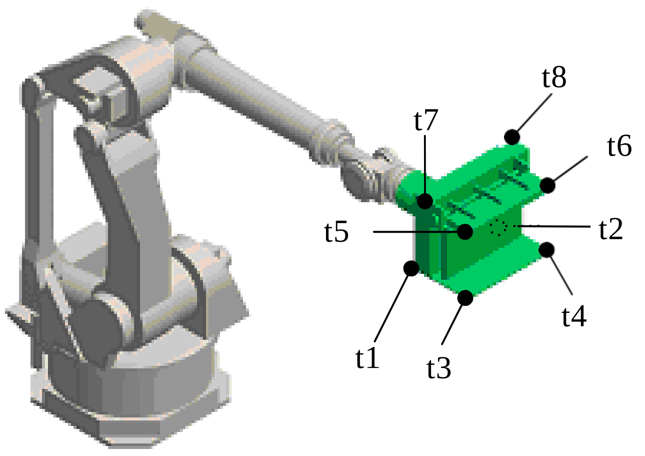
Transformation value variables 1-8

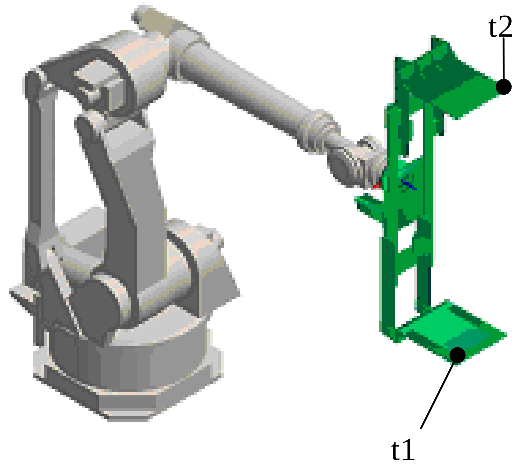
Specifies the points on the tool shape using transformation value variables. Maximum of 8 points can be specified. The points are specified in transformation values as seen from the center of the flange surface. However, only the X,Y, Z values of the transformation values are used for the tool shape registration.

**Explanation**

Defines the tool shape used for speed control in teach and check modes by maximum 8 points specified in pose variables defined by transformation values (t1 to t8 in the figure below). Speed should be controlled using tool shape in such cases where the tip of the tool is further away from the flange surface than the TCP, or when the shape of the workpiece attached to the tool should be put into consideration.

For tools registered via Aux. function 304, the tool shape can be registered via the screen that is displayed when pressing <Tool Shape> on the same Aux. function 304 screen.





**Example**

> SET_TOOLSHAPE 1=t1,t2, ↵ Specifies tool edge positions of tool shape no.1 by transformation value variables t1 and t2.

> TOOL tool1,1 ↵ Restricts the speed of edge points of tool shape no.1.

## ENA_TOOLSHAPE   tool shape no. = TRUE/ FALSE

**Function**

Enables/disables speed control in teach and check mode.

**Parameter**

Tool shape number

Specifies in integers from 1 to 9, the number of the tool shape to set enable/ disable.

TRUE/FALSE

Specify TRUE to enable speed control by the specified tool shape.  Specify FALSE to disable the speed control.

**Explanation**

Selects if speed control in teach and check modes are done by the specified tool shape or not. FALSE is selected for all tool shapes as default setting.  If TRUE is selected for a tool shape number with not even one point specified, error E1356 Tool shape not set occurs when the robot is operated in teach or check mode.  To avoid this, always set at least one tool point via SET_TOOLSHAPE command or change from TRUE to FALSE via this command an d then execute TOOL or TOOLSHAPE command specifying the relevant tool shape number.  (Once set to TRUE, the setting will not be changed to FALSE unless TOOL/TOOLSHAPE command is executed.)

## TOOLSHAPE tool shape no.

### Function

Selects the tool shape used to control speed in teach mode and check mode.

### Parameter

Tool shape no.

Specifies the number of the tool shape used for speed control. Setting range: 1 to 9.

### Explanation

To enable speed control in teach mode and check mode, the function must be enabled by ENA_TOOLSHAPE command/instruction (ENA_TOOLSHAPE n =TRUE). Error E1356 Tool shape not set occurs if a tool shape with no point registered (all points set to 0) is selected.

### Example

```
> TOOL tool1 ↵
> TOOLSHAPE 1 ↵
```

Specifies the tool transformation values for the relevant tool as “tool1” and controls the speed using the tool points registered for tool shape 1.

## SETHOME accuracy, HERE
## SET2HOME accuracy, HERE

**Function**

Sets and displays the HOME pose.

**Parameter**

Accuracy

Sets the accuracy range of the HOME pose in millimeters. The robot is at the HOME pose when it nears HOME by the distance specified here. If not specified, the default value 1 mm is assumed.

HERE

Sets the current pose as HOME.

**Explanation**

If no parameters are entered, the current values are displayed followed by the message "Change?" Enter the desired value and press the ↵ key. If no change is made, press only ↵.

Two HOME poses (HOME1 and HOME2) can be set in the AS system. HOME 1 is set using SETHOME command, HOME 2 using SET2HOME command.

**Example**

```
>SETHOME 2↵
```

Sets the accuracy at 2 mm, and changes the HOME pose by entering the new values.

```
    JT1   JT2      JT3      JT4      JT5      JT6  accuracy[mm]
    0.     0.       0.       0.       0.       0.       2.
    Change? (If not , hit RETURN only)
    ,90,-90
    JT1  JT2       JT3      JT4      JT5      JT6   accuracy[mm]
    0.     90.     -90.      0.       0.       0.       2.
    Change? (If not , hit RETURN only)↵
    >
```

```
>SETHOME  10,HERE↵
```

Sets the current pose as the HOME pose. The accuracy is set at 10 mm; i.e. the dedicated signal HOME will be output when the robot reaches the range of 10 mm from the HOME pose.

## ERRLOG

**Function**

Displays the error log.

**Explanation**

Displays the last one hundred errors. When the display reaches the end of the screen, press the Spacebar to continue viewing. Errors are listed in chronological order.
(Auxiliary function 0702)

**Example**

```
>ERRLOG↵

1-[02/07/17 09:55:45 (SIGNAL:00)
   (D1016)
:
```

## OPLOG

### Function

Displays the operation log.

### Explanation

Displays the last one hundred operations in the format shown below. When the display reaches the end of the screen, press the Spacebar to continue viewing.
(Auxiliary function 0703)

### Example

```
            >OPLOG↵
1-[02/07/17 10:04:46](SIGNAL:00) [ PNL ]
```

## SWITCH switch name, ……., switch name = ON
## SWITCH switch name, ……., switch name = OFF

**Function**

Displays and changes the system switches and their setting.

**Parameter**

Switch name

Displays the specified switch. If not specified, all the switches are displayed. More than one switch name can be entered separating each switch name by commas.

ON or OFF

Turns ON or OFF the specified system switch. If this parameter is not entered, the switch setting is displayed.

**Explanation**

Displays the setting state of the specified system switch with ON (enabled) or OFF (disabled). For the switches with * at the beginning of the names, only the display of the setting state is possible. The settings cannot be changed.

**Example**

```
>SWITCH↵                    Displays all system switches and their setting.
*POWER          ON      *REPEAT                 ON
*RUN            ON      *CS            OFF
*RGSO           OFF     *ERROR         OFF
*TRIGGER        ON      *TEACH_LOCK    OFF
CHECK.HOLD      OFF       CP           ON
CYCLE.STOP      OFF       OX.PREOUT    ON
PREFETCH.SIGINS         OFF     QTOOL           OFF
REP_ONCE                OFF     RPS             OFF
STP_ONCE                OFF     AFTER.WAIT.TMR        ON
MESSAGES                ON      SCREEN                ON
AUTOSTART.PC  OFF         AUTOSTART2.PC         OFF
AUTOSTART3.PCOFF          ERRSTART.PC OFF
DISPIO_01       OFF       HOLD.STEP    OFF
FLOWRATE        OFF       SPOT_OP      OFF
>↵
 ││
```

```
>SWITCH      SCREEN, MESSAGE = OFF  ↵   Turns OFF SCREEN and MESSAGE.
SCREEN            OFF
MESSAGE           OFF
>↵
```

## switch name, ……. ON

**Function**

Turns ON the specified system switch.

**Parameter**

Switch name

Turns ON the switch specified here. More than one switch name can be entered separating each switch name with a comma.

The current setting of the switch can be checked using the SWITCH command.

**Example**

|  |  |
|---|---|
| >MESSAGES ON ↵ | Turns ON the switch MESSAGES. |
| >SCREEN, MESSAGES ON ↵ | Turns ON the switches MESSAGES and SCREEN. |

## switch name, ……. OFF

**Function**

Turns OFF the specified system switch.

**Parameter**

Switch name

Turns OFF the switch specified here. More than one switch name can be entered separating each switch name with a comma.

The current setting of the switch can be checked using the SWITCH command.

**Example**

| | |
|---|---|
| >MESSAGES OFF ↵ | Turns OFF the switch MESSAGES. |
| >SCREEN, MESSAGES OFF ↵ | Turns OFF the switches MESSAGES and SCREEN. |

**Function**

Displays and sets max. number of external I/O signals.

**Explanation**

The current number of signals and message are displayed. (See Example)
FB1 gives the signal numbers for the FB1 port, and FB2 for the FB2 port.
When not changing the setting, just press ↵.

1. ZSIGSPEC sets the value only in the software. Setting is invalid unless hardware corresponds to it.
2. Set the signal numbers in multiples of 16.
3. Max. number of external I/O signals is 960:
   Input signals DI + FB1 + FB2 $\leq$ 960
   Output signals DO + FB1 + FB2 $\leq$ 960
4. Turn controller power OFF, then ON to activate the number of I/O signals set as I/O data length for the physical fieldbus interface.

**[ NOTE ]**

Ensure that the max. number of signals set by this command is consistent with the setting in [Aux. 0608 Signal Allocation]. If not, the max. number of signals cannot be set by this command.

**Example: When increasing the number of FB1 and FB2 port signals**

```
>ZSIGSPEC↵
                    DO,     DI,     INT,     FB1,     FB2
                    64      64      960       32       32
Change? (If not, Press RETURN only)
, , , 112, 64
                    DO,     DI,     INT,     FB1,     FB2
                    64      64      960      112       64
Change? (If not, Press RETURN only)
```

## ZSIGMAP type, AS signal start number, port signal start number, number of signals

**Function**

Specifies the signal allocation setting.

**Parameter**

1. Type
   Specifies the port type.
   0: DIO port
   1: FB1 port
   2: FB2 port

2. AS signal start number
   1 to AS max. number of signals

3. Port signal start number
   1 to port max. number of signals

4. Number of signals
   1 to max. number of signals of each port

**Explanation**

Specifies the signal allocation setting.
When the AS system switch, SIGMAPD, is OFF, a setting equivalent to Aux. 0608 setting is specified.
When a setting that cannot be achieved with a standard signal allocation setting is specified in Advanced Signal Allocation setting, error message P4706, "Cannot use this data by Signal Allocation Setting.," appears.
Specifying the ZSIGMAP/Z identifier will overwrite the setting in Advanced signal allocation setting.

**Example 1: When allocating 1 to 480 of FB1 port to 1 to 480 of AS signal**

```
ZSIGMAP 1,1,1,480
```

**Example 2: When allocating 1 to 480 of FB2 port to 481 to 960 of AS signal**

```
ZSIGMAP 2,481,1,480
```

## HSETCLAMP

**Function**

Assigns signal numbers to operate material handling clamps.

**Example**

In the example below, clamp 3 is set as a double solenoid.

```
>HSETCLAMP↵
                    Clamp 1       Clamp 2       Clamp 3       Clamp 4
                    Spot weld     Handling      Not used      Not used
'ON'out.signal        10             0            24            24
'OFF'out.signal        9            11             0             0
                    Clamp 5       Clamp 6       Clamp7        Clamp 8
                    Not used      Not used      Not used      Not used
'ON'out.signal        24            24            24            24
'OFF'out.signal        0             0             0             0
Clamp number (1~8, ENTER only:No change, CTRL+C:Exit)  3          ;Select clamp number
Define as Handling clamp ?
  (1:Defined as Handling clamp, 0:Not used, ENTER only:No change, CTRL+C:Exit)1
                                             ;Enter 1 to define as handling clamp.
  For single solenoid valve, define one signal.
  For double solenoid valve, define both.
'ON' out. signal
(0:Not used, ENTER only:No change, CTRL+C:Exit) Change ? 12;     Set so that ch12 is output
                                                                  if ON
'OFF' out. signal
(0:Not used, ENTER only:No change, CTRL+C:Exit) Change ?; 13     Set so that ch13 is output
                                                                  if OFF

                       Clamp 1       Clamp 2       Clamp 3       Clamp 4
                       Spot weld     Handling      Handling      Not used
   'ON'out.signal        10             0            12            24
   'OFF'out.signal        9            11            13             0
                       Clamp 5       Clamp 6       Clamp7        Clamp 8
                       Not used      Not used      Not used      Not used
   'ON'out.signal        24            24            24            24
   'OFF'out.signal        0             0             0             0
Clamp number (1~8, ENTER only:No change, CTRL+C:Exit)  ↵
```

[ **NOTE** ]

Always use the clamps in order from one to eight.  For example clamp 5 cannot be used without using clamp 4.

[ **NOTE** ]

$\boxed{\text{Ctrl}}+\boxed{\text{C}}$ (Exit) cannot be used from the teach pendant keyboard screen.

## DEFSIG INPUT
## DEFSIG OUTPUT

**Function**

Displays and changes the current setting of the software dedicated signals.

**Parameter**

INPUT, OUTPUT

OUTPUT (or only O) displays the output signals, INPUT (or only I) the INPUT signals. The setting can be changed when this parameter is entered. If this parameter is not entered, the signals currently used as dedicated signals are displayed in a list.

**Explanation**

The signals in the table below can be used as dedicated signals.

For details on dedicated signals, refer to the External I/O Manual.

| Software Dedicated Input Signal | Software Dedicated Output Signal |
|---|---|
| EXT. MOTOR ON | MOTOR_ON |
| EXT. ERROR RESET | ERROR |
| EXT. CYCLE START | AUTOMATIC |
| EXT. PROGRAM RESET | CYCLE START |
| Ext. prog. select (JUMP_ON, JUMP_OFF RPS_ON, RPSxx) | TEACH MODE HOME1, HOME2 |
| EXT_IT | POWER ON |
| EXT. SLOW REPEAT MODE | RGSO |
|  | Ext. prog. select (JMP_ST, RPS_ST) |

The following codes can be used with this command.

L: Go back to the previous signal.

N: Go to the next signal.

Q: Cancel operation (the data input are ignored)

E: Exit (Input data is enabled)

[ **NOTE** ]

1. External program selection
   (1) When selecting JMP as a dedicated signal, signals JMP-ON, JMP-OFF, JMP-ST are also automatically set as dedicated.  JMP-ST is an output signal but is set under DEFSIG INPUT command.

   (2) When selecting RPS signal as a dedicated signal, signals RPS-ON, RPS-ST are also automatically set as dedicated signals. RPS-ST is an output signal but is set under DEFSIG INPUT command.

2. RPS code
   If at least one of the following signals is selected as dedicated signal, a prompt appears and an RPS code must be input.
   JMP, RPS or EXT. PROGRAM RESET

3. Signal numbers
   Signals can be set within the following range:
   Dedicated output signals: 1 to number of signals installed
   Dedicated input signals: 1001 to number of signals installed

4. Others
   If a signal number is already assigned to a dedicated signal, it cannot be assigned to another dedicated signal or used as a general purpose signal.

**Example**

The following example displays the currently selected software dedicated signals.

```
>DEFSIG
Dedicated signals are set

  EXT. MOTOR ON = 1032
  EXT. ERROR RESET = 1031
  EXT. CYCLE START = 1030
  MOTOR ON = 32
  ERROR = 31
  AUTOMATIC = 30
    Condition : Panel switch in RUN.
    Condition : Panel switch in REPEAT.
    Condition : Repeat continuous.
    Condition : Step continuous.
```

```
     CYCLE START = 29
     TEACH MODE = 28
     HOME1 = 27
>↵
```

The following example resets the selection of the software dedicated output signal MOTOR_ON, changes the signal number of AUTOMATIC to 30, selects TEACH MODE as dedicated signal and sets the signal number 3.

```
>DEFSIG OUTPUT
MOTOR ON       Dedication cancel? (Enter 1 to cancel.)1 ↵
ERROR       Dedication cancel? (Enter 1 to cancel.)↵
      Signal number       31       Change ?    (1 - 32 )   ↵
AUTOMATIC       Dedication cancel? (Enter 1 to cancel.) ↵
      Signal number       2       Change ?    (1 - 32 ) 30      ↵
CYCLE START       Dedication cancel? (Enter 1 to cancel.) ↵
      Signal number       29       Change ?    (1 - 32 )   ↵
TEACH MODE       Dedication set? (Enter 1 to set.) 1 ↵
      Signal number       0       Change ?    (1 - 32 ) 3     ↵
HOME1       Dedication cancel? (Enter 1 to cancel.) ↵
>
```

## ZZERO joint number

**Function**

Sets the encoder value corresponding to the mechanical origin of each axis of a robot as zeroing data. Also, the current zeroing data and the value of encoder rotation counter can be displayed using this command.

**Parameter**

Joint number

1. To reset the encoder rotation counter:
   Enter the joint number plus 100. For example, to reset the encoder rotation counter on joint two, enter:

   ```
   ZZERO   102↵
   ```

If "100" is entered as the joint number, all the encoder counters are reset.

2. To set zeroing data:
   To set the encoder zeroing data for joint two, enter:

   ```
   ZZERO    2↵
   ```

If "0" is entered as the joint number, all the joints are zeroed.

If no joint number is specified, the current encoder data and the zeroing data are displayed.

**[ NOTE ]**

Reset the encoder rotation counter before setting the zeroing data.

**⚠ DANGER**

**Use this command only for the following purposes:**

**1. To check if the zeroing data has changed when the position of the arm is abnormal.**
**2. To correct the zeroing data when it has changed unexpectedly.**

**When the zeroing data is changed, the values detected for robot poses also change. Therefore be aware that the same program ends in different destination pose and trajectory before and after the zeroing data is changed.**

**Example**

1. The following command displays the zeroing data:

```
        >ZZERO↵
                 JT1         JT2         JT3         JT4         JT5         JT6
Set data     268435456   268435456   268435456   268435456   268435456   268435456
Current      268435456   268435456   268435456   268435456   268435456   268435456
data
   Change  (If not , hit RETURN only)↵

                 JT1         JT2         JT3         JT4         JT5         JT6
OFFSET            0           0           0           0           0           0
   Change? (If not , hit RETURN only)↵
       >
```

2. The following command resets the encoder rotation counter of all the joints.

```
        >ZZERO 100
        ** Encoder rot. counter reset (all joints) **
        Are you sure? (Enter 1 to execute)     1↵
        Setting complete.
        >
```

3. The following command resets the encoder rotation counter of joint 2 with the joint value specified for [Current angle] as the current value.

```
        >ZZERO 102
        ** Encoder rot. counter reset (joint 2) **
        Current angle (deg, mm) ?     0↵
        Are you sure? (Enter 1 to execute) 1↵
        Setting complete.
        >
```

4. The following command sets the zeroing data of all the joints simultaneously. The current value will become 0°.

```
        >ZZERO 0
        Set current values of all joints as zeroing data? (Enter 1 to set.)1↵
        Setting complete.
        >
```

5. Resets the zeroing data of joint 2 with the value specified for [Current angle] as the current value.

```
    >ZZERO 2  ↵
      Current angle (deg.mm)?      0 ↵
      Change? (If not, Press RETURN only.)   ↵
      Encoder value? (Current=268435456, Enter 1to set current value) 1 ↵
      Zeroing value=268435456 (268419072-268451840) OK?    (Enter 0 to change) ↵
      Setting complete.
     >
```

## ERESET

### Function

Resets the error condition. Identical to the $\boxed{\text{ERROR RESET}}$ button on the operation panel.

### Explanation

When the ERESET command is executed, the ERROR_RESET signal is output. However this command is ineffective when an error occurs continuously.

## SYSINIT

### Function

Deletes all program and data in the memory and initializes defined parameters.

### Explanation

Initializes the system and deletes all programs, pose variables, numeric variables, and string variable.

**[ NOTE ]**

All programs and variables are deleted from the memory when this command is executed.

**HELP alpha character**
**HELP/ M alpha character**
**HELP/ P alpha character**
**HELP/ F alpha character**
**HELP/ PPC alpha character**
**HELP/ MC alpha character**
**HELP/ DO alpha character**
**HELP/ SW alpha character**

**Function**

Displays a list of AS Language commands and instructions.

**Parameter**

Specifies with which letter in the alphabet the command, instruction, etc. begins. If omitted, all the commands, instructions are displayed.

For example, entering HELP command followed by an alpha character displays the monitor commands or program instructions starting with that alphabet. Entering HELP/F command followed by an alpha character command displays the functions starting with that alphabet.

**Explanation**

Entering HELP only, displays a list of monitor command and program instructions.
HELP/M lists the monitor commands.
HELP/P lists the program instructions.
HELP/F lists functions.
HELP/PPC lists program instructions usable in PC programs. (Option)
HELP/MC lists monitor commands usable with the MC instruction. (Option)
HELP/DO lists program instructions usable with DO command. (Option)
HELP/SW displays a list of system switches. (Option)

For some commands and instructions, the parameters are also displayed.

**Example**

```
    >HELP/M↵
        ABORT   BASE        BITS     BATCHK  CONTINUE   COPY      DEFSIG
        DELETE  DIRECTORY   DLYSIG   DO      EDIT       ERESET    ERRLOG

    >HELP/F↵
        #DEST      #PPOINT    $CHR      $DECODE      $ENCODE      $ERROR
        $LEFT        $MID      $RIGHT      $SPACE     ABS      ASC
```

## ID

**Function**

Displays the version information of the software installed in the robot controller.

**Explanation**

Displays the following information.

| | |
|---|---|
| Robot name: | the name of the robot arm controlled by AS software |
| Joint number: | number of joints of the controlled robot arm |
| Serial number: | serial number of the controlled robot arm |
| Number of signals: | maximum number of output, input, and internal signals available in this system |
| Clamp number: | maximum number of clamps available in this system |
| Motion type: | motion type currently set |
| Servo type: | type of servo software currently set |
| Acceleration speed change per load: | enable/ disable of acceleration speed change function per load |
| Servo specification: | control specification of the servo software |
| Software version: | version number of the AS software |

[ **NOTE** ]

If the above information does not match the actual robot, contact us immediately. Do not turn ON the motor power nor make the robot do any motion operations.

**Example**

```
>ID↵
        Robot name: RS010N-A001    Num of axes: 6        Serial No. 1
        Number of signals: output=32          input=32          internal=960
        Clamp number: 2    MOTION TYPE: 2    SERVO TYPE: 2
        ACC. & DEC. VARIABLE BY WEIGHT : OFF
        Servo Spec: 5
   [SOFT VERSION]
=== AS GROUP ===          : ASF_010000005 2016/09/16 11:33
USER IF AS                : UASF010000005 2016/09/16 11:33
USER IF TP                : UTPF010000005 2016/09/16 11:31
ARM CONTROL AS            : AASF010000005 2016/09/16 11:32
```

```
USER IF AS MESSAGE FILE : MASF0100005EN 2016/09/14 18:00
USER IF TP MESSAGE FILE : MTPF0100005EN 2016/09/14 18:00
ARM DATA FILE           : ARMF010000005 2016/09/16 11:31
KERNEL                  : _KNL000610000 2016/08/26
DRIVER                  : _DRV100500001 2016/08/26
RFS                     : _RFS100500001 2016/08/25
=== SERVO GROUP ===  : SVF_010000003 2016/09/14 18:42
ARM CONTROL SERVO : SVSF010000003 2016/09/14 18:21
SRV DATA FILE           : SVPF010000003 2016/09/14 18:42
ARM CONTROL SERVO FPGA: SVFF010000002 2016/09/14 18:08
```

[ **NOTE** ]

Displayed software version may vary from above according to the selected option.

## WEIGHT load mass, center of gravity X, center of gravity Y, center of gravity Z, inertia moment ab. X axis, inertia moment ab. Y axis, inertia moment ab. Z axis

### Function

Sets the load mass data (weight of tool and workpiece). The data is used to determine the optimum acceleration of the robot arm.

### Parameter

Load mass

The mass of the tool and workpiece (in kilograms). Range: 0.0 to the maximum load capacity (kg).

Center of gravity (unit = mm)

X: the x value of the center of gravity in tool coordinates

Y: the y value of the center of gravity in tool coordinates

Z: the z value of the center of gravity in tool coordinates

Inertia moment about X axis, inertia moment about Y axis, inertia moment about Z axis (Option)

Sets the inertia moment around each axis. Unit is kg·m$^2$. The inertia moment about each axis is defined as the moment around the coordinates axes parallel to the null tool coordinates with the center of rotation at the tool's center of gravity.

### Explanation

If no parameters are specified, the current value is displayed followed by the message "Change?"

**DANGER**

**Always set the correct load mass and center of gravity. Incorrect data may weaken or shorten the longevity of parts or cause overload/deviation errors.**

## ENCCHK_EMG

**Function**

Sets an acceptable deviation range when checking the robot’s pose at an emergency stop versus the pose when the robot is restarted.

**Explanation**

Deviation ＝|(pose after motor power is reapplied) –(pose after the emergency stop)|

The acceptable deviation range can be set at each joint.  If 0.0 is set as the range, deviation check is not performed.  Setting too small of a range may trigger an error when motor power is reapplied after an emergency stop even if the robot is operating within performance specifications.

## ENCCHK_PON

### Function

Sets the acceptable range for the difference in encoder value when the control power is turned ON versus the value when the power was turned OFF the last time.

### Explanation

Acceptable range = | (value when control power is turn ON) – (value when the control power was turned OFF the last time)|

The acceptable range can be set at each joint. Setting too small of a range may trigger an error when motor power is reapplied after an emergency stop even if the robot is operating within performance specifications.

## SLOW_REPEAT

### Function

Sets the repeat speed in slow repeat mode.

### Explanation

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

If no change is made, press only $\boxed{\hookleftarrow}$.    To change the speed, enter the new value and press $\boxed{\hookleftarrow}$.

**[ NOTE ]**

$\boxed{\text{Ctrl}}+\boxed{\text{C}}$ (Exit) cannot be used from the teach pendant keyboard screen.

## REC_ACCEPT

**Function**

Enables or disables RECORD and PROGRAM CHANGE functions.

**Explanation**

```
>REC ACCEPT  ↵
RECORD(0:Enable, 1:Disable)
        (Enter only: No change ^C:Exit): Now    0 Change ?
PROGRAM CHANGE(0:Enable, 1:Disable)
        (Enter only: No change ^C:Exit): Now    0 Change ?
```

Enter 0 to enable RECORD or PROGRAM CHANGE option.  Enter 1 to disable the options.

[ **NOTE** ]

1. Ctrl + C (Exit) cannot be used from the teach pendant keyboard screen.
2. If PROGRAM CHANGE is disabled, the following message appears when EDIT command is executed: "Program change inhibited. Set ACCEPT and operate again." REC_ACCEPT command cannot be used in EDIT mode.

## ENV_DATA

### Function

Sets hardware environmental data. (Auto servo OFF timer and status of teach pendant installation)

### Explanation

```
      >ENV DATA ↵
        AUTO SERVO OFF TIMER(0:Servo not off)
            (Enter only: No change ^C:Exit): Now     0 Change ?
```

If no change is to be made, press only ↵. Enter 0 to disable the auto servo OFF timer. To enable the timer, enter after how much time (in seconds) the servo turns OFF.

Next, a prompt for the teach pendant is displayed.

```
        TEACH PENDANT(0:Connect, 1:Disconnect)
            (Enter only: No change ^C:Exit): Now     0 Change ?
```

If no change is made, press only ↵. To operate the robot without connecting the teach pendant, enter 1. Connect the short circuit plug after disconnecting the teach pendant.

**[ NOTE ]**

$\boxed{\text{Ctrl}}+\boxed{\text{C}}$ (Exit) cannot be used from the teach pendant keyboard screen.

## ENV2_DATA

### Function

Sets software environmental data.

### Explanation

```
        >ENV2 DATA
        PANEL (0:Connect, 1:Disconnect)
                (Enter only: No change ^C:Exit): Now   0 Change ?
```

If no change is made, press only ↵.    Next, a prompt for the terminal setting is displayed.

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

If no change is made, press only ↵.    Usually the personal computer is set as the terminal.

**[ NOTE ]**

Ctrl＋C (Exit) cannot be used from the teach pendant keyboard screen.

## Function

Enables or disables the resetting of abnormal check sum error.

## Explanation

```
>CHSUM
  CLEAR CHECK SUM ERROR(0:Ineffect, 1:Effect)
         (Enter only: No change ^C:Exit): Now      0 Change ?
```

If “0” is entered, error cannot be reset. If “1” is entered the error is reset. The default value is “0”. CHSUM is reset to “0” when the control power is turned OFF.

The following message appears if the error cannot be reset:

```
>CHSUM
Cannot clear check sum error. Check the following command or auxiliary data.
     ZZERO
     DEFSIG
     :
     :
>
```

If any data still contains an abnormal check sum, the message in the first example (CLEAR CHECK SUM ERROR) does not appear. Instead, a message (as in the second example) is displayed identifying additional troubleshooting.

**[ NOTE ]**

Ctrl + C (Exit) cannot be used from the teach pendant keyboard screen.

## TPLIGHT

**Function**

Turns on the teach pendant backlight.

**Explanation**

If the backlight of the teach pendant screen is OFF, then this command turns ON the light. If this command is executed when the backlight is ON, the light stays on for the next 600 seconds.

## IPEAKLOG

Option

**Function**

Displays the peak current value for each joint.

**Explanation**

Displays the program name, step number, effective current value [Arms], and the ratio of peak value to mechanical limit or motor limit when the motor torque is at its highest for each joint.

**Example**

```
        >IPEAKLOG↵
        Log since   00/1/24   13:44:23

Joint      Program     Step      Effective current                          Date
JT1          pg223        5          4.1[Arms]          29.5[%]        00/1/24   13:44
JT2          pg223       13          1.4[Arms]           9.8[%]        00/1/24   13:44
JT3          pg223       10         13.7[Arms]          98.2[%]        00/1/24   13:44
JT4          pg223        1          2.1[Arms]          55.5[%]        00/1/24   13:45
JT5          pg223        7          2.4[Arms]          64.8[%]        00/1/24   13:45
JT6          pg223        4          1.0[Arms]          27.8[%]        00/1/24   13:45
```

## IPEAKCLR

Option

**Function**

Clears the peak current value log and restarts logging values.

**Explanation**

The logged values are reset using the IPEAKCLR command, the SYSINI command, or by initializing the system by turning on No.8 dip switch on 1TA board.

Example

```
>IPEAKCLR↵
Are you sure? (Yes:1,No;0) 1↵
>
```

## OPEINFO robot number: joint number

**Function**

Displays the operation information.

**Parameter**

Robot number

Specifies the robot if more than one robot is controlled by one controller.

Joint number

Specifies for which joint the information is to be displayed. If not specified, the data on all the joints are displayed.

**Example**

```
          >OPEINFO↵
Operation Info. (02/1/14   9:54:1 - )(FILE LOAD   02/1/14)      ;describes from which hour data
Control ON               0.2 [H]                                  accumulation started
Servo ON                 0.1 [H]
Motor ON                 4 times
Servo ON                 10 times
Emergency stop (while in motion)      2 times
JT1
Operation hour           0.1 [H]
Operation distance       301.28 [x100 deg, mm]
JT2
Operation hour           0.1 [H]
Operation distance       193.84 [x100 deg, mm]
:
```

**[ NOTE ]**

Limits on data accumulation

1. Hours[H]:69 years
2. Number of operation [times]: ab. 2000 million times (1176 years if operated 10000 times a day)
3. Distance [deg, mm]: 12.5 years if operated at 500 mm/s

## OPEINFOCLR

### Function

Resets the operation information to 0.

## REFFLTSET_STATUS

**Function**

Displays the default and current values for moving average span of the command values.

**Explanation**

This instruction allows checking the default and current values of the moving average span modified by REFFLTSET instruction.

**Example**

In the sample program below, the default value of the command value moving average span is 48 ms. The screen display will show as below if REFFLTSET_STATUS instruction is executed while the robot is moving from #p1 to #p2.

Program example

```
1 JMOVE #p1
2 REFFLTSET 128
3 JMOVE #p2
```

Example of screen display (REFFLTSET_STATUS instruction is executed while step 3 is being executed):

```
>REFFLTSET_STATUS
  Default value [ms] = 48 48 48 24
  Current value [ms] = 128 128 128 64
```

## FFSET_STATUS

**Function**

Displays the default and current values of robot speed/ acceleration speed feed forward gain.

**Explanation**

Allows checking the default and current values of speed/acceleration feed forward gain changed via FFSET instruction.

**Example**

In this example, the below sample program is executed with the default value of 0.7 for speed/ acceleration speed feed forward gain. The screen display shows as below when FFSET_STATUS instruction is executed while the robot is moving from #p1 to #p2.

Program example

```
1 JMOVE #p1
2 FFSET 0.5
3 JMOVE #p2
```

Example of screen display (FFSET_STATUS instruction is executed while step 3 is being executed):

```
>FFSET_STATUS
Default value: KVFF = 0.70, 0.70, 0.70, 0.70, 0.70, 0.70
Current value: KVFF = 0.50, 0.50, 0.50, 0.50, 0.50, 0.50
Default value: KAFF = 0.70, 0.70, 0.70, 0.70, 0.70, 0.70
Current value: KAFF = 0.50, 0.50, 0.50, 0.50, 0.50, 0.50
```

## ENC_TEMP robot number: joint number

**Function**

Displays the encoder temperature information of each joint.

**Parameter**

Robot No.

Specifies a robot number when one controller is controlling multiple robots.

Joint number

Specifies the joint number to display encoder temperature information. Specifying 0 resets the encoder temperature information of all joints. Displays encoder temperature information of all controlled joints when omitted.

**Explanation**

Displays each joint’s encoder temperature information (current temperature, lowest temperature, detection date/time of lowest temperature, highest temperature, detection date/time of highest temperature), as well as the ability to reset encoder temperature information.

**Example**

```
>ENC_TEMP↵
Robot No.1
JT      Current temp.   Lowest temp.    Lowest temp.           Highest temp.   Highest temp.
        (ºC)            (ºC)            Detection date/time    (ºC)            Detection date/time
1       35.000          25.000          [19/01/01 09:00:00]    60.000          [19/01/01 15:00:00]
2       35.000          25.000          [19/01/01 09:00:00]    60.000          [19/01/01 15:00:00]
3       35.000          25.000          [19/01/01 09:00:00]    60.000          [19/01/01 15:00:00]
4       35.000          25.000          [19/01/01 09:00:00]    60.000          [19/01/01 15:00:00]
5       35.000          25.000          [19/01/01 09:00:00]    60.000          [19/01/01 15:00:00]
6       35.000          25.000          [19/01/01 09:00:00]    60.000          [19/01/01 15:00:00]
```

## SETENCTEMP_THRES [/N] robot number: joint number, warning threshold, error threshold

**Function**

Performs the threshold setting of encoder temperature warning and temperature error of each joint.

**Parameter**

/N

Specifies the presence/absence of inquiries. Specifying /N means no inquiry.

Robot No.

Specifies a robot number when one controller is controlling multiple robots.

Joint number

Specifies the joint number to set the encoder temperature warning and temperature error thresholds. Sets encoder temperature warning and temperature error thresholds for all joints by joint when omitted.

Warning threshold

Sets the encoder temperature warning threshold. Warning threshold can be set between 0 and 125ºC. Specify -1 to reset to default value.

Error threshold

Sets the encoder temperature error threshold. Error threshold can be set between 0 and 125ºC. Specify -1 to reset to default value.

**Explanation**

Allows the setting and resetting of the encoder temperature warning and temperature error thresholds of each joint.

**[NOTE]**

Encoder temperature warning thresholds and temperature error thresholds set by this command may be reset to default values due to encoder replacement.

Temperature warning: (W1085) “Encoder temperature exceeded limit.(jt XX) (XX deg C)”

Temperature error: (E1564) “Encoder temperature exceeded limit.(jt XX) (XX deg C)”

**CAUTION**

**Do not raise threshold temperatures above default values, as any rise of temperature may affect arm components such as the encoder.**

**Example**

When setting the encoder temperature warning threshold as 80°C and encoder temperature error threshold as 90°C for JT2:

```
>SETENCTEMP_THRES 2↵
Robot No.1
JT2 Encoder temperature threshold
Warning[deg C]
   95 (DEFAULT SETTING: 95[deg C])
Change? (If not, Press RETURN only.)
80
Warning[deg C]
   80 (DEFAULT SETTING: 95[deg C])
Change? (If not, Press RETURN only.)

Error[deg C]
   95 (DEFAULT SETTING: 95[deg C])
Change? (If not, Press RETURN only.)
90
Error[deg C]
   90 (DEFAULT SETTING: 95[deg C])
Change? (If not, Press RETURN only.)
```

## RESET

### Function

Turns OFF all the external output signals.   Dedicated signals, clamp signals and antinomy signals for multifunction OX/WX are not affected by this command.

By using the optional setting, the signals used in the Interface Panel screen are not affected by this command.   (Option)

**[ NOTE ]**

Beware that this command turns OFF all the signals other than those mentioned above even in repeat mode.

## SIGNAL signal number, ……

**Function**

Turns ON (or OFF) the specified external or internal I/O signal.

**Parameter**

Signal number

Selects the number of external output signal or internal signal. Positive number turns ON the signal, negative number turns OFF.

**Explanation**

The signal number determines if the signal is an external or internal signal.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1 – actual number of signals |
| Internal signal | 2001–2960 |
| External input signal | Cannot be defined |

If the signal number is positive, the signal is turned ON; if negative, the signal is turned OFF. If "0" is given, all the signals are turned OFF. This command does not take effect upon dedicated signals, clamp signals, and multipurpose double OX/WX signals.

**Example**

| | |
|---|---|
| >SIGNAL -1,4,2010 ↵ | External output signal1is OFF, 4 is ON, Internal signal 2010 is ON. |

| | |
|---|---|
| `>SIGNAL  -reset,4 ↵` | If the value of the variable “reset” is positive, the output signal determined by that value is turned OFF, and output signal 4 is turned ON. |

## PULSE signal number, time

**Function**

Turns ON the specified signal for the given period of time.

**Parameter**

Signal number

Selects the number of the external output signal or internal signal (only positive values). Error occurs if the signal number is already used as a dedicated signal.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1 – actual number of signals |
| Internal signal | 2001–2960 |

Time

Sets for how long the signal is output (in seconds). If not specified, it is automatically set at 0.2 seconds.

**DLYSIG signal number, time**

**Function**

Outputs the specified signal after the given time has passed.

**Parameter**

Signal number

Selects the number of the external output signal or internal signal. If the signal number is positive, the signal is turned ON; if negative, the signal is turned OFF. Error occurs if the signal number is already used by a dedicated signal.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1 – actual number of signals |
| Internal signal | 2001–2960 |

Time

Specifies the time to hold the output of the signal in seconds.

---

## BITS starting signal number, number of signals = value

**Function**

Arranges a group of external output signals or internal signals in a binary pattern. The signal states are set ON/OFF according to the binary equivalent of the specified value. If the value is not specified, the current signal states are displayed.

**Parameter**

Starting signal number

Specifies the first signal to set the signal state.

Number of signals

Specifies the number of signals to be set ON/OFF. The maximum number allowed is 16. To set more than 16 signals, use BITS32 command, explained below.

Value

Specifies the value used to set the desired ON/OFF signal states. The value is transformed into binary notation and each bit of the binary value sets the signal state. The least significant bit corresponds to the signal with the smallest signal number, and so on. If the binary notation of this value has more bits than the number of signals, only the state of the given number of signals (starting from the specified signal number) is set and the remaining bits are ignored.

If this parameter is omitted, the current signal states are displayed.

**Explanation**

Sets (or resets) the signal state of one or more external output signals or internal signals according to the given value.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1 – actual number of signals |
| Internal signal | 2001–2960 |

Specifying a signal number greater than the number of signals actually installed results in error. Selecting a dedicated signal also results in error.

**Example**

| | |
|---|---|
| >BITS 2001,3 ↵ | Displays the values of internal signals 2001-2003. (3 bits starting from signal number 2001). |
| BITS 2001,3 = 5 | When internal signals 2001, 2003 are ON and 2002 is OFF, displays 5 of decimals in binary representation of 101. |
| >BITS 1,8=100 ↵ | External output signals 1–8 are set to output 01100100 (the binary notation of 100). |

## BITS32 starting signal number, number of signals = value

**Function**

Arranges a group of external output signals or internal signals in binary pattern. The signal states are set ON/OFF according to the binary equivalent to the specified value. If the value is not specified, the current signal states are displayed.

**Parameter**

Starting signal number
Specifies the first signal to set the signal state.

Number of signals
Specifies the number of signals to be set ON/OFF. The maximum number allowed is 32.

Value
Specifies the value used to set the desired ON/OFF signal states. The value is transformed into binary notation and each bit of the binary value sets the signal state. The least significant bit corresponds to the signal with the smallest signal number, and so on. If the binary notation of this value has more bits than the number of signals, only the state of the given number of signals (starting from the specified starting signal number) is set and the remaining bits are ignored.

If this parameter is omitted, the current signal states are displayed.

**Explanation**

Sets (or resets) the signal state of one or more external output signals or internal signals according to the given value.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1 – actual number of signals |
| Internal signal | 2001–2960 |

Specifying a signal number greater than the number of signals actually installed results in error. Selecting a dedicated signal also results in error.

**Example**

```
>BITS32 1,32=^H7FFFFFFF ↵
```

External output signals 1–32 are set to correspond to binary notation of 7FFFFFFF. External output signals 1–31 turn ON.

```
>BITS32   2001,32 ↵
```

Displays the numerical values represented by internal signals 2001 to 2032 (32 bits).

```
 BITS32   2001,32 = 2147483647
```

When internal signals 2001 to 2031 are ON and 2032 is OFF, displays the decimal display value of hexadecimal 7 FFFFFFF.

## SCNT counter signal number = count up signal, count down signal, counter clear signal, counter value

**Function**

Outputs counter signal when the specified counter value is reached.

**Parameter**

Counter signal number

Specifies the signal number to output. Setting range for counter signal numbers: 3097 to 3128.

Count up signal

Specified by signal number or logical expressions. Each time this signal changes from OFF to ON, the counter counts up by 1.

Count down signal

Specified by signal number or logical expressions. Each time this signal changes from OFF to ON, the counter counts down by 1.

Counter clear signal
Specified by signal number or logical expressions.  If this signal is turned ON, the internal counter is reset to 0.

Counter value
When the internal counter reaches this value, the specified signal is output.  If “0”is given, the counter signal is turned OFF.

**Explanation**
If the count up signal changes from OFF to ON when the SCNT command is executed, then the internal counter value increases by 1.  If the countdown signal changes from OFF to ON, the internal counter value decreases by 1.  When the internal counter value reaches the value specified in the parameter (counter value), the counter signal is output.  If the counter clear signal is output, value of the internal counter is set at 0.  Each counter signal has its own individual counter value.  To force reset of the internal counter to 0, use SCNTRESET command.

To check the states of signals 3001 to 3128, use the IO/E command. (Option)

---

## SCNTRESET　counter signal number

### Function

Resets the internal counter value of the specified counter signal number to 0.

### Parameter

Counter signal number

Select the number of the counter signal to reset.　Setting range for counter signal numbers: 3097 to 3128.

## SFLK  signal number = time

**Function**

Turns ON/OFF (flickers) the specified signal in specified time cycle.

**Parameter**

Signal number

Specifies the number of signal to flicker.  Setting range: 3065 to 3096.

Time

Specifies the time to cycle ON/OFF (real values).  If a negative value is set, flickering is canceled.

**Explanation**

The process of ON/ OFF is considered one cycle, and the cycle is executed in the specified time.

## SFLP    output signal = set signal expression, reset signal expression

**Function**

Turns ON/OFF an output signal using a set signal and a reset signal.

**Parameter**

Output signal

Specifies the signal number of the signal to output.    A positive number turns ON the signal; a negative number turns OFF.    Only the signal numbers for output signals can be specified (from 1 to actual number of signals).

Set signal expression

Specifies the signal number or logical expression to set the output signal.

Reset signal expression

Specifies the signal number or logical expression to reset the output signal.

**Explanation**

If the set signal is ON, the output signal is turned ON.    If the reset signal is ON, the output signal is turned OFF.    The output signal is turned ON or OFF when the SFLP command is executed, and not when the set signal or the reset signal is turned ON.

## SOUT signal number = signal expression

**Function**

Outputs the specified signal when the specified condition is set.

**Parameter**

Signal number

Specifies the signal number of the signal to output. Only the signal numbers for output signals can be specified. (1 to actual number of signals).

Signal expression

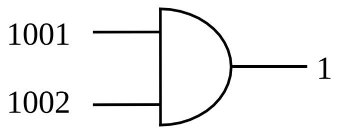
Specifies a signal number or a logical expression.

**Explanation**

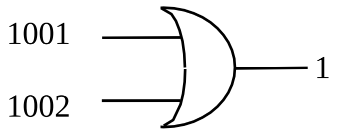
This command is for logical calculation of signals. Logical expressions such as AND and OR are used. The specified signal is output when that condition is set.

**Example**

SOUT 1 = 1001 AND 1002

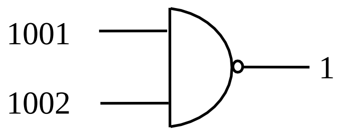


SOUT 1 = 1001 OR 1002

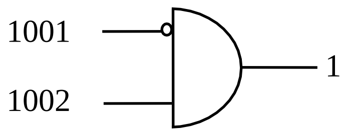


SOUT -1 = 1001 AND 1002

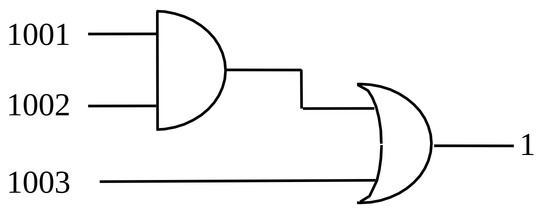
SOUT 1 = NOT(1001 AND 1002)



SOUT 1 = -1001 AND 1002

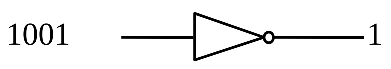


SOUT 1 = (1001 AND 1002) OR 1003



SOUT -1 = 1001 or SOUT 1= -1001 or

SOUT 1 = NOT(1001)



## STIM timer signal = input signal number, time

**Function**

Turns ON the timer signal if the specified input signal is ON for the given time.

**Parameter**

Timer signal

Selects the signal to turn ON.  Acceptable signal numbers are from 3001 to 3064.

Input signal number

Specifies in integers the input signal number or logical expression to monitor as a condition for turning ON the timer signal.  The value cannot exceed the number of signals actually installed.

Time

Specifies in real numbers the time (sec) the input signal must be ON before turning ON the timer signal.

**Explanation**

The monitored input signal has to be ON continuously in order for the timer signal to be turned ON.  If the input signal turns OFF before the given time passes, the time count restarts when that signal turns ON again.  If the input signal turns OFF, the timer signal turns OFF immediately. However, the input signal affects the timer signal only when STIM is executed.  Unless STIM is executed, the timer signal remains ON even when the input signal turns OFF.

To check the state of signals 3001 to 3064, use the IO/E command.

**Example**

```
STIM 3001 = 1,5 ↵                    sig2 turns ON if sig1 is ON for 5 seconds.

SOUT 2 = 3001 ↵

>PCEXECUTE↵
```

**SETPICK time1, time2, time3, time4, …, time8**

**SETPLACE time1, time2, time3, time4, …, time8**

**Function**

Sets the time to start clamp close control (SETPICK) or clamp open control (SETPLACE) for each of the 8 clamps.

**Parameter**

Time 1 to 8

Sets the control time to open/close clamps 1 to 8 in seconds. Setting range: 0.0 to 10.0 seconds.

**Explanation**

See CLAMP instruction for more details.

**HSENSESET no. = input signal number, output signal number, signal output delay time**

Option

**Function**

Declares the starting of signal detection to AS system. When this instruction is executed, AS system starts to watch the sensor signal and accumulates the data such as pose, etc., into the buffer memory at signal transaction. The data saved in the buffer memory can be read using HSENSE instruction. Buffer memory can save up to 20 data.

**Parameter**

No.

Specifies the number for the monitoring results. Up to 2 input signals can be monitored. Command for each signal is written as HSENSESET 1 or HSENSESET 2. Acceptable range is 1 or 2.

Input signal number

Set the signal number to monitor. Setting zero (0) terminates the monitoring.

Output signal number

Set the number of the signal to be output after system acquires the joint angle. The specified signal turns ON for 0.2 seconds. This may be omitted.

Signal output delay time

Set the time to delay the output of signal after acquiring the pose data. Acceptable range is 0 to 9999 ms. This may be omitted.

[ **NOTE** ]

Even when the control power becomes OFF during watching, buffer memory keeps the read data. It is possible to read the kept data by HSENSE instruction after turning ON the control power again. However, watching does not restart automatically, so HSENSESET should be executed again

**Example**

| | |
|---|---|
| >HSENSESET 1 = wx_sensor | Starts watching for input signal wx_sensor. |

**HSENSE no. result variable, signal status variable, pose variable, error variable, memory usage variable**

Option

**Function**

Reads the data saved in the buffer memory by HSENSESET instruction.

**Parameter**

No.

Sets the monitoring number. To read data saved by HSENSESET 1, specify HSENSE1. To read data saved by HSENSESET 2, specify HSENSE 2.

Result variable

Specifies the name of the real variable to which the watch result is assigned. After executing HSENSE instruction, numerical value is assigned to this variable. Zero (0) is assigned to this variable when AS system does not detect the signal transaction. –1 is assigned to this variable when AS system detects the signal transaction.

Signal status variable

Specifies the name of the real variable to which the status of signal transaction is assigned. After executing HSENSE instruction, a numerical value is assigned to this variable. When the signal(s) is turned from OFF to ON, ON (-1) is assigned. When the signal(s) is turned from ON to OFF, OFF (0) is assigned to this variable.

Pose variable

Specifies the name of the pose variable to which the joint values at time of HSENSE signal input are assigned.

Error variable

Specifies the name of the real variable to which the buffer overflow error result is assigned. When no error occurs, 0 is assigned to this variable. When buffer memory overflows, a numerical value (other than 0) is assigned to this variable. The buffer memory overflows after accumulating data from more than 20 transactions.

Memory usage variable

Specifies the name of the real variable to which the number of used memory in the buffer is assigned. The value assigned to this variable shows the number of memory in the buffer that is already used. When only one memory is used, 0 will be assigned to the variable. When all the memories are used, the value of the variable will be 19.

## RSIGRANGE

**Function**

Configures the display and settings of signal range to be used by RSIGPOINT instruction.

**Explanation**

Sets signal range by segments of every 32 points.

Specified value

Setting range: 1 to 30 (see below signal number correspondence table)

| Specified value | Output signals | Signal range |
|---|---|---|
| 1 | External output signals | 1 to 32 |
| 2 | External output signals | 33 to 64 |
| : | : | : |
| 29 | External output signals | 897 to 928 |
| 30 | External output signals | 929 to 960 |

After this command is entered, the current set value and message “Change?” are displayed. When using the RSIGPOINT instruction to change the signal range to be used, input the new value. Enter the ↵ key only for no changes.

**Example**

```
>RSIGRANGE
[Now: 1]                  The current set value is displayed.
Change? (If not, Press RETURN only.)
 2↵                       Specifies 2 when using external output signals of 33 to 64 as shutter
                          signals.
```

**PRINT device number: print data, …….**
**TYPE device number: print data, …….**

**Function**

Displays on the terminal the print data specified in the parameter.

**Parameter**

Device number

Select the device for displaying the data:

0: All terminals that are connected

1: Personal computer

2: Teach pendant

3: - 5: Terminals connected via Ethernet

If not specified, the data is displayed on the currently selected device.

Print data

Select one or more from below. Separate the data with commas when specifying more than one.

| | | |
|---|---|---|
| (1) character string | e.g. | "count =" |
| (2) real value expressions (the value is calculated and displayed) | e.g. | count |
| (3) Format information (controls the format of the output message) | e.g. | /D, /S |

A blank line is displayed if no parameter is specified.

**Explanation**

If "2" is entered for device number, the teach pendant screen changes automatically to keyboard screen.

The following codes are used to specify the output format of numeric expressions. The same format is used until a different code is specified. In any format, if the value is too large to be displayed in the given width, asterisks (*) will fill the space. In this case, change the number of characters that can be displayed. The maximum number of characters displayed in one line is 128. To display more than 128 characters in a line, use the /S code explained on the following page.

**[ NOTE ]**

If the MESSAGES switch is OFF, no message appears on the terminal screen.

Format Specification Codes

| | |
|---|---|
| **/D** | Uses the default format. This is the same as specifying the format as /G15.8 except that zeros following numeric values and all spaces but one between numeric values are removed. |
| **/Em.n** | Displays the numeric values in scientific notation（e.g. -1.234E+02）. "m" describes the total number of characters shown on the terminal and "n" the number of decimal places. "m" should be greater than n by five or more. |
| **/Fm.n** | Displays the numeric values in fixed point notation (e.g. -1.234). "m" describes the total number of characters shown on the terminal and "n" the number of digits in the fraction part. |
| **/Gm.n** | If the value is greater than 0.01 and can be displayed in Fm.n format within m digits, the value is displayed in that format. Otherwise, the value is displayed in Gm.n format. |
| **/Hn** | Displays the values as a hexadecimal number in the n digit field. |
| **/In** | Displays the values as a decimal number in the n digit field. |

The following parameters are used to insert certain characters between character strings.

| | |
|---|---|
| **/Cn** | Inserts line feed n times in the place where this code is entered, either in front or after the print data. If this code is placed within print data, n-1blank lines are inserted. |
| **/S** | The line is not fed. |
| **/Xn** | Inserts n spaces. |
| **/Jn** | Displays the value as a hexadecimal number in the n digit field. Zeros are displayed in place of blanks. (Option) |
| **/Kn** | Displays the value as a decimal number in the n digit field. Zeros are displayed in place of blanks. (Option) |
| **/L** | This is the same as /D except that all the spaces are removed with this code. (Option) |

**Example**

In this example the value of real variable “i” is 5, the fifth element of array variable “point” is 12.66666.

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

```
point 5 = 12.67
```

The display should look like this.

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

If the value of point[5] is 1000 (1000.00), the display should look like this.  The value is too large to display (i.e. the number of digits is greater than 5).

In the following example code /S is used to display the data without changing the lines after the data.

```
>PRINT “ABC”
>PRINT/S, “DEF”
>PRINT “GHI” ↵
```

```
| ABC
| DEFGHI            |
```

The display should look like this, with “GHI” displayed on the same line as “DEF”.

**IFPWPRINT window number, row, column, background color, label color = "character string", "character string", ……**

**Function**

Displays the specified character string in the string window set by Auxiliary Function 0509 (Interface panel screen).

**Parameter**

Window number

Corresponds to the window number specified in Auxiliary Function 0509 as the window specification used to display the string. Select from 1to 8 (standard).

Row

Specifies the row in the window for displaying the string. Acceptable number is from 1 to 4, though it depends on the window size. If not specified, 1 is assumed.

Column

Specifies the column in the window for displaying the string. Acceptable number is from 1 to 70, though it depends on the window size. If not specified, 1 is assumed.

Background color

Selects the color of the background of the selected window. Acceptable numbers are from 0 to 15. If not specified, the background is white.

| No. | Color | No. | Color | No. | Color | No. | Color |
|---|---|---|---|---|---|---|---|
| 0 | Grey | 4 | Green | 8 | Pink | 12 | Navy |
| 1 | Blue | 5 | Pale Blue | 9 | White | 13 | Reddish Brown |
| 2 | Red | 6 | Yellow | 10 | Black | 14 | Deep Green |
| 3 | Orange | 7 | White | 11 | Cyan | 15 | Lavender |

Label color

Selects the color of the characters displayed. Acceptable numbers are from 0 to 15 (See chart above). If not specified, the characters are displayed in black.

Character string

Specifies the character string to display. All strings after the first string are displayed on the next row starting at specified column. Execution of IFWPRINT clears all items, except the specified character strings, from the specified window.

**Explanation**

IFPWPRINT command can be used only when the interface panel is available for use.　If the parameters are not specified, the last setting of that particular window is selected (for first time use, the above default values are set).　If the character string does not fit in one row, its display overflows to the next line (indenting to the selected column).　Strings that extend beyond the size of the window are not displayed.　Control characters in the string are displayed as blanks.

**IFPWOVERWRITE mode window number, row, column, background color, label color = “character string”, “character string”, ……**

**Function**

Displays by overwriting the specified character string in the string window set by Auxiliary Function 0509 (Interface panel screen).

**Parameter**

Mode

(None) Overwrites the existing character string in unit of line.

/CUT Displays the character string by truncating the characters that do not fit in one line of the string window, without starting a new line. However, if the target window number is not allocated for the interface panel, the character string is saved to the full extent of the window and is displayed when allocated.

/CHAR Overwrites the existing character string in unit of character.
For two-byte characters, as a result of truncation or overwriting, are displayed within the correctly-displayable range.

Window number

Corresponds to the window number specified in Auxiliary Function 0509 as the window specification used to display the string. Select from 1to 8 (standard).

Row

Specifies the row in the window for displaying the string. Acceptable number is from 1 to 4, though it depends on the window size. If not specified, 1 is assumed.

Column

Specifies the column in the window for displaying the string. Acceptable number is from 1 to 78, though it depends on the window size. If not specified, 1 is assumed.

Background color

Selects the color of the background of the selected window. Acceptable numbers are from 0 to 15. If not specified, the background is white.

| No. | Color | No. | Color | No. | Color | No. | Color |
|---|---|---|---|---|---|---|---|
| 0 | Grey | 4 | Green | 8 | Pink | 12 | Navy |
| 1 | Blue | 5 | Pale Blue | 9 | White | 13 | Reddish Brown |
| 2 | Red | 6 | Yellow | 10 | Black | 14 | Deep Green |
| 3 | Orange | 7 | White | 11 | Cyan | 15 | Lavender |

Label color
Selects the color of the characters displayed.  Acceptable numbers are from 0 to 15 (See chart above).  If not specified, the characters are displayed in black.

Character string
Specifies the character string to display.  All strings after the first string are displayed on the next row starting at specified column.

**Explanation**
IFPWOVERWRITE command can be used only when the interface panel is available for use.  If the parameters are not specified, the last setting of that particular window is selected (for first time use, the above default values are set).  If the character string does not fit in one row, its display overflows to the next line (indenting to the selected column).  Strings that extend beyond the size of the window are not displayed.  Control characters in the string are displayed as blanks. Unlike IFPWPRINT command/ instruction, the rows other than the row specified for display are displayed unchanged as before executing IFPWOVERWRITE.

**Example**
The figures below show the screens displayed when executing IFPWPRINT and IFPWOVERWRITE command/ instruction from the screen showing “This is a pen”. The left figure is the figure after executing ”IFPWPRINT 1,3,1,7,10=”my”. The characters on lines 1, 2, and 4 disappear and line 3 shows ”my”.  On the other hand, the right figure shows the screen after executing “IFPWOVERWRITE 1,3,1,7,10=”my”.  The characters on lines 1, 2, and 4 are displayed as they were before executing the instruction and only line 3 has changed to ”my”.

## IFPLABEL position, "label 1"  "label 2", "label 3", "label 4"

**Function**

Sets and modifies the label of the icon at the specified position on the interface panel.

**Parameter**

Position

Specifies the display position on the interface panel of the icon to set/ modify the label.

Setting range: 1 – 112.

“Label 1”, “Label 2”…

Specifies the character string to display on the interface panel as the label of the specified icon. Omitted label will not be changed.

**Explanation**

Sets and modifies the label for the icons displayed on the interface panel. When a position with no icon set or when an icon with no label is specified, nothing occurs.

**Example**

```
>IFPLABEL 10, , ,"label 3"
```

Icon at position 10 on the interface panel is changed to “label 3” (label 1, label 2 is not changed because the parameters are omitted.)

## IFPTITLE page no., "title"

**Function**

Sets and modifies the title for the specified page of the interface panel.

**Parameter**

Page no.

Specifies the page of the interface panel to change the title. Setting range: 1- 4.

“Title”

Specifies the character string to display on the page as the title. Default setting is “Interface Panel”. When NULL string (“”) is specified, this default setting is also displayed.

**Explanation**

Sets and modifies the title for the specified page of the interface panel.

**Example**

| | |
|---|---|
| `>IFPTITLE 1, "Page 1"` | The title of the first page of the interface panel is changed to “Page 1”. |
| `>IFPTITLE 2, ""` | The title of the second page of the interface panel is changed to “Interface Panel”. |

## IFPDISP Page no.

**Function**

Displays the specified page of the interface panel.

**Parameter**

Page no.

Specifies the page number of the interface panel to be displayed.

**Explanation**

Executing this command allows the display of the specified page of the interface panel. To switch pages of the interface panel, choose <Prev Page> or <Next Page> in the teach pendant's pull-down menu.

**Example**

| | |
|---|---|
| >IFPDISP 2 | Displays the second page of the interface panel. |

## SETOUTDA channel No. = LSB, No. of bits, logic, max. voltage, min. voltage

Option

**Function**

Specifies the analog output environment including: channel number and LSB, number of bits and logic voltage for signal output, maximum and minimum voltage.

**Parameter**

Channel No.

Sets the analog output channel number. (Setting range: integers between 1 and 16; first 1TW/1UR board: 1 to 4; second 1TW/1UR board: 5 to 8; third 1TW/1UR board: 9 to 12; fourth 1TW/1UR board: 13 to 16)

LSB

Specifies the first analog output signal number for D/A conversion as an integer. Setting range: OUT1 to OUT125, 2001 to 2125, 3000 (first channel of 1TW/1UR board), 3001 (second channel of 1TW/1UR board), and subsequent 1TW/1UR board analog output channels can be specified. Default value is 3000. Previous setting remains in effect if not specified.

Number of bits

Sets the number of bits of analog output signals for D/A conversion as an integer. Setting range: 4 to 16 bits. Sets to 12 bits when 3000 onward [3000 + analog output channel number on 1TW/1UR board] is chosen for parameter LSB above. Default value is 8 bits. Previous setting remains in effect if not specified.

Logic

Sets the logic to either positive (1) or negative (0). Default value is 0 (negative). Previous setting remains in effect if not specified. Please set logic to positive for 1TW/1UR board.

Maximum voltage

Sets the maximum voltage of hardware (D/A output). Setting range: -15.0 to +15.0 V. Unit: V. Default value is 10 V. The value should be rounded off to the first decimal place. Previous setting remains in effect if not specified.

Minimum voltage

Sets the minimum voltage of hardware (D/A output). Setting range: -15.0 to +15.0 V. Unit: V. Default value is 0 V. The value should be rounded off to the first decimal place. Previous setting remains in effect if not specified.

See also 6.8 SETOUTDA, OUTDA instructions.

[ **NOTE** ]

1. Actual voltage output depends on the hardware used.
2. Error occurs if the value for maximum voltage is set lower than the minimum voltage.

## OUTDA voltage, channel number

Option

**Function**

Outputs the voltage at set conditions from the specified analog output channel.

**Parameter**

Voltage

Sets the analog output voltage. Setting range: -15.0 to +15.0 V. Unit: V. The value should be rounded off to the first decimal place.

Channel number

Specifies the analog output channel number. Setting range: integers between 1 and 16. If not specified, 1 is assumed.

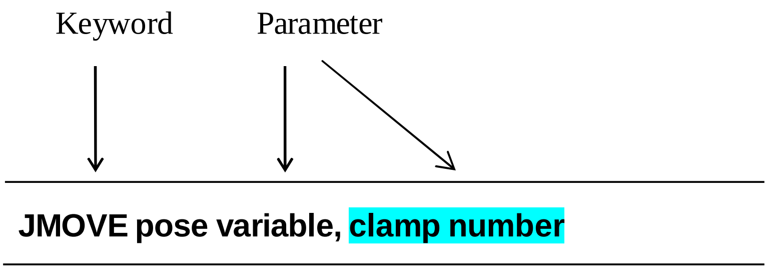
**[ NOTE ]**

Confirm that command voltage and actual output voltage are the same by setting output environment to correspond with the hardware settings via SETOUTDA instruction (command).

# 6 Program Instructions

This chapter groups the program instructions in the following categories, and describes each instruction in detail. A program instruction consists of a keyword expressing the instruction and parameter(s) following that key word, as shown in the example below.

**Example**



**JMOVE pose variable, clamp number**

Parameters marked with 🟦 can be omitted.

Always enter a space between the keyword and the parameter.

↵ in the examples represent the Enter key.

**JMOVE pose variable, clamp number**
**LMOVE pose variable, clamp number**

**Function**

Moves the robot to the specified pose.

JMOVE: Moves in joint interpolated motion.

LMOVE: Moves in linear interpolated motion.

**Parameter**

Pose variable

Specifies the destination pose of the robot. (Can be in transformation values, compound transformation values, joint displacement values or pose information functions.)

Clamp number

Specifies the clamp number to open or close at the destination pose. Positive number closes the clamp, and negative number opens it. Any clamp number can be set, up to the maximum number set via HSETCLAMP command (or auxiliary function 0605). If omitted, the clamp does not open or close.

**Explanation**

The robot moves in joint interpolated motion when JMOVE instruction is executed. The robot moves so that the ratios of distance traveled to the total distance are equal at all joints throughout the movement from the starting pose to the end pose.

The robot moves in linear interpolated motion when LMOVE instruction is executed. The origin of the tool coordinates (TCP) moves along a linear trajectory.

**Example**

| | |
|---|---|
| JMOVE #pick | Moves to pose described by joint displacement values “#pick” in joint interpolated motion. |
| LMOVE ref+place | Moves to the pose described by the compound transformation values “ref + place” in linear interpolated motion. |
| LMOVE #pick,1 | Moves to the pose described by joint displacement values “#pick” in linear interpolated motion. Upon reaching the pose, clamp 1 is closed. |

## DELAY time

**Function**

Stops the robot motion for the specified time.

**Parameter**

Time

Specifies in seconds for how long the robot motion is stopped.

**Explanation**

In AS system, DELAY instruction is considered as a motion instruction that “moves to nowhere”.

Even if the robot motion is stopped by DELAY instruction, all the program steps before the next motion instruction are executed before stopping.

**Example**

| | |
|---|---|
| DELAY 2.5 | Stops the robot motion for 2.5 seconds. |

## STABLE time

### Function

Postpones execution of next motion instruction until the specified time elapses after the axes coincide.　(Waits until the robot is stable.)

### Parameter

Time

Specifies in seconds for how long the robot motion is kept stable.

### Explanation

If coincidence of the axes fails while the robot is stopped by this command, the time is counted from when the axes coincide again.

## JAPPRO pose variable, distance
## LAPPRO pose variable, distance

**Function**

Moves in tool Z direction to a specified distance from the taught pose.

JAPPRO: Moves in joint interpolated motion.

LAPPRO: Moves in linear interpolated motion.

**Parameter**

Pose variable

Specifies the end pose (in transformation values or joint displacement values)

Distance

Specifies the offset distance between the end pose and the pose the robot actually reaches on the Z axis direction of the tool coordinates (in millimeters). If the specified distance is a positive value, the robot moves towards the negative direction of the Z axis. If the specified distance is a negative value, the robot moves towards the positive direction of the Z axis.

**Explanation**

In these commands, tool orientation is set at the orientation of the specified pose, and the position is set at the specified distance away from the specified pose in the direction of the Z axis of the tool coordinates.

**Example**

| | |
|---|---|
| JAPPRO place,100 | Moves in joint interpolated motion to a pose 100 mm away from the pose "place" in the direction of the Z axis of the tool coordinates. Pose "place" is described in transformation values. |
| LAPPRO place, offset | Moves in linear interpolated motion to a pose away from the pose "place", described in transformation values, at the distance defined by the variable "offset" in the direction of the Z axis of the tool coordinates. |

## JDEPART distance
## LDEPART distance

**Function**

Moves the robot to a pose at a specified distance away from the current pose along the Z axis of the tool coordinates.

JDEPART : Moves in joint interpolated motions.

LDEPART : Moves in linear interpolated motions.

**Parameter**

Distance

Specifies the distance in millimeters between the current pose and the destination pose along the Z axis of the tool coordinates. If the specified distance is a positive value, the robot moves “back” or towards the negative direction of the Z axis. If the specified distance is a negative value, the robot moves “forward” or towards the positive direction of the Z axis.

**Example**

| | |
|---|---|
| JDEPART 80 | The robot tool moves back 80 mm in –Z direction of the tool coordinates in joint interpolated motion. |
| LDEPART 2∗offset | The robot tool moves back 2∗offset (200 mm if offset = 100) in –Z direction of the tool coordinates in linear interpolated motion. |

## HOME home pose number

**Function**

Moves in joint interpolated motion to pose defined as HOME or HOME2.

**Parameter**

Home pose number

Specifies the home pose number (1 or 2). If omitted, HOME 1 is selected.

**Explanation**

Two home poses can be set (HOME 1 and HOME 2). This instruction moves the robot to one of the home poses in joint interpolated motion. The home pose should be defined beforehand using the SETHOME or SET2HOME command/ instruction. If the home pose is not defined, the null origin (all joints at 0°) is assumed as the home pose.

**Example**

| | |
|---|---|
| HOME | Moves to the home pose defined by SETHOME command/ instruction in joint interpolated motion. |
| HOME 2 | Moves to the home pose defined by SET2HOME command/ instruction in joint interpolated motion. |

## DRIVE joint number, displacement, speed

**Function**

Moves a single joint of the robot.

**Parameter**

Joint number

Specifies the joint number to move. (In a six-joint robot, the joints are numbered 1 to 6, starting from the joint furthest from the tool mounting flange.)

Displacement

Specifies the amount to move the joint, as either a positive or negative value.

The unit for this value is the same as the value that describes the pose of the joint; i.e. if the joint is a rotational joint, the value is expressed in degrees (°), and if the joint is a slide joint, the value is expressed in distance (mm).

Speed

Specifies the speed for this motion. As in regular program speed, it is expressed as a percentage of the monitor speed. If not specified, 100 % of the monitor speed is assumed.

**Explanation**

This instruction moves only one specified joint.

The motion speed for this instruction is combination of the speed specified in this instruction and the monitor speed. The program speed set in the program does not affect this instruction.

**Example**

| | |
|---|---|
| DRIVE 2,-10,75 | Moves joint 2 (JT2) –10° from the current pose. The speed is 75 % of the monitor speed. |

**DRAW** **X translation, Y translation, Z translation**
**X rotation, Y rotation, Z rotation, speed**

## TDRAW X translation, Y translation, Z translation, X rotation, Y rotation, Z rotation, speed

**Function**

Moves the robot in linear movement from the current pose and at the specified speed, the distance specified in the direction of the X, Y, Z axes and rotates the specified amount around each axis. DRAW instruction moves the robot based on the base coordinates, TDRAW instruction moves the robot based on the tool coordinates.

**Parameter**

X translation

Specifies the amount to move on the X axis in mm. If not specified, 0 mm is entered.

Y translation

Specifies the amount to move on the Y axis in mm. If not specified, 0 mm is entered.

Z translation

Specifies the amount to move on the Z axis in mm. If not specified, 0 mm is entered.

X rotation

Specifies the amount to rotate around the X axis in deg. Acceptable range is less than ±180°. If not specified, 0 deg is entered.

Y rotation

Specifies the amount to rotate around the Y axis in deg. Acceptable range is less than ±180°. If not specified, 0 deg is entered.

Z rotation

Specifies the amount to rotate around the Z axis in deg. Acceptable range is less than ±180°. If not specified, 0 deg is entered.

Speed

Specifies the speed in %, mm/s, mm/min, cm/min, or s. If not specified, the robot moves at the program speed.

**Explanation**

The robot moves from the current pose to the specified pose in linear movement.

**Example**

| | |
|---|---|
| DRAW 50,,-30 | Moves from the current pose in linear motion 50 mm in the direction of the X axis and –30 mm in the direction of the Z axis of the base coordinates. |

## ALIGN

**Function**

Moves the Z axis of the tool coordinates to be parallel with the closest axis of the base coordinates.

**Explanation**

In each application, if the reference motion direction is set along the tool Z direction, DO ALIGN enables easy alignment of the tool direction to the base coordinates before teaching the pose data.

## HMOVE pose variable, clamp number

**Function**

Moves the robot to the specified pose. The robot moves in hybrid motion: major axes in linear interpolation, and the wrist joints in joint interpolation.

**Parameter**

Pose variable

Specifies the destination of the robot motion. (Can be in transformation values, compound transformation values, joint displacement values or pose information functions.)

Clamp number

Specifies the clamp number to open or close at the destination pose. Positive number closes the clamp, and negative number opens it. Any clamp number can be set up to the maximum number set via HSETCLAMP command (or the auxiliary function 0605). If omitted, the clamp does not open or close.

**Explanation**

This instruction moves the robot in linear interpolated motion. The origin of the tool coordinates draws a linear trajectory. However, the wrist joints move in joint interpolation. This instruction is used when the robot is to be moved in linear motion but the angles of the wrist joints change greatly between the beginning and end of the motion.

## XMOVE mode pose variable TILL signal number

**Function**

Moves the robot towards the specified pose in linear movement, stops motion when the specified signal condition is set even if the pose has not been reached, and skips to the next step.

**Parameter**

Mode

(Not specified)

Monitors for the rising or trailing edge of the specified input signal. Positive signal number monitors rising edge, and negative number monitors trailing edge.

/ERR (Option)

Returns an error if the signal condition is already set when the monitoring starts.

/LVL (Option)

Immediately skips to the next step if the signal condition is already set when the monitoring starts.

Pose variable

Specifies the destination pose of the robot motion. (Can be in transformation values, compound transformation values, joint displacement values or pose information functions.)

Signal number

Specifies the number of external input signal or internal signal.

Acceptable signal numbers

| | |
|---|---|
| External input signal | 1001 to actual number of installed signals or 1256 (the smaller of the two). |
| Internal signal | 2001 to 2960 |

[ **NOTE** ]

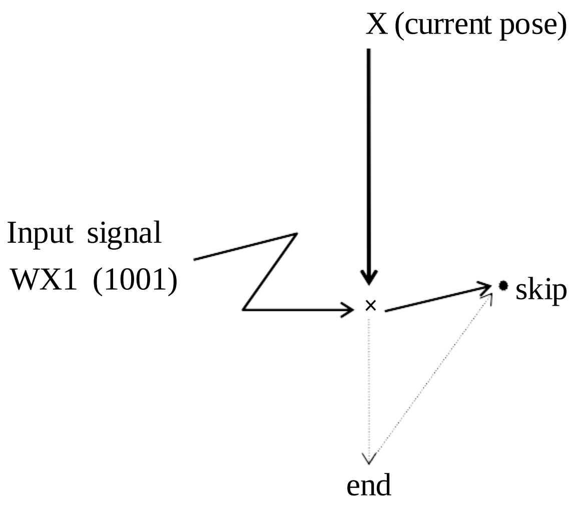
When monitoring the rising and trailing edge of the signal, the program branches only when there is a change in the signal status. Therefore, if the rising edge of signal is monitored and the signal is ON at the time XMOVE is executed, the program will not be interrupted until that signal turns OFF and then ON again.

The input signal should be stable for at least 50 msec for accurate monitoring.

**Example**

```
XMOVE   end   TILL   1001
LMOVE  skip
```

Moves from the current pose to pose “end” in linear motion. As soon as the input signal 1001 is turned ON, the program execution skips to the next step (LMOVE skip) even if the robot has not reached “end”.



**C1MOVE pose variable, clamp number**
**C2MOVE pose variable, clamp number**

Option

**Function**

Moves the robot to the specified pose following a circular path.

**Parameter**

Pose variable

Specifies the destination of the robot motion. (Can be in transformation values, compound transformation values, joint displacement values or pose information functions.)

Clamp number

Specifies the clamp number to open or close at the destination pose. Any clamp number can be set, up to the maximum number set via HSETCLAMP command. Positive number closes the clamp, and negative number opens it. If omitted, the clamp does not open or close.

**Explanation**

C1MOVE instruction moves to a point midway on the circular trajectory, C2MOVE instruction moves to the end of the trajectory.

To move the robot in a circular interpolated motion, three poses must be taught. The three poses differ in C1MOVE and C2MOVE instructions.

| | |
|---|---|
| C1MOVE : | 1. Pose of the latest motion instruction. |
| | 2. Pose to be used as the parameter of C1MOVE instruction. |
| | 3. Pose of the next motion instruction. (C1MOVE or C2MOVE instruction) |
| C2MOVE : | 1. Pose of the latest C1MOVE instruction. |
| | 2. Pose of the motion instruction before C1MOVE instruction. |
| | 3. Pose of C2MOVE instruction. |

**[ NOTE ]**

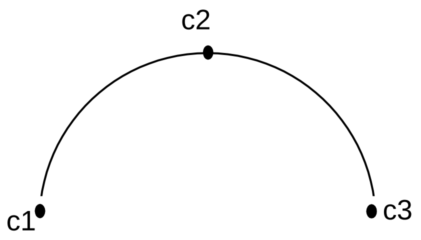
The following motion instructions are needed before the C1MOVE instruction:
ALIGN, C1MOVE, C2MOVE, DELAY, DRAW, TDRAW, DRIVE, HOME, JMOVE, JAPPRO, JDEPART, LMOVE, LAPPRO, LDEPART, STABLE, XMOVE

C1MOVE instruction must be followed by C1MOVE or C2MOVE instruction.

C1MOVE instruction must precede a C2MOVE instruction.

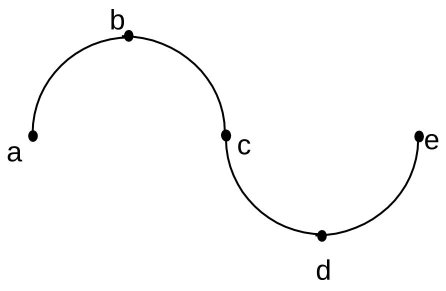
**Example**

```
JMOVE   c1
C1MOVE c2
C2MOVE c3
```

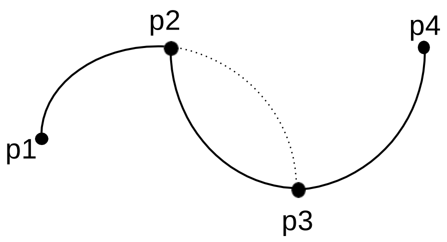


The robot moves in joint interpolated motion to c1 and then moves in a circular interpolated motion following the arc created by c1, c2, c3.

```
JMOVE   #a      ⎫
C1MOVE #b       ⎬ arc a,b,c
C2MOVE #c       ⎭
C1MOVE #d       ⎫ arc c,d,e
C2MOVE #e       ⎭
```



```
LMOVE   #p1     ⎫ arc p1,p2,p3
C1MOVE #p2      ⎭
C1MOVE #p3      ⎫ arc p2,p3,p4
C2MOVE #p4      ⎭
```



## 6.2 Speed and Accuracy Control Instructions

| | |
|---|---|
| SPEED | Sets the motion speed (program speed). |
| MON_SPEED | Sets the monitor speed. |
| ACCURACY | Sets the accuracy range. |
| ACCEL | Sets the acceleration. |
| DECEL | Sets the deceleration. |
| BREAK | Holds execution of the next step until the current motion is completed. |
| BRAKE | Stops the current motion and skips to the next step. |
| BSPEED | Sets the block speed. (Option) |
| REFFLTSET | Specifies the moving average for robot command values. |
| REFFLTRESET | Resets the robot’s moving average span. |
| FFSET | Sets the speed/ acceleration feed forward gain. |
| FFRESET | Resets the robot speed/ acceleration feed forward gain. |

## SPEED speed, rotational speed, ALWAYS

**Function**

Specifies the robot motion speed.

**Parameter**

Speed

Specifies the program speed. Usually it is specified in percentages between 0.01 and 100 (%). Absolute speed can be set by specifying the speeds with these units: MM/S and MM/MIN. The unit S (seconds) specifies the motion time. The input range for motion time is 0.1 S to 3601 S. If the unit is omitted, it is read as percent (%).

Rotational speed (Option)

Specifies the rotational speed of the tool orientation in linear and circular interpolated motions. Usually it is specified in percentages between 0.01 and 100 (%). Absolute speed can be set by specifying the speed with these units: DEG/S and DEG/MIN. If the unit is omitted, it is read as percent (%). If this parameter is omitted, the rotational speed is set at 100%.

ALWAYS

If this parameter is entered, the speed set in this instruction remains valid until the next SPEED instruction is executed. If not entered, the speed is effective only for the next motion instruction.

**Explanation**

The actual speed of the robot motion is determined by the product of the speed specified by this instruction and the monitor speed specified by the SPEED command or MON_SPEED instruction (Monitor speed × Program speed). However, full speed is not guaranteed in cases such as below:

1. when the distance between the two taught poses is too short,
2. when a linear motion exceeding the maximum speed of axis rotation is taught.

The motion speed is determined differently in joint interpolated motion and linear movement. In joint interpolated motion, the motion speed is determined as a percentage of the maximum speed of each axis. In linear movement, the motion speed is determined as a percentage of the maximum speed at the origin of the tool coordinates.

When the speed is specified in distance per unit time or in seconds, the speed in linear movement at the origin of the tool coordinates is set. When moving in joint interpolated motions, set the speed in percent. (Even if the speed is set in absolute speed or in motion time, the robot will not move in the set speed. Instead, the speed is processed as a percentage of the given value to the maximum speed.)

The absolute speed expressed in values with MM/ S and MM/MIN, and time specified speed expressed in values with S, describe the speed when the monitor speed is 100%. If the monitor speed is decreased, these speeds decrease in the same proportion.

**[ NOTE ]**

Even if the product of program speed and the speed set by SPEED command or MON_SPEED instruction (monitor speed) exceeds 100% the actual motion speed does not exceed 100%.

The rotational speed cannot be set without the rotational speed control option ON. If the option is not ON, error occurs.

**Example**

The speed is set as follows when the monitor speed is 100%:

| | |
|---|---|
| SPEED 50 | Sets the speed of the next motion to 50% of the maximum speed. |
| SPEED 100 | Sets the speed of the next motion to 100% of the maximum speed. |
| SPEED 200 | Sets the speed of the next motion to 100% of the maximum speed (speed over 100% is considered 100%). |
| SPEED 20MM/S ALWAYS | The speed of the origin of the tool coordinate (TCP) is set at 20mm/sec until it is changed by another SPEED instruction (when the monitor speed is 100%). |
| SPEED 6000 MM/MIN | Sets the speed of the next robot motion to 6000 mm/min. (The speed of linear motion of the origin of the tool coordinates). |
| SPEED 5 S | Sets the speed of the next robot motion so that the destination is reached in 5 seconds. (The speed of linear motion of the origin of the tool coordinates). |

## **MON_SPEED** monitor speed

**Function**

Sets or changes the monitor speed.

**Parameter**

Monitor speed

Specifies the speed to be set or changed as percentage of the maximum speed (unit in %). It is the normal maximum speed if this value is 100, and 1/2 of the maximum speed if it is 50. Percentage of the range up to 99999 can be specified, if the speed limit release option is enabled.

**Explanation**

The speed of the robot is determined by the product of the speed set by this program instruction and the speed set by the SPEED instruction.

For example, if the monitor speed has a value of 50 and program speed is set at 60, the maximum speed of the robot will be 30%.

[**NOTE**]

If the sum of the speed specified by this program instruction and the speed specified by SPEED command or MON_SPEED instruction exceeds 100%, motion speed of the robot will be forcibly set at 100%. (Only when speed limit option is released.)

The monitor speed is automatically set to 10% in default settings.

Robot operating instructions that are already running will not be affected by this instruction. The newly set speed will be effective after the current or the next motions are completed.

**Example**

When the program speed is 100%:

| | |
|---|---|
| `>MON_SPEED 30 ↵` | Robot speed is set to 30% of the maximum speed. |
| `>MON_SPEED 50 ↵` | Robot speed is set to 50% of the maximum speed. |
| `>MON_SPEED 100 ↵` | Robot speed is set to 100% of the maximum speed. |

## ACCURACY distance ALWAYS FINE

**Function**

Sets the accuracy when determining the robot pose.

**Parameter**

Distance

Specifies the distance of accuracy range in millimeters.

ALWAYS

If this parameter is entered, the accuracy setting remains valid until the next ACCURACY instruction is executed. If not entered, the accuracy setting is valid only for the next motion instruction.

FINE

If this parameter is entered, the robot pose is determined only when the current values match the taught pose. If omitted, the pose is determined as if the command value matches the taught pose.

**Explanation**

When the parameter ALWAYS is entered, all the proceeding motions are controlled by the accuracy set by this instruction. The default accuracy setting is 1 mm.

There is a limit to the effect of the accuracy setting, since in AS system the accuracy check is not started until the robot decelerates as it approaches the taught pose. (See also 4.5.4 Relation between CP Switch and ACCURACY, ACCEL, DECEL Instructions.)

**[ NOTE ]**

When the accuracy is set at 1 mm, the robot sets the pose after each motion instruction, coming to a pause in between the motion segments. To assure CP motion, set the accuracy range greater.

Setting the accuracy range too small may result in non-coincidence of the axes.

The accuracy set by this instruction is not the accuracy for repetition but for positioning the robot; therefore do not set values of 1 mm or less.

**Example**

| | |
|---|---|
| ACCURACY 10 ALWAYS | The accuracy range is set at 10 mm for all motion instructions after this instruction. |

## ACCEL acceleration ALWAYS
## DECEL deceleration ALWAYS

**Function**

Sets the acceleration (or deceleration) of the robot motion.

**Parameter**

Acceleration (ACCEL) / deceleration (DECEL)
Specifies the acceleration or deceleration in percentages of the maximum acceleration (deceleration). Acceptable range is from 0.01 to 100. Values over this limit are assumed as 100, values below the limit are assumed as 0.01.

ALWAYS
If this parameter is entered, the acceleration (or deceleration) here is valid until the next ACCEL (or DECEL) instruction. If not entered, this instruction affects only the next motion instruction.

**Explanation**

ACCEL instruction sets the acceleration when the robot starts a motion as a percentage of the maximum acceleration. DECEL instruction sets the deceleration when the robot is at the end of a motion as a percentage of the maximum deceleration.

**Example**

| | |
|---|---|
| ACCEL 80 ALWAYS | The acceleration is set at 80% for all motions after this instruction. |
| DECEL 50 | The deceleration for the next motion instruction is set at 50%. |

## BREAK

### Function

Holds execution of the next step in the program until the current robot motion is completed.

### Explanation

This instruction has the following two effects:

1. Holds the execution of the program until the robot reaches the destination of the current motion instruction.
2. The CP motion from the current motion to the next motion is interrupted. The robot comes to a stop in between the motion segments.

## BRAKE

**Function**

Stops current robot motion.

**Explanation**

Stops current robot motion immediately and skips to the next step in program.

## BSPEED speed

Option

**Function**

Sets the robot's motion speed (block speed). The robot motion speed is calculated by monitor speed × program speed × block speed.

**Parameter**

Speed

Sets the speed (acceptable range: 1 to 1000%). The speed set by this instruction is valid until the next BSPEED instruction is executed.

**Explanation**

The robot motion speed is calculated by monitor speed × program speed × block speed. However, the total speed cannot exceed 100%. Values up to 1000 can be entered for each speed, but if the total speed exceeds 100%, it is automatically cut down to 100%. For example, if the monitor speed is 100% and the program speed 50%, the motion speed is calculated by 100%×50%×block speed. If the block speed is less than 200%, the speed varies following the result of the above expression, but if it is over 200%, the motion speed always becomes 100%.

1. When the program selection is reset, and a new program is executed (e.g. via EXECUTE command or by program selection via the teach pendant), the block speed is set at the default value of 100%. When the program is selected externally, the block speed is set at the default value if the program is selected by external program reset, but not with RPS and JUMP signals.

2. Note that the robot may not move in the specified program speed if the program is not executed from the beginning of the program or when the steps are skipped. In the example below, the robot is stopped while in step 3 and the motion is resumed after jumping to step 25. Then, the block speed at step 25 will be the speed of block 1.

```
Step 1     BSPEED  block1          ; Sets the speed for block 1.
Step 2     Joint  Speed 9……
Step 3     Linear  Speed 9……
   :
Step 12    BSPEED  block2          ; Sets the speed for block 2.
Step 13    Joint  Speed 9……
Step 14    Linear  Speed 9……
   :
Step 24    BSPEED  block3          ; Sets the speed for block 3.
Step 25    Joint  Speed 9……
Step 26    Linear  Speed 9……
```

**Example**

Write the program as follows so that the speed is changed by 4 bits from an external signal.

```
a=BITS(first signal for external speed selection,4)
BSPEED block1[a]
```

The following program enables selecting speed from an external device:

```
BSPEED block1                              ; Sets the default value for the block.
IF SIG(External_speed ON)THEN              ; Determines if external speed selection is
                                             enabled.
a=BITS(first signal for external speed selection,4)  ; Acquires the number used for external
                                                       selection
IF(a<11)THEN                               ; Setting not possible if a is 11+
BSPEEDblock11[a]                           ; Sets the selected block speed.
END
END
Joint   Speed 9……                         ; Moves in selected block speed.
Joint  Speed 9……
```

Real number variable "block 1" must be defined in advance.

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

For example, if the first signal for external program selection is 1010, and the signals are inputs as:

```
1010･･･OFF
1011･･･ON
1012･･･OFF
1013･･･OFF
```

then a = 2, therefore block 11[2] is chosen and the motion speed becomes 30%.

## REFFLTSET joint value moving average span, position moving average span, orientation moving average span, signal moving average span

**Function**

Specifies the moving average for robot command values. This instruction is valid only when the moving average option for command value is enabled.

**Parameter**

Joint value moving average span

Specifies the moving average span for the joint angles when the robot is moving in joint interpolation motion. Unit: [ms]. Acceptable range: integer between 1 through 254. The specified value is rounded up depending on the AS system control cycle. (This is the same for parameters 2-4).

Position moving average span

Specifies the moving average span for position values when the robot is moving in linear interpolation or circular motion. Unit: [ms]. Acceptable range: integer between 1 through 254. When omitted, the same value as joint value moving average span is set.

Orientation moving average span

Specifies the moving average span for orientation values when the robot is moving in linear interpolation or circular motion. Unit: [ms]. Acceptable range: integer between 1 through 254. When omitted, the same value as position moving average span is set.

Signal moving average span (Option)

Specifies the moving average span for signal outputs. Unit: [ms]. Acceptable range: integer between 1 through 254. When omitted, this parameter value is set by multiplying the ratio between the default value of joint value moving average span and its current setting to the default value of the signal moving average span.

**[ NOTE ]**

The relative relation between the above four parameters are balanced for the default setting. Therefore, when making any modifications, it is usually only necessary to specify the parameter for the joint value moving average span. This way, the relative relation between the parameters will be kept adequate.

The default and current values for each parameter can be checked via REFFLTSET_STATUS command.

**Explanation**

Moving average span is set to make the robot reach the specified pose smoothly. When this span is set longer, the robot's vibration is reduced and the robot makes a smoother motion. However, the cycle time will also become longer and there is a tendency that the robot takes a greater shortcut than the taught path.

On the other hand, when the span is set shorter, the cycle time is shortened and the robot follows a trajectory that is more precise to the taught path. However, the robot vibration tends to be greater.

[ **NOTE** ]

Normally do not use this instruction, because using this instruction changes the dynamic characteristics. When using this instruction, follow the below procedure:

- Gradually change the values, in about 8 ms increments, and confirm the robot motion after making the changes.
- When checking the robot motion, start at a low monitor speed of about 20% and gradually raise the speed.

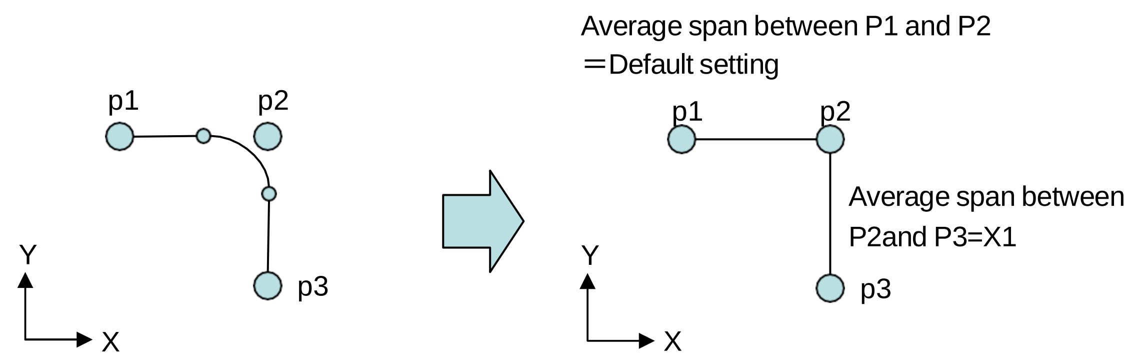
[ **NOTE** ]

The values set by executing this instruction apply to all robot motions, as is with SPEED ALWAYS instruction. To reset to the default value, use REFFLTRESET instruction.

The continuous path motion between the current motion and the next motion instruction is interrupted when REFFLTSET instruction is executed. That is, the two motions will not be followed in a one consecutive motion, but the robot will stop once at the end of the first motion before entering the second motion.

**Example**

| | |
|---|---|
| REFFLTSET 64 | Sets the robot's joint value, position, and orientation average span to 64 ms. The signal average span is set to the default value. |
| REFFLTSET 64, 64, 64, 32 | Sets the robot's joint value, position, and orientation average span to 64 ms and sets the signal average span to 32 ms. |



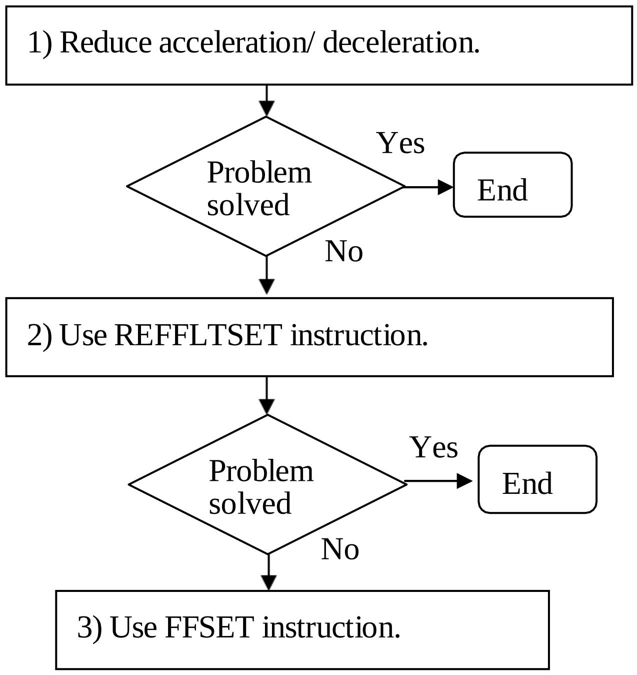
```
Robot motion program
JMOVE   p1
LMOVE   p2
LOMVE   p3
```

```
Robot motion program
JMOVE  p1
LMOVE  p2
REFFLTSET  X1
LOMVE  p3
```

Take the countermeasures for vibration following the below procedure:

1) Reduce the vibration by reducing the acceleration and deceleration.
   Use AS instructions ACCEL/DECEL to change them.
   For block teaching, change them via Aux. 0301 Acceleration/ Deceleration setting. (Aux.0301 can be used only when [ACCEL and DECEL] setting in Aux. 0399 is set to [Enable].)

   When the vibration is not reduced or the cycle time is too long, reset them to the original setting and perform the next adjustment.


1) Reduce acceleration/ deceleration.
Problem solved
Yes
End
No
2) Use REFFLTSET instruction.
Problem solved
Yes
End
No
3) Use FFSET instruction.

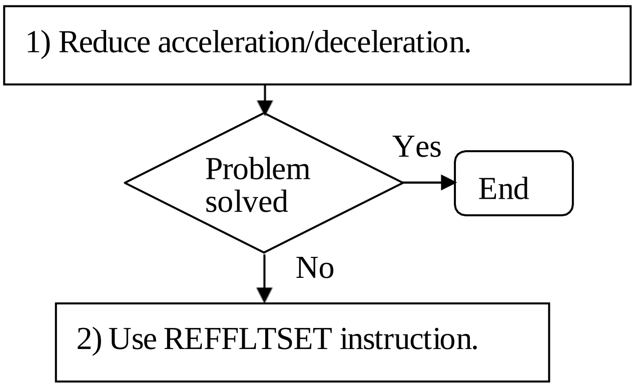
2) Smooth the robot motion via REFFLTSET instruction.
   Set the moving average span larger via REFFLTSET instruction.
   This is effective when the section where the average span is set larger (i.e. the section between REFFLTSET and REFFLTRESET) satisfies the below condition:

- There is more than one step.
- Few steps that require axis coincidence with the current pose.
- There are many steps with short distance below 50 mm.
- The accuracy setting is small in the motion step before REFFLTSET/REFFLTRESET instruction.

It is recommended to adjust the acceleration and deceleration together with the moving average span. Executing REFFLTSET instruction changes the robot's path, so be careful when confirming the motion.

When the vibration is not reduced or the cycle time is too long after using REFFLTSET instruction and changing the acceleration and deceleration, reset the REFFLTSET instruction and acceleration/ deceleration setting to the original setting and perform the next adjustment using FFSET instruction. FFSET instruction is explained later in this section.

Take the countermeasures for accuracy following the below procedure:


1) Reduce acceleration/deceleration.
Problem solved
Yes
End
No
2) Use REFFLTSET instruction.

1) Confirm the path accuracy at a low speed. Use AS instruction SPEED or speed instruction in block teaching to change the speed.

   If the path accuracy does not improve or the speed setting does not match with the application conditions, set the speed back to the original setting and perform the next adjustment.

2) Improve the path accuracy via REFFLTSET instruction. Use REFFLTSET instruction to reduce the moving average span and improve the path accuracy.

   The vibration tends to increase when this setting is done, so be careful when checking the robot motion.

## REFFLTRESET

### Function

Resets the robot's moving average span to the default value.

### Parameter

Resets to the default value the moving average span changed by REFFLTSET instruction.

As with the REFFLTSET instruction, the continuous path motion between the current motion and the next motion instruction is interrupted when this instruction is executed.   That is, the two motions will not be followed in a one consecutive motion, but the robot will stop once at the end of the first motion before entering the second motion.

### Example

In the sample program below, the moving average span from p2 to p3 is X1, and the moving average span between p3 to p1 is set at the default value.

```
Robot motion program
JMOVE p1
LMOVE p2
REFFLTSET X1
LOMVE p3
REFFLTRESET
JMOVE p1
```

**FFSET JT1 gain, JT2 gain, JT3 gain, JT4 gain, JT5 gain,**
**JT6 gain, JT7 gain, JT8 gain, JT9 gain**

**Function**

Sets the speed/ acceleration feed forward gain for when the robot starts moving.

**Parameter**

JT 1 gain, JT 2 gain, JT 3 gain, JT 4 gain, JT 5 gain, JT 6 gain, JT 7 gain, JT 8 gain, JT 9 gain
Specifies the speed/ acceleration feed forward gain for each axis in real values. Acceptable range: 0 -1 (valid to the third decimal place). Values for the axes other than JT1 can be omitted. The values will be set as follow when omitted.

| | |
|---|---|
| When parameters for robot axes are omitted: | Sets the same value as the setting for JT1. |
| When parameters for external axes are omitted: | The ratio between the default and current value of JT1 is multiplied to the default value for the omitted external axis. |

**[ NOTE ]**

The gains for the robot axes should be set equal to JT1. The default and current setting for each parameter can be checked via FFSET_STATUS command.

**Explanation**

When the speed/acceleration feed forward gain is set smaller, the robot’s vibration is reduced and the robot makes a smoother motion. However, the cycle time will also become longer and there is a tendency that the robot takes a greater shortcut than the taught path.

On the other hand, when the gain is set greater, the cycle time is shortened and the robot follows a trajectory that is more precise to the taught path. However, the robot vibration tends to be greater.

**[ NOTE ]**

Normally do not use this instruction, because using this instruction changes the dynamic characteristics. When using this instruction, follow the below procedure:

- Gradually change the values, in about 0.1 increments, and confirm the robot motion after making the changes.
- When checking the robot motion, start with a low monitor speed of about 20% and gradually raise the speed.

This instruction changes the dynamic characteristics; therefore confirm the robot speed does not change suddenly at the beginning and end of the modification.

**[ NOTE ]**

The values set by executing this instruction apply to all robot motions, as is with SPEED ALWAYS instruction. To reset to the default value, use FFRESET instruction.

**Example**

| | |
|---|---|
| FFSET 0.5 | Sets all the speed/ acceleration feed forward value to 0.5. |
| FFSET 0.5,0.49 | Sets the speed/ acceleration feed forward value to 0.5 for all the axes except for JT2, and sets the speed/ acceleration feed forward value to 0.5 for JT2 to 0.49. |

In the sample robot motion program below, the speed/acceleration feed forward gain is changed to 0.5 after axis coincidence in #p1.

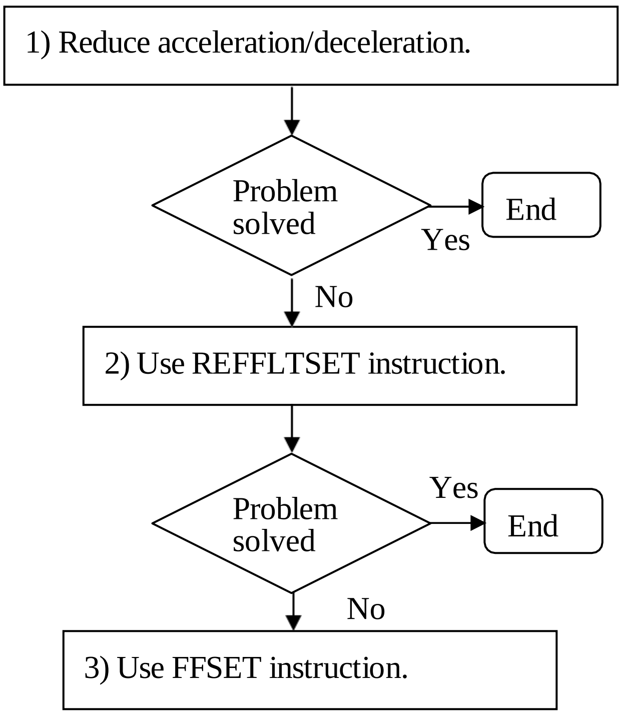
```
JMOVE  #p1
FFSET 0.5
JMOVE #p2
FFRESET
JMOVE #p3
```

Take the countermeasures for vibration following the below procedure:

1) Reduce the vibration by reducing the deceleration.
Follow the procedures explained for REFFLTSET instruction.

2) Smoothen the motion via REFFLTSET instruction.
Follow the procedures explained for REFFLTSET instruction.

3) Smoothen the motion via FFSET instruction.
Set the speed/acceleration feed forward gain via FFSET instruction to smoothen the robot motion.
Using FFSET instruction changes the robot path so carefully confirm the robot motion after executing the instruction.


1) Reduce acceleration/deceleration.
Problem solved
Yes
End
No
2) Use REFFLTSET instruction.
Problem solved
Yes
End
No
3) Use FFSET instruction.

## FFRESET

**Function**

Resets the robot speed/acceleration feed forward gain to the default value.

**Explanation**

Resets the robot speed/acceleration feed forward gain changed by FFSET instruction to the default value.　This works as setting the default value via FFSET instruction.

**Example**

In the sample robot motion program below, the speed/ acceleration feed forward gain is changed to 0.5 after axis coincidence in #p1.　The feed forward gain is reset to the default value after axis coincidence in #p2.

```
JMOVE  #p1
FFSET 0.5
JMOVE #p2
FFRESET
JMOVE #p3
```

## OPEN clamp number
## OPENI clamp number

**Function**

Opens robot clamps (outputs clamp open signal).

**Parameter**

Clamp number

Specifies the number of the clamp.    If omitted, 1 is assumed.

**Explanation**

This instruction outputs signals to the control valve of pneumatic hand to open the clamp.

With the OPEN instruction, the signal is not output until the next motion starts.

The timing for signal output using the OPENI instruction is as follows:

1. If the robot is currently in motion, the signal is output after that motion is completed.    If the robot is moving in CP motion, the CP motion is suspended (BREAK).
2. If the robot is not in motion, the signal is sent immediately to the control valve.

**Example**

| | |
|---|---|
| OPEN | The clamp open signal is sent to the control valve of clamp 1 when the robot starts the next motion. |
| OPENI 2 | The clamp open signal is sent to the control valve of clamp 2 as soon as the robot completes the current motion. |

## CLOSE clamp number
## CLOSEI clamp number

**Function**

Closes robot clamps (outputs clamp close signal).

**Parameter**

Clamp number

Specifies the number of the clamp. If omitted, 1 is assumed.

**Explanation**

This instruction outputs signals to the control valve of pneumatic hand to close the clamp.

With the CLOSE instruction, the signal is not output until the next motion starts.

The timing for signal output using the CLOSEI instruction is as follows:

1. If the robot is currently in motion, the signal is output after that motion is completed. If the robot is moving in CP motion, the CP motion is suspended (BREAK).
2. If the robot is not in motion, the signal is sent immediately to the control valve.

**Example**

| | |
|---|---|
| CLOSE 3 | The clamp close signal is sent to the control valve of clamp 3 when the robot starts the next motion. |
| CLOSEI | The clamp close signal is sent to the control valve of clamp 1 as soon as the robot completes the current motion. |

## RELAX clamp number
## RELAXI clamp number

**Function**

Turns OFF the pneumatic solenoid valves for both OPEN and CLOSE signals (turns the clamp signal OFF. In double solenoid specification, both clamp open and close signals are turned OFF).

**Parameter**

Clamp number

Specifies the clamp number. If omitted, 1 is assumed.

**Explanation**

With the RELAX instruction, the signal is not output until the next motion starts.

The timing for signal output using the RELAXI instruction is as follows:

1. If the robot is currently in motion, the signal is output after that motion is completed. If the robot is moving in CP motion, the CP motion is suspended (BREAK).
2. If the robot is not in motion, the signal is sent immediately to the control valve.

**OPENS clamp number**
**CLOSES clamp number**
**RELAXS clamp number**

**Function**

Turns ON/OFF the open and close signals of the pneumatic solenoid valves.

Clamp number

Specifies the number of the clamp. If omitted, 1 is assumed.

**Explanation**

This instruction is different from the OPEN/CLOSE/RELAX and OPENI/CLOSEI/RELAXI instruction in the following ways:

1. OPEN/CLOSE/RELAX instructions:
   The signal is output when the next motion starts.
2. OPENI/CLOSEI/RELAXI instructions:
   If the robot is in motion, the signal is output when that motion is completed. The CP motion is interrupted (BREAK).
3. OPENS/CLOSES/RELAXS instructions:
   The signal is output immediately after this instruction is executed.

This instruction is not affected by the PREFETCH.SIGINS switch.

**GUNON gun number, distance**
**GUNOFF gun number, distance**

Option

**Function**

Turns ON/OFF the gun signal and controls the gun output timing by the specified distance.

**Parameter**

Gun number

Specifies gun number 1 or 2.

Distance

Specifies the distance (in mm) to adjust the ON/OFF timing of the GUN. Negative value advances the timing, and the positive value delays the timing. If not specified, 0 is assumed.

**Explanation**

The gun signal is turned ON/ OFF when the motion instruction after the GUNON/GUNOFF instruction is executed. The output timing is determined by the distance specified in the instruction and the time set by GUNONTIMER/GUNOFFTIMER.

**Example**

| | |
|---|---|
| GUNON 2,100 | Turns ON the ON signal of gun 2 delaying the ON timing by 100 mm. |

## GUNONTIMER gun number, time
## GUNOFFTIMER gun number, time

Option

**Function**

Adjusts the timing of the gun output (timing at which gun is turned ON/OFF) by the specified time.

**Parameter**

Gun number

Specifies gun number 1 or 2.

Time

Specifies the time (in seconds) to adjust the ON/OFF timing of the GUN. Negative value advances the timing, and the positive value delays the timing. If not specified, 0 second is assumed.

**Explanation**

The adjustment time is determined by the environment of the gun system (e.g. the distance from the valve to the tip of the gun, the type of the paint, climate, etc.) so set the timing in the beginning of the program. To change the timing outside the program, use a variable for the time parameter (e.g. when the timing has to be changed due to paint color or viscosity).

This instruction only adjusts the output timing of the gun and does not actually turn the gun ON/OFF.

**Example**

| | |
|---|---|
| GUNONTIMER 1,-0.5 | Advances the timing of ON signal for gun 1 by 0.5 seconds. |

## 6.4 Configuration Instructions

| | |
|---|---|
| RIGHTY | Changes configuration so the robot arm resembles a person’s right arm. |
| LEFTY | Changes configuration so the robot arm resembles a person’s left arm. |
| ABOVE | Changes configuration so the elbow joint is in the above position. |
| BELOW | Changes configuration so the elbow joint is in the below position. |
| UWRIST | Changes configuration so the angle of JT5 has a positive value. |
| DWRIST | Changes configuration so the angle of JT5 has a negative value. |

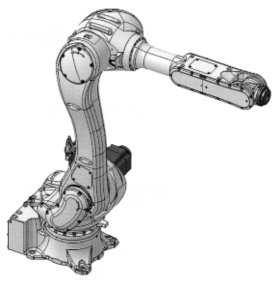
## RIGHTY
## LEFTY

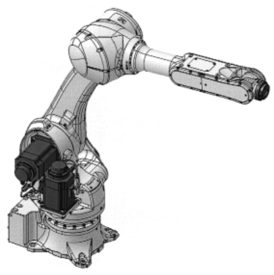
### Function

Forces a robot configuration change during the next motion so the robot arm is configured to resemble a person’s right (RIGHTY) or left (LEFTY) arm. The configuration may not be changed during a linear interpolated movement, or when the destination of the next motion is expressed in joint displacement values.

See also 11.7 Setting Robot Configuration.

### Example





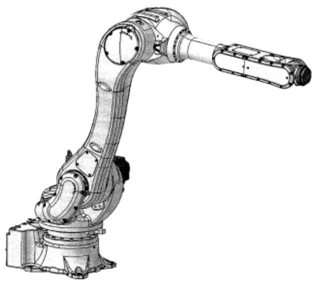
| | | |
|---|---|---|
| **ABOVE BELOW** | RIGHTY | LEFTY |

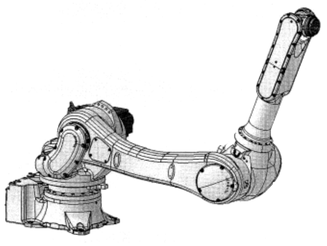
**Function**

Forces a robot configuration change during the next motion so the “elbow joint” (joint 3) is configured to resemble a person’s arm when the elbow is in above or below position relative to the wrist. The configuration may not be changed during a linear interpolated movement, or when the destination of the next motion is expressed in joint displacement values.

See also 11.7 Setting Robot Configuration.

**Example**


ABOVE


BELOW

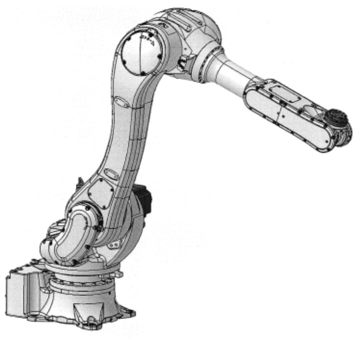
## UWRIST
## DWRIST

**Function**

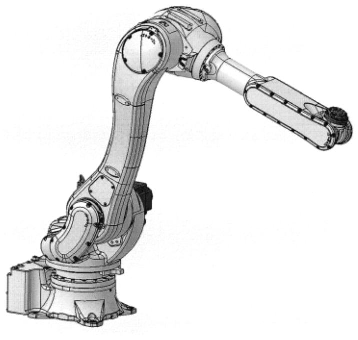
Forces a robot configuration change during the next motion so the angle of joint 5 (JT5) has a positive or negative value. The configuration may not be changed during a linear interpolated movement, or when the destination of the next motion is expressed in joint displacement values.

See also 11.7 Setting Robot Configuration.

**Example**



UWRIST
(Joint 5 is 90°)



DWRIST
(Joint 5 is -90°)*
**NOTE*** Joint 4 has rotated 180°.

## GOTO label IF condition

**Function**

Jumps to the program step with the specified label.

**Parameter**

Label

Specifies label of the program step to jump to. The label can be any character string within 15 letters (including alphanumeric characters, periods, underscores) that starts with an integer or alphabetical letter and is followed by a colon (:).

Condition

Specifies the condition to jump. This parameter and the keyword IF can be omitted. If omitted, the program jumps whenever the instruction is executed.

**Explanation**

Jumps to the step specified by the label. If the condition is specified, the program jumps when the condition is set. If the condition is not set, the execution goes on to the next step after this instruction.

Note that the label and the step number are different. Step numbers are assigned to all program steps automatically by the system. Labels are purposely given to program steps and are entered after the step number.

This instruction functions the same as the IF GOTO instruction when a condition is specified.

**Example**

| | |
|---|---|
| GOTO 100 | Jumps to label 100, there is no condition. If there is no step labeled 100, error occurs. |
| GOTO 200 IF n==3 | When variable "n" is equal to 3, then the program jumps to label 200. If not, the step after this step is executed. |

## IF condition GOTO label

**Function**

Jumps to the step with the specified label when the given condition is set.

**Parameter**

Condition

Specifies the condition in expressions, e.g. n = = 0, n>3, m + n<0.

Label

Specifies the label of the step to jump to (not the step number). The label must be within the same program.

**Explanation**

The program jumps to the step specified by the label, when the given condition is set. If the condition is not satisfied, the step after this instruction is executed.

If the specified label does not exist, error occurs.

**Example**

| | |
|---|---|
| IF n>3 GOTO 100 | If the value of integer variable "n" is greater than 3, then the program jumps to the step labeled 100. If n is not greater than 3, then the step after this step is executed. |
| IF flag GOTO 25 | If the value of integer number variable "flag" is not 0, the program jumps to the step labeled 25. If the value of the variable "flag" is equal to 0, the step after this step is executed. This is the same as writing: IF flag<>0 GOTO 25. |

## CALL program name

**Function**

Holds execution of the current program and jumps to a new program (subroutine). When the execution of the subroutine is completed, the processing returns to the original program and executes the step after the CALL instruction.

**Parameter**

Program name
Specifies the subroutine to execute.

**Explanation**

This instruction temporarily holds the execution of the current program and jumps to the first step of the specified subroutine.

**[ NOTE ]**

The same subroutine cannot be called from a robot control program and a PC program at the same time. Also, a subroutine cannot call itself.

Up to 20 programs can be held while subroutines are called.

**Example**

| | |
|---|---|
| CALL sub1 | Jumps to the subroutine named “sub1”. When the RETURN instruction in “sub1” is executed, the program execution returns to the original program and executes the program from the step after this CALL instruction. |

## RETURN

### Function

Ends execution of a subroutine and returns to the step after the CALL instruction in the program that called the subroutine.

### Explanation

This instruction ends execution of a subroutine and returns to the program that called that subroutine. If the subroutine is not called from another program (e.g. when the subroutine is executed by EXECUTE command) the program execution is ended.

At the end of the subroutine, the program execution returns to the original program even if there is no RETURN instruction. However, the RETURN instruction should be written as the last step of the subroutine (or at any place the subroutine is to be ended).

## WAIT condition

**Function**

Makes program execution wait until the specified condition is set (condition becomes TRUE).

**Parameter**

Condition

Specifies the stand-by condition. (real number expressions)

**Explanation**

This instruction holds execution of the program until the specified condition is set. CONTINUE NEXT command resumes the program execution before the condition is set (skips the WAIT instruction being executed).

**Example**

| | |
|---|---|
| WAIT SIG (1001, – 1003) | Holds execution of the program until external input signal 1001 (WX1) is ON and signal 1003(WX3) is OFF. |
| WAIT TIMER(1)>10 | Holds program execution until the value of timer1 is over 10 (seconds). |
| WAIT n>100 | Holds program execution until the value of variable "n" exceeds 100. (In this example, suppose variable "n" is a value that is counted up by a PC program or program interruptions.) |

## TWAIT time

**Function**

Holds program execution until the specified time elapses.

**Parameter**

Time

Specifies the time, in seconds, for how long the program execution is held.

**Explanation**

This instruction holds the program execution until the specified time elapses.

A TWAIT instruction in execution can be skipped using the CONTINUE NEXT command.

WAIT instruction can be used instead of the TWAIT instruction to gain the same result.

**Example**

| | |
|---|---|
| TWAIT 0.5 | Waits for 0.5 seconds. |
| TWAIT deltat | Waits until the value of variable “deltat” elapses. |

## MVWAIT value

### Function

Holds program execution until the remaining distance (or time) of the current motion becomes shorter than the specified distance (or time).

### Parameter

Value

Specifies the distance or time. The distance is expressed in millimeters (mm) and the time in seconds (S). If the unit is not specified, it is considered as millimeters.

### Explanation

This instruction is used to synchronize program execution with the robot motion. However, note that since this instruction monitors the remaining distance (or time) based on the command values, it may be different from the actual remaining distance (or time) due to response lag. When the robot is moving in joint interpolated motion, the distance specified and the actual distance may differ greatly. If the current motion is completed when this instruction is executed, the execution goes on to the next instruction without waiting. CONTINUE NEXT instruction can be used to skip the MVWAIT instruction while it is in execution.

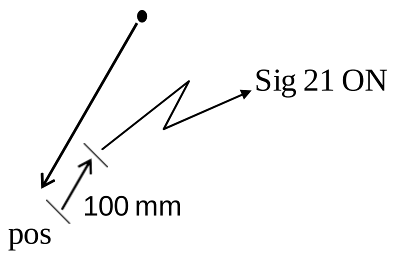
**[ NOTE ]**

MVWAIT instruction cannot be used in PC programs. Also, this instruction cannot be used with DO command.

### Example

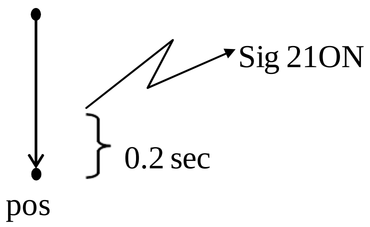
In the diagram below, the robot moves towards pose "pos", and when coming within 100 mm to "pos", the signal 21 is turned ON. This is true only when system switch PREFETCH.SIGINS is ON (reads the signal before axes coincidence) and the robot is within the accuracy range.

```
LMOVE pos
MVWAIT 100mm
SIGNAL 21
```



In the diagram below, the robot moves towards pose “pos”, and when the required time to reach “pos” becomes 0.2 seconds, signal 21 is turned ON. This is true only when PREFETCH.SIGINS is ON and the robot is in the accuracy range.

```
LMOVE pos
MVWAIT 0.2S
SIGNAL 21
```



## LOCK priority

**Function**

Changes the priority of the robot program currently selected on the stack.

**Parameter**

Priority

Specifies the priority in real numbers from 0 to 127.

**Explanation**

Normally, the priority of robot program is 0. The priority can be changed using this instruction. The greater the number the higher the priority will be.

**Example**

| | |
|---|---|
| LOCK 2 | Changes the priority to 2. |

## PAUSE

**Function**

Temporarily holds (pauses) the program execution.

**Explanation**

This instruction temporarily holds the program execution and displays a message on the terminal. Execution can be resumed using the CONTINUE command.

This instruction is convenient when checking a program. The values of the variables can be checked while the program is held by the PAUSE instruction.

## HALT

**Function**

Stops the program execution. The program cannot be resumed after this instruction is executed.

**Explanation**

Stops the program execution regardless of the remaining steps. A message is displayed on the terminal.

Program execution stopped by this instruction cannot be resumed using the CONTINUE command.

## STOP

**Function**

Terminates the current execution cycle.

**Explanation**

If there are cycles remaining to be completed, execution returns to the first step, otherwise execution ends. This instruction marks the end of the execution path and has a different effect than the HALT instruction.

If there are execution cycles remaining, execution continues with the first step of the main program* (even if STOP instruction was processed during execution of a subroutine or another interrupting program, execution returns to the main program).

**NOTE*** A main program is the program executed using the EXECUTE, STEP, PCEXECUTE commands. A subroutine is a program called from another program by CALL, ON or ONI instructions.

A RETURN instruction in a main program functions in the same way as a STOP instruction.

Program execution stopped by a STOP instruction cannot be resumed by CONTINUE command.

## SCALL string expression, variable

### Function

Jumps to the subroutine with the name given by the string expression.

### Parameter

String expression

Specifies the subroutine name in the form of a string expression.

Variable

If the subroutine call is executed normally, then the value 0 is assigned to this variable. If some abnormality occurred during the subroutine call, the error code (≠ 0) is assigned. If omitted, the execution comes to an error stop when an abnormality occurs in the subroutine call.

### Explanation

This instruction functions the same as the CALL instruction except that the program name is expressed as a string expression. (See CALL instruction).

### Example

```
$prog="sub1"
SCALL $prog                      Jumps to a subroutine named "sub1".

num=12
$temp1=$ENCODE(/I2,num)          Converts into a string expression, the real value
$temp2=""                        given to “num”, and jumps to the subroutine
                                 named “sub12”.
FOR i=1 to LEN($temp1)
$temp3=$MID($temp1,i,1)
IF $temp3<> "" THEN
$temp2=$temp2+$temp3
END
END
SCALL "sub"+$temp2
```

## ONE program name

**Function**

Calls the specified program when an error occurs.

**Parameter**

Program name

Specifies the name of the program to execute when an error occurs.

**Explanation**

This instruction calls the specified program when an error occurs. PC programs can be called too.

To return to the original program from the called program, RETURN (or RETURNE) instruction is used. RETURN instruction returns the execution to the step where the error occurred. RETURNE instruction returns the execution to the step after the error. (If neither RETURN nor RETURNE instruction exists within the program, the execution cycle stops at the end of the called program.)

Motion instructions cannot be used in the program called by ONE instruction.

If error occurs in the program called by ONE, the program execution stops there.

**[ NOTE ]**

As long as the main program containing the ONE instruction is in execution, the instruction is effective on errors in the subroutines, as well as the main program. When the main program ends execution, ONE becomes ineffective.

When an error arises, the Error lamp does not illuminate if a program is called by the ONE instruction.

## RETURNE

### Function

Returns to the step after the error.

### Explanation

This instruction is commonly paired with the ONE instruction. With ONE instruction, the program jumps to a subroutine when an error occurs. Then, the execution returns to the step after the error in the original program when the RETURNE instruction in the subroutine is executed.

## JUMP program name

**Function**

Ends the current program and moves on to a different program.

**Parameter**

Program name

Specifies the program to change to.

**Explanation**

This instruction ends the execution of the current program and moves to the first step of the specified program. After the execution of specified program is completed, it does not return to the original program. However, if this instruction were executed in a subroutine program called by CALL instruction, the execution returns to the next step in the source program after the program execution is completed. The execution cannot jump from a subroutine program to the source program. Error (E0121) “Cannot specify the jump source program as jump destination.” occurs.

## SJUMP program name, status variable

### Function

Ends the current program and moves on to a different program.

### Parameter

Program name

Specifies the program to change to in character string.

Status variable

When the program change is done normally, 0 is written. If not, the error code (≠0) is written. When omitted, the program execution comes to an error stop when switching is not done normally.

### Explanation

This instruction ends the execution of the current program and moves to the first step of the specified program by the character string. After the execution of specified program is completed, it does not return to the original program. However, if this instruction were executed in a subroutine program called by CALL instruction, the execution returns to the next step in the source program after the program execution is completed. The execution cannot jump from a subroutine program to the source program. Error (E0121) "Cannot specify the jump source program as jump destination." occurs.

## MON_TWAIT time

Option

**Function**

When the command value path constant move function (option) is valid, holds program execution until the specified time (seconds) times the ratio between monitor speed 100% and speed setting value (monitor speed or check speed) elapses.

**Parameter**

Time

Specifies the time, in seconds, for how long the program execution is held.

**Explanation**

When the command value path constant move function (option) is invalid, holds program execution until the specified time (seconds), same as in TWAIT instruction.

When the command value path constant move function (option) is valid, holds program execution until the specified time (seconds) times the monitor speed 100%/speed setting value (monitor speed check speed) elapses.

The wait time is as follows:
When monitor speed is set to 100% in repeat mode, the specified time (seconds).
When monitor speed is set to 10% in repeat mode, 10 times the specified time (seconds).
In check mode, the specified time (seconds) multiplied by the ratio between the maximum speed in straight linear motion and specified check speed.

However, even if the command value path constant move function (option) is valid, the program waits for the specified time ignoring the monitor speed or check speed in cases such as when no motion step in motion exist (i.e. this instruction is used at the beginning of the program), or in check once mode.

The MON_TWAIT instruction in execution can be skipped using CONTINUE NEXT instruction.

**Example**

When the command value path constant move function (option) is valid

| | |
|---|---|
| MON_TWAIT 0.5 | Waits for 0.5 seconds if monitor speed in repeat mode is 100%. |
| | Waits for 5 seconds if monitor speed in repeat mode is 10%. |

| | |
|---|---|
| MON_TWAIT deltat | Waits until the value of variable “deltat” elapses if monitor speed in repeat mode is 100%.<br>Waits until 10 times the value of variable “deltat” elapses if monitor speed in repeat mode is 10%. |

```
IF  logical expression THEN
program instructions(1)
ELSE
program instructions(2)
END
```

**Function**

Executes a group of program steps according to the result of a logical expression.

**Parameter**

Logical expression

Logical expression or real value expression. Tests if this value is TRUE (not 0) or FALSE(0).

Program instructions (1)

The program instructions entered here are executed if the above logical expression is TRUE.

Program instructions (2)

The program instructions entered here are executed if the above logical expression is FALSE.

**Explanation**

This control flow structure executes one of the two groups of instructions according to the value of the logical expression. The execution procedure is as follows:

1. Calculates the logical expression, and jumps to step 4 if the resulting value is 0 (FALSE).
2. Calculates the logical expression, and executes program instructions (1) if the resulting value is 1 (TRUE).
3. Jumps to 5.
4. If there is the ELSE statement, program instructions (2) is executed.
5. Continues program execution from the step after END.

[ **NOTE** ]

1. ELSE and END statements each must be entered in a line on its own.
2. The IF…THEN structure must end with END statement.

**Example**

In the example below, if n is greater than 5, the program speed is set at 10%, if not it is set at 20%.

```
21      IF n>5 THEN
22              sp=10
23      ELSE
24              sp=20
25      END
26      SPEED sp ALWAYS
```

The program below first checks the value of variable “m”. If “m” is not 0, the program checks the external input signal 1001(WX1) and displays a different message according to the status of the signal. In this example, the outer IF structure does not have an ELSE statement.

```
71      IF m THEN
72              IF SIG(1001) THEN
73                      PRINT"Input signal is TRUE"
74              ELSE
75                      PRINT"Input signal is FALSE"
76              END
77      END
```

```
WHILE  condition  DO
program instructions
END
```

**Function**

While the specified condition is TRUE, the program instructions are executed. When the condition is FALSE, the WHILE statement is skipped.

**Parameter**

Condition

Logical expression or real value expression. Checks if this value is TRUE (not 0) or FALSE (0).

Program instructions

Specifies the group of instructions to be executed when the condition is TRUE.

**Explanation**

This control flow structure repeats the given program steps while the specified condition is TRUE. The execution procedure is as follows:

1. Calculates the logical expression, and jumps to step 4 if the resulting value is 0 (FALSE).
2. Calculates the logical expression, and executes program instructions if the resulting value is 1 (TRUE).
3. Jumps to 1.
4. Continues program execution from the step after END.

**[ NOTE ]**

Unlike the DO structure, if the condition is FALSE, none of the program steps in the WHILE structure is executed.

When this structure is used, the condition must eventually change from TRUE to FALSE.

**Example**

In the following example, input signals 1001 and 1002 are monitored and robot motion is stopped based on their condition. When either of the signals from the two parts feeders changes to 0 (feeder is emptied), the robot stops and the execution continues from the step after the END statement (step 27 in this example).

If one of the feeders is empty at the time the WHILE structure begins (external input signal OFF=0), none of the steps in the structure is executed, and processing jumps to step 27.

```
.
.
.
WHILE SIG(1001,1002) DO
        CALL part1
        CALL part2
END
.
.
.
.
```

**DO**
**program instructions**
**UNTIL  logical expression**

**Function**

Creates a DO loop.

**Parameter**

Program instructions

These instructions are repeated as long as the logical expression is FALSE.

Logical expression

Logical expression or real value expression.  When the result of this logical expression changes to TRUE, execution of the program instructions in this structure is stopped.

**Explanation**

This control flow structure executes a group of program instructions while the given condition (logical expression) is FALSE.

The execution procedures are as follows:

1. Executes the program instructions.
2. Checks the value of the logical expression and if the result is FALSE, procedure 1 is repeated. If the result is TRUE, it jumps to procedure 3.
3. Continues program execution from the step after UNTIL statement.

The execution exits the DO structure when the value of the logical expression changes from FALSE to TRUE.

**[ NOTE ]**

Unlike the WHILE structure, the program instructions in the DO structure are executed at least once.

The program instructions between DO statement and UNTIL statement can be omitted. If there are no instructions, the logical expression after UNTIL is evaluated repeatedly. When the value of the logical expression changes to TRUE, then the execution exits the loop and goes on to the step after the DO structure.

The DO structure must end with an UNTIL statement.

**Example**

In the example below, the DO structure controls the following task: a part is picked up, and carried to the buffer. When the buffer becomes full, the binary input signal “buffer.full” is turned ON. When the signal turns ON, the robot stops and starts a different operation.

```
10      .
11      .
12      .
13      DO
14          CALL get.part
15          CALL put.part
16      UNTIL SIG(buffer.full)
17      .
18      .
19      .
```

```
FOR   loop variable = start value   TO   end value   STEP   step value
program instructions
END
```

**Function**

Repeats program execution.

**Parameter**

Loop variable

Variable or real value. This variable is first set at an initial value, and 1 is added each time the loop is executed.

Starting value

Real value or expression.   Sets the first value of the loop variable.

End value

Real value or expression. This value is compared to the present value of the loop variable and if the value of the loop variable reaches this value, the program exits the loop.

Step value

Real value or expression that can be omitted. This value is added or subtracted to the loop variable after each loop. Enter this parameter when using the STEP statement, unless the loop variable is to increment by 1. If step value is not specified, 1 is added to the loop variable. In this case, the STEP statement can be omitted too.

**Explanation**

This control flow structure repeats execution of the program instructions between the FOR and END statements. Loop variable is incremented by the given step value each time the loop is executed.

The execution procedures are as follows:

1. The start value is assigned to the loop variable.
2. Calculates the end value and the step value.
3. Compares the value of the loop variable with the end value.
   a. If the step value is positive, and the loop variable is greater than the end value, then jump to procedure 7.
   b. If the step value is negative and the loop variable is smaller than the end value, jump to procedure 7.

   In other cases, goes on to procedure 4.

4. Executes the program instructions after the FOR statement.
5. When the END statement is reached, the step value is added to the loop variable.
6. Returns to procedure 3.
7. Executes the program instructions after the END statement. (The value for the loop variable at the time of the comparison test at procedure 3 above does not change.)

**[ NOTE ]**

There must be an END statement for each FOR statement.

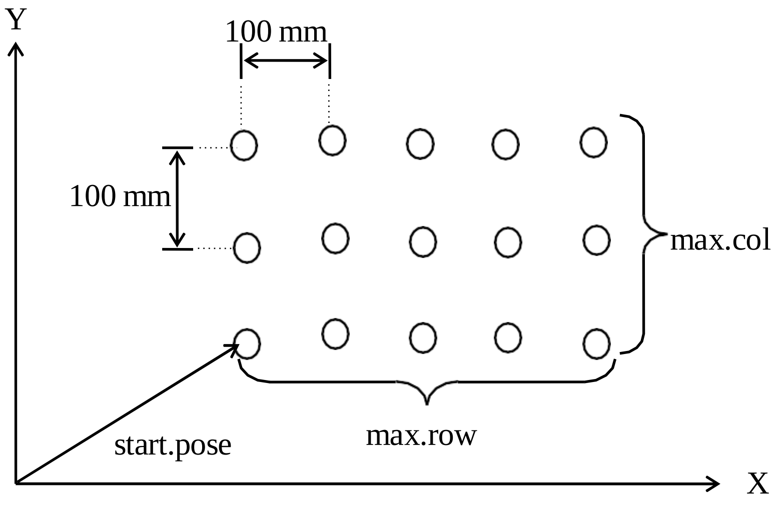
Beware that if the loop variable is greater than the end value (or less if the step value is negative) at the first check, none of the program instructions between FOR and END is executed.

The value for the number of loops (loop variable) must not be changed by other programming (operators, expressions, etc.) within the FOR loop.

**Example**

The subroutine "pick.place" picks up a part and places it on "hole". The parts are placed as shown in the figure below. (The pallet is placed parallel to X, Y axes of the world coordinates, and the distance between the parts is 100 mm.

```
FOR row = 1 TO max.row
POINT hole = SHIFT (start.pose BY (row-1)*100,0,0)
FOR col = 1 TO max.col
CALL pick.place
POINT hole = SHIFT(hole BY 0,100,0)
END
END
```



```
CASE    index variable    OF
VALUE    case number 1，……:
program instructions
```

**VALUE case number 2, ……:**
**program instructions**

**:**

**VALUE case number n, ……:**

```
program instructions
ANY   :
program instructions
END
```

**Function**

Executes the program according to a particular case number.

**Parameter**

Index variable

Real value variable or expression. Decides which CASE structure to execute according to the value of this variable.

Program instructions

Executes these program instructions when the value of the index variable equals one of the values after the VALUE statement.

**Explanation**

This structure enables the program to select from among several groups of instructions and to process the selected group. This is a powerful tool in AS language that provides a convenient method for allowing several alternatives within the program.

The execution procedure is as follows:

1. Checks the value of the index variable entered after the CASE statement.
2. Checks through the VALUE steps and finds the first step that includes the value equal to the value of the index variable.
3. Executes the instructions after that VALUE step.
4. Goes on to the instructions after the END statement.

If there is no value that matches the index variable, the program instructions after the ANY statement are executed. If there is not an ANY statement, none of the steps in the CASE structure is executed.

[ **NOTE** ]

ANY statement and its program instructions can be omitted.

ANY statement can be used only once in the structure. The statement must be at the end of the structure as shown in the example below.

The colon ":" after the ANY statement can be omitted. When entering the colon, always leave a space after ANY. Without a space, ANY: is taken as a label.

Both the ANY and END statements must be entered on their own line.

**Example**

In the program below, if the value of real variable x is negative, the program execution stops after the message is displayed. If the value is positive, the program is processed according to these 3 cases:

1. if the value is an even number between 0 and 10.
2. if the value is an odd number between 1 and 9.
3. if the value is a positive number other than the above.

```
    IF x<0 GOTO 10

    CASE x OF
    VALUE 0,2,4,6,8,10:
    PRINT "The number x is EVEN"
    VALUE 1,3,5,7,9:
    PRINT "The number x is ODD”
    ANY :
    PRINT "The number x is larger than 10"
    END

    STOP

10  PRINT "Stopping because of negative value"
    STOP
```

```
SCASE   index variable   OF
SVALUE   string_1, ……:
program instructions
SVALUE   string_2, ……:
```

**program instructions**

**:**

```
SVALUE   string_n, ……:
program instructions
ANY   :
program instructions
END
```

Option

**Function**

Executes program based on condition specified by the character string.

**Parameter**

Index variable

Specifies character string variable or expression. Decides which SCASE structure to execute according to character string of this variable.

Program instructions

Executes these program instructions when the string of the index variable equals one of the values after the SVALUE statement.

**Explanation**

Unlike CASE structure described before, the execution condition for SCASE structure is set as character string. See also CASE structure.

If there is no string that matches the string character, the program instructions after the ANY statement are executed. If there is not an ANY statement, none of the steps in the SCASE structure is executed.

[ **NOTE** ]

ANY statement and its program instructions can be omitted.

ANY statement can be used only once in the structure.  The statement must be at the end of the structure.

The colon “:” after the ANY statement can be omitted.  When entering the colon, always leave a space after ANY.  Without a space, ANY: is taken as a label.

Both the ANY and END statements must be entered on their own line.

**Example**

In the program below, if character string variable $str is equal to the string of $a+”c”, the program pc is executed.  If character string variable $str is equal to the string of $a+”g”, the program pg is executed.

```
SCASE $str OF
SVALUE $a+”c”:
CALL pc
SVALUE $a+”g”:
CALL pg
SVALUE $a+”c”:
END
```

## RESET

**Function**

Turns OFF all the external output signals.  This command does not have effect on signals used as dedicated signals, clamp signals and antinomy of multifunction OX/WX.

By using the optional setting, the signals used in the Interface Panel screen are not affected by this command. (Option)

## SIGNAL　signal number, ……

**Function**

Turns ON/OFF the specified external output signals (OX) or internal signals.

**Parameter**

Signal number

Selects the number of external output signal or internal signal.　Selecting a dedicated signal results in error.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001–2960 |

See also 5.7 SIGNAL monitor command.

## PULSE signal number, time

**Function**

Turns ON the specified external output signal or internal signal for the given period of time.

**Parameter**

Signal number

Selects the number of external output signal or internal signal. Selecting a dedicated signal results in error.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001–2960 |

Time

Sets for how long the signal is output (in seconds). If not specified, it is automatically set at 0.2 seconds.

See also 5.7 PULSE monitor command

## DLYSIG signal number, time

**Function**

Outputs the specified signal after the given time has passed.

**Parameter**

Signal number

Selects the number of the external output signal or internal signal. If the signal number is positive, the signal is turned ON; if negative, the signal is turned OFF Selecting a dedicated signal results in error.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001–2960 |

Time

Specifies the time to delay the output of the signal in seconds.

See also 5.7 DLYSIG monitor command.

## RUNMASK starting signal number, number of signals

### Function

Allows signals to be ON only while the program is executing. The signals can be turned ON using the SIGNAL, PULSE or DLYSIG command, but the signal turns OFF when the program execution stops (if this instruction is not used, the signals remain ON once they are turned ON).

### Parameter

Starting signal number

Specifies the number of the first external output signal or internal signal in the group of signals to mask. Entering a negative number cancels the mask function for that signal number and the signal does not become OFF when the program stops.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001–2960 |

Number of signals

Specifies how many signals are masked. If not specified 1 is assumed.

### Explanation

The signals selected by this instruction always turns OFF when the program execution stops. However, dedicated signals are not affected by this instruction.

If the program execution is interrupted, the masked signals turn OFF. When the program is resumed using the CONTINUE command, the signals return to the status they were in when the program was running. The same occurs with DO command or STEP command. (Restarting program via EXECUTE command nullifies the RUNMASK instruction.)

### Example

RUNMASK 5,2 Masks the external output signal 5 and the next signal 6, specified by 2 bits. While the program is running these signals can be turned ON by SIGNAL, PULSE, or DLYSIG command. They are turned OFF when the program execution stops.

## BITS starting signal number, number of signals = decimal value

**Function**

Arranges a group of external output signals or internal signals in a binary pattern. The signal states are set ON/OFF according to the binary equivalent of the decimal value specified.

**Parameter**

Starting signal number

Specifies the first signal to set the signal state.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001–2960 |

Number of signals

Specifies the number of signals to be set ON/OFF. The maximum number allowed is 16. To set more than 16 signals, use BITS32 instruction, explained next.

Decimal value

Specifies the decimal value used to set the desired ON/OFF signal states. The decimal value is transformed into binary notation and each bit of the binary value sets the signal state starting from the least significant bit. If the binary notation of this value has more bits than the number of signals, only the state of the given number of signals (starting from the specified signal number) is set and the remaining bits are ignored.

See also 5.7 BITS monitor command.

## BITS32 starting signal number, number of signals = value

**Function**

Arranges a group of external output signals or internal signals in binary pattern. The signal states are set ON/OFF according to the binary equivalent to the specified value.

**Parameter**

Starting signal number

Specifies the first signal to set the signal state.

Number of signals

Specifies the number of signals to be set ON/OFF. The maximum number allowed is 32.

Decimal value

Specifies the value used to set the desired ON/OFF signal states. The value is transformed into binary notation and each bit of the binary value sets the signal state. The least significant bit corresponds to the smallest signal number, and so on. If the binary notation of this value has more bits than the number of signals, only the state of the given number of signals (starting from the specified signal number) is set and the remaining bits are ignored.

**Explanation**

Sets (or resets) the signal state of one or more external output signals or internal signals according to the given value.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001 – 2960 |

Specifying a signal number greater than the number of signals actually installed results in error. Selecting a dedicated signal also results in error.

See also 5.7 BITS, BITS32 commands.

## SWAIT signal number, ……

### Function

Waits until the specified external I/O or internal signal meets the set condition.

### Parameter

Signal number

Specifies the number of the external I/O or internal signal to monitor. Negative numbers indicate that the conditions are satisfied when the signals are OFF.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| Internal signal | 2001–2960 |

### Explanation

If all the specified signals meet the set conditions, this instruction is ended and the program executes the next step. If the conditions are not satisfied, the program waits in that step until they are set.

SWAIT instruction in execution can be skipped using CONTINUE NEXT command.

The same result can be gained using the WAIT instruction.

### Example

| | |
|---|---|
| `SWAIT 1001,1002` | Waits until the external input signals 1001(WX) and 1002 (WX2) are turned ON. |
| `SWAIT 1,-2001` | Waits until external output signal 1(OX1) is ON and the internal signal 2001(WX1) is OFF. |

## EXTCALL

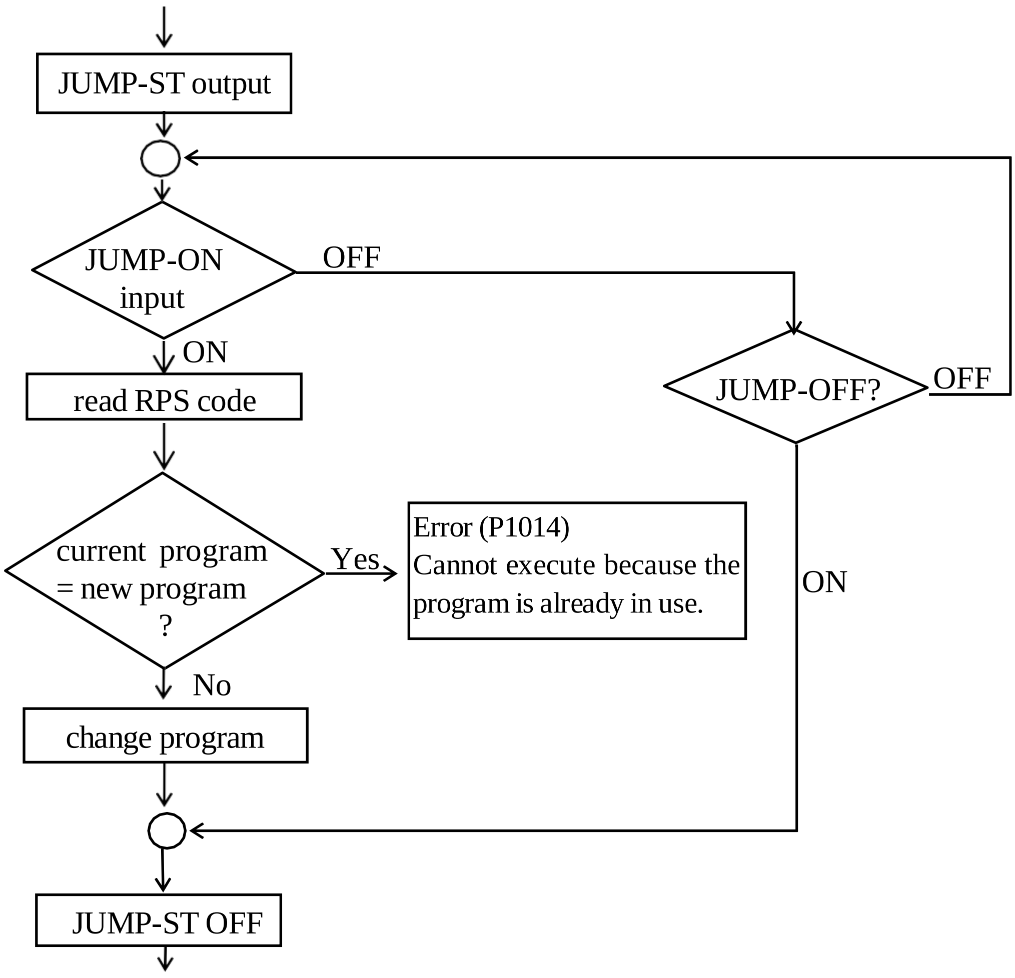
**Function**

Calls the program selected by the external input signal.

**Explanation**

EXTCALL instruction is processed as shown in the following procedure:

1. Outputs JUMP-ST signal, allowing input at an external program.
2. Waits for JUMP-ON signal to be input.
3. When JUMP-ON is input, the program number input by RPS-CODE is read. If the number input is 100 or higher, programs pgxxx are called. If the number is 99 – 10, programs pgxx are called and if the number is smaller than 9, pgx.


JUMP-ST output
JUMP-ON input
OFF
ON
read RPS code
JUMP-OFF?
OFF
current program = new program ?
Yes
Error (P1014) Cannot execute because the program is already in use.
ON
No
change program
JUMP-ST OFF

This instruction can be skipped by entering the CONTINUE NEXT command when waiting for JUMP_ON signal.

[ **NOTE** ]

This instruction can be skipped by entering the CONTINUE NEXT command when waiting for JUMP_ON signal.

This instruction is effective only when RPS mode is ON and the RPS signal is set as software dedicated signal. An error occurs if this instruction is executed when RPS is not defined as a dedicated signal.

If RPS mode is OFF, this instruction is ignored.

EXTCALL is used to call a subroutine. After the completion of this subroutine (or when a RETURN instruction is processed in the subroutine), the execution returns to the original program.

**ON mode signal number CALL program name, priority**
**ON mode signal number GOTO label, priority**
**ONI mode signal number CALL program name, priority**
**ONI mode signal number GOTO label, priority**

**Function**

Monitors the specified external input signal or internal signal and upon input of the signal, branches to the specified subroutine (CALL) or jumps to the specified label (GOTO).

ONI stops the current motion instruction, while ON waits for the current motion to be completed before jumping to the subroutine or label.

**Parameter**

Mode

| | |
|---|---|
| (not specified) | Monitors the rising and trailing edges of the specified signal. |
| /ERR (option) | Returns an error if the status of the signal already meets the set condition when monitoring starts. |
| /LVL (option) | Immediately jumps to the specified subroutine or label if the status of the signal already meets the set condition when monitoring starts. |

Signal number

Specifies the number of the signal to monitor.

If the number is positive, the rising edge of signal or the change from OFF to ON is monitored. If the number is negative, the trailing edge or the change from ON to OFF is monitored.

Acceptable signal numbers

| | |
|---|---|
| External input signal | 1001–actual number of signals |
| Internal signal | 2001–2256 |

Program name

Specifies the name of the subroutine to branch to when the specified signal is input. If omitted, the program goes on to the next step in the program and does not branch to a subroutine.

Label

Specifies which label to jump to when the specified signal is input.

Priority

Specifies the priority of the program, setting range: 1 to 127. If not specified, 1 is assumed. The greater the number is, the higher the priority becomes. Priority is ignored when a label is entered as the destination.

**Explanation**

For ON…CALL instruction, if change is detected in the monitored signal, the program is interrupted and the specified subroutine is executed. This functions the same as CALL instruction after the monitored signal is detected. (See also 6.5 CALL program instruction).

If the RETURN instruction is executed in the called subroutine, the execution returns to the program step after the step that was running before the subroutine was called. (See also 11.3 External Interlock.)

ONI instruction can be used only in robot motion programs and not in PC programs.

Signal monitoring is canceled in any of the following cases:

1. IGNORE instruction is executed for the signal specified in ON and ONI instructions.
2. The ON and ONI instructions are executed and the program has branched to a subroutine.
3. A new ON or ONI instruction specifies the same signal as an earlier ON (ONI) instruction (the older setting is canceled).

[ **NOTE** ]

1. When monitoring the rising and trailing edge of the signal, the program branches only when there is a change in the signal state. Therefore, if the leading edge is to be detected, branching does not occur if that signal is already ON when the ON instruction is executed. No branching will occur until the signal is turned OFF then turned ON again.

2. To detect signal changes accurately, the signal must be stable for at least 50 msec.

3. Monitoring starts as soon as the ON (ONI) instruction is executed. Since in the AS system, non-motion instructions are read and executed together with the preceding motion, the monitoring starts at the same time as the motion right before ON (ONI) instruction is executed.

4. The signals are not monitored while the program is not executed.

5. For ON and ONI instruction set in the main program, the signal status is monitored also in the subroutine. However, when the signal signified in the subroutine is input, the execution timing of the interruption process will be right after returning to the main program. To execute the interruption process immediately in the subroutine, set ON and ONI instructions in the subroutine in the same way as in the main program.

6. The robot moves in standard motion type instead of motion type 2 while the signals are monitored by ON instruction. Therefore, the robot motion may differ when monitoring and not monitoring the signals.

**Example**

| | |
|---|---|
| ONI -1001 CALL alarm | Monitors external input signal 1001(WX1). As soon as this signal changes from ON to OFF (the signal number is negative so the trailing edge is detected), the motion stops and the program branches to subroutine “alarm”. |
| ON test CALL delay | Monitors the signal assigned to the variable “test”. If the signal changes as desired (the condition depends on the value of “test”, since it could be negative or positive), the program branches to subroutine “delay” after execution of the current motion step is completed. It returns to the original program when the subroutine “delay” is completed. |

## IGNORE signal number

**Function**

Cancels the monitoring of signals set by ON or ONI instruction.

**Parameter**

Signal numbers

Specifies the number of the signal to cancel monitoring.

Acceptable signal numbers

| | |
|---|---|
| External input signal | 1001–actual number of signals |
| Internal signal | 2001–2960 |

**Explanation**

This instruction nullifies the effect of the recent ON or ONI instruction set to the specified signal. (See also 11.3 External Interlock.)

**[ NOTE ]**

The ON(ONI) monitoring function is only effective with binary I/O signals actually installed as input signal.

**Example**

| | |
|---|---|
| IGNORE 1005 | Cancels monitoring of external input signal (Channel 5). |
| IGNORE test | Cancels the monitoring of the signal specified by the value of variable “test”. |

## SCNT counter signal number = count up signal, count down signal, counter clear signal, counter value

**Function**

Outputs counter signal when the specified counter value is reached.

**Parameter**

Counter signal number

Specifies the signal number to output. Setting range for counter signal numbers: 3097 to 3128.

Count up signal

Specified by signal number or logical expressions. Each time this signal changes from OFF to ON, the counter counts up by 1.

Count down signals

Specified by signal number or logical expressions. Each time this signal changes from OFF to ON, the counter counts down by 1.

Counter clear signals

Specified by signal number or logical expressions. If this signal is turned ON, the internal counter is reset to 0.

Counter value

When the internal counter reaches this value, the specified counter signal is output. If "0"is given, the signal is turned OFF.

**Explanation**

If the count up signal changes from OFF to ON when the SCNT command is executed, then the internal counter value increases by 1. If the countdown signal changes from OFF to ON, the internal counter value decreases by 1. When the internal counter value reaches the value specified in the parameter (counter value), the counter signal is output. If the counter clear signal is output, value of the internal counter is set at 0. Each counter signal has its own individual counter value. To force reset of the internal counter to 0, use SCNTRESET command.

To check the states of signals 3001 to 3128, use the IO/E command. (Option)

See 5.7 also SCNT monitor command.

## SCNTRESET counter signal number

**Function**

Resets to 0 the internal counter value corresponding to the specified counter signal.

**Parameter**

Counter signal number

Selects the number of the counter signal to reset. Setting range for counter signal numbers: 3097 to 3128.

See also 5.7 SCNTRESET monitor command.

## SFLK signal number = time

**Function**

Turns ON/OFF (flicker) the specified signal in specified time cycle.

**Parameter**

Signal number

Specifies the number of the signal to flicker. Setting range: 3065 to 3096.

Time

Specifies the time to cycle ON/OFF (real values). If a negative value is set, flickering is canceled.

**Explanation**

The process of ON/ OFF is considered one cycle, and the cycle is executed in the specified time.

See also5.7 SFLK monitor command.

## SFLP output signal = set signal expression, reset signal expression

**Function**

Turns ON/OFF an output signal using a set signal and a reset signal.

**Parameter**

Output signal

Specifies the number of the signal to output. A positive number turns ON the signal; a negative number turns it OFF. Only output signals can be specified (1 to actual number of signals).

Set signal expression

Specifies the signal number or logical expression to set the output signal.

Reset signal expression

Specifies the signal or logical expression to reset the output signal.

**Explanation**

If the set signal is ON, the output signal is turned ON. If the reset signal is ON, the output signal is turned OFF. If both the set and reset signal are ON, then the output signal turns OFF. The output signal is turned ON or OFF when the SFLP command is executed, and not when the set signal or the reset signal is turned ON.

See also 5.7 SFLP monitor command.

**SOUT signal number = signal expression**

**Function**

Outputs the specified signal when the specified condition is set.

**Parameter**

Signal number

Specifies the number of the signal to output. Only output signals can be specified (1 to actual number of signals).

Signal expressions

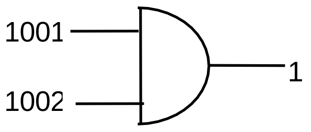
Specifies a signal number or a logical expression.

**Explanation**

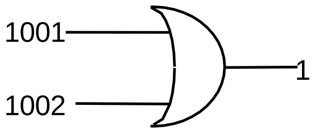
This instruction is for logical calculation of signals. Logical expressions such as AND and OR are used. The specified signal is output when that condition is set. (See also 5.7 SOUT monitor command.)

**Example**

SOUT 1 = 1001 AND 1002

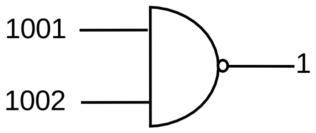


SOUT 1 = 1001 OR 1002

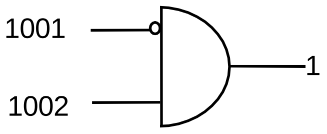


SOUT –1 = 1001 AND 1002

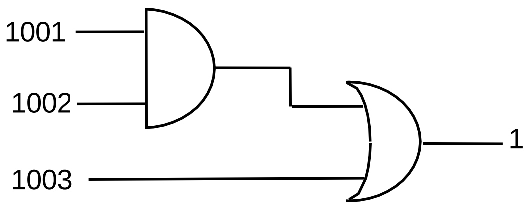
SOUT 1 = NOT(1001 AND 1002)



SOUT 1 = –1001 AND 1002

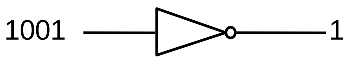


SOUT 1 = (1001 AND 1002) OR 1003



SOUT –1 = 1001 or SOUT 1= –1001 or

SOUT 1 = NOT(1001)



## STIM timer signal = input signal number, time

**Function**

Turns ON the timer signal if the specified input signal is ON for the given time.

**Parameter**

Timer signal

Selects the signal number to turn ON. Setting range: 3001 to 3064.

Input signal number

Specifies in integers the input signal number or logical expression to monitor as a condition to turn ON the timer signal. The value cannot exceed the number of signals actually installed.

Time

Specifies in real values the time (sec) the input signal is to be ON.

See also 5.7 STIM monitor command.

**SETPICK time1, time2, time3, time4, ⋯, time8**
**SETPLACE time1, time2, time3, time4, ⋯, time8**

Option

**Function**

Sets the time to start clamp close control (SETPICK) or clamp open control (SETPLACE) for each of the 8 clamps.

**Parameter**

Time 1 to 8

Sets the control time to open/close clamps 1 to 8 in seconds. Setting range: 0.0 to 10.0 seconds.

**Explanation**

See also CLAMP instruction.

## CLAMP clamp number 1, clamp number 2, clamp number 3, clamp number 4, ……, clamp number 8

Option

**Function**

Outputs clamp signal for opening/closing the hand specified by the parameter clamp number x. The output timing is set by the SETPICK/SETPLACE instruction; i.e. the signal is output x seconds before the current motion is completed.

**Parameter**

Clamp number 1 to 8

Specifies the clamp number. If the number is positive, the robot hand is opened. If the number is negative, the robot hand is closed.

**Explanation**

This instruction outputs signals to the control valve to open and close the pneumatic hand. The signal is output immediately if the robot is not in motion, or if the remaining motion time is less than the time set by SETPICK/SETPLACE instructions. The signal is output when the axes coincide if the superposing of the next motion begins before the time set by SETPICK/SETPLACE instructions is reached. If an irrational setting such as “CLAMP 1, -1” is made, the latter clamp number will be valid.

**Example**

```
12 SETPICK 4, 3, 2, 1
13 SETPLACE 0.2, 0.4, 0.6, 0.8
14 LMOVE a
15 CLAMP -1, 2, 3, -4
```

By executing the above program, the robot will move as follows:

Closes clamp 2, 3 seconds before reaching pose a.
Closes clamp 3, 2 seconds before reaching pose a.
Opens clamp 4, 0.8 seconds before reaching pose a.
Opens clamp 1, 0.2 seconds before reaching pose a.

**HSENSESET no. = input signal number, output signal number, signal output delay time**

Option

**Function**

Declares the starting of signal detection to AS system. When this instruction is executed, AS system starts to watch the sensor signal and accumulates the data such as pose, etc., into the buffer memory at signal transaction. The data saved in the buffer memory can be read using HSENSE instruction. Buffer memory can save up to 20 data.

**Parameter**

No.

Specifies the number for the monitoring results. Up to 2 input signals can be monitored. Instruction for each signal is written as HSENSESET 1 or HSENSESET 2. Acceptable range is 1 or 2.

Input signal number

Set the signal number to monitor. Setting zero (0) terminates the monitoring.

Output signal number

Set the number of the signal to be output after system gets the joint angle. The specified signal turns ON for 0.2 seconds. This may be omitted.

Signal output delay time

Set the time to delay the output of signal after acquiring the pose data. Acceptable range is 0 to 9999 ms. This may be omitted.

**[ NOTE ]**

Even when the controller power becomes OFF during watching, buffer memory keeps the read data. It is possible to read the kept data by HSENSE instruction after turning ON the controller power again. However, watching does not restart automatically, so HSENSESET should be executed again.

See also 5.7 HSENSESET monitor command.

**Example**

| | |
|---|---|
| HSENSESET 1 = wx_sensor | Starts watching for input signal wx_sensor. |

## HSENSE no. result variable, signal status variable, pose variable, error variable, memory remainder variable

Option

**Function**

Reads the data saved in the buffer memory by HSENSESET instruction.

**Parameter**

No.

Specifies the monitoring number. To read data saved by HSENSESET 1 specify HSENSE 1. To read data saved by HSENSESET 2, specify HSENSE 2.

Result variable

Specifies the name of the real variable to which the watch result is assigned. After executing HSENSE instruction, numerical value is assigned to this variable. Zero (0) is assigned to this variable when AS system does not detect the signal transaction. –1 is assigned to this variable when AS system detects the signal transaction.

Signal status variable

Specifies the name of the real variable to which the status of signal transaction is assigned. After executing HSENSE instruction, a numerical value is assigned to this variable. When the signal(s) is turned from OFF to ON, ON (-1) is assigned. When the signal(s) is turned from ON to OFF, OFF (0) is assigned to this variable.

Pose variable

Specifies the name of the pose variable to which the joint values at time of HSENSE signal input are assigned.

Error variable

Specifies the name of the real variable to which the buffer overflow error result is assigned. When no error occurs, 0 is assigned to this variable. When buffer memory overflows, a numerical value (other than 0) is assigned to this variable. The buffer memory overflows after accumulating data from more than 20 transactions.

Memory remainder variable

Specifies the name of the real variable to which the number of used memory in the buffer is assigned. The value assigned to this variable shows the number of memory in the buffer that is already used. When only one memory is used, 0 will be assigned to the variable. When all the memories are used, the value of the variable will be 19.

**Example**

In this program, sensor signal wx_sensor is monitored while the robot moves from #p2_1 to #p4_1 and the pose data of JT3 is saved in the array hsens_jt[ ] when the signal is detected. This program uses many local variables (local variable names are written with a period (.) at the beginning of the name).

```
.err = 0                                        ;Initialize
IF SIG(wx_sensor) THEN                          ;When sensor signal keeps ON
  .err=4                                        ;Incorrect starting point
  RETURN
END
;
HSENSESET 1 = wx_sensor                         ;Start watching

JMOVE #p2_1                                     ;Move to starting point
BREAK
SPEED sens_sp
ABS.SPEED ON
JMOVE #p4_1                                     ;Move to finishing point

.num = 0
loop:
HSENSE 1 .stat,hsens_onoff[.num+1],#hsens[.num+1] ,.serr,.rest
IF .serr <> 0 THEN
     .err = 3                                   ;Memory buffer over "(HSENS)"
    RETURN
END
IF .stat==ON THEN
  hsens_jt[.num+1] = DEXT(#hsens[.num+1],3)   ;Save pose in Z direction when signal detected
  .num = .num + 1
END
IF .rest GOTO loop                              ;Loop when buffer keeps data
IF DISTANSE(DEST,HERE) > 0.1 GOTO loop
;
HSENSESET 1 = 0                                 ;Finish watching
```

## RSIGPOINT signal number, output time, distance, correction time

**Function**

Outputs the specified signals before specified distance from the teaching point of next motion step of this instruction.

**Parameter**

Signal number

Specifies the external output signal number. Input range: 1 to 960

During instruction execution, if signals other than the signal number specified to output by RSIGRANGE command are specified, or if dedicated signals are specified, an error will occur.

Output time

Specifies the signal output time. (Unit: seconds)

Input range: 0.00 sec to 9.99 sec or -1.00 sec

Pulse output: Specifies between 0.00 sec and 9.99 sec

Level output: -1.00 sec

An error will occur if the output time outside of the input range is specified. When the level output signal is to be turned OFF, execute the step specifying output time as 0.00 sec. Specifying distance from the teaching point can turn the level signal OFF as well. Level signal cannot be turned OFF with the use of SIGNAL related instructions/commands.

Distance

Specifies the distance from the teaching point of next motion instruction to the position where the signals are to be output. (Unit: mm)

Input range: 0 mm to 9999 mm

Omissible. When omitted, the distance will be 0 mm.

Correction time

Using the signal output position set by specified distance as basis, corrects the signal output timing according to this correction time and outputs the signal. (Unit: seconds)

Input range: -9.99 sec to 9.99 sec

Positive: Signal is output ahead of time for the specified time.

Negative: Signal is output with delay for the specified time.

Omissible. When omitted, the correction time will be 0 sec.

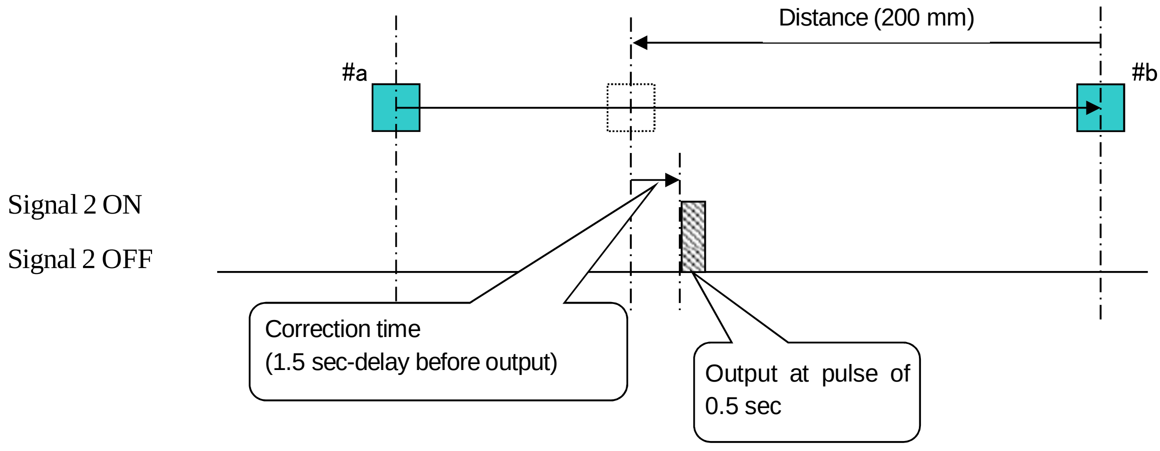
**Explanation**

Determines the specified position by current instead of command value. Specifying the correction time allows signals to be output ahead or with delay for the specified correction time, using the position of the specified distance as basis. Between the steps of motion instructions, it is possible to use this instruction to teach up to five steps, and to output signals up to five times between two points.

**Example**

In the following program example, during the linear interpolation operation moving from position #a to position #b, after having reached the position 200 mm before position #b, it delays for 1.5 sec and outputs external output signal 2 at pulse of 0.5 sec.

```
LMOVE #a
RSIGPOINT 2, 0.5, 200, -1.5
LMOVE #b
```



## RSIGCORRECT correction time

### Function

Advances the output timing of all RSIGPOINT instructions to be executed after the execution of this instruction, by the portion of correction time specified by this instruction.

### Parameter

Correction time

Specifies the time to advance the signal output. (Unit: seconds)

### Explanation

For example, to correct the time gap between the robot’s signal output and the machine’s motion, it advances the signal output timing by the portion of correction time only. If the teaching point of the previous step is exceeded in result of the advancement, the teaching point will be the limit. Delaying of timing is not possible.

Input range: 0.00 sec to 9.99 sec

### Example

| | |
|---|---|
| >RSIGCORRECT 1 | Advances the output timing of external output signals specified by the subsequent RSIGPOINT instructions by 1 sec. |

## PRINT device number: print data
## TYPE device number: print data

**Function**

Displays on the terminal the print data specified in the parameter.

**Parameter**

Device number

Select the device to display the data from below:

0: All terminals that are connected

1: Personal computer

2: Teach pendant

3: - 5: Terminals connected via Ethernet

If not specified, the data will be displayed on the currently selected device.

Print data

Select one or more from below. Separate the data with commas when specifying more than one.

| | | |
|---|---|---|
| (1) Character string | e.g. | "count =" |
| (2) Real value expressions (the value is calculated and displayed) | e.g. | count |
| (3) Format information (controls the format of the output message) | e.g. | /D, /S |

A blank line is displayed if no parameter is specified.

**Explanation**

See 5.8 PRINT/TYPE Monitor Command.

**[ NOTE ]**

If the MESSAGES switch is OFF, no message appears on the terminal screen.

## PROMPT device number: character string, variables

**Function**

Displays the specified character strings on the terminal followed by the prompt ”>” and waits for input from the keyboard.

**Parameter**

Device number

Select the device to display the data from below:

1: Personal computer

2: Teach pendant

If not specified, the data will be displayed on the currently selected device.

Character string

Specifies the characters to display on the terminal.

Variables

Specifies to which variable the data input from the keyboard is substituted. It can be a series of real variables or a single string variable.

**Explanation**

The specified character strings are displayed on the terminal and waits for data and ↵ to be input from the keyboard.

The data input is processed in one of the following ways.

1. When PROMPT is used to ask for values for a series of real variables, the system reads the input line as a series of numbers separated by spaces or commas. Each input number is converted into internal expressions according to its notation, and then they are assigned to the variables one by one.
2. If the number of values input is greater than the number of variables, the excess values are ignored. If the number of values input is less than the number of variables, “0” is assigned to the remaining variables. If data other than numeric values are input, an error occurs, and the program stops execution. To avoid confusion and error, it is advisable that one PROMPT instruction is used to assign one value to one variable.

When using a character string variable as the variable parameter for PROMPT, the characters input are read as a single data unit and all the characters are assigned to the character string variable. At the screen prompt, if only the $\boxed{\hookleftarrow}$ key or $\boxed{\text{CTRL}}$ + $\boxed{\text{C}}$ is pressed, “0” is assigned to real variables, and a null string is assigned to character string variables.

If “2” is entered for device number, the teach pendant screen changes automatically to keyboard screen.

**Example**

The character string in quotations is displayed on the terminal, and asks for data to be input. When the data (number of parts) is input and the $\boxed{\hookleftarrow}$ key is pressed, the value entered is substituted to the variable “part.count”.  The program execution then proceeds.

```
PROMPT "Enter the number of parts: ", part.count
```

The instruction below asks for the value of a character string variable.  Alphanumeric characters can be input without causing an error.

```
PROMPT "Enter the number of parts: ", $input
```

## IFPWPRINT window number, row, column, background color, label color, = "character string", "character string", ……

Option

**Function**

Displays the specified character string in the string window set in Auxiliary Function 0509 (Interface panel screen).

**Parameter**

Window number

Corresponds to the window number specified in Auxiliary Function 0509 as the window specification used to display the string. Select from 1to 8 (standard).

Row

Specifies the row in the selected window to display the string. Enter from 1 to 4; available rows depend on the window size. If not specified, 1 is assumed.

Column

Specifies the column in the selected window to display the string. Enter from 1 to 70, though available columns depend on the window size. If not specified, 1 is assumed.

Background color

Selects the background color of the selected window. Colors are numbered from 0 to 15. If not specified, the background is white.

| No. | Color | No. | Color | No. | Color | No. | Color |
|---|---|---|---|---|---|---|---|
| 0 | Grey | 4 | Green | 8 | Pink | 12 | Navy |
| 1 | Blue | 5 | Pale Blue | 9 | White | 13 | Reddish Brown |
| 2 | Red | 6 | Yellow | 10 | Black | 14 | Deep Green |
| 3 | Orange | 7 | White | 11 | Cyan | 15 | Lavender |

Label color

Selects the color of the characters displayed. Colors are numbered from 0 to 15 (See chart above). If not specified, the characters are displayed in black.

Character string

Specifies the character string to display. All strings after the first string are displayed on the next row starting at specified column. Execution of IFWPRINT clears the non-display area in the specified window.

**Explanation**

IFPWPRINT command can be used only when the interface panel is available for use. If the parameters are not specified, the last setting of that particular window is selected (for first time use, the above default values are set). If the character string does not fit in one row, its display overflows to the next line (indenting to the selected column). Strings that extend beyond the size of the window are not displayed. Control characters in the string are displayed as blanks.

**IFPWOVERWRITE mode window number, row, column, background color, label color = “character string”, “character string”, ……**

**Function**

Displays by overwriting the specified character string in the string window set by Auxiliary Function 0509 (Interface panel screen).

**Parameter**

Mode

(None) Overwrites the existing character string in unit of line.

/CUT Displays the character string by truncating the characters that do not fit in one line of the string window, without starting a new line. However, if the target window number is not allocated for the interface panel, the character string is saved to the full extent of the window and is displayed when allocated.

/CHAR Overwrites the existing character string in unit of character.
For two-byte characters, as a result of truncation or overwriting, are displayed within the correctly-displayable range.

Window number

Corresponds to the window number specified in Auxiliary Function 0509 as the window specification used to display the string. Select from 1to 8 (standard).

Row

Specifies the row in the window for displaying the string. Acceptable number is from 1 to 4, though it depends on the window size. If not specified, 1 is assumed.

Column

Specifies the column in the window for displaying the string. Acceptable number is from 1 to 78, though it depends on the window size. If not specified, 1 is assumed.

Background color

Selects the color of the background of the selected window. Acceptable numbers are from 0 to 15. If not specified, the background is white.

| No. | Color | No. | Color | No. | Color | No. | Color |
|---|---|---|---|---|---|---|---|
| 0 | Grey | 4 | Green | 8 | Pink | 12 | Navy |
| 1 | Blue | 5 | Pale Blue | 9 | White | 13 | Reddish Brown |
| 2 | Red | 6 | Yellow | 10 | Black | 14 | Deep Green |
| 3 | Orange | 7 | White | 11 | Cyan | 15 | Lavender |

Label color
Selects the color of the characters displayed. Acceptable numbers are from 0 to 15 (See chart above). If not specified, the characters are displayed in black.

Character string
Specifies the character string to display. All strings after the first string are displayed on the next row starting at specified column.

**Explanation**
IFPWOVERWRITE command can be used only when the interface panel is available for use. If the parameters are not specified, the last setting of that particular window is selected (for first time use, the above default values are set). If the character string does not fit in one row, its display overflows to the next line (indenting to the selected column). Strings that extend beyond the size of the window are not displayed. Control characters in the string are displayed as blanks. Unlike IFPWPRINT command/ instruction, the rows other than the row specified for display are displayed unchanged as before executing IFPWOVERWRITE.

**Example**
The figures below show the screens displayed when executing IFPWPRINT and IFPWOVERWRITE command/ instruction from the screen showing “This is a pen”. The left figure is the figure after executing ”IFPWPRINT 1,3,1,7,10=”my”. The characters on lines 1, 2, and 4 disappear and line 3 shows ”my”. On the other hand, the right figure shows the screen after executing “IFPWOVERWRITE 1,3,1,7,10=”my”. The characters on lines 1, 2, and 4 are displayed as they were before executing the instruction and only line 3 has changed to ”my”.

**IFPLABEL position, "label 1" "label 2", "label 3", "label 4"**

**Function**

Sets and modifies the label of the icon at the specified position on the interface panel.

**Parameter**

Position

Specifies the display position on the interface panel of the icon to set/ modify the label.

Setting range: 1 – 112.

“Label 1”, “Label 2”…

Specifies the character string to display on the interface panel as the label of the specified icon. Omitted label will not be changed.

**Explanation**

Sets and modifies the label for the icons displayed on the interface panel. When a position with no icon set or when an icon with no label is specified, nothing occurs.

See also 5.8 IFPLABEL monitor command.

## IFPTITLE page no., "title"

**Function**

Sets and modifies the title for the specified page of the interface panel.

**Parameter**

Page no.

Specifies the page of the interface panel to change the title. Setting range: 1- 4.

“Title”

Specifies the character string to display on the page as the title. Default setting is “Interface Panel”. When NULL string (“”) is specified, this default setting is also displayed.

**Explanation**

Sets and modifies the title for the specified page of the interface panel.

See also 5.8 IFPTITLE monitor command.

## IFPDISP page no.

**Function**

Displays the specified page of the interface panel.

**Parameter**

Page no.

Displays the page number of the interface panel to be displayed.

**Explanation**

Executing this command allows the display of the specified page of the interface panel.

See also 5.8 IFPDISP monitor command.

## SETOUTDA channel No. = LSB, No. of bits, logic, max. voltage, min. voltage

Option

**Function**

Specifies the analog output environment including: channel number and LSB, number of bits and logic voltage for signal output, maximum and minimum voltage.

**Parameter**

Channel No.

Sets the analog output channel number. (Setting range: integers between 1 and 16; first 1TW/1UR board: 1 to 4; second 1TW/1UR board: 5 to 8; third 1TW/1UR board: 9 to 12; fourth 1TW/1UR board: 13 to 16)

LSB

Specifies the first analog output signal number for D/A conversion as an integer. Setting range: OUT1 to OUT125, 2001 to 2125, 3000 (first channel of 1TW/1UR board), 3001 (second channel of 1TW/1UR board), and subsequent 1TW/1UR board analog output channels can be specified. Default value is 3000. Previous setting remains in effect if not specified.

Number of bits

Sets the number of bits of analog output signals for D/A conversion as an integer. Setting range: 4 to 16 bits. Sets to 12 bits when 3000 onward [3000 + analog output channel number on 1TW/1UR board] is chosen for parameter LSB above. Default value is 8 bits. Previous setting remains in effect if not specified.

Logic

Sets the logic to either positive (1) or negative (0). Default value is 0 (negative). Previous setting remains in effect if not specified. Please set logic to positive for 1TW/1UR board.

Maximum voltage

Sets the maximum voltage of hardware (D/A output). Setting range: -15.0 to +15.0 V. Unit: V. Default value is 10 V. The value should be rounded off to the first decimal place. Previous setting remains in effect if not specified.

Minimum voltage

Sets the minimum voltage of hardware (D/A output). Setting range: -15.0 to +15.0 V. Unit: V. Default value is 0 V. The value should be rounded off to the first decimal place. Previous setting remains in effect if not specified.

[ **NOTE** ]

1. Actual voltage output depends on the hardware used.
2. An error will occur if the value for maximum voltage is set lower than the minimum voltage.

## OUTDA voltage, channel no.

Option

**Function**

Outputs the voltage at set conditions from the specified analog output channel.

**Parameter**

Voltage

Sets the analog output voltage. Setting range: -15.0 to +15.0 V. Unit: V. The value should be rounded off to the first decimal place.

Channel No.

Specifies the analog output channel number. Setting range: integers between 1 and 16. If not specified, 1 is assumed.

**[NOTE]**

Confirm that command voltage and actual output voltage are the same by setting output environment to correspond with the hardware settings via SETOUTDA instruction (command).

## HERE pose variable

**Function**

Assigns the current pose to the specified variable. The pose may be expressed in transformation values, joint displacement values or compound transformation values.

**Parameter**

Pose variable

Can be defined in transformation values, joint displacement values, or compound transformation values.

**Explanation**

Assigns the current pose values to the specified variable in joint displacement values, transformation values, or compound transformation values.

**[ NOTE ]**

Only the right most variable in the compound transformation value is defined. If the other variables used in the compound value are not defined, this command results in an error.

See also 5.5 HERE monitor command.

## POINT pose variable 1= pose variable 2, joint displacement value variable

**Function**

Assigns the values of pose variable 2 to pose variable 1.

**Parameter**

Pose variable 1

Specifies the name of pose variable to be defined (by joint displacement values, transformation values, or compound transformation values). In the case of compound transformation values, POINT specifies the rightmost variable value.

Pose variable 2

Specify the name of variable defined by joint displacement values or transformation values.

Joint displacement value variable

This parameter must be set if the values of pose variable 1 are in joint displacement values and the values of pose variable 2 are in transformation values (if pose variable 1 is not defined by joint displacement values, this parameter cannot be set). The joint displacement values specified here expresses the configuration of the robot at the pose. If not specified, the current configuration is used to define the pose variable.

**Explanation**

An error is returned if pose variable 2 is not defined.

When pose variable 1 is defined in compound transformation values, the right most variable is defined. If the other variables used in the compound variables are not defined, this command will result in an error.

See also 5.5 POINT monitor command.

**POINT/ X transformation value variable 1 = transformation value variable 2**

**POINT/ Y transformation value variable 1 = transformation value variable 2**

**POINT/ Z transformation value variable 1 = transformation value variable 2**

**POINT/ OAT transformation value variable 1 = transformation value variable 2**

**POINT/ O transformation value variable 1 = transformation value variable 2**

**POINT/ A transformation value variable 1 = transformation value variable 2**

**POINT/ T transformation value variable 1 = transformation value variable 2**

**POINT/ 7 transformation value variable 1 = transformation value variable 2**

**.**
**.**
**.**

**POINT/18 transformation value variable 1 = transformation value variable 2**
**POINT/EXT transformation value variable 1 = transformation value variable 2**

**Function**

Assigns the specified component(s) of transformation value variable 2 to the corresponding component(s) of transformation value variable 1. The values will be displayed on the terminal for correction.

**Parameter**

Transformation value variable 1
Specifies a single transformation value variable, or a variable defined by compound transformation values with the last component defined in transformation values.

Transformation value variable 2
Specifies the name of a variable defined by transformation value, compound transformation value or transformation value function. The name of this variable must be defined beforehand.

**Explanation**

Error occurs if any pose variable on the right side of the "=" is not defined.
If compound transformation value variables are specified for transformation value variable 1, this instruction assigns values to only the rightmost variable. Also, error occurs if any variable other than the rightmost variable is undefined.

**Example**

| | |
|---|---|
| POINT/X temp=tempx | Assigns the x component of transformation value variable "tempx" to the x component of transformation value variable "temp". |
| POINT/EXT temp=tempx | Assigns the external axes components of transformation value variable tempx to the external axes components of transformation value variable temp. |

## DECOMPOSE array variable [element number] = pose variable

**Function**

Stores as elements of an array variable, each component of the values of the specified pose variable (X, Y, Z, O, A, T for transformation values; JT1, JT2, JT3, JT4, JT5, JT6 for joint displacement values).

**Parameter**

Array variable

Specifies the name of the array variable into which the values of each component will be assigned.

Element number

Specifies the first element in which to store the components.

Pose variable

Specifies the name of the pose variable from which to extract each component (transformation values, joint displacement values).

**Explanation**

This assigns the components of the specified pose information to the elements of the array variable.

In case of transformation values, six elements are assigned from each of the XYZOAT values. In case of joint displacement values, the elements are each assigned from the values of each joint in the robot arm.

**Example**

| | |
|---|---|
| DECOMPOSE X[0]=part | Assigns the components of transformation value variable "part" to the first six elements in the array variable "x". |
| DECOMPOSE angles[4]=#pick | Assigns the components of joint displacement value variable "#pick" to array variable "angles", starting from element number 4. |

For example, in the above instruction, if the values of #pick are (10,20,30,40,50,60) then,

```
angles[4]=10    angles[7]=40
angles[5]=20    angles[8]=50
angles[6]=30    angles[9]=60
```

## BASE transformation value variable

**Function**

Defines the base transformation values.

**Parameter**

Transformation value variable

Specifies a transformation value variable or compound transformation value variables. Defines the new base coordinates. The pose variable here describes the pose of the base coordinates with respect to the null base coordinates, expressed in null base coordinates.

**Explanation**

The CP movement is stopped (BREAK) and the base transformation values are changed to the specified transformation values when this instruction is executed.

See also 5.6 BASE monitor command.

## TOOL transformation value variable, tool shape number

**Function**

Defines the transformation values that shows the pose of the tool tip (tool transformation values) as seen from the tool mounting flange surface (null tool coordinates).

**Parameter**

Transformation value variable

Specifies a transformation value variable or compound transformation value variable to define the new tool coordinates. The transformation value variable here describes the pose of the tool coordinates with respect to the null tool coordinates, expressed in null tool coordinates. If "NULL" is specified for the parameter, the tool coordinates will be set same as the null tool coordinates.

Tool shape number

Specifies the tool shape to use for speed control in teach and check mode. This may be omitted. If not specified, speed control by tool shape will not be done.

**Explanation**

The continuous path (CP) movement is stopped (BREAK) and the tool transformation values are changed to the specified transformation values when this instruction is executed.

See also 5.6 TOOL monitor command.

## SET_TOOLSHAPE tool shape no. = transformation value variable 1, transformation value variable 2, …, transformation value variable 8

**Function**

Registers the tool shape used to control speed in teach mode and check mode.

**Parameter**

Tool shape no.

Specifies the number of tool shape to register. Setting range: 1 to 9.

Transformation value variables 1-8

Specifies transformation value variables for the points on the tool shape. Maximum of 8 points can be specified. The points are specified in transformation values as seen from the center of the flange surface. However, only the X,Y,Z values of the transformation values are used for the tool shape registration.

**Explanation**

Defines the tool shape used for speed control in teach and check modes by maximum 8 points specified in pose variables defined by transformation values.

See also 5.6 SET_TOOLSHAPE command.

## ENA_TOOLSHAPE tool shape no. = TRUE/ FALSE

**Function**

Enables/ disables speed control in teach and check mode.

**Parameter**

Tool shape number

Specifies the tool shape number to set enable/disable in integer from 1 to 9.

TRUE/FALSE

Specify TRUE to enable speed control by the specified tool shape. Specify FALSE to disable the speed control.

**Explanation**

Selects if speed control in teach and check modes are done by the specified tool shape or not. FALSE is selected for all tool shapes as default setting. If TRUE is selected for a tool shape number with not even one point specified, error E1356 Tool shape not set occurs when the robot is operated in teach or check mode. To avoid this, always set at least one tool point via SET_TOOLSHAPE command or change from TRUE to FALSE via this command an d then execute TOOL or TOOLSHAPE command specifying the relevant tool shape number. (Once set to TRUE, the setting will not be changed to FALSE unless TOOL/TOOLSHAPE command is executed.)

## TOOLSHAPE tool shape no.

**Function**

Selects the tool shape used to control speed in teach mode and check mode.

**Parameter**

Tool shape no.

Specifies the number of the tool shape used for speed control. Setting range: 1 to 9.

**Explanation**

To enable speed control in teach mode and check mode, the function must be enabled by ENA_TOOLSHAPE command/ instruction (ENA_TOOLSHAPE n =TRUE). Error E1356 Tool shape not set occurs if a tool shape with no point registered (all points set to 0) is selected.

See also 5.6 TOOLSHAPE command.

## SETHOME accuracy, joint displacement value variable
## SET2HOME accuracy, joint displacement value variable

**Function**

Sets HOME pose.

**Parameter**

Accuracy

Sets the accuracy range of the HOME pose in millimeters.

Joint displacement value variable

Specifies a joint displacement value variable to set the pose for HOME.

**Explanation**

Sets robot's HOME pose by specifying its joint displacement values (in units of degrees). HOME 1 is set using SETHOME command, HOME 2 using SET2HOME command.

See also 5.6 SETHOME/ SET2HOME command.

**Example**

| | |
|---|---|
| SETHOME 10, #place | Records the pose determined by joint displacement value variable #place as HOME. The accuracy is set to the range of within 10 mm of HOME. |

**ULIMIT joint displacement value variable**
**LLIMIT joint displacement value variable**

**Function**

Sets and displays the upper/lower limit of the robot motion range.

**Parameter**

Joint displacement value variable

Specifies a joint displacement value variable for software limit (upper or lower).

**Explanation**

Sets the upper (lower) limit of the robot motion range in joint displacement values (in degrees).

See also 5.6 ULIMIT/LLIMIT monitor commands.

## TIMER timer number = time

**Function**

Sets the time of the specified timer.

**Parameter**

Timer number

Specifies the number of the timer to set the time. Acceptable numbers are 1 to 10.

Time

Specifies the time to set to the timer in seconds.

**Explanation**

When this instruction is executed, the timer is immediately set to the specified time. Check the value of the timer using TIMER function.

**Example**

The example below holds the program execution for the specified time using the TIMER instruction and TIMER function.

| | |
|---|---|
| `TIMER 1=0` | Sets Timer 1 to 0 (seconds). |
| `WAIT TIMER(1)>delay` | Waits until the value of Timer 1 is greater than delay. |

## UTIMER @timer variable = timer value

**Function**

Sets the default value for the user timer.  The user timer can be named freely using the timer variable.  More than one user timer can be used at a time.

**Parameter**

@Timer variable

Specified the variable or array variable name to use as the timer name.  Enter @ in front of a integer variable.

Timer value

Specifies the default value for the timer.  Acceptable numbers are 0 to 2147483647 (seconds).

```
switch name, ……ON
switch name, ……OFF
```

**Function**

Turns ON (OFF) the specified system switch.

**Parameter**

Switch name

Specifies the name of the system switch to turn ON (OFF).

**Explanation**

Turns ON (OFF) the switch specified here.  More than one switch name can be entered separating each switch name by commas.

See also 5.6 ON/OFF monitor commands.

The current setting of the switch can be checked using the SWITCH command.

## WEIGHT load mass, center of gravity X, center of gravity Y, center of gravity Z, inertia moment ab. X axis, inertia moment ab. Y axis, inertia moment ab. Z axis

**Function**

Sets the load mass data (weight of the tool and workpiece). The data is used to determine the optimal acceleration/deceleration of the robot arm.

**Parameter**

Load mass

The mass of the tool and workpiece (in kilograms). Range: 0.0 to the maximum load capacity (kg).

Center of gravity (unit = mm)

X: the x value of the center of gravity in tool coordinates

Y: the y value of the center of gravity in tool coordinates

Z: the z value of the center of gravity in tool coordinates

Inertia moment about X axis, inertia moment about Y axis, inertia moment about Z axis (Option)

Sets the inertia moment around each axis. Unit is kg·m2. The inertia moment about each axis is defined as the moment around the coordinates axes parallel to the null tool coordinates with the center of rotation at the tool's center of gravity.

**Explanation**

If the parameters are not specified when using WEIGHT as a program instruction, the setting defaults to the maximum load capacity for that robot model.

**DANGER**

**Always set the correct load mass and center of gravity. Incorrect data may weaken or shorten the longevity of parts or cause overload/deviation errors.**

## MC monitor command

**Function**

Enables execution of monitor commands from PC programs. Monitor commands that can be used with this instruction are: ABORT, CONTINUE, ERESET, EXECUTE, HOLD, and SPEED.

**Explanation**

This instruction is used in cases when a robot program is executed (EXECUTE command) from program AUTOSTART.PC. MC instruction cannot be used in robot programs.

**Example**

Robot programs can be executed from AUTOSTART.PC using this command. However, the motor power has to be ON to execute robot programs, so when using MC instruction, add the following steps to check if the power is ON.

```
autostart.pc()
1 10 IF SWITCH(POWER)==FALSE GO TO 10
2 MC EXECUTE pg1
.END
```

## TPLIGHT

### Function

Turns on the teach pendant backlight.

### Explanation

If the backlight of the teach pendant screen is OFF, this instruction turns ON the light. If this instruction is executed when the backlight is ON, the light stays ON for the next 600 seconds.

## CURLIM axis number, positive current limit, negative current limit

**Function**

Changes the motor current limit of the external axis.

**Parameter**

Axis number

Specifies the number of the external axis. Acceptable range: 7-18.

Positive current limit

Specifies the positive limit of the motor current. The limit is set as percentage of current limit set in the external axis servo parameter. Unit: %. Acceptable range: greater than 0 and less than equal to 100.

Negative current limit

Specifies the negative limit of the motor current. The limit is set as percentage of current limit set in the external axis servo parameter. Unit: %. Acceptable range: greater than 0 and less than equal to 100.

**Explanation**

This instruction is valid only for external axis with KHI amplifier.

The changed limits are reflected on the current limit values of external axis servo parameters at the following timing:

- When program is executed via EXECUTE command.
- When a new program is reselected and executed via cycle start.
- When program execution is reset.
- When controller power is turned OFF/ON.

## SETTIME year, month, day, hour, minute, second

**Function**

Sets the current time and date.

**Parameter**

year, month, day, hour, minute, second

Sets the time and date as shown below. Both the date and the time have to be entered as the parameter even when changing only the time. The parameter values must be real values or real variables.

**Explanation**

Specifying the time and date via this instruction changes the internal calendar of the robot.
Each element for setting the calendar is as below:

| | | |
|---|---|---|
| Year | (00 - 99) | The last two digits of the year |
| Month | (01 - 12) | |
| Day | (01 - 31) | |
| Hour | (0 - 23) | |
| Minute | (0 - 59) | |
| Second | (0 - 59) | |

**Example**

| | |
|---|---|
| `SETTIME 16, 10, 20, 9, 45, 46` | Sets the current time to October 20, 2016 9:45:46 |

## ENVCHKRATE axis number, coefficient

### Function

Specifies the magnification ratio to the initial value of the threshold for detection of deviation abnormality in the external axis.

### Parameter

Axis number

Specifies the number of the external axis. Acceptable range: 7-18.

Coefficient

Specifies the desired magnification ratio to the initial value of the deviation abnormality detection threshold. Acceptable range: 0 - 9.999. The default value is 1.0. The deviation abnormality detection threshold is in its initial value when the magnification is set to the default. Specifying 0 nullifies the deviation abnormality check.

**⚠ CAUTION**

**When the coefficient is set to greater than 1.0, the deviation error detection may be delayed. If set to 0, deviation error check is invalidated. Be careful with the motion of the set axis when setting the coefficient larger than 1.0 or to 0.**

### Explanation

This instruction is valid only for external axis with KHI amplifier. Also, this instruction is executed only in repeat mode. This instruction is not executed in check mode.

The changed limits return to their initial values at the following timings:

- When program is executed via EXECUTE command.
- When a new program is reselected and executed via cycle start.
- When program execution is reset.
- When the control power is turned OFF then ON.
- When switched to teach mode.

## SETENCTEMP_THRES robot number: joint number, warning threshold, error threshold

**Function**

Performs the threshold setting of encoder temperature warning and temperature error of each joint.

**Parameter**

Robot No.
Specifies a robot number when one controller is controlling multiple robots.

Joint number
Specifies the joint number to set the encoder temperature warning and temperature error thresholds. Sets encoder temperature warning and temperature error thresholds for all joints by joint when omitted.

Warning threshold
Sets the encoder temperature warning threshold. Warning threshold can be set between 0 and 125ºC. Specify -1 to reset to default value.

Error threshold
Sets the encoder temperature error threshold. Error threshold can be set between 0 and 125ºC. Specify -1 to reset to default value.

**Explanation**

Allows the setting and resetting of the encoder temperature warning and temperature error thresholds of each joint.

**[NOTE]**

Encoder temperature warning thresholds and temperature error thresholds set by this command may be reset to default values due to encoder replacement.

Temperature warning: (W1085) “Encoder temperature exceeded limit.(jt XX) (XX deg C)”
Temperature error: (E1564) “Encoder temperature exceeded limit.(jt XX) (XX deg C)”

**CAUTION**

**Do not raise threshold temperatures above default values, as any rise of temperature may affect arm components such as the encoder.**

**Example**

When setting the encoder temperature warning threshold as 80°C and encoder temperature error threshold as 90°C for JT2:

```
>SETENCTEMP_THRES 2,80,90↵
```

DELETE program name, .........

DELETE/P program name, .........

**DELETE/L    pose variable[array elements] , .........**

**DELETE/R    real variable [array elements] , .........**

**DELETE/S string variable [array elements] , .........**

**DELETE/INT string variable [array elements] , ........**

**Function**

Deletes the specified data from the memory.

**Parameters**

Program name (/P), pose variable (/L), real variable (/R), string variable (/S), integer variable (/INT)

Specifies the type and name of the data to delete.

See also 5.2 DELETE monitor commands.

## TRACE stepper number: ON/OFF

**Function**

Starts (ON) or ends (OFF) logging of robot or PC program to allow program tracing.

**Parameters**

Stepper number

Specifies the program to log using the following number selection:

1: Robot program

1001: PC program 1

1002: PC program 2

1003: PC program 3

1004: PC program 4

1005: PC program5

If the program is not specified, the program currently in execution is logged.

ON/OFF

Starts/ ends logging.

**Explanation**

If the necessary memory is not reserved using the SETTRACE command before TRACE ON, the error (P2034) “Memory undefined” occurs. Execute SETTRACE before retrying.

See also 5.2 TRACE/SETTRACE monitor commands.

**NLOAD/IF/ARC device number = file name + file name + ···, status variable**

**Function**

Reads the specified file from the USB memory or the CFast stored in the controller and loads it onto robot memory.

**Parameter**

Device number

Specifies where to display the current processing situation. Specify either 1 or 2:

1: Standard terminal (PC)

2: Teach pendant

When omitted, it is displayed where the program is executed.

File name

Specifies the file to load onto the robot memory. When specifying a folder, specify “folder name¥”before the file name. Indicating “USB1¥” to “USB4¥”in the folder name allows specifying files on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the front of the controller, USB#3 indicates the back and USB#4 indicates the USB slot inside the board. Also, indicating “CFast¥” allows specifying files on the CFast. When loading multiple files, separate the file names using +.

Status variable

Specifying 0 allows continuing program execution even when error occurs. The error code is saved. When omitted, program execution stops at error occurrence.

**Explanation**

Loads the contents (program and variables) of the specified file onto robot memory.
Specifying a program with the same name as a program in the robot memory will result in such program being overwritten by the loaded program.

1. Specifying NLOAD/IF loads interface panel data.
2. Specifying NLOAD/ARC loads arc welding data.

When there is a step that is not readable in a program being loaded, error message and message “Step format incorrect. (0: Comment-out the step and continue reading, 1: Delete program and terminate) appears. When continuing by selecting zero (0), modify the program via editor after loading is completed.

**[ NOTE ]**

Pose variable, real variable, or a string variable with the same name as the variable being read is over-written by the new data without any warning.

**Example**

| | |
|---|---|
| NLOAD Pallet | Loads the contents of file pallet.as from USB#1 onto the robot memory (main memory). |
| NLOAD USB2¥Pallet2 | Loads the contents of file pallet2.as from USB#2 onto the robot memory. |
| NLOAD CFAST¥f3.pg | Loads all the programs in file f3.pg from the CFast onto the robot memory. |

## SLOAD/IF/ARC device number = character string, status variable

### Function

Reads the file name represented by character string from the USB memory or the CFast stored in the controller and loads it onto the robot memory.

### Parameter

Device number

Specifies where to display the current processing situation. Specify either 1 or 2:

1: Standard terminal (PC)

2: Teach pendant

When omitted, it is displayed where the program is executed.

Character string

Specifies the file to load onto robot memory. When specifying a folder, specify "folder name¥¥"before the file name. Indicating "USB1¥¥" to "USB4¥¥" in the folder name allows specifying files on the USB memory of USB#1 to USB#4. USB#1 and USB#2 indicate the front of the controller, USB#3 indicates the back and USB#4 indicates the USB slot inside the board. Also, indicating "CFast¥¥" allows specifying files on the CFast. Only one file can be specified.

Status variable

Specifying 0 for this parameter allows continuing program execution even when error occurs. The error code is saved. When omitted, program execution stops at error occurrence.

### Explanation

Loads the contents (program and variables) of the specified file onto robot memory. Specifying a program with the same name as a program already in the robot memory will result in such program being overwritten by the loaded program.

1. Specifying SLOAD/IF loads interface panel data.
2. Specifying SLOAD/ARC loads arc welding data.

When there is a step that is not readable in a program being loaded, error message and message "Step format incorrect. (0: Comment-out the step and continue reading, 1: Delete program and terminate) appears. When continuing by selecting "0", modify the program via editor after loading is completed.

**[ NOTE ]**

Pose variable, real variable, or a string variable with the same name as the variable being read is over-written by the new data without any warning.

**Example**

| | |
|---|---|
| SLOAD $str1 | When $str1="pallet.as", loads contents of file pallet.as from USB#1. |
| SLOAD $str2 | When $str2="USB2¥¥pallet2.as", loads contents of file pallet2.as from USB#2. |
| SLOAD $str23 | When $str2="CFAST¥¥f3.pg", loads contents of file f3.pg from the CFast. |

## SHUTDOWN

**Function**

Executes the data backup to the CFast in the controller.

**Explanation**

The data backup to the CFast starts when the SHUTDOWN command is executed.
If the data backup is completed normally, the message (D0906) “Data backup to CF is completed. Turn OFF the controller power.” appears.
If the data backup is not completed after the specified time (10[sec]), the message (D0907) “Data backup to CF is failed.  Turn OFF & ON the controller power.” appears.
Executing following operations becomes impossible after the SHUTDOWN command is executed, because the memory of the controller is locked.  Turn OFF and ON the controller power.
The SHUTDOWN command cannot be used while a robot motion program is running.
If the SHUTDOWN command is executed while the robot motion program is running, the error (P1012) “Robot is moving now.” occurs.

## CP

**Function**

Enables or disables continuous path (CP) function.

**Explanation**

This switch is used to turn ON (enable) or OFF (disable) the CP function. When this switch is changed in a program, the CP function is enabled /disabled starting from the next motion. The default setting for this switch is ON.

**Example**

| | |
|---|---|
| CP OFF | Disables the CP function. |

## CHECK. HOLD

**Function**

Enables or disables the use of the keyboard to enter the EXECUTE, DO, STEP, MSTEP, and CONTINUE commands when the HOLD/RUN state is in HOLD.

**Explanation**

When this switch is ON, the following commands entered from the keyboard are accepted only when HOLD/RUN is in HOLD.  (The commands are accepted, but the motion does not start unless HOLD/RUN is changed to RUN.)

EXECUTE, DO, STEP, MSTEP, CONTINUE

When this switch is OFF, the commands above are accepted regardless of the state of HOLD/RUN.  Operations not done by keyboard are not affected by this command.  (If in RUN state, the robot starts moving as soon as the command is input.)

Default setting is OFF.

## CYCLE. STOP

**Function**

Determines whether or not to continue repeating the execution cycle after an HOLD is applied.

**Explanation**

When this switch is OFF, the robot stops when the external HOLD signal is input, but CYCLE START remains ON. Therefore, when the external HOLD signal is turned OFF, the robot resumes program execution.

When this switch is ON, the robot stops when the external HOLD signal is input, and CYCLE START is turned OFF. Therefore, even if the external HOLD is released, CYCLE START remains OFF, so the robot does not resume program execution. To resume program execution, follow regular program execution procedures (i.e. press $\boxed{\text{A}}$+$\boxed{\text{CYCLE START}}$ on teach pendant).

This switch has effect only on external HOLD and not when HOLD/RUN state is changed to HOLD. If HOLD/RUN is changed to HOLD, robot stops, but CYCLE START remains ON, regardless of the condition of this system switch. The robot resumes program execution when HOLD/RUN is changed to RUN.

Default setting is OFF.

## MESSAGES

**Function**

Enables or disables the output of message on the terminal.

**Explanation**

When this switch is ON, the messages are displayed on the terminal when the PRINT or TYPE command is used.    When this switch is OFF, messages are not displayed on the terminal.

Default setting is ON.

## OX. PREOUT

**Function**

Determines the timing for turning ON/OFF OX signals in block step instruction.

**Explanation**

When running programs taught by block step instructions and this switch is ON, the OX signal taught for a given pose is turned ON as soon as the robot begins motion toward the pose.

When this switch is OFF, the signal is not turned ON until the axes coincide with the pose taught at that step.

Default setting is ON.

## PREFETCH. SIGINS

**Function**

Enables or disables early processing of signal input and output commands in AS programs.

**Explanation**

When this switch is OFF, commands for signal input/ output and synchronization listed below are not executed until the axis coincides with the pose taught to the current motion instruction.

SWAIT, SIGNAL, TWAIT, PULSE, DLYSIG, RUNMASK, RESET, BITS

When this switch is ON, all commands including the commands above are executed and the program is processed up to the step before the next motion instruction as soon as the movement towards a taught point starts.

Default setting is OFF.

## RPS

**Function**

Enables or disables the random selection of programs (JMP, RPS).

**Explanation**

When this switch is OFF, the EXTCALL instruction and JUMP/ END of the auxiliary data are ignored.

Default setting is OFF.

## SCREEN

**Function**

Enables or disables scrolling of the screen.

**Explanation**

If the switch is ON, the scrolling of the screen stops when the display is full. Press $\boxed{\text{Spacebar}}$ to show the next page.

Default setting is ON.

## REP_ONCE

### Function

Determines whether the program is run one time or continuously.

### Explanation

When this switch is ON, the program runs one time.

When this switch is OFF, the program runs continuously.

Default setting is OFF.

## STP_ ONCE

**Function**

Determines whether the program steps are run one step at a time or continuously.

**Explanation**

When this switch is ON, the program steps are run one step at a time.

When this switch is OFF, the program runs continuously through all the steps.

Default setting is OFF.

**AUTOSTART. PC**
**AUTOSTART2. PC**
**AUTOSTART3. PC**
**AUTOSTART4. PC**
**AUTOSTART5. PC**

**Function**

Determines if the selected PC program starts automatically when the controller power is turned ON.

**Explanation**

When this switch is ON, the PC program named AUTOSTART.PC starts automatically when the controller power is turned ON.　Program AUTOSTART.PC should be created beforehand.　Five different PC programs can be selected to start automatically, each named AUTOSTART.PC and AUTOSTART2.PC to AUTOSTART5.PC.　Create a program named accordingly and turn ON each corresponding switch to start the desired program.

Default setting is OFF.

[ **NOTE** ]

When this switch is turned ON but the corresponding program does not exist, an error occurs and the message “Program does not exist” appears.

## ERRSTART. PC

Option

**Function**

Determines if the selected PC program is executed or not when an error occurs.

**Explanation**

When this switch is ON, the PC program named ERRSTART.PC is automatically executed when an error occurs.　Create a PC program named ERRSTART.PC in advance.

Default setting is OFF. Once the ERRSTART.PC program is executed, this switch is turned OFF automatically. To reset automatic execution of the ERRSTART.PC program, be sure to set this switch to ON again. Beware that ERRSTART.PC program is executed as the fifth PC program, so if five PC programs are already running, ERRSTART.PC cannot be executed.

[ **NOTE** ]

When this switch is turned ON but the corresponding ERRSTART. PC program does not exist, an error occurs and the message “Program does not exist” appears.

## SWITCH(TRIGGER)

**Function**

Displays if $\boxed{\text{TRIGGER (DEADMAN)}}$ switch on the teach pendant is ON or OFF.

This function does not turn ON/OFF the switch.

When used with SWITCH function, –1 is returned if the $\boxed{\text{TRIGGER}}$ switch is ON, and 0 if it is OFF.

## SWITCH (CS)

**Function**

Displays if CYCLE START is ON or OFF.

This function does not turn ON/OFF the switch.

When used with SWITCH function, −1 is returned if CYCLE START is ON, and 0 if it is OFF.

## SWITCH (POWER)

**Function**

Displays if the motor power is ON or OFF.

This function does not turn ON/OFF the switch.

When used with SWITCH function, –1 is returned if the motor power is ON, and 0 if it is OFF.

## SWITCH (RGSO)

**Function**

Displays if servo motor power is ON or OFF.

This function does not turn ON/OFF the switch.

When used with SWITCH function, –1 is returned if the servo motor power is ON, and 0 if it is OFF.

## SWITCH (TEACH_LOCK)

**Function**

Displays if $\boxed{\text{TEACH LOCK}}$ switch is ON or OFF.

This function does not turn ON/OFF the switch.

When used with SWITCH function, –1 is returned if the switch is ON, and 0 if it is OFF.

## SWITCH (ERROR)

**Function**

Displays whether or not an error is currently occurring.

This function does not turn ON/OFF the switch.

When used with SWITCH function, –1 is returned if error is occurring, and 0 if not occurring.

## SWITCH (REPEAT)

**Function**

Displays if $\boxed{\text{TEACH/REPEAT}}$ is in TEACH position or in REPEAT position.

This function does not turn ON/OFF the switch.

When used with SWITCH function, –1 is returned if the switch is in REPEAT position, and 0 if it is in TEACH position.

## SWITCH (RUN)

### Function

Displays if HOLD/ RUN is in RUN state or in HOLD state.

This function does not the state of the switch.

When used with SWITCH function, −1 is returned if the switch is in RUN state, and 0 if it is in HOLD state.

## SWITCH (PNL_CYCST)

**Function**

Displays if CYCLE START is ON or OFF.

The switch cannot be turned ON/OFF with this instruction.

When used with SWITCH function, –1 is returned when the switch is ON, and 0 when it is OFF.

## SWITCH (PNL_MPOWER)

**Function**

Displays if the motor power is ON or not OFF.

The switch cannot be turned ON/OFF with this instruction.

When used with SWITCH function, –1 is returned when the switch is ON, and 0 when it is OFF.

## SWITCH (PNL_ERESET)

**Function**

Displays if ERROR RESET is ON or not (OFF).

The switch cannot be turned ON/OFF with this instruction.

When used with SWITCH function, –1 is returned when the switch is ON, and 0 when it is OFF.

## SWITCH (TPKEY_A)

**Function**

Displays if $\boxed{\text{A}}$ switch on the teach pendant is pressed (ON) or not (OFF).

The switch cannot be turned ON/OFF with this instruction.

When used with SWITCH function, –1 is returned when the switch is ON, and 0 when it is OFF.

## DISPIO_ 01

**Function**

Changes how signals (external I/O and internal signals) are displayed with the IO command.

**Explanation**

If the system switch DISPIO_01 is OFF, “o” is displayed for signals that are ON.  “x” is displayed for signals that are OFF.  The dedicated signals are displayed in uppercase letters (“O” and “X”).

If the system switch DISPIO_01 is ON, “1” is displayed for the signals that are ON and “0” for those that are OFF.  “-” is displayed for external I/O signals that are not installed.

Default setting is OFF.  (See also 5.6 IO monitor command.)

**Example**

When DISPIO_01 is OFF

```
>IO↵
  32 -    1    xxxx  xxxx  xxxx  xxXX  xxxx  XXXX  XXXO  XXXO
1032 - 1001    xxxx  xxxx  xxxx  xxXX  xxxx  XXXX  XXXX  OXXX
2032 - 2001    xxxx  xxxx  xxxx  xxxx  xxxx  xxxx  xxxx  xxxx
>
```

When DISPIO_01 is ON

```
>IO↵
32-     0      0000  0000  0000  0000  0000  0000  0001  0001
1032-1001      0000  0000  0000  0000  0000  0000  0000  1000
2032-2001      0000  0000  0000  0000  0000  0000  0000  0000
>
```

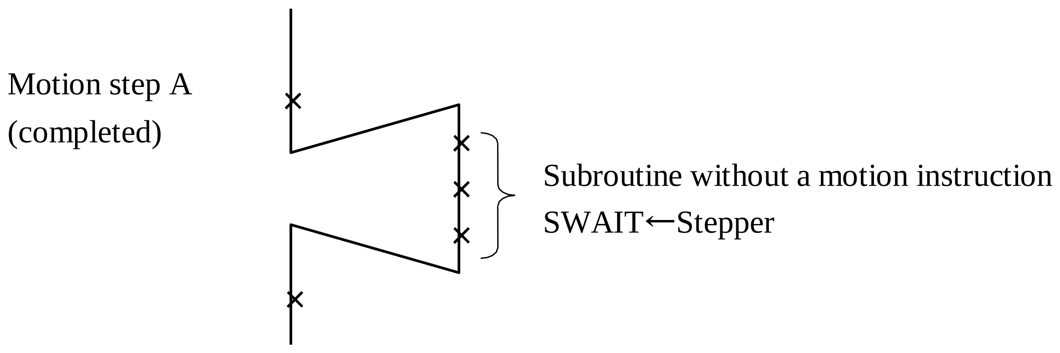
## HOLD.STEP

**Function**

Selects the step to display when the program execution is held.

**Explanation**

When this switch is ON and program execution is held during execution of a non-motion step, the step in the currently executed stepper is displayed instead of the motion instruction just completed.    For example in the situation below, if the switch is ON, the stepper step of SWAIT is displayed (SWAIT can be skipped).    If the switch is OFF, the motion step A is displayed.



Default setting is OFF.

## WS_COMPOFF

Option

**Function**

Changes the output condition of the welding signal (WS).

**Explanation**

When this switch is ON, WS signal is output from the moment memory change occurs until the welding complete signal is input.

When this switch is OFF, WS signal is output from the moment memory change occurs until the next memory change.

Default setting is OFF.

## FLOWRATE

**Function**

Switches between flow rate control mode and speed output mode.

**Explanation**

When this switch is ON, the flow rate control mode is selected.

When this switch is OFF, the speed output mode is selected.

Default setting is OFF.

Option

**Function**

Changes the operation done when the WS signal is 0.

**Explanation**

When this switch is ON, the welding is done when WS=0, as well as when WS≠0. (Pressurizing and welding)

When this switch is OFF, welding is not done when WS=0. (Pressurizing only)

Default setting is OFF.

Option

## ABS.SPEED

**Function**

Enables or disables the use of absolute speed.  This function enables execution of motion steps at a low, pre-defined speed setting, taking precedence over the monitor speed setting.

**Explanation**

When this switch is ON, the robot moves at the absolute speed specified for the program when the below condition is true.

Maximum speed × Monitor Speed > Program Speed

Also, when this switch is ON, the acceleration speed stays at the speed equivalent to monitor speed 100%, independent from the change in monitor speed.

Default setting is OFF.

**Example**

If Max speed 2400 mm/s, Monitor Speed 10%, Program Speed 100 mm/s, then

$2400 \times 0.1 > 100$

The robot moves at the program speed 100 mm/s.

If Max speed 100%, Monitor Speed 10%, Program Speed 5%, then

$100 \times 0.1 > 5$

The robot moves at the program speed 5%.

If Max speed 2400 mm/s, Monitor Speed 2%, Program Speed 100 mm/s, then

$2400 \times 0.02 < 100$

The robot moves at the monitor speed 48 mm/s.

## SLOW_START

**Function**

Enables (ON) or disables (OFF) the slow start function.  If the slow start function is enabled, the robot starts motion at slow repeat speed in the first motion step or during a specified time at the beginning of the motion.

**Explanation**

When this switch is turned ON, the slow start function is enabled in repeat mode. The slow start function will not be enabled in teach or check mode.

Set the speed in slow repeat mode using SLOW_REPEAT command or Aux. 0508 Slow Repeat.

Also, in Aux. 0508, the time to operate in slow speed can be set (slow speed repeat time at startup).  When slow speed repeat time at startup is set as 0, only the first motion step is executed in slow speed.

If a program is stopped and then restarted, the first motion step to be executed will start at the slow repeat speed.

Default setting is OFF.

## AFTER.WAIT.TMR

**Function**

Sets the timing for starting timers in block instructions.

**Explanation**

When this switch is OFF, the timer starts when the axes coincide.    If it is ON, the timer starts when the axes coincide and all the set conditions (e.g. WX, WAIT, RPS ON) are fulfilled.

Default setting is OFF.

## STAT_ON_KYBD

**Function**

Sets if the status information is displayed on the keyboard screen or not.

**Explanation**

When this switch is OFF, the status information is not displayed and the keyboard screen is displayed at full-screen.  If it is ON, the status information is displayed on the top part of the keyboard screen, as in the teach screen.  The keyboard screen will be smaller by 4 lines than the full-screen display.

Default setting is ON.

## WAITREL_AUTO

**Function**

Sets if the wait release popup window is displayed or not when the robot comes to a wait in teach or check mode.

**Explanation**

When this switch is ON, the popup window appears when the robot is in wait status.    If it is OFF, the popup window is not displayed.

Default setting is ON.

## REP_ONCE.RPS_LAST

**Function**

Sets on which step to end the program execution when the program is repeated once via RPS function.

**Explanation**

When this switch is ON with REP_ONCE switch ON (repeat once) the program is repeated once and then ends execution with the program that the END step is taught, without moving on to the next program.

If it is OFF, the program moves on to the next program after executing the END step and stops at the first step of that program.

Default setting is OFF.

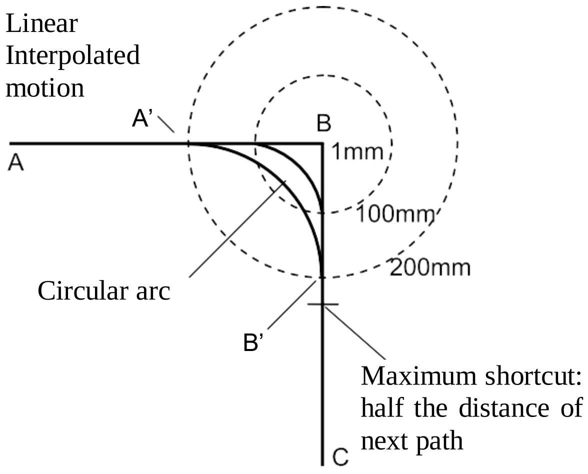
## DEST_CIRINT

### Function

Determines the pose where the DEST/#DEST function returns on the circular trajectory within the accuracy circle in Motion type 2.

### Explanation

The robot's motion trajectory when moving in linear interpolation motion along A→B→C in motion type 2 will be as shown in the figure below. This figure shows the example when the accuracy setting at point B is 1 mm, 100 mm, 200 mm. The robot starts to shortcut to the next path when as soon as it enters the accuracy circle of the set accuracy. The robot will follow a circular trajectory within the accuracy circle. See also "4.5.4 Relation between CP Switch and ACCURACY, ACCEL, and DECEL Instructions".


Linear Interpolated motion
A'
B
A
1mm
100mm
200mm
Circular arc
B'
Maximum shortcut: half the distance of next path
C

When DEST/#DEST functions are used in the following two timing, the returned pose can be changed by setting this switch ON or OFF.

1. When the robot is between point A and the point starting circular motion
2. When the robot is following the circular trajectory (within the accuracy circle)

When this switch is ON, at accuracy setting of 200 mm, point A' is returned at timing 1, and point B' is returned for timing 2.

When this switch is OFF, at accuracy setting of 200 mm, point B is returned at timing 1, and point C is returned for timing 2.

Default setting is ON.

## PROG.DATE

**Function**

Determines if the date of program modification is added to the program information or not.

**Explanation**

When this switch is ON, an item selection button is displayed in the program list screen. Pressing this button will display the total number of execution times and the program modification date.

When this switch is OFF, the item selection button is not displayed on the program list screen. The number of steps in the program and its comment are displayed instead of the program list.

Default setting is OFF.

## INSERT_NO_CONFIRM

**Function**

Selects whether confirmation message is displayed or not when insertion operation is done in teach operation.

**Explanation**

When this switch is ON, the confirmation message is not displayed when insertion operation is done in teach operation.

When this switch is OFF, the confirmation message is displayed when insertion operation is done in teach operation.

Default setting is OFF.

## SIGMON_TEACH

**Function**

Selects whether operation of hard key in O signal.  I/O name and KLogic monitor is nullify or not in teach repeat or teach lock disabled.

**Explanation**

When this switch is ON, signal operation from monitor screen is not allowed when repeat mode is enabled or teach lock is disabled. Operation can be done only when in teach/check mode or when teach lock is valid.

When this switch is OFF, the robot can be operated both in repeat mode and teach mode.

Default setting is OFF.

## TOUCH.ENA

### Function
Selects whether touch panel operation of the repeat conditions are enabled or disabled. When set to disable, avoids mistaken operation of the touch panel when the touch panel is touched accidently.

### Explanation
When this switch is ON, all the items of repeat conditions can be operated from the touch panel.

When this switch is OFF, touch panel operation is disable except for “speed specification”, “▲+10%”, “▲-10%”.  However operations using the cursor keys are possible.

Default setting is ON.

## TOUCHST.ENA

**Function**

Selects whether touch panel operation of status lamps (HOLD/RUN, Motor power, Cycle start) are enabled or disabled.  When set to disable, avoids mistaken operation of the touch panel when the touch panel is touched accidently.

**Explanation**

When this switch is ON, status lamp can be operated from touch panel.

When this switch is OFF, status lamp cannot be operated from touch panel.

Default setting is ON.

## TPSPEED.RESET

**Function**

Selects whether teach speed and check speed is set to slow speed automatically or not.

The timing slow speed is set is as follows:
(1) When switched from repeat mode to teach mode
(2) When Controller power is turned ON
(3) When emergency stop switch is pressed
Sow speed is not set when in inching operation.

**Explanation**

When this switch is ON, teach and check speeds are automatically changed to slow speed.

When this switch is OFF, teach and check speeds are not automatically changed to slow speed.

Default setting is ON.

## OXZERO

Option

**Function**

Selects whether external output signal (OX) collective reset function is enabled or disabled. When the function is enabled, “OX=0” can be taught and the OX signals of the taught step are reset together.

However, dedicated signals and interface panel signals cannot be reset.

This function is valid only when multi-function OXWX function option is valid.

**Explanation**

When this switch is ON, collective reset function is enabled.

When this switch is OFF, collective reset function is disabled.

Default setting is OFF.

## IFAKEY

**Function**

Selects whether the press buttons on the interface screen and press buttons with lamp are enabled only with A key or not.  Buttons in operation disable status and those set to operation disables cannot be operated.

**Explanation**

When this switch is ON, allows operation only when pressed together with A key.

When this switch is OFF, allows operation without pressing A key.

Default setting is OFF.

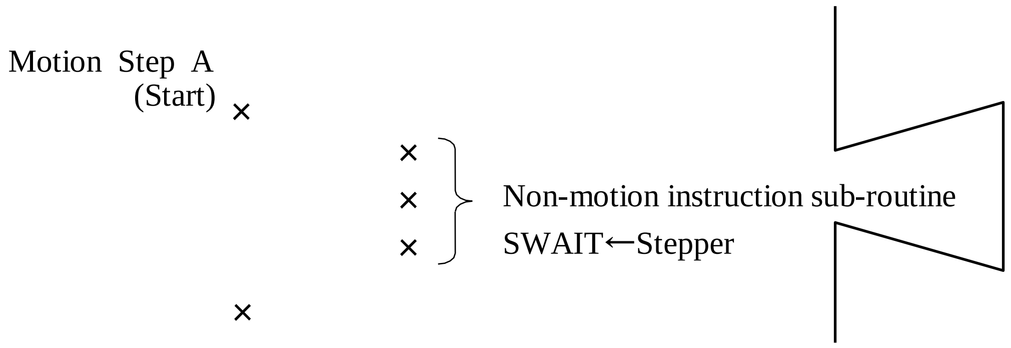
## DISP.EXESTEP

### Function

Displays the current stepper of the execution step during program execution instead of current motion step.

### Explanation

For example, if the program is executed in condition as below, when this switch is ON, the stepper step of SWAIT is displayed.　(SWAIT can be skipped.)　When this switch is OFF, motion instruction A is displayed.



Default setting is ON.

## NO_SJISCONV

**Function**

Selects whether the function to convert the character code between SJIS and EUC at time save/ load is enabled or disabled.

**Explanation**

When this switch is ON, character code conversion is conducted.

When this switch is OFF, character code conversion is not conducted.

Default setting is ON.

## CONF_VARIABLE

**Function**

Selects whether the configuration change is allowed at time of linear motion or not. Normally, the configuration is kept the same during the linear motion, but when the robot passes a singular point, some axes change greatly in the command value and robot may not be able to move.  Configuration can be changed to avoid this and pass the singular point.

**Explanation**

When this switch is ON, allows configuration during linear motion.

When this switch is OFF, does not allow configuration during linear motion.

Default setting is ON.

## INVALID.TPKEY_S

**Function**

Selects whether or not the shift status is invalidated when A key is pressed.

**Explanation**

When this switch is ON, pressing A key only turns ON “TPKEY_A” and “TPKEY_S” remains OFF.

When this switch is OFF, pressing A key turns ON both “TPKEY_A” and “TPKEY_S”.

Default setting is OFF.

## DIVIDE.TPKEY_S

**Function**

Selects whether the information of the two A keys on the teach pendant are allocated to switches “TPKEY_A” and “TPKEY_S” or not.

**Explanation**

When this switch is ON, the information of the two A keys are allocated to the switches as shown below:

Left A key: TPKEY_S

Right A key: TPKEY_A

When INVALID.TPKEY_S switch is ON, TPKEY_S switch is invalidated.

When INVALID.TPKEY_S switch is the information from the two A keys are united to one and allocated to both TPKEY_A and TPKEY_S switches.

Default setting is OFF.

## SIGRSTCONF

**Function**

Changes the number of signals that are reset when signal 0 is output via SIGNAL instruction or manual signal output.

**Explanation**

When this switch is ON, outputting signal 0 resets 1 to the number of external output signals set by ZSIGSPEC instruction (maximum of 960).

When this switch is OFF, outputting signal 0 resets 1 to 64 signals (1 to 256 when OX/WX signal expansion function is ON).

Default setting is ON.

## SINGULAR

### Function

Switches between singular point check function enable and disable.

### Explanation

When this switch is ON, checks whether the taught point or the command values for JT 5 are within range of $0^{\circ}$ to the threshold values (for example $5^{\circ}$) when moving in linear or circular motion.    If the value is within the range, error (E6007) “Wrist can't be straightened any more (Singular point 1).” occurs.

Default setting is ON (in some model the default setting is OFF).

## USE_ISO8859_5

**Function**

Changes the display font of ASCII8 bit from Latin font to Cyrillic font.

**Explanation**

When this switch is OFF, ASCII8 bit (0xA1 - 0xFF) is displayed in Latin font (ISO8859-1).
When this switch is ON, the font is displayed in Cyrillic font (ISO8859-5).

Default setting is ON.

## PCENDMSG_MASK

**Function**

Switches the ON/OFF of the message display at the time of the PC program completion.

**Explanation**

When this switch is OFF, a message “PC program completed.” appears at the time of the PC program completion.　When this switch is ON, the message does not appear.

Default setting is OFF.

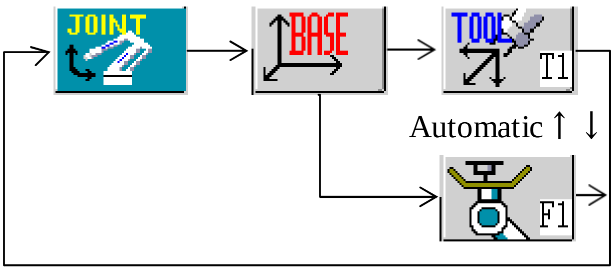
## INTERP_FTOOL

**Function**

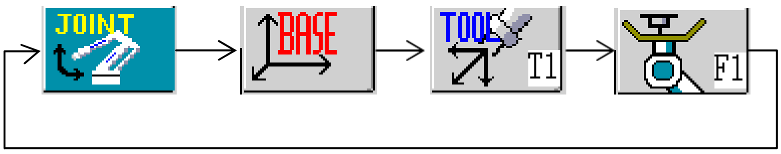
Switches between enabling and disabling the function to always select the fixed tool coordinates with the motion coordinates key.

**Explanation**

When this switch is disabled (OFF), the motion coordinates key switches to three levels by the pushing of the key button. TOOL or FTOOL is automatically determined by the interpolation type.



When this switch is enabled (ON), the motion coordinates key switches to four levels by the pushing of the key button.



This switch is off upon initialization.

# 8 Operators

This chapter describes how the operators function in AS language. These operators are used in conjunction with monitor commands and program instructions.

## 8.1 Arithmetic Operators

Arithmetic operators are used to perform general mathematic calculations.

| Operator | Function | Example |
|---|---|---|
| + | Addition | i = i + 1 |
| – | Subtraction | j= i – 1 |
| ∗ | Multiplication | i = i ∗ 3 |
| / | Division | i = i/2 |
| MOD | Remainder | i= i MOD 2 |
| ^ | Power | i = i ^ 3 |

**Example**

| | |
|---|---|
| i = i + 1 | The value of i plus 1 is assigned to i; e.g. when i is 5, 6 will be assigned to i as the result of the expression i + 1. |
| i = i MOD 2 | When i is 5, operator calculates 5 ÷ 2 and assigns to i the remainder of 1. |
| i = iˆ3 | The value of $i^3$ is assigned to i. When i is 2, 8 is assigned to i on the left side of the instruction. |

In division (/) and MOD, using 0 as the rightmost value of the instruction results in an error.

Example
i = i/0
i = i MOD 0 etc.

## 8.2 Relational Operators

Relational operators are used with instructions such as IF and WAIT to verify if a condition is set.

| Operator | Function | Example |
|---|---|---|
| < | TRUE (−1) when left side value is less than right side | i < j |
| > | TRUE (−1) when left side value is greater than right side | i > j |
| <= | TRUE (−1) when left side value is less than or equal to right side | i<= j |
| =< | Same as above | i =< j |
| >= | TRUE (−1) when left side value is greater than or equal to right side | i >= j |
| => | Same as above | i=> j |
| == | TRUE (−1) when the two sides are equal | i == j |
| <> | TRUE (−1) when the two sides are not equal | i <> j |

**Example**

| | |
|---|---|
| `IF i<j GOTO 10` | When j is greater than i, (i.e. the instruction i<j is true), the program jumps to the step labeled 10. If not, the program proceeds to the next step. |
| `WAIT INT(t)==INT(5)` | When t is 5 (i.e. t==5 is true), the program proceeds to the next step, if not, program execution is suspended until the condition is set. |
| `IF i+j>100 GOTO 20` | When i + j is greater than 100 (i.e. the expression i + j > 100 is true), the program jumps to the step labeled 20. If not, the program proceeds to the next step. |
| `IF $a =="abc" GOTO 20` | When $a is "abc" (i.e. $a == "abc" is true), the program jumps to the step labeled 20. If not, the program proceeds to the next step. |

**[ NOTE ]**

When comparing real values, do not use "==". In the AS software, real values are treated as decimal floating point. In decimal floating points, real values cannot be checked is they are equal using "==". To compare real values and integer values, use INT or ROUND functions.

## 8.3 Logical Operators

Logical operators are used in Boolean operations such as 0 + 1 = 1, 1 + 1 = 1, 0 + 0 = 0 (logical OR), or 0 × 1 = 0, 1 × 1 = 1, 0 × 0 = 0 (logical AND). There are two types of logical operators in AS language, logical operators and binary operators.

Logical operators are not used for calculating numeric values, but for determining if a value or an expression is true or false. If a numeric value is 0, it is considered FALSE (OFF). All nonzero values are considered TRUE (ON). Take note that the calculation using this operator returns −1 as TRUE.

| Operator | Function | Example |
|---|---|---|
| AND | Logical AND | i AND j |
| OR | Logical OR | i OR j |
| XOR | Exclusive logical OR | i XOR j |
| NOT | Logical complement | NOT i |

**Example**

i AND j Evaluates the logical AND between i and j. The variables i and j are generally logical values, but they can also be real number values. In this case, all real number values other than 0 are considered ON (TRUE).

| i | j | Result |
|---|---|---|
| 0 | 0 | 0 (OFF) |
| 0 | not 0 | 0 (OFF) |
| not 0 | 0 | 0 (OFF) |
| not 0 | not 0 | −1 (ON) |

The result is ON (TRUE) only when both values are ON (TRUE).

i OR j Evaluates the logical OR between i and j.

| i | j | Result |
|---|---|---|
| 0 | 0 | 0 (OFF) |
| 0 | not 0 | −1 (ON) |
| not 0 | 0 | −1 (ON) |
| not 0 | not 0 | −1 (ON) |

The result is ON (TRUE) when both or either of the two values are ON (TRUE).

i XOR j Evaluates the exclusive logical OR between i and j.

| i | j | Result |
|---|---|---|
| 0 | 0 | 0 (OFF) |
| 0 | not 0 | −1 (ON) |
| not 0 | 0 | −1 (ON) |
| not 0 | not 0 | 0 (OFF) |

The result is ON (TRUE) when only one of the two values is ON (TRUE).

NOT i Evaluates the logical complement of i.

| i | Result |
|---|---|
| 0 | −1 (ON) |
| not 0 | 0 (OFF) |

In AS, the logical status of a value or expression may be expressed as following:

| | |
|---|---|
| True: | not 0, ON, TRUE |
| False: | 0, OFF, FALSE |

## 8.4 Binary Operators

Binary logical operators perform logical operations for each respective bit of two numeric values. For example, if a number is composed of 4 bits, the values that will be calculated will be 0000, 0001, 0010, 0011, ……, 1111 (In AS, the numeric values are composed of 32 bits).

| Binary expression | Decimal expression |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| : | : |
| : | : |
| 1111 | 15 |

| Operator | Function | Example |
|---|---|---|
| BOR | Binary OR | i BOR j |
| BAND | Binary AND | i BAND j |
| BXOR | Binary XOR | i BXOR j |
| COM | Binary complement | COM i |

**Example**

i BOR j

If i=5, j=9, then the result is 13.

$$\begin{array}{ll} i=5 & 0101 \\ j=9 & \underline{1001\quad\quad} \\ & 1101 \Rightarrow 13 \end{array}$$

i BAND j

If i=5, j=9, then the result is 1.

$$\begin{array}{ll} i=5 & 0101 \\ j=9 & \underline{1001\quad\quad} \\ & 0001 \Rightarrow \quad 1 \end{array}$$

i BXOR j

If i=5, j=9, then the result is 12.

$$\begin{array}{ll} i=5 & 0101 \\ j=9 & \underline{1001\quad\quad} \\ & 1100 \Rightarrow 12 \end{array}$$

COM i

If i=5 then the result is –6.

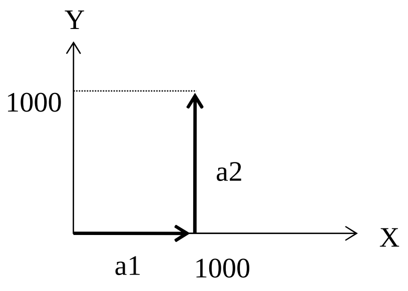
$$\begin{array}{ll} i=5\ 0\ldots & \underline{0101\quad\quad} \\ \quad\ \ 1\ldots & 1010 \Rightarrow -6 \end{array}$$

## 8.5 Transformation Value Operators

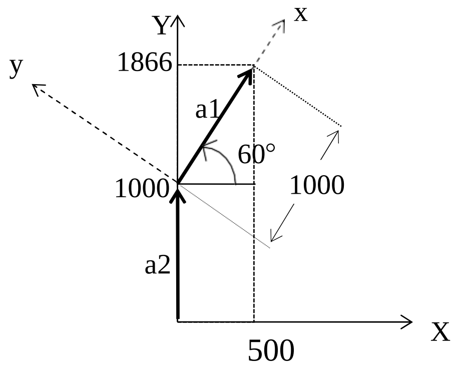
In the AS system, operators + and – are used to determine the compound transformation values (the XYZOAT values). However note that unlike the usual addition or subtraction, the commutative law does not hold true with the transformation operation. Arithmetic expression “a + b” and “b + a” will result the same, but “pose a + pose b” will not necessarily equal “pose b + pose a”. This is because in transformation operations, the values of the axes are taken into consideration. An example of this is shown below:

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

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



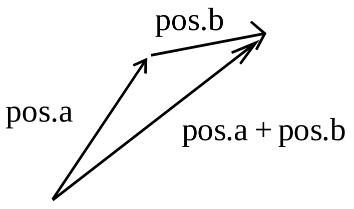
a2 + a1 = (500, 1866, 0, 60, 0, 0)



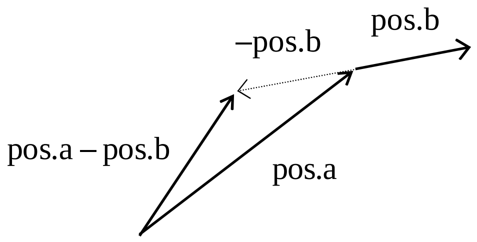
| Operator | Function | Example |
|---|---|---|
| + | Addition of two transformation values | pos.a + pos.b |
| – | Subtraction of two transformation values | pos.a – pos.b |

**Example**

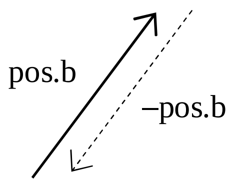
pos.a + pos.b



pos.a – pos.b



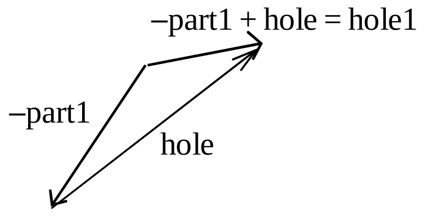
Transformation operator "–" used with a single value (e.g. –x) signifies the inverse value of x. For example, when the transformation value variable pos.b defines the pose of object B relative to object A, then –pos.b defines the pose of object A relative to object B.



In the example below, "hole1" is to be defined relative to "part1" (defined in advance).  This can be done using the compound transformation value variable part1+hole.

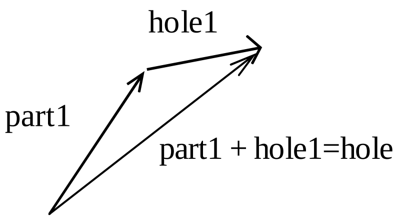
Move the robot to the pose to be defined "hole1" and teach that pose as "hole" using the HERE command.  Using this pose ("hole"), "hole1" can be defined.

POINT hole1 = – (part1) + hole



Another way to define "hole1" without using the operator "–" is by writing "hole1" in the left side of the expression in POINT command.  The following command also defines "hole1".

POINT part1 + hole1 = hole



## 8.6 String Operators

| Operator | Function | Example |
|---|---|---|
| + | Combines two strings | $a = $b + $c |

**Example**

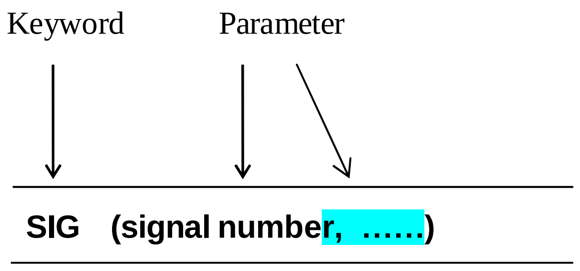
| | |
|---|---|
| $a=$b + $c | Combines $b + $c and assigns that string to $a. |

For example, when $b="abc" and $c="123", $a becomes "abc123".

# 9 Functions

This chapter describes the functions used in AS system. Functions are generally used in combination with monitor commands and program instructions. They are expressed in format described below. The keyword specifies the function, and the parameters entered in parentheses determine the value.

**EXAMPLE**



**SIG (signal number, ……)**

Parameters marked by can be omitted.

Always enter a space (blank) after the keyword.

## SIG (signal number, ……)

**Function**

Returns the logical AND of the specified binary signal status.

**Parameter**

Signal Number

Specifies the number of the external or internal I/O signal.

**Explanation**

Calculates logical AND of all the specified binary signal states and returns the resulting value. If all the specified signal states are TRUE, –1 (the value of TRUE) is returned. Otherwise 0 (the value of FALSE) is returned. External I/O signals or internal I/O signals as shown below, are specified by their numbers.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| External input signal | 1001–actual number of signals |
| Internal signal | 2001–2960 |

Signals specified by positive numbers are considered TRUE when they are ON, while signals specified by negative number are considered TRUE when they are OFF. No signal corresponds with signal number “0”, so it is considered always TRUE.

**[ NOTE ]**

There is a timing restriction when evaluating more than one signal at the same time. When more than one signal is input at the same time, note that there is approx. 2 ms difference in stabilization time of each signal.

**Example**

If the binary I/O signals 1001=ON, 1004=OFF, 20=OFF, then

| | Results | |
|---|---|---|
| SIG(1001) | –1 | TRUE |
| SIG(1004) | 0 | FALSE |
| SIG(–1004) | –1 | TRUE |
| SIG(1001,1004) | 0 | FALSE |
| SIG(1001,–1004) | –1 | TRUE |
| SIG(1001,–1004,–20) | –1 | TRUE |

## BITS (starting signal number, number of signals)

### Function

Reads consecutive binary signals and returns the decimal value corresponding to the bit patterns of the specified binary signals.

### Parameter

Starting signal number
Specifies the first signal to read.

Number of signals
Specifies the number of signals to read. The maximum number accepted is 16. To read more than 16 signals, use BIT32 function, explained next.

### Explanation

This function returns the decimal value corresponding to the bit pattern of the specified signals. In the binary expression of the value returned by this function, the least significant bit corresponds to the starting signal number.

Acceptable Signal Numbers

| | |
|---|---|
| External output signal | 1–actual number of signals |
| External input signal | 1001–actual number of signals |
| Internal signal | 2001–2960 |

**[ NOTE ]**

There is a timing restriction when evaluating more than one signal at the same time. When more than one signal is input at the same time, note that there is approx. 2 ms difference in stabilization time of each signal.

### Example

If the signal states are as follows, the result of the expression below will be 5.

```
x= BITS(1003,4)
```

The logical values of 4 signals starting from 1003 (i.e. 1003, 1004, 1005 and 1006) are read as a binary representation of the value, and x is 5 or 0101.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Signals: | 1008 | 1007 | 1006 | 1005 | 1004 | 1003 | 1002 | 1001 |
| State: | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

## BITS32 (starting signal number, number of signals)

### Function

Reads consecutive binary signals and returns the decimal value corresponding to the bit patterns of the specified binary signals.

### Parameter

Starting signal number
Specifies the first signal to read.

Number of signals
Specifies the number of signals to read. The maximum number accepted is 32.

### Explanation

This function reads the signal status of signal numbers specified from the starting signal number, and corresponding to the bit value arranged in ascending order, returns as the integer variable in decimals. In the binary expression of the returned value, the least significant bit corresponds to the starting signal number. When assigning the returned value to a variable, add @ to the front of the variable name to make it an integer variable name.

### Example

@x=BITS32(2001, 32) is interpreted as the binary representation of the value with 32 bits starting from 2001 (or signals from 2001 to 2032). The first 32-bit bit represents the sign in two's complement, so the returned value may be negative. Please refer to the below table for values returned for signal statuses.

| Internal signals 2032-2001 | | | | | | | | Returned value |
|---|---|---|---|---|---|---|---|---|
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0 |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | 1 |
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0011 | 3 |
| 0111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 2147483647 |
| 1000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | -2147483648 |
| 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | -1 |
| 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1111 | 1101 | -3 |

## TIMER (timer number)

### Function

Returns the value of the specified timer in seconds. Expresses timer value of the moment this TIMER function was executed.

### Parameter

Timer number
Specifies which timer to read. Acceptable numbers are 0 to10.

### Explanation

By using the TIMER function the timer value can be read at any time. Read and returns the time (in seconds) elapsed from the value previously set by the TIMER instruction. If no value has been set by the TIMER instruction, the value of TIMER 0 is returned.

Timer number 0 is for the system clock. The value returned by specifying this timer is the time elapsed since the system start up.

### Example

In the below example, TIMER instruction and real value function is used to measure the execution time of a subroutine.

```
TIMER (1)=0                                      Sets Timer 1 to 0.
CALL   test.routine                              Calls the subroutine.
PRINT "Elapsed time=", TIMER(1),"seconds"
```

## DISTANCE (transformation value variable 1, transformation value variable 2)

**Function**

Calculates the distance between two poses that are expressed in transformation values.

**Parameter**

Transformation values variable 1, transformation values variable 2

Specifies names of the two transformation value variables of which the distance between them is to be calculated.

**Explanation**

Returns the distance between two poses in millimeters. (The two poses can be entered in any order.)

**Example**

| | |
|---|---|
| k=DISTANCE(HERE,part) | Calculates the distance between the current TCP and the pose "part", and substitutes the result into k. |

## DX (transformation value variable)
## DY (transformation value variable)
## DZ (transformation value variable)

**Function**

Returns the transformation values (X, Y, Z) of the position defined by the specified pose variable.

**Parameter**

Transformation value variable

Specifies the name of the transformation value variable whose X, Y, or Z component is required.

**Explanation**

These three functions each returns the X, Y, or Z component of the specified pose.

**[ NOTE ]**

Each component of the transformation values can also be obtained using the DECOMPOSE instruction. The values for O, A, and T are obtained using the DECOMPOSE instruction.

**Example**

If the pose "start" has the transformation values of:

| X | Y | Z | O | A | T |
|---|---|---|---|---|---|
| 125, | 250, | –50, | 135, | 50, | 75 |

```
x=DX(start)        DX function returns x = 125.00
y=DY(start)        DY function returns y = 250.00
Z=DZ(start)        DZ function returns z = –50.00
```

## DEXT (pose variable, element number)

**Function**

Returns the specified element of the specified pose.

**Parameter**

Pose variable

Specifies the name of pose variable defined by joint displacement values or transformation values.

Element number

Specifies the element to be returned in real numbers, as shown in the figure below.

| Element number | Pose | |
|---|---|---|
| | Transformation values | Joint displacement values |
| 1 | X component | JT1 |
| 2 | Y component | JT2 |
| 3 | Z component | JT3 |
| 4 | O component | JT4 |
| 5 | A component | JT5 |
| 6 | T component | JT6 |
| 7 | JT7 | JT7 |

**Example**

If the transformation values for “aa” are 0, 0, 0, –160, 0, 0, 300, then inputting this function as:

```
type   DEXT(aa, 7)
```

This returns 300, the value of JT7.

## ASC (string, character number)

**Function**

Returns the ASCII value of the specified character in a string expression.

**Parameter**

String

Specifies the string that contains the character for which the ASCII value is required. If the string is a null string, or the number specified for the parameter “character number” exceeds the actual number of characters in the string, –1 is returned.

Character number

Specifies the number of the character counting from the beginning of the string. If not specified, or if 0 or 1 is specified, ASCII value of the first character of the string is returned.

**Explanation**

The ASCII value is returned in real values.

**Example**

| | |
|---|---|
| ASC("sample", 2) | Returns the ASCII value for character “a”. |
| ASC($name) | Returns the ASCII value for the first character of string “$name”. |
| ASC($system, i) | Returns the ASCII value of the i$^{th}$ character in the string variable “$system”. |

## LEN (string)

**Function**

Returns the number of characters in the specified string.

**Parameter**

String

Specifies a character string, character string variable, or string expression.

**Example**

| | |
|---|---|
| LEN("sample") | Returns the number of characters of the string “sample”, which is 6. |

. . . **TRUE**. . .
. . . **ON**. . .

**Function**

Returns the logical value for TRUE (–1).

**Explanation**

This function is convenient when it is necessary to specify the logical condition TRUE.

The results of functions TRUE and ON are the same. (Choose the function that best fits the needs of the program.)

. . . **FALSE**. . .
. . . **OFF**. . .

**Function**

Returns the logical value for FALSE (0).

**Explanation**

This function is convenient when it is necessary to specify the logical condition FALSE.

The results of functions FALSE and OFF are the same. (Choose the function that best fits the needs of the program.)

## VAL (string, code)

**Function**

Returns the real value in the specified string.

**Parameter**

String

Specifies character string, character string variable, or string expression.

Code

Expressed in real value or expression, specifies the notation of the value returned. If not specified, or if number other than 0, 1, or 2 is specified, 0 (decimal notation) is assumed.

| Numbers | Notation |
|---|---|
| 0 | Decimal |
| 1 | Binary |
| 2 | Hexadecimal |

**[ NOTE ]**

Scientific notation can be used in the string.

Codes that specify the notation (e.g. ^B and ^H) can be added to the beginning of the string.

All characters not read as a numeric value or code for notation are interpreted as characters marking the end of the string.

**Example**

| | |
|---|---|
| VAL("123 ") | Returns the real value 123. |
| VAL("123abc ") | Returns the real value 123. |
| VAL("12 ab 3 ") | Returns the real value 12. |
| VAL("1.2E-5") | Returns the real value 0.00001. |
| VAL("ˆHFF") | Returns the real value 255. (^H means hexadecimal notation. 16×15+15=255) |

## INSTR (starting point, string 1, string 2)

**Function**

Returns the place (in real value) where the specified string starts in the given string.

**Parameter**

Starting point

Specifies from where in string 1 to search for string 2. If not specified, the search starts from the beginning of string 1.

String 1

Expressed in character string, character string variable, or string expression, specifies the string where string 2 is searched.

String 2

Expressed in character string, character string variable, or string expression, specifies the string to search for. If a null string is specified, the value of the starting point is returned. 1 is returned if this string is not specified.

**Explanation**

This function returns the value of the starting point of string 2 in string 1, if string 2 is included in string 1.

The value 0 is returned if string 2 is not included in string 1.

If the value specified as the starting point is equal to or smaller than 1, the search starts from the beginning of string 1. If the value of the starting point is larger than the number of characters in string 1, 0 is returned.

Lower and upper case letters are not differentiated.

**Example**

| | |
|---|---|
| INSTR("file.ext",".") | Real value 5 is returned. |
| INSTR("file",".") | Real value 0 is returned. |
| INSTR("abcdefgh","DE") | Real value 4 is returned. |
| INSTR(5,"1-2-3-4","-") | Real value 6 is returned. |

## MAXVAL (real value 1, real value 2, ……)

**Function**

Compares the given real values and returns the largest among them.

**Parameter**

Real value 1, real value 2

Specifies the real values to compare.

## MINVAL (real value 1, real value 2, ……)

**Function**

Compares the given real values and returns the smallest among them.

**Parameter**

Real value 1, real value 2

Specifies the real values to compare.

## INT (numeric expression)

**Function**

Returns the integer of the specified numeric expression.

**Parameter**

Numeric expression

Returns the integer of the value.

**Explanation**

Returns the integer (i.e. left side of the decimal point if the value is not in scientific notation).
The negative sign remains with the integer unless the integer is 0.

**Example**

| | |
|---|---|
| INT(0.123) | 0 is returned. |
| INT(10.8) | 10 is returned. |
| INT(-5.462) | –5 is returned. |
| INT(1.3125E+2) | 131 is returned. |
| INT(cost+0.5) | The value of “cost” rounded down to the nearest integer is returned. |

## MAXINDEX (string variable, dimension number)

**Function**

Returns the value of the largest element in the specified dimension number of an array.

**Parameter**

String variable

Specifies the name of the array variable.

Dimension number

Specifies the dimension number. (1-3)

If the value is not specified between one and three, an error occurs. If not specified, 1 is assumed.

**Explanation**

Returns the value of the largest element in the specified dimension number of the array if the array has been already defined. Returns –1 if the variable is not an array. Returns –2 if the variable has not been defined.

**Example**

Variable #pos is expressed by joint displacement values and is an one-dimensional array from #pos[0] to #pos[100]. The dimension number is omitted.

ret= MAXINDEX (“#pos”) The value for variable ret is 100.

Variable #place is expressed by joint displacement values and is a two-dimensional array from #place[1,1] to #place[1,5].

ret=MAXINDEX (“#place”, 2) The value for variable ret is 5.

Variable #place is expressed by joint displacement values and is a three-dimensional array from #place[2,1,10] to #place[2,1,20].

ret=MAXINDEX (“#place”, 3) The value for variable ret is 20.

This program displays transformation values for variable pos. pos is a one-dimensional array.

```
.PROGRAM index()
max = MAXINDEX("pos",1)
min = MININDEX("pos",1)
FOR j = min TO max
        $val = "pos"+"["+$ENCODE(/I2,j)+"]"
        ret = EXISTTRANS($val)
        IF ret==FALSE THEN
                GOTO continue
        END
        DECOMPOSE x[0] = pos[j]
        TYPE "pos[",j,"] =",/F8.3,x[0],/X3,/F8.3,x[1],/X3,/F8.3,x[2]
continue:
END
.END
```

## MININDEX (string variable, dimension number)

**Function**

Returns the value of the smallest element in the specified dimension number of an array.

Parameter

String variable

Specifies the name of the array variable.

Dimension number

Specifies the dimension number. (1-3)

If the value is not specified between one and three, an error occurs. If not specified, 1 is assumed.

**Explanation**

Returns the value of the smallest element in the specified dimension number of the array if the array has been already defined.

Returns –1 if the variable is not arrays.

Returns –2 if the variable has not been defined.

**Example**

Variable #pos is expressed by joint displacement values and is a one-dimensional array from #pos[0] to #pos[100].

ret=MININDEX("#pos", 1) The value for variable ret is 0.

Variable #place is expressed by joint displacement values and is a two-dimensional array from #place[1,1] to #place[1,5].

ret=MININDEX("#place", 2) The value for variable ret is 1.

Variable #place is expressed by joint displacement values and is a three-dimensional array from #place[2,1,10] to #place[2,1,20].

ret=MININDEX("#place", 3) The value for variable ret is 10.

## SWITCH (switch name)

**Function**

Returns the current condition of the specified system switch.

**Explanation**

−1 is returned if the switch is ON, 0 is returned if it is OFF.

## WHICHTASK program name

**Function**

Returns the task number selected by the specified program (subroutine).

**Parameter**

Program name

Specifies the name of the program or subroutine in form of string variable.

**Explanation**

Returns the task number in real values.

1: Robot program (Robot 1)

2: Robot program (Robot 2)

1001: PC program 1 1004: PC program 4

1002: PC program 2 1005: PC program 5

1003: PC program 3

-1: Executed task does not exist.

**Example**

| | |
|---|---|
| `task_no=WHICHTASK($pg_name)` | Stores the tasks number in variable if the task selected by program $pg_name exists. If it does not exists, task_no= -1. |

## ERROR

### Function

Returns the error code of the current error.

### Explanation

Returns the error code when an error is currently occurring.   The value 0 is returned when no error is occurring.

Reread the error number as below:

| | |
|---|---|
| -4xxxx: | Dxxxx |
| -3xxxx: | Exxxx |
| -2xxxx: | Wxxxx |
| -1xxxx: | Pxxxx |

When -41500 is returned, error D1500 "Encoder misread error. JtXX" is displayed.

### Example

| | |
|---|---|
| type $ERROR (ERROR) | TYPE instruction displays the error message of the error code returned by the function ERROR. |

## PRIORITY

**Function**

Returns the priority number of the current robot program.

**Explanation**

Returns the priority number (in real value) of robot program currently selected on the stack. There is no priority setting among PC programs.

The default value for robot program priority is 0. The priority number can be changed via LOCK instruction.

## UTIMER (@timer variable)

**Function**

Returns the current value of the @timer variable set by UTIMER instruction.

**Parameter**

@timer variable

Specifies the name of the variable set by UTIMER instruction. An @ sign is added to the beginning of the variable name so that a integer variable can be specified.

## MSPEED
## MSPEED2

**Function**

Returns the current monitor speed (0 to 100%).

MSPEED is for Robot 1 and MSPEED2, for Robot 2.

## INRANGE (pose variable 1, joint displacement value variable)

**Function**

Checks if a pose is within the robot's motion range and returns a value depending on the result of this check (see the table below).

**Parameter**

Pose variable 1

Specifies which pose to check. (Joint displacement values, transformation values, or compound transformation values).

Joint displacement value variable

Specify a pose defined by joint displacement values. This parameter is entered only when the specified pose variable 1 is defined by transformation values or compound transformation values. The robot configuration is calculated by the pose variable 2 defined by joint displacement values. If not specified, the current configuration is used.

**Explanation**

The values returned by this function are as follows:

| | |
|---|---|
| 0 | Out of motion range. |
| 1 | JT1 is out of motion range. |
| 2 | JT2 is out of motion range. |
| 4 | JT3 is out of motion range. |
| 8 | JT4 is out of motion range. |
| 16 | JT5 is out of motion range. |
| 32 | JT6 is out of motion range. |
| 16384 | Beyond the collision check range. |
| 32768 | Out of reach of robot arm. |

**[ NOTE ]**

This function checks if the pose is in the motion range but does not check if the path to that pose is within the motion range.

**Example**

```
IF INRANGE(pos1, #p) GOTO ERR STOP
:
ERR_STOP:
TYPE "pose pos1is out of motion range."
PAUSE
```

Jumps to label ERR_STOP if pose pos1 is out of motion range, displays the message, and stops.

## SYSDATA (keyword, opt1, opt2)

**Function**

Returns specified parameters in the AS system according to the given keyword.

**Parameter**

Keyword, opt1, opt2

M.SPEED

Returns monitor speed (in percentage). If no motion step is being executed, –1 is returned.

Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.

Opt 2: Not used.

MSTEP

Returns the step number of the motion step in execution or the last executed motion step in the program in execution. If no such step exists, –1 is returned.

Opt 1: Robot number (1 to number of robot). If not entered, 1 is assumed.

Opt 2: Not used.

STEP

Returns the step number of the motion step in execution or the last executed motion step in the program in execution. If no such step exists, –1 is returned.

Opt 1: Robot number (1 to number of robot) or PC task number (1001 to number of PC programs). If not entered, 1 is assumed.

Opt 2: Not used.

P.SPEED

Returns the motion speed (in percentage) of the current motion or the next motion executed. If the speed is set in seconds, –1 is returned.

Opt 1: Robot number (1 to number of robot). If not entered, 1 is assumed.

Opt 2: Not used.

P.SPEED.M

Returns the motion speed (in MM/S) of the current motion or the next motion executed. If the speed is set in seconds, –1 is returned.

Opt 1: Robot number (1 to number of robot). If not entered, 1 is assumed.

Opt 2: Not used.

P.ACCEL
Returns the acceleration (in percentage) of the motion step in execution or the last executed motion step in the program in execution. If the motion speed is set in time (unit: S), –1 is returned.
Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.
Opt 2: Not used.

P.DECEL
Returns the deceleration (in percentage) of the motion step in execution or the last executed motion step in the program in execution. If the motion speed is set in time (unit: S), –1 is returned.
Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.
Opt 2: Not used.

MTR.RPM
Returns the rpm value for the motor speed (actual value) of the specified axis.
Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.
Opt 2: Axis number. JT 1 is selected when omitted.

MTR.RPM.CMD
Returns the rpm value for the motor speed (command value) of the specified axis.
Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.
Opt 2: Axis number. JT 1 is selected when omitted.

TOOL.VEL.CMD
Returns the mm/s value for the tool center point speed (command value) of the specified axis.
Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.
Opt 2: Not used.

JT.VEL.CMD
Returns the deg/s value for rotation axis or mm/s value for linear axis for the speed (command value) of the specified axis.
Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.
Opt 2: Axis number. JT 1 is selected when omitted.

NUMROBOT
Returns the number of robots connected.
Opt1: Not used.
Opt2: Not used.

ZROB.MGFNO

Returns the robot number.

opt1: Robot number (1 to number of robot). If not entered, 1 is assumed.

opt2: Not used.

ZROB.NOWAXIS

Returns the number of axis of the robot.

Opt1: Robot number (1 to number of robot). If not entered, 1 is assumed.

Opt2: Not used.

SIG.DO

Returns the number of external output signal.

Opt1: Not used.

Opt2: Not used.

SIG.DI

Returns the number of external input signal.

Opt1: Not used.

Opt2: Not used.

SIG.INT

Returns the number of internal signal.

Opt1: Not used.

Opt2: Not used.

LANGUAGE

Returns the number of the language selected for display. The language numbers are as follows.

| | | | | | |
|---|---|---|---|---|---|
| Japanese | 1 | English | 2 | Italian | 3 |
| French | 4 | German | 5 | Chinese | 6 |
| Korean | 7 | Polish | 9 | Spanish | 10 |
| Dutch | 11 | | | | |

Opt1: Not used.

Opt2: Not used.

MEM.FREE

Returns the size of the memory currently available in percentage.

Opt1: Not used.

Opt2: Not used.

CONT.NO

Returns the controller number.

Opt1: Not used.

Opt2: Not used.

MTR.CURR.CMD

Returns the current value (command value) as Arms value.

opt1: Robot number (1 to number of robots). Robot 1 is assumed if omitted.

opt2: Joint number. JT1 is assumed if omitted.

MTR.CURR

Returns the current value (feed-back) as Arms value.

opt1: Robot number (1 to number of robots). Robot 1 is assumed if omitted.

opt2: Joint number. JT1 is assumed if omitted.

POWER

Returns the integral power of operating data as kWh value.

opt1: 0: Integral power of consumption

1: Integral power of supply (power regeneration compatible models only)

2: Integral power of regeneration (power regeneration compatible models only)

opt2: Not used.

## EXISTDATA (“variable name or program name”, type)

**Function**

Checks if the variable or program of specified name exists in specified type and if it is set with an AND value.

**Parameter**

“Variable name or program name”

Specifies the variable name or program name to be checked whether it exists or not.

Type

Specifies the data type of variable or program to be checked whether it exists or not.

Data types are as follows:

| Specification | Data type |
|---|---|
| P | Joint displacement value |
| T | Converted value |
| R | Real-value |
| S | Character string |
| I | Integer |
| G | Program |

**Explanation**

Returns -1 if the specified variable or program exists, and 0 (zero) if doesn’t.

**[NOTE]**

Please be noted that array specifications have the following limitations:

1. Omission is impossible.
2. For the range, specify the small value on the left of a colon (:) and the big value on the right.

For example, when a real value r is three-dimensional and r[1,1,1], r[1,1,2], r[1,1,3] exist:

| | | |
|---|---|---|
| · When existence check is OK | r[1,1,1:3] | |
| · When existence check is not OK | r[1,1,1:] | ; omitted |
| | r[1,1,*] | ; omitted |
| | r[1,1,3:1] | ; range |

specification from large to small

**Example**

| | |
|---|---|
| ret=EXISTDATA(“#data”, P) | If the joint displacement value variable #data exists, ret is -1.<br>If it doesn’t exist, ret is 0 (zero). |
| ret=EXISTDATA(“data”, T) | If the converted value variable data exists, ret is -1.<br>If it doesn’t exist, ret is 0 (zero). |
| ret=EXISTDATA(“data”, R) | If the real variable data exists, ret is -1.<br>If it doesn’t exist, ret is 0 (zero). |
| ret=EXISTDATA(“$data”, S) | If the character string variable $data exists, ret is -1.<br>If it doesn’t exist, ret is 0 (zero). |
| ret=EXISTDATA(“@data”, I) | If the integer variable @data exists, ret is -1.<br>If it doesn’t exist, ret is 0 (zero). |
| ret=EXISTDATA(“data”, G) | If the program data exists, ret is -1.<br>If it doesn’t exist, ret is 0 (zero). |

## EXISTJOINT (“name of joint displacement value variable”)

**Function**

Checks if the specified pose variable exists as variable defined by joint displacement values.

**Parameter**

“Name of joint displacement value variable”

Specifies the name of joint displacement value variable in form of character string. The variable name should be enclosed in quotations. Start the name with #.

**Explanation**

If the variable exists, returns –1. If it does not exist, returns 0.

**[ NOTE ]**

There are restrictions as described below when specifying array variables.

1. Specify the area of array elements. This cannot be omitted.
2. The minimum value is written to the left of the colon “:” and the largest values is written to the right.

For example, if real variable r is a three-dimensional array and elements r[1,1,1], r[1,1,2], and r[1,1,3] exist,

| | | |
|---|---|---|
| Check will result OK | r[1,1,1:3] | |
| Check will result NG | r[1,1,1:] | ; Area not specified |
| | r[1,1,*] | ; Area not specified |
| | r[1,1,3:1] | ; Area specified in wrong order (from largest to smallest) |

**Example**

| | |
|---|---|
| ret=EXISTJOINT(“#pos”) | If joint displacement value variable #pos exists, ret= –1. If not, ret=0. |

The following shows a case where joint displacement value variable #place is a two-dimensional array from #place[1,1] to #place[1,5].

| | |
|---|---|
| ret=EXISTJOINT(“#place”) | ret will equal –1. |
| ret=EXISTJOINT(“#place[1,1]”) | #place[1,1]exists, so ret=-1. |
| ret=EXISTJOINT(“#place[1,1:5]”) | #place[1,1] to #place[1,5] exists, so ret=-1. |
| ret=EXISTJOINT(“#place[1,1:6]”) | #place[1,6] does not exist, so ret = 0. |

## EXISTTRANS ("name of transformation value variable")

**Function**

Checks if the specified pose variable exists as variable defined by transformation values.

**Parameter**

"Name of transformation value variable"

Specifies the name of transformation value variable in form of character string. The variable name should be enclosed in quotations.

**Explanation**

If the variable exists, returns –1. If it does not exist, returns 0.

See EXISTJOINT for restrictions and examples for specifying array variable.

**Example**

| | |
|---|---|
| ret=EXISTTRANS("pos1") | If transformation value variable pos1 defined by transformation values exists, ret=-1. If not, ret=0. |

## EXISTREAL (“real variable name”)

**Function**

Checks if the specified variable exists as real variable.

**Parameter**

“Real variable name”

Specifies the name of real variable in form of character string. The variable name should be enclosed in quotations.

**Explanation**

If the variable name exists, returns –1. If it does not exist, returns 0.

See EXISTJOINT for restrictions and examples for specifying array variable.

**Example**

ret=EXISTREAL(“pp”) If real variable pp exists, ret=-1. If not, ret=0.

## EXISTCHAR (“string variable name”)

**Function**

Checks if the specified variable exists as string variable.

**Parameter**

“String variable name”

Specifies the name of string variable in form of character string. The variable name should be enclosed in quotations. Start the name with $.

**Explanation**

If the string variable exists, returns –1. If it does not exist, returns 0.

See EXISTJOINT for restrictions and examples for specifying array variable.

**Example**

| | |
|---|---|
| ret=EXISTCHAR(“$val”) | If string variable $val exists, ret=-1.<br>If not, ret=0. |

## EXISTINTEGER ("integer variable name")

**Function**

Checks if the specified variable exists as integer variable.

**Parameter**

"Integer variable name"

Specifies the name of integer variable in form of character string. The variable name should be enclosed in quotations. Start the name with @.

**Explanation**

If the integer variable exists, returns –1. If it does not exist, returns 0.

See EXISTJOINT for restrictions and examples for specifying array variable.

**Example**

`ret=EXISTINTEGER("@abc")`  If integer variable @abc exists, ret=-1. If not, ret=0.

## EXISTPGM (“program name”)

### Function

Checks if the specified program exists or not.

### Parameter

“Program name”

Specifies program (or subroutine) name in form of string variable.

### Explanation

If the specified program or subroutine exists, returns –1. If it does not exist, returns 0 (zero).

### Example

| | |
|---|---|
| `ret=EXISTPGM(“pg1”)` | If program pg1 exists, ret=-1. If not, ret=0. |

## EXISTLOCALJOINT   (“name of local joint displacement value variable”)

**Function**

Checks if the specified local pose variable exists as variable defined by joint displacement values.

**Parameter**

“Name of local joint displacement value variable”

Specifies the name of local joint displacement value variable in form of character string.  The variable name should be enclosed in quotations.  Start the name with #.  Error occurs if variable other than local joint displacement value variable is specified.

**Explanation**

If the variable exists, returns –1.  If it does not exist, returns 0.

**[ NOTE ]**

There are restrictions as described below when specifying array variables.

1. Specify the area of array elements. This cannot be omitted.
2. The minimum value is written to the left of the colon ":" and the largest values is written to the right.

For example, if real variable r is a three-dimensional array and elements r[1,1,1], r[1,1,2], and r[1,1,3] exist,

| | | |
|---|---|---|
| Check will result OK | r[1,1,1:3] | |
| Check will result NG | r[1,1,1:] | ; Area not specified |
| | r[1,1,*] | ; Area not specified |
| | r[1,1,3:1] | ; Area specified in wrong order (from largest to smallest) |

**Example**

| | |
|---|---|
| ret=EXISTLOCALJOINT(".#pos") | If local joint displacement value variable #pos exists, ret= –1. If not, ret=0. |

The following shows a case where local joint displacement value variable #place is a two-dimensional array from #place[1,1] to #place[1,5].

| | |
|---|---|
| ret=EXISTLOCALJOINT(".#place") | .#place does not exist, so ret = 0. |
| ret=EXISTLOCALJOINT(".#place[1,1]") | .#place[1,1] exists, so ret=-1. |
| ret=EXISTLOCALJOINT(".#place[1,1:5]") | .#place[1,1] to .#place[1,5] exist, so ret=-1. |
| ret = EXISTLOCALJOINT(".#place[1,1:6]") | .#place[1,6] does not exist, so ret = 0. |

## EXISTLOCALTRANS (“name of local transformation value variable”)

**Function**

Checks if the specified pose variable exists as local variable defined by transformation values.

**Parameter**

“Name of transformation value variable”

Specifies the name of local transformation value variable in form of character string. The variable name should be enclosed in quotations. Error occurs if variable other than local transformation value variable is specified.

**Explanation**

If the variable exists, returns –1. If it does not exist, returns 0.

See EXISTLOCALJOINT for restrictions and examples for specifying array variable.

**Example**

| | |
|---|---|
| ret=EXISTLOCALTRANS(“.pos1”) | If local variable pos1 defined by transformation values exists, ret=-1. If not, ret=0. |

## EXISTLOCALREAL (“local real variable name”)

**Function**

Checks if the specified variable exists as local real variable.

**Parameter**

“Local real variable name”

Specifies the name of local real variable in form of character string. The variable name should be enclosed in quotations. Error occurs if variable other than local real value variable is specified.

**Explanation**

If the variable exists, returns –1. If it does not exist, returns 0.

See EXISTLOCALJOINT for restrictions and examples for specifying array variable.

**Example**

| | |
|---|---|
| ret=EXISTLOCALREAL(“.pp”) | If local real value variable .pp exists, ret=-1. If not, ret=0. |

## EXISTLOCALCHAR (“local string variable name”)

**Function**

Checks if the specified local variable exists as string variable.

**Parameter**

“Local string variable name”

Specifies the name of local string variable in form of character string. The variable name should be enclosed in quotations. Start the name with $. Error occurs if variable other than local string variable is specified.

**Explanation**

If the string variable exists, returns –1. If it does not exist, returns 0.

See EXISTLOCALJOINT for restrictions and examples for specifying array variable.

**Example**

| | |
|---|---|
| ret=EXISTLOCALCHAR(“.$val”) | If local string variable .$val exists, ret=-1.<br>If not, ret=0. |

## EXISTLOCALINTEGER (“local integer variable name”)

**Function**

Checks if the specified local variable exists as integer variable.

**Parameter**

“Local integer variable name”

Specifies the name of local integer variable in form of character string. The variable name should be enclosed in quotations. Start the name with @. Error occurs if variable other than local integer variable is specified.

**Explanation**

If the integer variable exists, returns –1. If it does not exist, returns 0.

See EXISTLOCALJOINT for restrictions and examples for specifying array variable.

**Example**

| | |
|---|---|
| ret=EXISTLOCALINTEGER (“.@abc”) | If integer variable @abc exists, ret=-1.<br>If not, ret=0. |

## STRTOPOS (string variable)

**Function**

Returns the value of the pose variable that is specified by the string variable.

**Parameter**

String variable

Specifies a character string variable to get the specified pose values.

**Explanation**

Returns the value of the pose variable if a pose variable has been already assigned to the string variable. If a pose variable has not been assigned, an error occurs.

**Example**

```
HERE  #pos
$A = “#pos”
```

| | |
|---|---|
| JMOVE STRTOPOS($A) | The string value “$A” specifies “#pos”. The robot moves to the destination which was described by joint displacement values “#pos”. If “#pos” has not been defined, an error occurs. |

## STRTOVAL (string variable)

**Function**

Returns the real value specified by the string variable.

**Parameter**

String variable

Specifies character string variable to get the specified real value.

**Explanation**

Returns the real value if a real variable has been already assigned to the string variable. If a real variable has not been assigned, an error occurs.

**Example**

```
VAR = 5
$VA = “VAR”
```

| | |
|---|---|
| `total = STRTOVAL($VA)+6` | The string variable “$VA” specifies the real variable “VAR”. The real variable “total” is eleven as “VAR” is five. If the variable “VAR” has not been defined, an error occurs. |

## ROUND (numeric value)

**Function**

Returns the value rounded at the first decimal place.

**Parameter**

Numeric value

This value is rounded at the first decimal place.

**Explanation**

Returns the value rounded at the first decimal place of the value specified as the parameter. When the specified value is a negative value, the value is rounded as an absolute value and then, the negative sign is added. The sign of the numeric value specified as the parameter remains unchanged unless the result is 0.

**Example**

| | |
|---|---|
| ROUND (0.123) | Returns 0. |
| ROUND (10.8) | Returns 11. |
| ROUND (-5.462) | Returns -5. |
| ROUND (-5.662) | Returns -6. |

## IQARM (axis number)

**Function**

Returns the motor current value of the axis with the specified number.

**Parameter**

Axis number

Specify the number of the axis to acquire the motor current value. Acceptable range: 1- to the number of axes set.

**Explanation**

Returns the motor current value for the axis with the number specified in the parameter. Unit is in Arms.

Error occurs when used under the below condition:

When axis number of Mitsubishi motor is specified, error “(E1145) Cannot use specified channel, already in use.” occurs if this function is used when monitoring of the motor current value is conducted by WHERE command, etc.

**Example**

| | |
|---|---|
| a = IQARM(1) | Returns the motor current value of JT1 and substitutes it to a. |

## TRQNM　(axis number)

### Function

Returns the torque value of the axis with the specified number.

### Parameter

Axis number

Specify the number of the axis to acquire the torque value.　Acceptable range: 1- to the number of axes set.　This function cannot be used for axis with Mitsubishi motor.

### Explanation

Returns the torque value for the axis with the number specified in the parameter.　Unit is in N·m.

### Example

```
        a = TRQNM(1)          Returns the torque value of JT1 and substitutes it to a.
```

## CURLIMM (axis number)

**Function**

Acquires the negative limit value for the motor current of the external axis.

**Parameter**

Axis number

Specify the number of the external axis. Acceptable range: 7- 18.

**Explanation**

Acquires the limit value for the negative current of the external axis motor set by CURLIM instruction in form of percentage to the servo parameter current limit value. Unit is in %. Range of acquisition: 0 – 100.

This function is valid only for external axis using KHI amplifier.

Refer to CURLIM instruction for setting of current limit value.

**Example**

| | |
|---|---|
| CURLIM 7,10,20 | Sets the current limit value for JT7. |
| curm7 = CURLIMM(7) | Acquires the negative value for the set current limit value and stores it in the variable.<br><br>In this example, 20 is stored in “curm7”. |

## CURLIMP (axis number)

**Function**

Acquires the positive limit value for the motor current of the external axis.

**Parameter**

Axis number

Specify the number of the external axis. Acceptable range: 7-18.

**Explanation**

Acquires the limit value for the positive current of the external axis motor set by CURLIM instruction in form of percentage to the servo parameter current limit value. Unit is in %. Range of acquisition: 0 – 100.

This function is valid only for external axis using KHI amplifier.

Refer to CURLIM instruction for setting of current limit value.

**Example**

| | |
|---|---|
| CURLIM 7,10,20 | Sets the current limit value for JT7. |
| curp7 = CURLIMP(7) | Acquires the positive value for the set current limit value and stores it in the variable.<br><br>In this example, 10 is stored in “curp7”. |

## ENVCHKRATE (axis number)

**Function**

Acquires the set value for the magnification ratio to the initial threshold value to detect the deviation abnormality of the external axis.

**Parameter**

Axis number

Specify the number of the external axis. Acceptable range: 7- 18.

**Explanation**

Acquires the value set in ENVCHKRATE instruction for the magnification ratio to the initial threshold value for detection of deviation abnormality in external axis.

This function is valid only for external axis using KHI amplifier.

Refer to ENVCHKRATE instruction for the setting of magnification ratio to the deviation error.

**Example**

| | |
|---|---|
| `ENVCHKRATE 7, 0.1` | Sets the magnification ratio to the deviation error detection threshold in JT7. |
| `env7=ENVCHKRATE(7)` | Stores the acquired magnification ratio to the variable.<br><br>In this example, 0.1 is stored in “env 7”. |

## GETENCTEMP (axis number)

**Function**

Returns the temperature [°C] of the encoder of the axis of the specified number.

**Parameter**

Axis number

Specify the number of axis to acquire the encoder temperature. Acceptable range: 1- to the number of axes set.

**Explanation**

Returns the encoder temperature [°C] of the specified axis.

If the specified axis is disconnected, 0 is returned.
When the axis number is omitted, the specified axis number does not exist, or if the specified axis is an external axis not using KHI amplifier, error occurs and the program stops.

**Example**

The examples below acquire the encoder temperature for JT4 of Robot 1.

Example of monitor command

```
>x = GETENCTEMP(4)
>PRINT x
>55.75
```

Example of program

```
.PROGRAM enctemp.pc()
   X = GETENCTEMP(4)
   TYPE X
.END
```

## OPEINFO (data number, robot number, joint number)

**Function**

Returns operating data corresponding to data numbers.

**Parameter**

Data number

Specifies the acquiring operating data by data numbers.

Data numbers and correspondence of operating data are described in the below table.

Robot number

When one controller is controlling multiple robots, specifies the robot number of operating data to be acquired.

When omitted, 1 is assumed.

Joint number

Specifies the joint number of operating data to be acquired. When omitted, 1 is assumed.

| Data number | Operating data | Robot number | Joint number | Unit |
|---|---|---|---|---|
| 1 | Hour meter | Not used | Not used | H |
| 2 | Controller power-ON time | 1 - | Not used | H |
| 3 | Servo-ON time | 1 - | Not used | H |
| 4 | Frequency of motor-ON | 1 - | Not used | Number of times |
| 5 | Frequency of servo-ON | 1 - | Not used | Number of times |
| 6 | Frequency of emergency stop (moving) | 1 - | Not used | Number of times |
| 7 | Integral power of consumption | 1 - | Not used | kWh |
| 8 | Integral power of supply | 1 - | Not used | kWh |
| 9 | Integral power of regeneration | 1 - | Not used | kWh |
| 10 | Joint moving time | 1 - | 1 - | H |
| 11 | Joint total displacement | 1 - | 1 - | X1000 deg, mm |
| 12 | Joint total displacement (+) | 1 - | 1 - | X1000 deg, mm |
| 13 | Joint total displacement (-) | 1 - | 1 - | X1000 deg, mm |

**Explanation**

This function returns the operating data displayed by the OPEINFO command as a real value.

**Example**

| | |
|---|---|
| OPEINFO(10,1,1) | Returns the joint moving time of JT1, robot number 1. |

## INS_POWER (power number)

### Function

Returns the instantaneous power corresponding to the power number.

### Parameter

Power number

Specifies the instantaneous power to be acquired by power number. Power numbers and correspondence of instantaneous power are described in the table below.

| Power number | Instantaneous power | Unit |
|---|---|---|
| 0 | Instantaneous power of consumption | kW |
| 1 | Instantaneous power of supply | kW |
| 2 | Instantaneous power of regeneration | kW |

### Explanation

Returns the instantaneous power [kW] corresponding to the power number specified by the parameter. For power regeneration-incompatible models, 0 is returned for instantaneous power of regeneration.

### Example

power = INS_POWER(0)      Returns instantaneous power of consumption to power.

## DEST

## #DEST

### Function

DEST: Returns the transformation values of the destination of current robot motion.
#DEST: Returns the joint displacement values of the destination of current robot motion.

### Explanation

By using these functions, the robot destination can be found out after the robot motion is interrupted for some reason. These functions can be used with all robot motions.

**[ NOTE ]**

The pose where the robot stops and the pose returned by DEST/#DEST functions are not always the same. For example, if the HOLD/RUN state is changed from RUN to HOLD, the robot stops immediately, but the pose returned by DEST/#DEST functions describes the pose the robot was heading for at that moment.

### Example

| | |
|---|---|
| POINT #old=#DEST | Stores the destination as a pose variable called #old. |
| dist=DISTANCE(DEST,HERE) | Calculates the distance between current position and destination, and assigns to dist. |

**FRAME (transformation value variable 1, transformation value variable 2, transformation value variable 3, transformation value variable 4)**

**Function**

Returns the transformation values of the frame (relative) coordinates with respect to the base coordinates. Note that only the translational components of transformation values of the pose variables are used as positional information to determine the frame coordinates.

**Parameter**

Transformation value variable1, transformation value variable 2

Specifies transformation value variables to determine the direction of the X axis. The X axis of the frame coordinates is set so that it passes through these two poses. The positive direction of the X axis is set in the direction from pose determined by transformation values variable 1 to pose determined by transformation value variable 2.

Transformation value variable 3

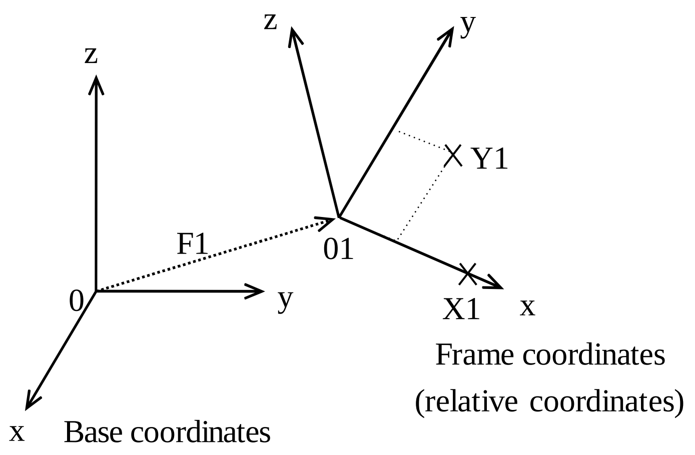
Specifies a transformation value variable to determine the direction of the Y axis. The Y axis of the frame coordinates is set so that the three points, pose1, pose 2, and pose 3, each determined by transformation value variables 1, 2 and 3, are on the XY plane. Also, pose 3 is set so it has the positive Y value.

Transformation value variable 4

Specifies a transformation value variable to specify the origin of the frame coordinates, which equals the values of X,Y,Z returned by this function.

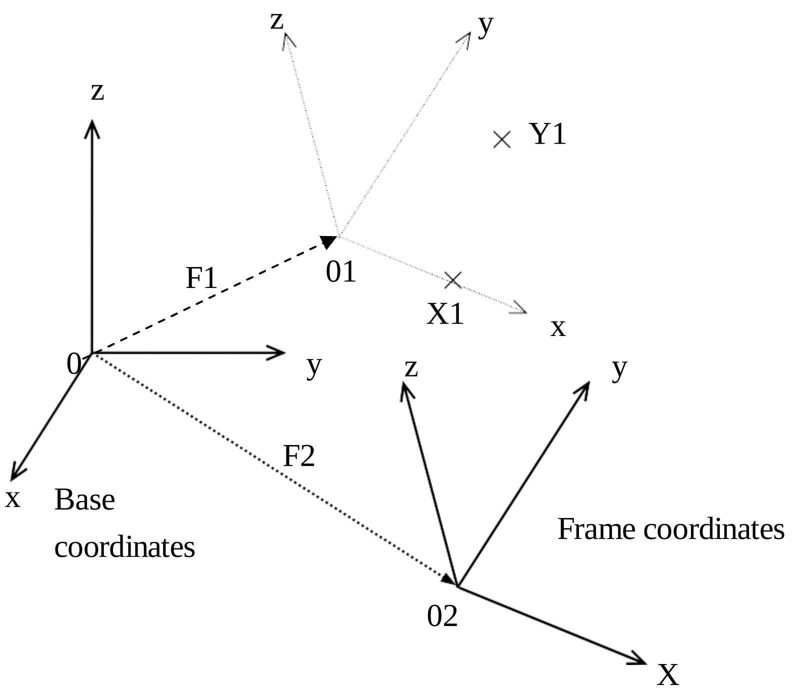
**Explanation**

POINT F1=FRAME(O1, X1, Y1, O1) Sets frame coordinates as in the diagram below.



If the poses in the frame coordinates are taught as F1+A, then only F1 needs re-teaching if the coordinates change, as when the parts station is moved. (See 11.6 Relative Pose Using the Frame Coordinates.)

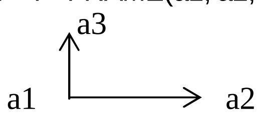
POINT F2=FRAME(O1, X1, Y1, O2) Sets frame coordinates as shown in the diagram below.

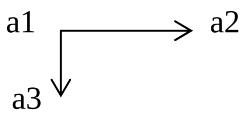


[ **NOTE** ]

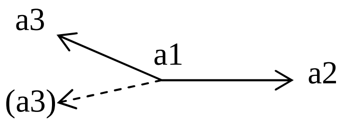
Pay attention to the following points when defining frame coordinates.

```
POINT   F=FRAME(a1, a2, a3, a1)
```





Note that the Y and Z axes in the above two frame coordinates face opposite directions, depending on where a3 is taught. Therefore, when a3 is taught close to the X axis (see diagram below), the direction of the Z axis may not result in the desired direction, so always check the frame coordinates before using the coordinates in any programs.



The three points a1, a2, a3 defines the position of the tool coordinates origin. When redefining the frame coordinates, the tool transformation must be the same as when a1, a2, a3 were first taught.

For better accuracy, teach the three points a1, a2, a3 as far apart as possible. Especially, a3 should be a point near the Y axis but as far away from the X axis as possible.

When teaching a1, a2, a3, the origin of the tool coordinates should be defined at a point that is easy to see, e.g. the tip of the tool, etc.

With some tools, it is difficult to determine the origin of the tool coordinates even when it has been moved by tool transformation. In such case, a1, a2, a3 are taught at null tool, but note that in this case, the origin of the tool coordinates is at the center of the flange surface.

## NULL

**Function**

Returns the null transformation values.

**Explanation**

Returns the transformation values in which both the translational components and the rotational components are all 0 (X=Y=Z=0, O=A=T=0). This function is convenient when used with the SHIFT function enabling easy redefinition of transformation values. Coordinates can be shifted in translation movement without any change in rotational components (OAT).

**Example**

```
POINT   new=SHIFT(NULL BY x.shift,y.shift,z.shift)+old
```

Defines the variable “new” by shifting the pose defined as “old” a specified distance along the base coordinates.

```
dist=DISTANCE(NULL, test.pos)
```

Calculates the distance between the pose “test.pos” and the null origin of the robot (0,0,0,0,0,0), and assigns that value to dist.

## HERE

## #HERE

**Function**

HERE: Returns the transformation values which describe the current pose of the tool coordinates.

#HERE: Returns the joint displacement values which describe the current pose of the tool coordinates.

**Explanation**

The encoder values at the moment this function is executed are read. Therefore, note that the values returned by this function represent the pose of the robot when the function was executed.

**[ NOTE ]**

The name "here" cannot be used as a program name or a variable name.

**Example**

| | |
|---|---|
| dist=DISTANCE(HERE, pos1) | Calculates the distance between the pose "pos1" and the current pose and assigns the value to "dist". |

## TRANS (X component, Y component, Z component, O component, A component, T component)

**Function**

Returns the transformation value that has the specified translational and rotational components.

**Parameter**

X component, Y component, Z component

Specifies the translation components X, Y, Z.  If not specified 0 is assumed.

O components, A component, T component

Specifies the rotation components O, A, T.  If not specified 0 is assumed.

**Explanation**

The transformation values are calculated from the values specified in each parameter.  The new transformation values can be used then to define pose variables, in compound transformation values, or in motion instructions.  This function is convenient when used with DECOMPOSE instruction (see the example for #PPOINT function).

**Example**

POINT temp.pos=TRANS(v[0], v[1]+100, v[2], v[3], v[4], v[5])

Array variable v[0] –

**RX (angle)**
**RY (angle)**
**RZ (angle)**

**Function**

Returns the transformation values that represent the rotation around the specified axis.

**Parameter**

Angle

Specifies the value of the rotation in degrees.

**Explanation**

The X, Y, Z in this function represents the axes of coordinates. The rotational value around the specified axis is returned. The translational values are not returned by this function (X, Y, Z = 0).

**Example**

| | |
|---|---|
| POINT x_rev =RX(30) | Returns the transformation value that represent 30° rotation around the X axis and assigns the value to "x_rev". |

## #PPOINT (jt1, jt2, jt3, jt4, jt5, jt6)

**Function**

Returns the specified joint displacement values.

**Parameter**

| | |
|---|---|
| jt1<br>jt2<br>jt3<br>:<br>jt6 | Specifies the value of each angle (in degrees for rotational joints)<br>If not specified, 0 is assumed. |

**Explanation**

This function returns the specified joint displacement values. The values represent the displacement of each joint, from joint 1 to the last joint (not necessarily six).

**[ NOTE ]**

#PPOINT function is only processed in joint displacement values. Therefore, always enter "#" at the beginning of the function.

**Example**

In the program below, joints 2 and 3 of a six-joint robot are moved the specified amount from the current pose.

```
HERE  #ref                      Stores the current pose in memory.(#ref)
DECOMPOSE x[0]=#ref             Decomposes each component into elements of
                                array variable x[0],……x[5].

JMOVE #PPOINT (X[0], x[1]+a, x[2]-a/2, x[3], x[4], x[5])
```

These two instructions result in the same pose as the program above, but unlike the program, the two joints do not move at the same time.

```
DRIVE   2, a, 100               Move jt2 by a°.
DRIVE 3, -a/2, 100              Move jt3 by – a/2°.
```

## SHIFT (transformation value variable BY X shift, Y shift, Z shift)

**Function**

Returns the transformation values of the pose shifted by the distance specified for each base axis (X,Y,Z) from the specified pose.

**Parameter**

Transformation value variable

Specifies a transformation value variable for the pose to be shifted.

X shift, Y shift, Z shift

Specifies the shift amount in X, Y, Z directions of the base coordinates. If any value is omitted, 0 is assumed.

**Explanation**

The X shift, Y shift, Z shift amounts are added to each of the X, Y, Z component of the specified transformation value variable. The result is returned in transformation values.

**Example**

If the values of the transformation value variable x are (200, 150, 100, 10, 20, 30), then

```
POINT    y=SHIFT(x BY 5, -5, 10)
```

“x” is shifted by the specified values to (205, 145, 110,10, 20, 30) and those values are assigned to transformation value variable “y”.

## AVE_TRANS (transformation value variable 1, transformation value variable 2)

**Function**

Returns the average values of the two transformation value variables 1 and 2.

**Parameter**

Transformation value variable 1, transformation value variable 2
Specifies the two transformation value variables to calculate the average between them.

**Explanation**

This function calculates the transformation values of the pose which defines the midpoint for each of the components of transformation value variables 1 and 2.

This function is commonly used for calculating the average of pose information gained from sensor checks.

**[ NOTE ]**

For the XYZ components, the average is given by adding each of the components and dividing them by 2. However, for the OAT components, the average is not necessarily given in that manner.

**Example**

```
POINT x = AVE_TRANS(p,q)
JMOVE AVE TRANS(p,q)
```

## BASE

### Function

Returns the current base transformation values.

### Example

| | |
|---|---|
| point a = BASE | Assigns to variable “a” the current base transformation values. |

## TOOL

### Function

Returns the current tool transformation values.

### Example

| | |
|---|---|
| `point aa = TOOL` | Assigns to variable “aa” the current tool transformation values. |

## TRADD (transformation value variable)

Option

**Function**

Returns the sum of the traverse axis value and the X component of the specified transformation value variable.

**Parameter**

Transformation value variable

Specifies the name of the transformation value variable to whose X component the traverse axis value is added.

## TRSUB (transformation value variable)

Option

**Function**

Returns the value gained by subtracting traverse axis value from the X component of the specified transformation value variable.

**Parameter**

Transformation value variable

Specifies the name of the transformation value variable from whose X component the traverse axis is subtracted.

## #HOME (home pose number)

**Function**

Returns the currently set home pose.

**Parameter**

Home pose number

Specifies the home pose number.

1: Specifies home pose 1 set by SETHOME command.

2: Specifies home pose 2 set by SET2HOME command.

If not specified, 1 is selected.

**Explanation**

Returns the pose of the currently set home pose in joint displacement values.

**[ NOTE ]**

#HOME function is only processed in joint displacement values. Therefore, always enter "#" at the beginning of the function.

**Example**

In the program below, the robot does not move in the direct path but first moves to the same height as the home pose (in the direction of the Z axis only), and then it moves to the home pose.

```
POINT homepos = #HOME(1)
IF DZ(homepos) > DZ(HERE) THEN
HERE tmp
POINT/Z tmp = homepos
LMOVE tmp
END
HOME
```

## CCENTER (transformation value variable1, transformation value variable 2, transformation value variable 3, transformation value variable 4)

Option

**Function**

Returns the center of the arc created by the three specified pose.

**Parameter**

Transformation value variable 1, transformation value variable 2, transformation value variable 3
Specifies pose variables defined by transformation values to determine the three points on the arc.

Transformation value variable 4
Specifies the pose variable to determine the orientation of the robot.

**CSHIFT (transformation value variable1, transformation value variable 2, transformation value variable 3, transformation value variable 4 BY shift amount)**

Option

**Function**

Returns the pose that was shifted the specified amount from the object pose.

**Parameter**

Transformation value variables 1, 2, 3

Specifies transformation value variables to determine the three points on the arc.

Transformation value variables 4

Specifies a transformation value variable to specify the object pose in transformation values.

Shift amount

Specifies the amount to shift in real values.

## ABS(real value)

**SQRT(real value)**

**PI**

**SIN(real value)**

**COS(real value)**

## ATAN2(real value1, real value 2)

## RANDOM

| Keyword | Function | Example |
|---|---|---|
| ABS | Returns the absolute value of a numerical expression. | ABS(value) |
| SQRT | Returns the square root of a numerical expression. | SQRT(value) |
| PI | Returns the constant π(3.1415‥‥‥). | PI |
| SIN | Returns the sine value of a given angle. | SIN(value) |
| COS | Returns the cosine value of a given angle. | COS(value) |
| ATAN2 | Returns the values of an angle (in degrees) whose tangent equals v1/v2. | ATAN2(v1,v2) |
| RANDOM | Returns a random number from 0.0 to 1.0. | RANDOM |

**Example**

| | |
|---|---|
| x=ABS(y) | substitutes the value \|y\| into x |
| x=SQRT(y) | substitutes the square root of y into x |
| en=2∗PI∗r | substitutes the result of 2πr into en Z=(SIN(x) |
| ˆ 2)+(COS(y) ˆ 2) | substitutes the result of $(\sin(x))^2+(\cos(y))^2$ into z |
| slope=ATAN2(rise,run) | substitutes the result of $\tan^{-1}$(rise/run) into slope |
| r=RANDOM∗10 | substitutes a random number from 0 to 10 into r |

## $CHR (real value)

### Function

Returns the ASCII character string corresponding to the specified ASCII value.

### Parameter

Real value (or numeric expression)

Specifies the value to change into an ASCII character. Acceptable range: 0 to 255.

### Explanation

Returns a corresponding ASCII character string if the specified ASCII value is between 0 and 255.

### Example

| | |
|---|---|
| $CHR(65) | Returns “A” for ASCII value 65. |
| $CHR(ˆH61) | Returns “a” for ASCII value 97 (16×6+1). |

## $SPACE (number of blanks)

### Function

Returns the specified number of blanks.

### Parameter

Number of blanks

Specifies the number of blanks. Specify 0 or a positive value.

### Example

| | |
|---|---|
| Type ”a” + $SPACE(1) + “dog” | Displays “a dog” (1 space is entered between “a” and “dog”. |

## $LEFT (string, number of characters)

**Function**

Returns the specified number of characters starting from the leftmost character of the specified string.

**Parameter**

String

Character string, string variable, or string expression.

Number of characters

Specifies how many characters to return counting from the leftmost (or first) character of the entered string. If 0 or a negative number is specified, blank is returned. If the number specified is larger than the number of characters in the string, the whole string is returned.

**Explanation**

Returns the character strings for the number of characters counting from the left of the string.

**Example**

| | |
|---|---|
| $LEFT ("abcdefgh",3) | Returns the string "abc". |
| $LEFT ("*1*2*3*4*5",15) | Returns "*1*2*3*4*5" ( the whole string). |

## $RIGHT (string, number of characters)

**Function**

Returns the specified number of characters starting from the rightmost character of the specified string.

**Parameter**

String

Character string, string variable, or string expression.

Number of characters

Specifies how many characters to return counting from the rightmost (or last) character of the entered string. If 0 or a negative number is specified, blank is returned. If the number specified is larger than the number of characters in the string, the whole string is returned.

**Explanation**

Returns the character strings for the number of characters counting from the right of the string.

**Example**

| | |
|---|---|
| $RIGHT("abcdefgh",3) | Returns the string "fgh". |

## $MID (string, real value, number of characters)

**Function**

Returns the specified number of characters from the specified string.

**Parameter**

String

Character string, string variable, or string expression.

Real values (or numeric expression)

Specifies the starting position of the string is to be taken.

Number of character

Specifies the number of characters to extract.

**Explanation**

If the starting position is not specified, or specified by a value of 1 or less, the characters are extracted from the first character of the string. If the starting position is specified by a value of 0 or less, or if the number is larger than the number of characters in the string, blank is returned.

If the number of characters to extract is not specified, or when it is larger than the number of characters in the string, the characters from the specified starting position to the end of the string is returned.

**Example**

In the instruction below, the $MID function returns “cd” (two characters starting from the third character in the string “abcdef”). Then the result is substituted into string variable $substring.

```
$substring＝$MID(“abcdef”,3,2)
```

## $DECODE (string variable, separator character, mode)

**Function**

Returns the string separated by “separator characters”.

**Parameter**

String variable

Specifies the string from where the characters are taken. Characters extracted as a result of this function are removed from this string.

Separator character

Specifies the character to read as separator. (Any character in the string can be specified as separator.)

Mode

Specifies the real number for operation done by this function.

If the mode is a negative number or 0, or if it is not specified, the characters starting from the first character in the string variable to the separator are returned. The returned string is removed from the string variable. If a positive number specifies the mode, the first separator that appears in the string is returned. The returned separator is removed from the value of the string variable. If more than one separator characters exist in the string consecutively, all the separator characters are returned and removed from the string variable.

**Explanation**

This function searches the specified string for the separator character and extracts the characters from the beginning of the string to the separator. The extracted characters are returned as the result of the function, and at the same time they are removed from the original string.

The string returned as the result of the function (string removed from the original string) could be either the characters before the separator or the separator itself.

**[ NOTE ]**

This function changes the original string at the same time it returns the characters.

The separator character is not case-sensitive.

**Example**

In the instructions below, the numbers separated by commas or blanks are removed from the string “$input”. The first instruction in the DO structure removes the first set of characters in $input and substitutes them into variable “$temp”. Next the function VAL changes the string gained in the previous instruction into a real value. The real value is then substituted into the array variable “value”. Then, the program execution goes on to the next $DECODE function and searches for the next separator (the separator is removed from $input).

```
i=0                              ;Resets counter
DO
$temp=$DECODE($input,”,”,0)      ;Extracts characters up to the separator ”,”
value[i]=VAL($temp)              ;Converts the characters to real values.
if $input ==” ” GOTO 100
$temp = $DECODE($input,”,”,1)    ;Extracts the separator “,”
i=i+1                            ;Counter increment
UNTIL $input==“”                 ;Continues program execution until there are no
100 TYPE “END”                   more characters
```

If the values of $input are as below, each separated by space and comma then the result of the above program are as follows:

```
1234, 93465.2,   .4358, 3458103,

value[0]        1234.0
value[1]        93465.2
value[2]        0.4358
value[3]        3458103.0
```

As the result of executing the program, the value of string variable $input becomes “” (blank).

## $ENCODE (print data, print data, ……)

**Function**

Returns the string created from the print data specified in the parameters. The string is created in the same way as when using TYPE command.

**Parameter**

Print data

Select one or more from below. Separate the data with commas when specifying more than one.

(1) character string

(2) real value expressions (the value is calculated and displayed)

(3) Format information (controls the format of the output message)

**Explanation**

This function enables creating strings within programs using the same print data as in the TYPE command. Unlike TYPE, the $ENCODE function does not display the created strings, but instead the results are used as values in programs.

The following codes are used to specify the output format of numeric expressions. The same format is used until a different code is specified. In any format, if the value is too large to be displayed in the given width, asterisks (∗) are shown instead of the values.

Format Specification Codes

| | |
|---|---|
| /D | Uses the default format. This is the same as specifying the format as /G15.8 except that zeros following numeric values and all spaces but one between numeric values are removed. |
| /Em.n | Expresses the numeric value in scientific notation (e.g. -1.234E＋02). “m” describes the total number of characters shown on the terminal and “n” the number of decimal places. “m” should be greater than “n” by five or more, and smaller than by 32. |
| /Fm.n | Expresses the numeric values in fixed point notation (e.g. -1.234). “m” describes the total number of characters shown on the terminal and “n” the number of decimal places. |
| /Gm.n | If the value can be expressed in Fm.n format within “m” digits (including “n” digits after the decimal point), the value is expressed in that format. Otherwise, the value is expressed in Em.n format. |

| | |
|---|---|
| /Hn | Expresses the values as a hexadecimal number in the “n” digit field. |
| /In | Expresses the values as a decimal number in the “n” digit field. |

The following parameters are used to insert certain characters between character strings.

| | |
|---|---|
| /Cn | Inserts line feed n times in the place where this code is entered, either in front or after the print data. If this code is placed within print data, n–1 blank lines are inserted. |
| /S | The line is not fed |
| /Xn | Inserts n spaces. |
| /Jn | Expresses the value as a hexadecimal number in the n digit field. Zeros are used in place of blanks. (Option) |
| /Kn | Expresses the value as a decimal number in the n digit field. Zeros are used in place of blanks. (Option) |
| /L | This is the same as /D except that all the spaces are removed with this code. (Option) |

**Example**

```
$output = $output + $ENCODE(/F6.2,count)
```

The value of the real variable “count” is converted into a string in the format specified by /F6.2, and added to the end of the string “$output”. Then the combined string is substituted back in the string variable “$output”.

## $ERRORS (“error code”)

**Function**

Returns the error message for the specified error code. The error code is returned as a character string with the error message.

**Parameter**

Error code

Specifies the error code in the following format: Pxxxx, Wxxxx, Exxxx, or Dxxxx.

## $ERROR (error number)

**Function**

Returns the error message for the specified error code.

**Parameter**

Error number

Specifies the error number by a negative number (starting with –). The error codes are converted into negative error numbers as shown below:

| | | |
|---|---|---|
| Dxxxx | : | –4xxxx |
| Exxxx | : | –3xxxx |
| Wxxxx | : | –2xxxx |
| Pxxxx | : | –1xxxx |

## $DATE (date form)

**Function**

Returns the system date in the specified string format.

**Parameter**

Date form

Specifies by numbers 1 - 3, the date format to be output.

**Explanation**

The types of date forms are as follows.

$DATE(1)     mm/dd/yyyy

(If the date is “October 20, 2016” then the value returned is 10/20/2016).

$DATE(2)     dd/mmm/yyyy

(If the date is “October 20, 2016” then the value returned is 20/OCT/2016).

mmm is JAN/FEB/MAR/APR/MAY/JUN/JUL/AUG/SEP/OCT/NOV/DEC in order from January to December.

$DATE(3)     yyyy/mm/dd

(If the date is “October 20, 2016” then the value returned is 2016/10/20.)

## $TIME

### Function

Returns the system time in following string format:

hh:mm:ss

(For example, for the time 18:27:50, it becomes 18:27:50.)

hh is in 24-hour time system.

## $TIME_MS

**Function**

Returns the system time including milliseconds in a string.

In form of hh:mm:ss.xxx

(For example, for the time 18:27:50, it becomes 18:27:50.0000.)

hh is in 24-hour time system. Accuracy of the time acquired by this function is ±2 ms.

## $SYSDATA (keyword, opt1, opt2)

**Function**

Returns specified parameters in the AS system according to the given keyword.

**Parameter**

Keyword, opt1, opt2

ZROB.NAME

Returns the model name of the robot.

Opt 1: Robot number (1 to number of robots). If not entered, 1 is assumed.

Opt 2: Not used.

## $ERRLOG (error log number)/$ERRORLOG (error log number) *1

**Function**

Returns the character string of error message of the specified log number.

**Parameter**

Error log number

Specifies the error log number of the log to return.

**Example**

When the error is recorded in the error log as below,

```
1 - [14/11/17 12:05:02 SIGNAL:00 Monitor speed: 10 Teach mode]
      (E1162) Buffer overflow occurred in the gravity comp. value channel 2.
2 - [14/11/17 12:05:02 SIGNAL:00 Monitor speed: 10 Teach mode]
      (E1352) Jt3 4 5 6 Codes set in software and power block do not match.
3 - [14/10/16 17:19:22 SIGNAL:00 Monitor speed: 10 Teach mode]
      (D2023) Failed to load arm data.
```

When

```
> TYPE  $ERRLOG(2)
```

is input, the error message logged second "(E1352) Jt3 4 5 6 Codes set in software and power block do not match." is displayed.

*1 $ERRORLOG is used for explosion proof software.

## $STR_ID (number)
## $STR_ID2 (number)

**Function**

Returns the character string for the robot information.

| | |
|---|---|
| $STR_ID | Robot 1 |
| $STR_ID2 | Robot 2 |

**Parameter**

Number

Specifies the robot information to return by integer values 0 to 1.

0 : Robot name  Returns the robot name of the controlled robot arm.

1: Machine number:  Returns the machine number of the controlled robot arm.

**Example**

When

Robot 1 Robot name: RS020N-A001 Machine number: 1

Robot 2 Robot name: BX200L-B001 Machine number: 2

then,

| | |
|---|---|
| $STR_ID(0) | String “RS020N-A001” is returned. |
| $STR_ID(1) | String “1” is returned. |
| $STR_ID2(0) | String “BX200L-B001” is returned. |
| $STR_ID2(1) | String “2” is returned. |

## PCSTATUS PC program number:

**Function**

Displays the status of PC programs. (M)

**Parameter**

PC program number

Selects the PC program number to display.  Acceptable range: 1 to 5.  If not specified, 1 is assumed.

**Explanation**

The PC program status is displayed in the following format.

```
(1) ············PC program status:       Program is not running
                 Execution cycles:
(2) ············Completed cycles:        11
(3) ············Remaining cycles:        Infinite
(4) ············Program name             Priority   Step No. ...................................................... (5)
                 pc_test0                 1          PRINT "step1"
```

(1) Program status

The PC program status as described as one of the following:

| | |
|---|---|
| Program is not running | Program is not currently running. |
| Program running | Program is currently running. |
| Program WAIT | Program is running, but waiting for the condition set in WAIT command to fulfill. |

(2) Completed cycles

Displays the number of execution cycles completed.

(3) Remaining cycles

Displays the numbers of cycles not yet executed.  If the execution cycle is set as negative number (−1) in PCEXECUTE command, the display will be “infinite”.

(4) Program name

Name of the PC program in execution is displayed.

(5) Step

Displays the number of the step currently being executed and the instruction written in that step.

## PCEXECUTE PC program number : program name, execution cycle, step number

**Function**

Executes PC programs. (M, P)

**Parameter**

PC Program number

Selects the number of the PC program to execute. Acceptable range: 1 to 5. If not specified, 1 is assumed. Up to 5 PC programs can be executed at the same time. The PC program number is not the order of priority.

Program name

Selects the name of the program to execute at that PC program number. If not specified, the program last executed using the PCEXECUTE command is selected.

Execution cycle

Specifies how many times the PC program is to be executed. If not specified, 1 is assumed. If –1 is entered, the program is executed continuously.

Step number

Selects the step from which to start execution. If not specified, the execution starts from the first step in the program.

**Explanation**

This command is identical to EXECUTE monitor command, except that this command executes PC programs instead of robot control programs. The PC program currently in execution is displayed with a blinking "∗ " at the end of its name.

PCEXECUTE can be used as either a monitor command or an instruction in a robot control program.

**Example**

| | |
|---|---|
| PCEXECUTE control, -1 | The program "control" is executed continuously; i.e. program execution continues until PCABORT command is executed, PAUSE or HALT instruction is executed in the program, or an error occurs. |

## PCABORT PC program number:

**Function**

Stops the execution of the currently running program. (M, P)

**Parameter**

PC program number

Selects the number of the PC program to be stopped. Acceptable range: 1 to 5. If not specified, 1 is assumed.

**Explanation**

PCABORT is identical to ABORT command except this command stops PC programs instead of robot control programs.

The program currently running is stopped, and the execution can be resumed using PCCONTINUE command.

PCABORT can be used as either a monitor command or an instruction in a robot control program.

## PCKILL PC program number:

**Function**

Initializes the stack of PC programs. (M)

**Parameter**

PC program number

Selects the number of the PC program to initialize. Acceptable numbers are from 1 to 5. If not specified, 1 is assumed.

**Explanation**

This command initializes the program stack of PC programs.

When a program is suspended by PAUSE or PCABORT command, or by an error, the program remains in the program stack. As long as the program is in the stack, it cannot be deleted (DELETE command). In this case, first use PCKILL to remove the program from the stack.

## PCEND PC program number: task number

### Function

Ends execution of the PC program currently running upon execution of the next STOP instruction in that program. (M, P)

### Parameter

PC program number

Selects the number of the PC program to end. Acceptable range: 1 to 5. If not specified, 1 is assumed.

Task number

Specifies 1 or –1. If not specified, 1 is assumed.

### Explanation

If the task number is not specified or specified as 1, the program execution is stopped as soon as the next STOP or RETURN instruction (or similar instruction) is executed, regardless of remaining cycles. The remaining cycles can be executed using PCCONTINUE.

If –1 is specified as the task number, the PCEND command entered previously is canceled. When a program loop occurs or the program runs infinitely without a STOP instruction, PCEND is ineffective and must be canceled by PCEND –1. (To cancel the loop, PCABORT must be used).

PCEND can be used as either a monitor command or an instruction in robot control programs.

## PCCONTINUE PC program number NEXT

**Function**

Resumes execution of a suspended PC program.  Or, skips the WAIT instruction in the PC program. (M)

**Parameter**

PC program number
Selects the number of the PC program to resume execution.  Acceptable range: 1 to 5.  If not specified, 1 is assumed.

NEXT
If this parameter is specified, the execution is resumed from the step after the step that was suspended.  If not specified, the execution resumes from the same step that was suspended. With the parameter NEXT, this command can be used to skip the WAIT instruction in the currently running PC program and to resume execution of that PC program.

**Explanation**

PCCONTINUE is identical to CONTINUE command except this command is used to continue execution of PC programs instead of robot control programs.

Execution is resumed from the step where the execution was stopped by PAUSE or PCABORT command, or by an error, and from the step after that when the parameter NEXT is specified.

## PCSTEP PC program number: program name, execution cycles, step number

**Function**

Executes a single step of a PC program. (M)

**Parameter**

PC program number

Selects the number of the PC program containing the desired step. Acceptable range: 1 to 5. If not specified, 1 is assumed.

Program name

Selects the name of the program to execute at that PC program number. If not specified, the program currently in execution or the program last executed is selected.

Execution cycle

Specifies how many times the program step is to be executed. If not specified, 1 is assumed.

Step number

Selects the number of the program step to execute. If not specified, the first step of the program is selected. If none of the parameters are specified, the next step is executed.

**Explanation**

PCSTEP command, like the PCCONTINUE command, can be used without parameters only in the following conditions:

1. when PCSTEP command was used in the last executed step
2. after a PAUSE instruction
3. when the program was suspended by reasons other than error.

**Example**

| | |
|---|---|
| >PCSTEP sequence,,23 ↵ | Executes step 23 of the PC program no.1 named "sequence" one time. |

Enter PCSTEP↵ after this, and then the next step (step 24) is executed.

## PCSCAN time

**Function**

Sets the cycle time for executing the PC program. (P)

**Parameter**

Time

Sets how long the program repetition cycle takes. The time is specified in seconds, 0 or greater.

**Explanation**

This command is used to execute the PC program in the specified cycle time. If the execution time is longer than the specified time, the time specified here is ignored.

**Example**

```
program
    PCSCAN 1
    IF sig(1) THEN
    SIGNAL -1
    ELSE
    SIGNAL 1
    END
```

If the above program is executed continuously using the PCEXECUTE command (execution cycle: –1), SIGNAL 1 turns ON → OFF every second.