1112, TFDEN-012-001, 001

NACHi

FD CONTROLLER INSTRUCTION MANUAL Robot language

1st edition

- Before attempting to operate the robot, please read through this operating manual carefully, and comply with all the safety-related items and instructions in the text.
- The installation, operation and maintenance of this robot should be undertaken only by those individuals who have attended one of our robot course.
- When using this robot, observe the low related with industrial robot and with safety issues in each country.
- This operating manual must be given without fail to the individual who will be actually operating the robot.
- Please direct any queries about parts of this operating manual which may not be completely clear or any inquiries concerning the after-sale service of this robot to any of the service centers listed on the back cover.

NACHI-FUJIKOSHI CORP.

Table of Contents

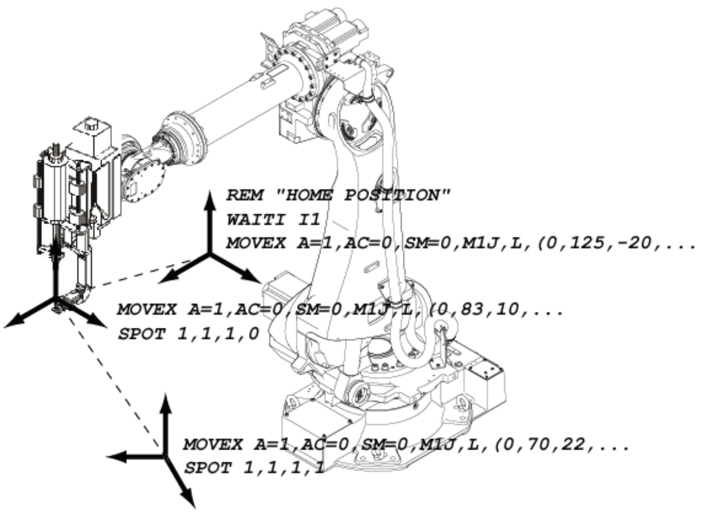
Chapter 1 Outline

This chapter describes the outline of the robot language.

1.1 What is "Robot language" ?

1.1.1 Outline

"Robot language" is a programming language for this robot controller. This function can make (teach) a work-program of a robot by combining move commands (`MOVEX`) and application commands like input signal waiting command (`WAITI`) etc. Because it is also possible to use various functions and calculation formula, this function is suitable for making a work-program that requires complicated flow controls or complicated coordinate calculations that are difficult to achieve when using normal teaching operation via teach pendant keys. Furthermore, it is also possible to create a customized monitor window by using the original commands for the robot language function.

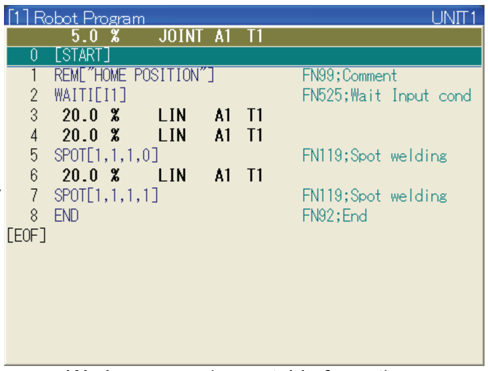


```
REM "HOME POSITION"
WAITI I1
MOVEX A=1,AC=0,SM=0,M1J,L,(0,125,-20,...

MOVEX A=1,AC=0,SM=0,M1J,L,(0,83,10,...
SPOT 1,1,1,0

MOVEX A=1,AC=0,SM=0,M1J,L,(0,70,22,...
SPOT 1,1,1,1
```

An image of robot motion

```
REM "HOME POSITION"
WAITI I1
MOVEX A=1,AC=0,SM=0,M1J,L,(0,125,-20,…
MOVEX A=1,AC=0,SM=0,M1J,L,(0,83,10,…
SPOT 1,1,1,0
MOVEX A=1,AC=0,SM=0,M1J,L,(0,70,22,…
SPOT 1,1,1,1
END
```

Robot language program (source program)



```
[1] Robot Program                                      UNIT1
       5.0 %     JOINT  A1  T1
  0  [START]
  1  REM["HOME POSITION"]                  FN99;Comment
  2  WAITI[I1]                             FN525;Wait Input cond
  3   20.0 %     LIN    A1  T1
  4   20.0 %     LIN    A1  T1
  5  SPOT[1,1,1,0]                         FN119;Spot welding
  6   20.0 %     LIN    A1  T1
  7  SPOT[1,1,1,1]                         FN119;Spot welding
  8  END                                   FN92;End
[EOF]
```

Work program (executable format)

Fig. 1.1.1 An image of a work program generated using robot language

Concerning the syntax of robot language, refer to the chapter 2. And, while you are not accustomed to the robot language, we recommend you to try to make several simple work-programs with normal teaching operation by referring to the instruction manual "BASIC OPERATIONS MANUAL" and then try to "reverse compile" those programs to get robot language format source programs. This trial will help you to understand the relationship between the robot language and the work-program. Concerning how to make "reverse-compile" operation, refer to the chapter 4.

INFO.

In this instruction manual, the details of the application commands are not described. For details, please refer to the online help displayed on the teach pendant screen or instruction manual "Command reference".

INFO.

For details of customized monitor window, refer to the instruction manual of "User task".

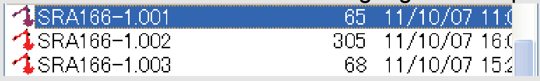
1.1.2 Characteristics and precautions

- The source program format of robot language is plain text format. The source program can be made using a text editor of your PC or "ASCII File Edit" screen of this robot controller.

- For example, if the robot type if SRA166-01, the file name of the robot language program (source program) and the work-program (executable format) are like the following examples. If the filename does not follow this style, the file is not recognized by this controller.

Robot language program (source program)	`SRA166-1-A.001`
Work-program (executable format)	`SRA166-1.001`

To confirm the file name that must be used for the source program file, open the "PROGRAM" folder in the <Service Utilities> - [8 File manager] - [2 Directory] screen.
If there are work-programs that were already made, their file names will be displayed.
To make a filename of a robot language source program, attach "-A" to the filename.



```
SRA166-1.001      65  11/10/07 11:0
SRA166-1.002     305  11/10/07 16:0
SRA166-1.003      68  11/10/07 15:2
```

(In this case, "`SRA166-1-`**`A`**`.001`" is the source program name for "`SRA166-1.001`")

- Robot language is based on the SLIM (**S**tandard **L**anguage for **I**ndustrial **M**anipulator) language which complies with the JIS B9439 standard.

- The robot language source program must be converted (compiled) to an executable format in advance to use. The robot language format and the executable format can be converted to each other. (These operations are referred to "Compile" and "Reverse-compile"). And, when making a work-program with normal teaching operation using the teach pendant, the file is recorded to the internal memory automatically in a style of executable format from the beginning. So it is not necessary to compile the file to use.

- The work-programs made from robot language and the work-programs made from normal teaching operation can mingle with each other. It is also possible to use a robot in a way in which only complicated calculation and condition jump/call process etc. are controlled with robot language logic and the other robot program created with normal teaching operation are called from those robot language programs.

- This robot language function supports 2 types of move command teaching method. The first one is a method in which the position data is directly described in the move command and the second one is a method in which a pose variable is called using the parameter in the move command. The pose variables can be made via position record operation using the teach pendant or robot language's "Pose operation" etc. It is also possible to manage the plural pose variables (made via position record operation) altogether in a form of "Pose file". For details of "Pose file", refer to the chapter 3.

How to describe the pose constant in a program "2.4.3 Pose constants"

Example 1 : MOVEX-X
```
MOVEX A=1,AC=0,SM=0,M1X,L,(1800,0,2000,0,-90,-180),R=5.0,H=1,MS
```

Example 2 : MOVEX-J
```
MOVEX A=1,AC=0,SM=0,M1J,L,(0,90,0,0,0,0),R=5.0,H=1,MS
```

How to designate the position using pose variable "2.5.7 Pose variable","3.3 Creating pose files"
```
MOVEX A=1,AC=0,SM=0,M1X,P,P1,R=5.0,H=1,MS
```

- After converting to an executable format, it is possible to modify the position data of a move command using the normal position modification operation with the teach pendant even if the move command is made from robot language. However, in this case, please be sure that in spite of the format of the original move command, the robot position data is forcibly converted to encoder value format (this is the same format of which a move command that is recorded with normal position record operation).

- Even in case of a work program that is generated from robot language, it is possible to make step addition or step deletion in the same way with normal work-program. But, it is recommended to modify the original source program (not the executable format work-program) and make compile operation again. And, to make small modifications to the source program file, "ASCII File Edit" function of this robot controller is convenient. For details, refer to the chapter 3.

1.2 Teaching (programming) and playback

Outline of the operation

1 Create a source program file using a text file editor on your PC or "ASCII File Edit" function of this controller.

☞"3.1 Editing using a personal computer "

2 For example, if the robot type is "SRA166-01", save the source program to the USB memory using a file name like the following;

`SRA166-1-A.001` (The file name of the source program)

- "`SRA166-1`" shows the robot type.
- "`-A`" shows that the file is robot language source program.
- The file name extension "`001`" shows the program number. (MAX is 9999)

☞See the item of file name in "1.2.2 Characteristics and precautions"
☞See "3.1.2 Loading a robot language source program into this controller "
☞See the Chapter 6 of "BASIC OPERATIONS MANUAL"

3 Copy the source program file from the USB memory to the internal memory of this controller.

This operation can be done in the <Service Utilities> - [7 File Manager] - [1 File Copy] screen.
☞See the Chapter 6 of "BASIC OPERATIONS MANUAL"

4 Convert the source program to an "executable format" by executing "Compile" operation.

This operation can be done in the <Service Utilities> - [9 Program Conversion] - [8 Language] screen.
☞See "4.1 Compiling" of this instruction manual.

When the compile operation is normally finished, an executable program file will be generated.
In this example, the file name would be the following;

`SRA166-1.001` (The filename of a executable format)

5 For the generated executable format program, CHECK GO/BACK or playback operation etc. can be applied in the same procedures with the normal work-program.

☞See the Chapter 4 and 5 of "BASIC OPERATIONS MANUAL"

6 If there exists problems in the robot motion, revise the source program file and compile it again. For small modifications, "ASCII File Edit" function is convenient.

☞See "3.2 Editing using the teach pendant"

INFO.

When using integer variables etc., it is possible to check their values with "Monitor" function. This "Monitor" function is included in "Service Utilities" menu.

WARNING

Because the robot language can create the coordinate freely using calculation formula etc., please pay special attention when moving the robot. If the robot makes unexpected motion, death or serious injury may result.

NOTE

Chapter 2 Syntax

This chapter describes the syntax of the robot language when the robot language is used to prepare programs.

2.1 An example of robot language program

A robot language source program

```
100 'SHIFT
110 USE 100
120 FOR V1%=1 TO 7
130 R1=(10,1,0,0,0,0)
140 P100=P[V1%]+R1
150 MOVEX A=1,M1X,P,P100,R=5.0,H=1,MS
160 NEXT
170 END
```

LINE 100 : Comment statement

Starting a line with a single quotation mark (') turns the statement on the line into a comment statement, thus making it possible to facilitate reading of the program. "REM" can be used instead of single quotation mark.

"2.7.1 Comment statement"

LINE 110 : USE command

This statement selects the Pose file No. 100.
From this line downward, "Pose variable" will be retrieved from the Pose file No. 100.

"2.5.7 Pose variable"

LINE 120 : FOR command

A flow control statement includes a jump between lines (GOTO), a conditional branch (IF), and others. "V1%" is a variable to which an integer can be assigned. In this line, this is used as a loop counter.

"2.7.4 Flow control statement"

LINE 130 : Assignment statement

In an assignment statement, values are directly assigned to the shift variable "R1".

"2.5.8 Shift variable", "2.7.3 Substitution statement"

LINE 140 : Assignment statement

The operation of the pose variable "P[V1%]" and the shift variable "R1" is carried out to assign the result to the pose variable "P100".

"2.5.7 Pose variable", "2.5.8 Shift variable"

LINE 150 : MOVEX command

A move command (MOVEX command) is executed using pose variable P100, thus making the robot move.

"2.5.7 Pose variable"

LINE 160 : NEXT command

This flow control statement pairs up with the FOR statement. In this case a jump to LINE 120 will be made.

"2.7.4 Flow control statement"

LINE 170 : END command

The program is ended. The END command is always necessary at the bottom line of the program.

(*) The line numbers can be omitted.

"2.2 Line numbers"

Fig. 2.1.1 Robot language syntax (Example)

2.2 Line numbers

A line is the smallest unit that makes up a robot language program. A robot language is executed on a line by line basis.
A total of **254 characters** may be written on one line, and up to **9999 lines** can be written in a program. A line consists of a comment statement, label, assignment statement, flow control statement or command statement. (☞2.7Statements) Including two or more statements in a line (e.g. "`FOR V1%=0 TO 100, NEXT`") is not permitted.

POINT

- Two or more of statements cannot be written on one line.
- It is not permitted to enter a line feed code anywhere in a statement.
- The maximum number of characters allowed in a statement is 254. A compiling error results when a line with 255 or more characters is written.

The line numbers indicate the execution positions in the program. Numbers from 1 to 9999 can be specified. Line numbers can also be used as the jump destinations of line jump instructions, etc. They are allocated in numerical order so that the lowest numbers come at the start of a program and progressively increase as the program advances.

POINT

Line numbers can be omitted. In this case, use labels for the jump destinations of line jump instructions, etc.
When a line number is omitted, any line number must not be written anywhere in the program. Programs will not run properly if some lines are numbered while others are not.

POINT

When reverse-compiled, the line number will be changed.

2.3 Characters

The table below lists the characters which can be used by the robot language.

Table 2.1 Characters which can be used by the robot language

Item	Characters which can be used
Alpha numerics	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 01234567890
Symbols	! ″ ′ # $ % & ( ) * + − . , / : ; = < > ? @ ` [ ] ¥ ^ { }~ | _
Blanks	(Half-size space) (tab)

POINT

No differentiation is made between upper-case and lower-case letters of the alphabet. Use half-size characters for all text except for comments and character string variables.

2.4 Constants

The following types of constants are available.
The constants shown below will be retained as the power failure retention data, even if the main power supply is turned OFF.

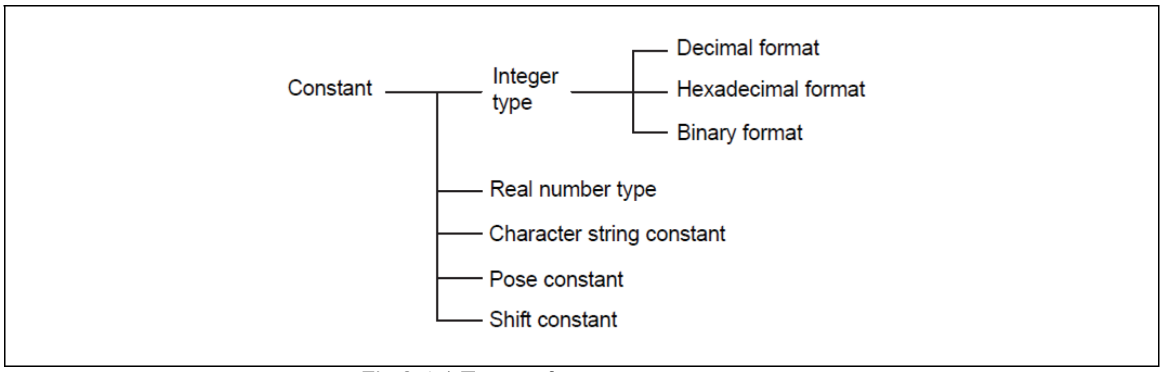


Fig 2.4.1 Types of constants

2.4.1 Numeric constants

Numeric constants are numeric data used for executing operations.
Whether they are integer or real number constants is automatically determined internally.

<Examples>

・ `-10`	Integers
・ `1.23ーE12`	Real numbers
・ `&H1F3`	Hexadecimal format
・ `&B10101`	Binary format
・ `90DEG`	Angle value (real number) expressed in degrees

Integers range from –2147483648 to +2147483648.
Real numbers range from –1.0+E38 to +1.0+E38.

2.4.2 Character string constants

Alphanumeric characters and symbols enclosed in double quotation marks (") are referred to as a character string.

<Examples>

・ `PRINT #0, "ABCDE"`

2.4.3 Pose constants

"Pose constants" is a constant that represents the robot position and angle. This constant is used combining `MOVEX` command. There are 3 types of Pose constants.

`MOVEX-X`	This represents the robot posture using `(X,Y,Z,r,p,y)`
`MOVEX-J`	This represents the robot position using a set of angles or positions of each axis.
`MOVEX-E`	This represents the robot position using a set of encoder values of each axis.

`MOVEX-X`

A MOVEX-X style pose constant represents the robot posture using (X,Y,Z,r,p,y).
The reference point is the origin of the mechanism coordinate system of the robot.

Example : `(1834, 0, 2030, 0, -90, -180)`

Table 2.2 The parameters for `MOVEX-X`

Parameter	`MOVEX-X`
X	X coordinate of the TCP [mm]
Y	Y coordinate of the TCP [mm]
Z	Z coordinate of the TCP [mm]
r	"Roll" angle of the tool [deg] (Around Z axis)
p	"Pitch" angle of the tool [deg] (Around Y axis)
y	"Yaw" angle of the tool [deg] (Around X axis)

Designate the coordinates of the TCP in (X, Y, Z) based on the machine coordinate system.
And, designate the angle of the tool in (Roll, Pitch, Yaw) based on the machine coordinate system's direction. Please be sure that the tool direction depends on the order of the rotation. (The rotation should be done in an order of Roll (Z), Pitch (Y), and Yaw (X))

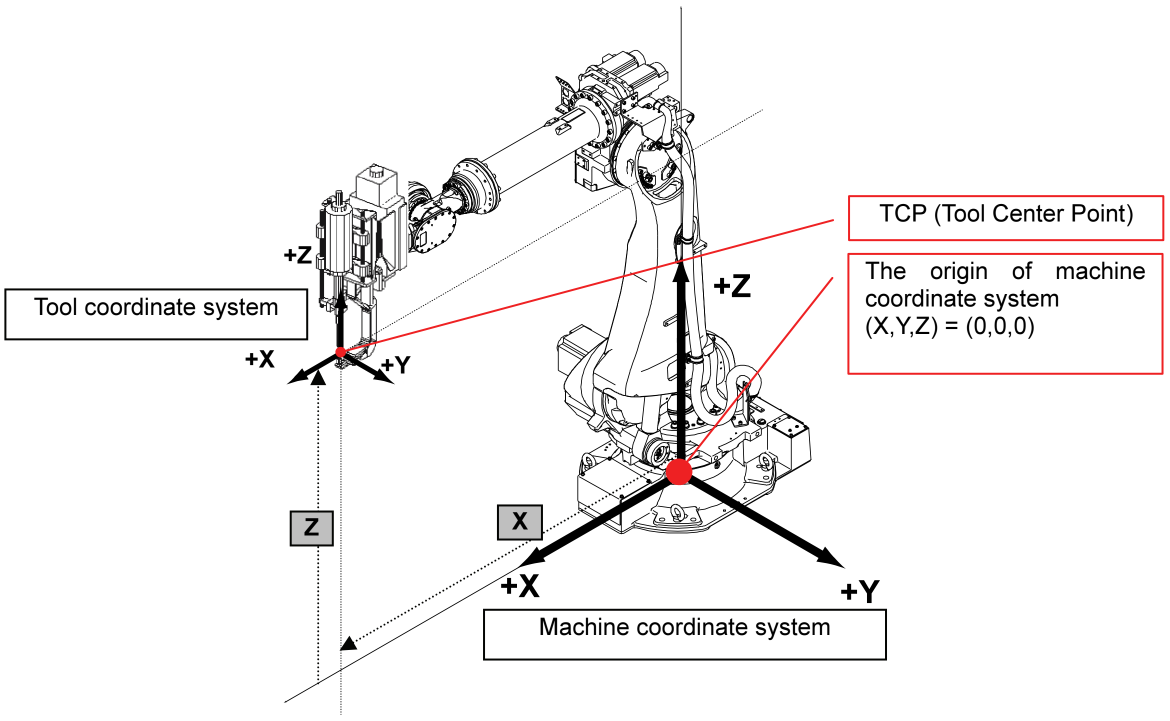

TCP (Tool Center Point)
The origin of machine coordinate system (X,Y,Z) = (0,0,0)
Tool coordinate system
+Z
+X
+Y
+Z
Z
X
+X
+Y
Machine coordinate system

Fig. 2.4.2 A machine coordinate system and a tool coordinate system

CAUTION

- Because the TCP position and the tool angle are written in the machine coordinate system, in a move command using this representation, the position where the actual TCP reaches will not change even if the tool constant is changed. However, please do not forget that the angle of each axis will change at the step. So please pay special attention for e.g. interference etc.
- When not using CONF setting, the robot may make unexpected posture (configuration). For details of the CONF setting, please refer to the online help of the MOVEX command. And, it is recommended to use MOVEX-J for the move command step that will be executed at first time. (Because MOVEX-J does not have redundancy of the robot posture to be generated and can make a robot unique posture)

MOVEX-J

This represents the robot position using a set of angles [deg] or positions [mm] of each axis. The number of the parameters differs from the mechanism type. (The maximum number of the axes for 1 mechanism is 6)

Example: `(0, 90, 0, 0, 0, 0)`

Table 2.3 The parameters for `MOVEX-J`

Parameter	`MOVEX-J`
1st axis	Angle [deg] or position [mm] for the 1st axis
2nd axis	Angle [deg] or position [mm] for the 2nd axis
3rd axis	Angle [deg] or position [mm] for the 3rd axis
4th axis	Angle [deg] or position [mm] for the 4th axis
5th axis	Angle [deg] or position [mm] for the 5th axis
6th axis	Angle [deg] or position [mm] for the 6th axis

In case of a normal 6-axes robot, write 6 values. If the robot has additional axes like servo gun or slider (traverse axis) etc. refer to the example shown as below.

An example of 6-axes robot that has slider (traverse axis)

```
MOVEX A=1,AC=0,SM=0,M1J,P,(0,90,0,0,0,0),R= 5.0,H=1,MS,CONF=0000,M2J,P,(100),R= 100,H=1
```

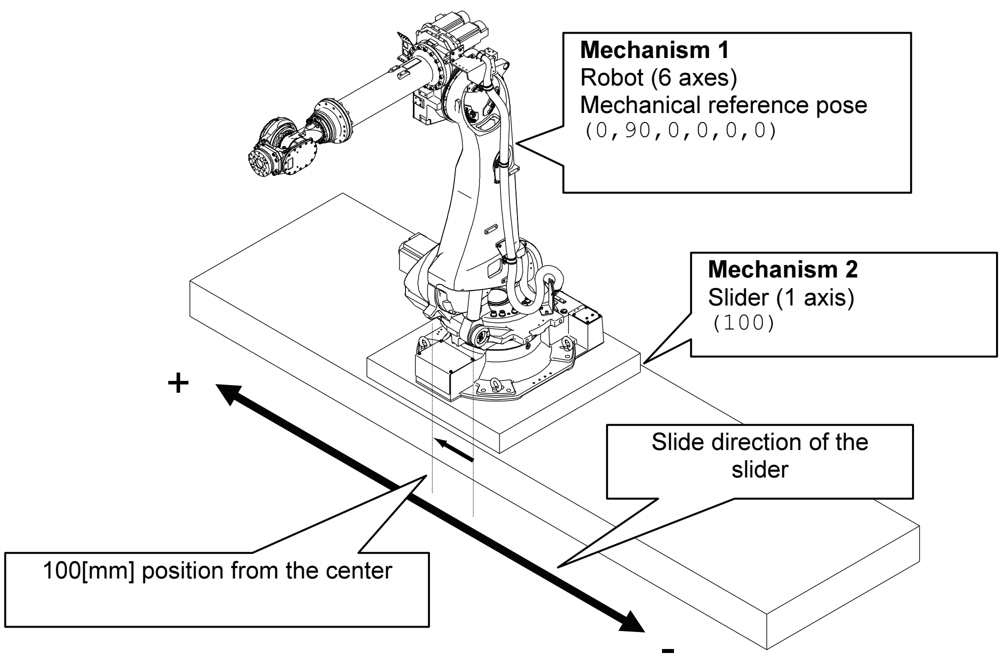

Mechanism 1
Robot (6 axes)
Mechanical reference pose
(0,90,0,0,0,0)
Mechanism 2
Slider (1 axis)
(100)
+
-
Slide direction of the slider
100[mm] position from the center

Fig. 2.4.3 A robot with a slider

In this example, the mechanism 1 (6-axes robot) makes its mechanical reference pose (0, 90, 0, 0, 0, 0) and the mechanism 2 (slider with 1 axis) makes 100 [mm] position.

MOVEX-E

This represents the robot position using a set of encoder values of each axis. When there are 2 or more mechanisms in the UNIT, please refer to the example of MOVEX-J.

Example: `(&H80000,&H80000,&H80000,&H80000,&H80000,&H80000)` (NOTE)"`&H80000`" is hexadecimal

Table 2.4 The parameters for `MOVEX-E`

Parameter	`MOVEX-E`
1st axis	Encoder value for the 1st axis
2nd axis	Encoder value for the 2nd axis
3rd axis	Encoder value for the 3rd axis
4th axis	Encoder value for the 4th axis
5th axis	Encoder value for the 5th axis
6th axis	Encoder value for the 6th axis

2.4.4 Shift constants

The shift constants are used for adding the translational amounts for the robot poses. They are described using the (X, Y, Z, r, p, y) format.

Table 2.5 Shift constant

Parameter	Shift amount
X	Shift in X-axis direction [mm]
Y	Shift in Y-axis direction [mm]
Z	Shift in Z-axis direction [mm]
R	Rotational shift around Z axis [deg]
P	Rotational shift around Y axis [deg]
Y	Rotational shift around X axis [deg]

2.5 Variables

The structures for holding the values used in programming are called variables. Variables are used as references for operations and other such uses. Their names are reserved in advance, and users cannot assign the names of their choice to them.
The following variables are available.

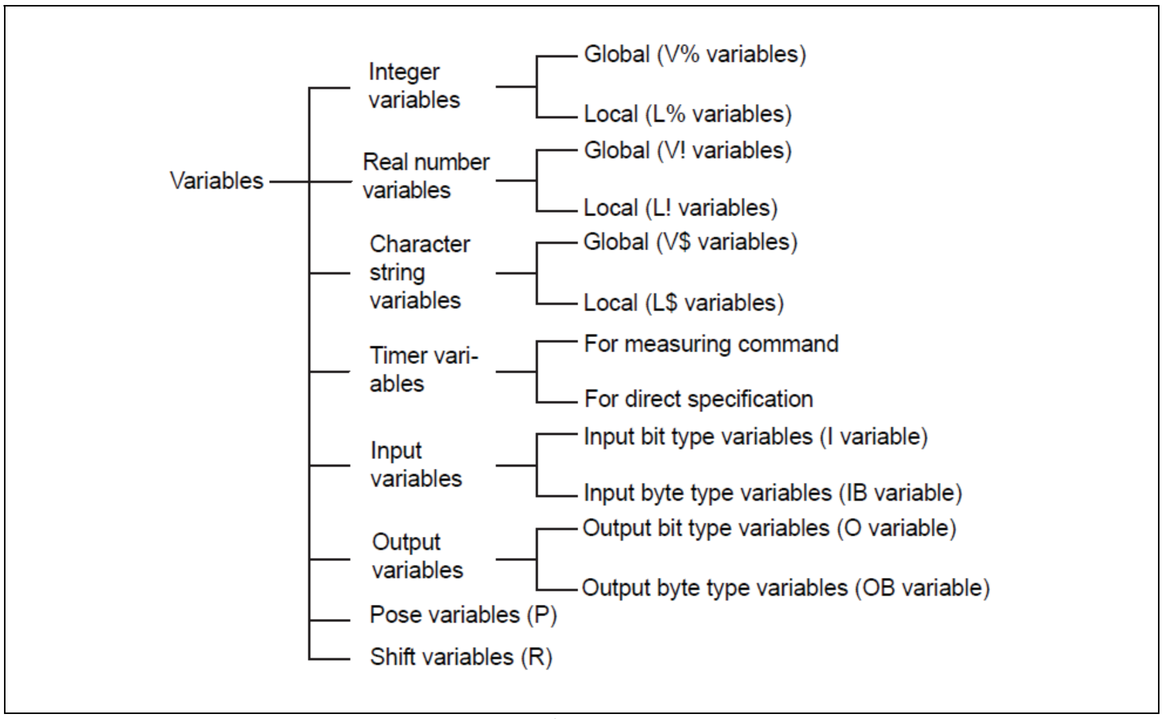


Fig 2.5.1 Types of variables

Table 2.6 Global variables and local variables

Global variables	These can be referenced from any unit or user task. They start from `V`.
Local variable	These are allocated to individual units and user tasks, and they cannot be referenced from other units. They start from `L`.

2.5.1 Integer variables

The "Integer variables" are used to handle values with no decimal point included.

Table 2.7 Integer variables

Format	`Vn%, V%[n]` n=1 to 250, 301 to 500 (Variables can be also used.) `Ln%, L%[n]` n=1 to 200, 301 to 500(Variables can be also used.) `V1%` and `V%[1]` denote the same variable area.
Range	`−2147483648 to +2147483647`

Sample	`V1%=V2%+1`
	`L%[1]=10`
	`CALLN 10, V1%, 10`
	`JMPN 10, V1%, 20`

Storage	All global integer variables are retained as power failure retention data even if the main power supply is turned OFF. In contrast, all local integer variables are not retained.

POINT

• The global integer variables 201 to 250 are used to execute passing of PLC's internal integer variables and integer values. Writing values in global variables will write their corresponding values in the PLC's internal integer variables. Furthermore, reading the said global variables will read the value of the PLC's internal integer variables. Table 2.5.3 below shows correspondence of global integer variables to PLC's internal integer variables.

• Writing values in the global integer variables 201 to 250 aforementioned will cause delay until the changes affect the PLC's internal integer variables. In contrast, writing values in the PLC's internal integer variables will cause delay until the changes affect global integer variables 201 to 250. The delay duration varies with the robot configuration, usage, and others. Consequently, interlock should be provided between the PLC and the robot language.

• For details of PLC's internal integer variables, refer to the instruction manual "Software PLC".

Table 2.8 Correspondence of global integer variables to PLC's internal integer variables

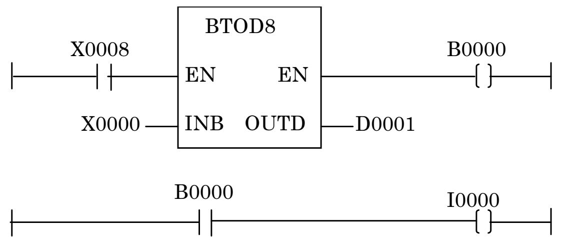
Global integer variable number	PLC's internal integer variable number
V201% or V%[201]	D0001
V202% or V%[202]	D0002
V203% or V%[203]	D0003
· · ·	· · ·
V248% or V%[248]	D0048
V249% or V%[249]	D0049
V250% or V%[250]	D0050

Precautions for using global integer variables V201% to V250%

The global integer variables `V201%` to `V250%` are used to read/write integer variables out of the PLC's internal variables when reading/writing values. In other works, the global integer variables are not available to advance execution unlike conventional global integer variables. Furthermore, delay occurs after values are set to the global integer variables `V201%` to `V250%` until the set values affect the PLC's internal integer variables. In contrast, if changes are made to the PLC's internal integer variables, delay will occur before the changes affect the global integer variables `V201%` to `V250%`. Furthermore, the delay duration varies with the robot configuration, usage, and others.
To prevent the influence of the delay aforementioned, be sure to provide interlock between the PLC and the robot language according to the procedure shown below.

The following section shows the interlocking procedure for passing the PLC's internal variables with the use of the global integer variables `V201%` to `V250%`. Furthermore, note that the interlocking procedure varies with reading or writing of the internal variables.

<<For reading of PLC's internal variables>>



```
WAIT (I1)
V1%=V201%
END
```

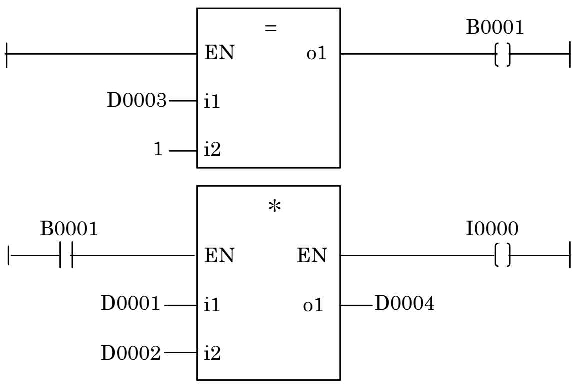
Fig 2.5.2 Example of use of global integer variables 201 to 250

The above figure shows the case where eight bits of this controller's standard inputs X0000 to X0007 are assigned to the global integer variable V1% as integer values.

In the above figure, external signals are input in the order presented below.
(1) Input integer values set with this controller to X0000 to X0007.
(2) Turn ON X0008 indicating that the said inputs have been determined.

In the above figure, since the external signals are input in the order of X0000 to X0007 and then X0008, D0001 has been determined at the time when I1 (I0000 for PLC) turns ON. Consequently, interlock between the PLC and the robot language is provided through turning ON the I1.

<<For writing in PLC's internal variables>>



```
V201%=V1%
V202%=V2%
V203%=1
*WAIT_PLC
WAITI(I1)
V203%=0
END
```

Fig 2.5.3 Example of use of global real number variables 201 to 250

The above figure shows the case where the global integer variables V1% and V2% are stored in the PLC's internal integer variables D0001 and D0002 respectively, and then the product of those variables are stored in the PLC's internal integer variable D0004.
In the above figure, since the order in which values are set to V201%, V202% and V203% will never be reversed, set "1" to V203% (D0003 for PLC). The contents of D0001 and D0002 have been determined at the time when the I1 (I0000 for PLC) turns ON. Consequently, interlock between the PLC and the robot language is providing through setting "1" to V203% and turning ON the I1.

2.5.2 Real number variables

The "Real number variables" are used to handle values with decimal point included.

Table 2.9 Real number variables

Format	`Vn!, V![n]` n=1 to 250, 301 to 500 (Variables can be also used.) `Ln!, L![n]` n=1 to 200, 301 to 500 (Variables can be also used.) `V1!` and `V![1]` denote the same variable area.
Range	`−1.0E38 to +1.0E38`

Sample	`V1!=V2!*103.45`
	`L![1]=1.567E-2`
	`GETANGLE V1!`

Storage	All global integer variables are retained as power failure retention data even if the main power supply is turned OFF. In contrast, all local integer variables are not retained.

If the real number variables are integers, the calculation results may become integers. For robot language calculations, follow the types of variables on the right-hand side not those of variables on the left-hand side. As a specific example, refer to the calculation results from
"Table 2.10 Example of calculations with mixed variables of real numbers and integers (1)" and
"Table 2.11 Example of calculations with mixed variables of real numbers and integers (2)".

Table 2.10 Examples of calculations with mixed variables of real numbers and integers (1)

Format	`V42!` result and display	Expected value	Remark
`V42!=152/10`	15	15.2	−
`V42!=152.0/10`	15	15.2	"`152.0`" is handled as an integer.
`V42!=152/10.0`	15	15.2	"`10.0`" is handled as an integer.
`V42!=152.0/10.0`	15	15.2	The numerator and denominator are both handled as an integer.
`V42!=152/10+ V41!`	25	25.2	`V41!=10` or `V41!=10.0`
`V42!=152/10+ V41!`	25.1	25.3	`V41!=10.1`
`V42!=152/10+10`	25	25.2	−
`V42!=152/10+10.0`	25	25.2	−
`V42!=152/10+10.1`	25.1	25.3	−

The table above lists examples showing that the divisions produce results in integers.
These calculations do not produce results as expected.

Table 2.11 Examples of calculations with mixed variables of real numbers and integers (2)

Format	`V42!` result and display	Expected value	Remarks
`V42!=V41!/10`	15.2	15.2	`V41!=152`
`V42!=V41!/10`	15.2	15.2	`V41!=152.0`
`V42!=V41!/10`	15.21	15.21	`V41!=152.1`
`V42!=152*0.1`	15.2	15.2	Example of an alternate solution to `V42! 152/10`

The table above lists examples showing that the divisions produce results in real numbers.
These calculations produce results as expected.

POINT

• The global real number variables `201` to `250` are used to execute passing of PLC's internal integer variables and real number values. Writing values in global variables aforementioned will write their corresponding values in the PLC's internal real number variables. Furthermore, reading the said global variables will read the value of the PLC's internal real number variables. The table below shows correspondence of global real number variables to PLC's internal real number variables.

• Writing values in the global real number variables `201` to 250 aforementioned will cause delay until the changes affect the PLC's internal real number variables. In contrast, writing values in the PLC's internal real number variables will cause delay until the changes affect global real number variables 201 to 250. The delay duration varies with the robot configuration, usage, and others. Consequently, interlock should be provided between the PLC and the robot language.

• For details of PLC's internal real number variables, refer to the instruction manual "Software PLC".

Table 2.12 Correspondence of global real number variables to PLC's internal real number variables

Global real number variable number	PLC's internal real number variable number
`V201! or V![201]`	`R0001`
`V202! or V![202]`	`R0002`
`V203! or V![203]`	`R0003`
. . .	. . .
`V248! or V![248]`	`R0048`
`V249! or V![249]`	`R0049`
`V250! or V![250]`	`R0050`

CAUTION

For precautions for using global real number variables `V201!` to `V250!`, refer to [Precautions for using global integer variables `V201%` to `V250%`].

2.5.3 Character string variables

"Character string variables" handle character strings. ASCII and Shift JIS are handled. The 2-byte codes specified by shift JIS can also be handled also.

Table 2.13 Character string variables

Format	`Vn$, V$[n]` n=1 to 50 (Variables can be also used.) `Ln$, L$[n]` n=1 to 50 (Variables can be also used.) `V1$` and `V$[1]` denote the same variable area.
Range	0 to 199 characters (199 bytes)

Sample	`V1$="Execution state"`
	`L$[1]=V1$+"Under suspension"`
	`LETVS V1$, "1A"`

Storage	The first ten global character string variables are retained as power failure retention data even if the main power supply is turned OFF. In contrast, all local character string variables are not retained.

2.5.4 Timer variables

Timer variables are not supported at the present time.

2.5.5 Input signal variables

These handle the input ports in bit or byte units.

Table 2.14 Input signal variables

	When the ports are handled in bits	When the ports are handled in bytes

Range	`In, I[n]`
	`n=1 to 2048, 5101 to 5196`
	(variables can be used)
		`IBn, IB[n]`
		`n=1 to 204`
		(variables can be used)

Range	`0,1`	`0 to 255`
Sample	`WAIT I[1],0,0`	`WAITAD IB[1],255,0,100`

The input signal variables are referenced only: they cannot be written.
Refer to the table below when the ports are handled in bytes. The numbers of the groups in the table correspond to "n" of `IB[n]`.

Table 2.15 Group numbers of input signals

Group	Input signal	Group	Input signal	Group	Input signal
1	1 to 10	11	101 to 110	21	201 to 210
2	11 to 20	12	111 to 120	…	…
3	21 to 30	13	121 to 130	30	291 to 300
4	31 to 40	14	131 to 140	…	…
5	41 to 50	15	141 to 150	50	491 to 500
6	51 to 60	16	151 to 160	…	…
7	61 to 70	17	161 to 170	100	991 to 1000
8	71 to 80	18	171 to 180	…	…
9	81 to 90	19	181 to 190	…	…
10	91 to 100	20	191 to 200	204	2031 to 2040

2.5.6 Output signal variables

These variables handle the output ports in bit or byte units.
The output signals cannot be operated by assigning them directly to the output signal variables.
These variables are used as parameters of the OUT and other application instructions.

Table 2.16 Output signal variables

	When the ports are handled in bits	When the ports are handled in bytes
Range	`O, O[n]` `n=1 to 2048` (variables can be used)	`OBn, OB[n]` `n=1 to 204` (variables can be used)
Range	`0,1`	`0 to 255`
Examples	`SET O1／RESET O[2]`	`OUT OB205, 0`

Refer to the table below when the ports are handled in bytes. The number of the group in the table corresponds to "n" of OB[n].

Table 2.17 Group numbers of output signals

Group	Output signal
1	1 to 10
2	11 to 20
3	21 to 30
4	31 to 40
5	41 to 50
6	51 to 60
7	61 to 70
8	71 to 80
9	81 to 90
10	91 to 100
11	101 to 110
12	111 to 120
13	121 to 130
14	131 to 140
15	141 to 150
16	151 to 160
17	161 to 170
18	171 to 180
19	181 to 190
20	191 to 200
21	201 to 210
…	…
30	291 to 300
…	…
50	491 to 500
…	…
100	991 to 1000
…	…
…	…
204	2031 to 2040

2.5.7 Pose variable

The "Pose variables" hold the robot poses.
Please write the values using `MOVEX-X`, `MOVEX-J`, or `MOVEX-E` representation.
If there are additional axes like e.g. servo gun etc, write the joint value at the end.
Example: `P1=(1800,0,2000,0,-90,-180`**`,-100`**`)`

Table 2.18 Pose variable

Format	`Pn, P[n]` n=1 to 999 (variables can be used)
Examples	`P1=(100,0,100,0,0,90)`

(Example) The mechanism 1 is a robot (6-axes) and the mechanism 2 is a servo gun (1 axis)

```
REM "MOVEX-X"
P1=(1800,0,2000,0,-90,-180,-100)
REM "MOVEX-J"
P2=(0,90,0,0,0,0,-100)
REM "MOVEX-E"
P3=(&H80000,&H80000,&H80000,&H80000,&H80000,&H80000,&H80000)
MOVEX A=1,M1X,P,P1,R=25.0,H=1,MS,M2X,P,P1,R= 100,H=1
MOVEX A=1,M1J,P,P2,R=25.0,H=1,MS,M2J,P,P2,R= 100,H=1
MOVEX A=1,M1E,P,P3,R=25.0,H=1,MS,M2E,P,P3,R= 100,H=1
END
```

In this example, because the total number of the axes is 7, the number of the parameters of pose variable is 7. Please be sure that the pose variable is used for both of mechanism 1 and 2. The mechanism 1 gets the first 6 values and the mechanism 2 gets the 7th data.

Pose variables is not kept when the main power is lost.
Even when number is set to pose variable, pose file is not renewed.
By executing function `FN74 POSESAVE`, pose variable can be copied to pose file.

2.5.8 Shift variable

The "Shift variables" hold the shift amounts.
This variable uses `MOVEX-X` representation (X, Y, Z, r, p, y).

Table 2.19 Shift variable

Format	`Rn, R[n]` n=1 to 8 (variables can be used)
Examples	`R1=(10,1,0,0,0,0)`

2.6 Expressions and operations

An "expression" refers to general numerical formulas and functions which connect variables in operation expressions or simply to characters and numerical values or to variables only.

<Eamples>

- `"ABC"`
- `3＊5/4+10`
- `V1%+V2%*V1!`
- `SIN(V3!)`
- `1023`
- `V$[4]`

Operations use operators and functions to operate the expressions, and they are classified into the six types below.

- Arithmetic operations
- Relational operations
- Logical operations
- Character string operations
- Pose operations
- Functions

2.6.1 Arithmetic operations

The following operators can be used as arithmetic operators.

Table 2.20 Arithmetic operators

Operator	Description of operation	Example
`^`	Exponential (power) operations	`10^2`
`-`	Minus sign	`-100`
`*`	Multiplication	`3*5`
`/`	Division	`3/2`
`¥`	Division (quotient)	`5¥2`
`MOD`	Division (remainder)	`5 MOD 2`
`+`	Addition	`10+12`
`-`	Subtraction	`103-99`

2.6.2 Relational operations

Relational operations are used when comparing two numerical values. The results are obtained as true (1) or false (0) and used, for instance, to change the execution sequence of conditional decision statements and other programs.

Table 2.21 Relational operator

Operator	Description of operation	Example
`=`	Is equal to	`V1%=V2%`
`<>`	Is not equal to	`V1%<>V2%`
`<`	Less than	`V1%<V2%`
`>`	Greater than	`V1%>V2%`
`<=`	Less than or equal to	`V1%<=V2%`
`>=`	Greater than or equal to	`V1%>=V2%`

2.6.3 Logical operations

Logical operations connect multiple numbers of conditions, and they are used for bit operations.

Table 2.22 Logical operators

Operator	Description of operation	Example
`NOT`	Negation	`NOT V1%`
`AND`	Logical product	`V1% AND V2%`
`OR`	Logical sum	`V1% OR V2%`
`XOR`	Exclusive `OR`	`V1% XOR V2%`

2.6.4 Character string operations

Character string operations are restricted to connecting character strings using the "+" operator and comparing character strings using relational operators. Character strings are connected by inserting the "+" operator between two character strings.

For example:

```
V1$=″ABC″
V2$=V1$+″123″
```

In this case, "`ABC123`" character string is assigned to `V2$`. In addition, character strings can be compared using the "=" and "<>" relational operators. If the two character strings are equal, an equality sign is used (an equality is established). On the other hand, if they are not equal, an inequality sign is used (an inequality is established).

2.6.5 Pose operations

Pose operations are limited solely to the addition of pose constants (variables) and shift constants (variables). And calculation result is not stored to the recorded pose variables.

<Examples>

・ `P1=P1+R1`
・ `P1=P1+(100,0,0,0,0,0)`
・ `P1=(0,1000,1200,90,0,90)+(80,0,0,0,0,0)`
・ `P1=(0,2000,100,0,0,0)+R1`

2.6.6 General functions

Functions perform specific operations for specific arguments, and they return the results of those operations.

Types and descriptions of the functions

Variables can be used for `f/p/s` and or so.
Each input range is as followed.

- In case of integer variables ; `-2147483648` to `+2147483647`
- In case of real variables ; `-1.0E38` to +1.0E38
- In case of characters ; 0 to 199 characters (199 bytes)

Table 2.23 Functions

Name of function	Description of function	Attribute
`DATE$`	Converts the current date into a character string, and returns it.	Character string
`TIME$`	Converts the current time into a character string, and returns it.	Character string
`TIMER`	Returns the time (in milliseconds) which has elapsed since the power was turned on.	Real number value
`SQR(f)`	Obtains the square root.	Real number value
`SIN(f)`	Obtains sin(f) (where angle f is a radian).	Real number value
`COS(f)`	Obtains cos(f) (where angle f is a radian).	Real number value
`TAN(f)`	Obtains tan(f) (where angle f is a radian).	Real number value
`ATN(f)`	Obtains atan(f).	Real number value
`ATN2(f1, f2)`	Obtains atan(f1/f2).	Real number value
`ABS(f)`	Obtains the absolute value.	Real number value
`DEGRAD(f)`	Converts a degree value into a radian value.	Real number value
`RADDEG(f)`	Converts a radian value into a degree value.	Real number value
`MIN(f1, f2)`	Obtains whichever is lower, f1 or f2.	Real number value
`MAX(f1, f2)`	Obtains whichever is higher, f1 or f2.	Real number value
`ORD(s)`	Returns the first character code of a character string.	Integer value
`CHR$(f)`	Returns the character string of length 1 in which value f is the character code.	Character string
`VAL(s)`	Converts the numerical value expressed by a character string into a value.	Real number value
`STR$(f)`	Converts a numerical value into a character string.	Character string
`BIN$(I)`	Converts a numerical value into a character string expressed in binary notation.	Character string
`HEX$(i)`	Converts a numerical value into a character string expressed in hexadecimal notation.	Character string
`MIRROR$(s)`	Returns the character string in which the character string s list has been reversed.	Character string
`LEFT$(s, f)`	Cuts out the character string of length f from the left of character string s.	Character string
`RIGHT$(s, f)`	Cuts out the character string of length f from the right of character string s.	Character string
`MID$(s,f1, f2)`	Cuts out the character string of length f2 starting with the f1-th character from the left of character string s.	Character string
`STRPOS(s1, s2)`	Returns the position that first coincides with character string s2 in character string s1 (199 if the position is not found; 0 if there is no s1 or s2 data).	Integer value
`LEN(s)`	Returns the length of a character string.	Integer value

Types and descriptions of input functions

Table 2.24 Input functions

Name of function	Description of function	Attribute
`INP(i)`	Used to return the value of input signal specified with the argument "I" through setting it "0" (OFF) or "1" (ON). ("`i`" can be specified with `1` to `2048` and `5101` to `5196`.)	Integer value
`INPB(i)`	Used to get input signals specified with the argument "i" in bytes and then covert the signals into decimal integer values to return them. ("`i`" can be specified with `1` to `256`.) Refer to "Table 2.25 Argument "i" of input function INPB".	Integer value
`GETSIG(i1,i2)`	Used to get input/output signals specified with the argument "i1" in bits. ("i1" can be specified with 1 to 2048 and 5101 to 5196. For output signals, however, 1 to 2048 are only available.) When `i2=0`, input signals will be got. When `i2=1`, output signals will be got.	Integer value
`GETSIGB(i1,i2)`	Used to get input/output signals specified with the argument "i1" in bytes. ("`i1`" can be specified with `0` to `259`.) When i2=0, input signals will be got. When i2=1, output signals will be got. Refer to "Table 2.26 Argument "i1" of input function GETSIGB"	Integer value

Table 2.25 Argument "`i`" of input function `INPB`

i	Input number
1	0001 to 0008
2	0009 to 0016
3	0017 to 0024
4	0025 to 0032
5	0033 to 0040
6	0041 to 0048
7	0049 to 0056
8	0057 to 0064
9	0065 to 0072
10	0073 to 0080
11	0081 to 0088
12	0089 to 0096
13	0097 to 0104
14	0105 to 0112
15	0113 to 0120
	*** to ***
	*** to ***
242	1929 to 1936
243	1937 to 1944
244	1945 to 1952
245	1953 to 1960
246	1961 to 1968
247	1969 to 1976
248	1977 to 1984
249	1985 to 1992
250	1993 to 2000
251	2001 to 2008
252	2009 to 2016
253	2017 to 2024
254	2025 to 2032
255	2033 to 2040
256	2041 to 2048

Input numbers 1 to 2048 are grouped by every eight inputs.

Table 2.26 Argument "`i1`" of input function `GETSIGB`

i1	Input/output number
0	Board internal I/O
1	Board internal I/O
2	Board internal I/O
3	Board internal I/O
4	0001 to 0008
5	0009 to 0016
6	0017 to 0024
7	0025 to 0032
8	0033 to 0040
9	0041 to 0048
10	0049 to 0056
11	0057 to 0064
12	0065 to 0072
13	0073 to 0080
14	0081 to 0088
	*** to ***
	*** to ***
245	1929 to 1936
246	1937 to 1944
247	1945 to 1952
248	1953 to 1960
249	1961 to 1968
250	1969 to 1976
251	1977 to 1984
252	1985 to 1992
253	1993 to 2000
254	2001 to 2008
255	2009 to 2016
256	2017 to 2024
257	2025 to 2032
258	2033 to 2040
259	2041 to 2048

The board internal I/O and input/output numbers 1 to 2048 are grouped by every eight inputs/outputs.
The same numbers are assigned to input and output signals.

2.6.7 System functions

"System functions" are provided in advance for reading the information inside this controller such as the positions of the robot axes. The following types are available.

Table 2.27 System functions

Name of function	Description of function	Attribute
`ERRMES$(i)`	Returns an error message of the control unit that corresponds to error number `i`.	Character string
`SYSTEM%(i)`	Returns the integer information held by the control unit that corresponds to `i`. ☞ Refer to "Table 2.28 Return value of system function `SYSTEM%`"	Integer value
`SYSTEM!(i)`	Returns the real number information held by the control unit that corresponds to `i`. ☞ Refer to "Table 2.29 Return value of system function SYSTEM!"	Real number value
`SYSTEM$(i)`	Returns the character string information held by the control unit that corresponds to `i`. ☞ Refer to "Table 2.30 Return value of system function SYSTEM$"	Character string

Table 2.28 Return value of system function `SYSTEM%`

Argument	Return value
0	User task status Bit 0: User task program now starting Bit 1: 2nd or subsequent cycle from startup Bit 2: User screen now open Bit 3: User screen now temporarily closed Bit 4: 2nd or subsequent cycle after user screen opened Bit 5: User screen active status (key information is coming) Bit 6 to 31: Not used
1	Number of units
2	Mode (0 = teach, 1 = playback, 2 = high-speed teach)
4	Playback mode (0 = 1-step, 1 = 1-cycle, 2 = continuous)
100	Number of user task program now being executed
101 to 109	Numbers of task programs now being executed in each unit 1 to 9
110(120)	Number of user task 1 playback line
111 to 119	Numbers of task programs now being executed in each unit 1 to 9
131 to 139	Numbers of unit axes for each unit 1 to 9
141 to 149	Numbers of unit errors for each unit 1 to 9
151 to 159	Shift status for each unit 1 to 9 (0; not in shift, 1; now shifting)
161 to 169	In-position status for each unit 1 to 9 (0; not in-position,1; in-position)
200 to 205	Current encoder values of axes J1 to J6 of mechanism 1
210 to 215	Current encoder values of axes J1 to J6 of mechanism 2
220 to 225	Current encoder values of axes J1 to J6 of mechanism 3
230 to 235	Current encoder values of axes J1 to J6 of mechanism 4
240 to 245	Current encoder values of axes J1 to J6 of mechanism 5
250 to 255	Current encoder values of axes J1 to J6 of mechanism 6
260 to 265	Current encoder values of axes J1 to J6 of mechanism 7
270 to 275	Current encoder values of axes J1 to J6 of mechanism 8
280 to 285	Current encoder values of axes J1 to J6 of mechanism 9
1000 to 1003	Playback program number of each user task 1 to 4
1010 to 1013	Playback line number of each user task 1 to 4

Table 2.29 Return value of system function `SYSTEM!`

Argument	Return value	Units
`100 to 105`	Current joint angles of axes J1 to J6 of mechanism 1	rad
`110 to 115`	Current joint angles of axes J1 to J6 of mechanism 2	rad
`120 to 125`	Current joint angles of axes J1 to J6 of mechanism 3	rad
`130 to 135`	Current joint angles of axes J1 to J6 of mechanism 4	rad
`140 to 145`	Current joint angles of axes J1 to J6 of mechanism 5	rad
`150 to 152`	Current fingertip positions (X, Y, Z coordinates) of mechanism 1	mm
`153 to 155`	Current fingertip positions (X, Y, Z coordinates) of mechanism 2	mm
`156 to 158`	Current fingertip positions (X, Y, Z coordinates) of mechanism 3	mm
`159 to 161`	Current fingertip positions (X, Y, Z coordinates) of mechanism 4	mm
`162 to 167`	Current fingertip positions (X, Y, Z coordinates) of mechanism 5	mm
`200 to 205`	Current joint angles of axes J1 to J6 of mechanism 6	rad
`210 to 215`	Current joint angles of axes J1 to J6 of mechanism 7	rad
`220 to 225`	Current joint angles of axes J1 to J6 of mechanism 8	rad
`230 to 235`	Current joint angles of axes J1 to J6 of mechanism 9	rad
`250 to 252`	Current fingertip positions (X, Y, Z coordinates) of mechanism 6	mm
`253 to 255`	Current fingertip positions (X, Y, Z coordinates) of mechanism 7	mm
`256 to 258`	Current fingertip positions (X, Y, Z coordinates) of mechanism 8	mm
`259 to 261`	Current fingertip positions (X, Y, Z coordinates) of mechanism 9	mm

In the case of a servo gun, slider or other rectilinear axis, the current joint angle is expressed in [mm] rather than [rad].

Even in the inch mode, the return values for lengths are expressed in [mm].

Table 2.30 Return value of system function `SYSTEM$`

Argument	Explanation
`0`	Software version
`101`	Unit name of unit 1
`102`	Unit name of unit 2
`103`	Unit name of unit 3
`104`	Unit name of unit 4
`105`	Unit name of unit 5
`106`	Unit name of unit 6
`107`	Unit name of unit 7
`108`	Unit name of unit 8
`109`	Unit name of unit 9
`300 to 2347`	Names of input signals
`2348 to 4098`	Names of output signals

2.6.8 Priority of operators

When a multiple number of operators are present in an expression, the sequence in which they are applied is determined by the sequence of priority which has been set for each operator. The sequence of priority is set as shown below.

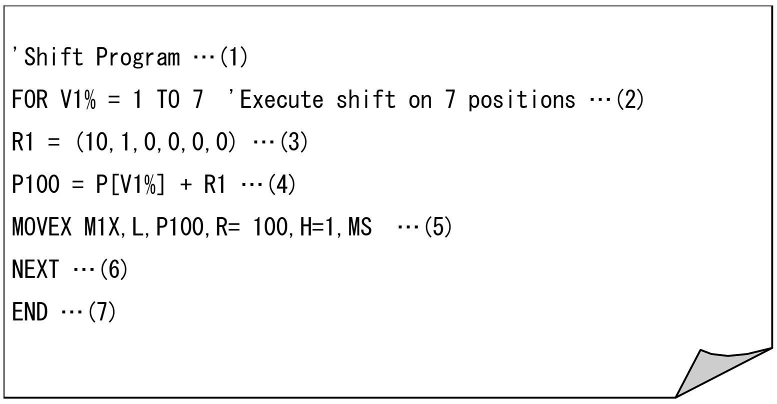
Table 2.31 Priority of operators

Operator	Priority
+,- (signs)	Higher
NOT	
^	
/,*,¥,MOD	
+(addition), -(subtraction)	
=,<>,>,<,>=,<=	
AND	
OR	
XOR	Lower

2.7 Statements

Statements must be allocated on a one statement per line basis in source programs. In this manual, one line of a program and one statement are considered to be the same unless otherwise specified.

Robot Language Program

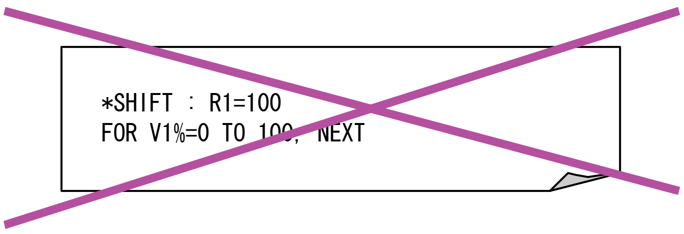


```
'Shift Program …(1)
FOR V1% = 1 TO 7  'Execute shift on 7 positions …(2)
R1 = (10,1,0,0,0,0) …(3)
P100 = P[V1%] + R1 …(4)
MOVEX M1X,L,P100,R= 100,H=1,MS  …(5)
NEXT …(6)
END …(7)
```

(1) Comment statement	(2) Flow control statement + Comment statement
(3) Substitution statement	(4) Substitution statement
(5) Command statement	(6) Flow control statement
(7) Flow control statement	

Fig 2.7.1 Statements

Robot Language Program



```
*SHIFT : R1=100
FOR V1%=0 TO 100, NEXT
```

Two or more statements cannot be described on a single line.

Fig 2.7.2 Description of only one statement on a single line

POINT

- Two or more statements cannot be described on a single line.
- Insertion of any line feed code in a statement is prohibited.
- A maximum of 254 characters can be used in a statement. Writing 255 or more characters on a single line will result in a compiling error.

2.7.1 Comment statement

If a statement starts with "'" (a single quotation mark) or the description "REM," the line concerned is a comment statement. Furthermore, comment statement can be described after the command statement and flow control statement with a single quotation mark.
Comment statements are written to make it easier to understand the program. Therefore, the contents of comment statements are not executed. Comments may be up to 199 characters long.

<Examples>

- `’SPOT1 Program`
- `REM "ARC ON"`
- `SET O1  ‘Task complete`

CAUTION

Maximum line number of the program with comment statement is fewer than that of the program without comment statement, because comment statement needs memory to store them. Maximum line number varies depending on the length of comment statement.

2.7.2 Labels

Any statement starting with "*" (an asterisk) and followed by letters of the alphabet is identified as a label. GOTO, IF and other statements are provided as instructions that control the program flow, and label names are specified as their branch destinations.
Labels are written using up to 16 characters starting with a letter of the alphabet. The character sthat may be used are alphanumerics, "." (period) and "_" (underbar). The total maximum number of characters which can be written in labels in a program is 1024.

<Examples>

- `*HANDLING1`
- `*ARC_WELDING`

2.7.3 Substitution statement

Values can be substituted for the respective variables in a program whether the variables are integer variables, real number variables, character string variables, shift variables or pose variables.
Any statement starting with the name of a variable and followed by "=" is identified as a substitution statement.

<Examples>

- `V1%=10`
- `V3$=V1$`
- `R1=(100,0,0,0,0,0)`
- `P1=(100,1000,1000,0,90,90)`

2.7.4 Flow control statement

The instructions used to change the program flow such as the repeat execution, branch, and subroutine call instructions are called flow control statements. Each type of flow control statement is described below.

Table 2.7.3 Flow control statement

Command	Outline
`GOTO` line number/label	This transfers control unconditionally to the line indicated by the line number or label. Example) `GOTO 100` `GOTO *LOOP_END`
`GOSUB` line number/label	This transfers control to the subroutine indicated by the line number or label name. Example) `GOSUB 120` `GOSUB *FUNC1`
`RETURN`	This returns control from the subroutine to the call source (line following the line on which the subroutine was called by `GOSUB`).
`IF` conditional expression `THEN` line number/label `ELSE` line number/label	This transfers control to the line number or label written after THEN if the conditions are satisfied. It transfers control to the line number or label written after ELSE if the conditions are not satisfied. If ELSE is not written and the conditions are not satisfied, it transfers control to the next line. A line number or label must be written after THEN or ELSE without fail. Execution statements cannot be written. Example) `IF V1%=1 THEN 100 ELSE 200` `IF V1%>100 AND V1%<200 THEN *START`

`FOR` variable name = initial value `TO` end value `STEP` increment
`NEXT`
	This repeatedly executes the instruction written in the loop between `FOR` and `NEXT`. The variable name must be an integer variable (`V%`, `L%`) or real number variable (`V!, L!`). The variable must be specified directly (as `V1%` not as `V%[1]`). The initial value is the value which is set into the variable in the initial loop. When the instruction reaches `NEXT`, the increment is added to the variable and compared with the end value. If the instruction has exceeded the end value, control is transferred to the line following `NEXT`. In all other cases, it transfers to the instruction following `FOR`. "+1" is used as the increment when `STEP` has been omitted. It is possible to write another loop between `FOR` and `NEXT` inside a loop between FOR and `NEXT`. This is referred to as nesting. In nesting, different variable names must be used without fail. The maximum number of nesting level is 4. The following precautions must be heeded in order for `FOR` statements to be used.
	(1) It is not possible to make changes to the variables used by `FOR` statements inside the same loop between `FOR` and `NEXT`.
	(2) It is not possible to write instructions for jumping out of a `FOR`-`NEXT` loop inside the loop (such as `GOTO` and `RETURN`).
	(3) If increment is 0, this loop never ends because variable never changes.
	(4) If sigh of increments and initial/end number does not match, loop is never started and program is aborted.
	(5) If reverse conversion is executed, increment is surely recorded.
	(6) If increment is not 1, loop will end when variable becomes larger than the end value.
	(7) If variable is integer and increment is decimal (smaller than 1), this loop never ends because decimal is ignored.
	(8) If increment includes decimal value, loop may act incorrectly because of miscalculation.
	Beware that above situations are never detected as compile error.
	
	Example) `FOR V1%=0 TO 100`
	  `FOR V2%=0 TO 10`
	    `V3%=V1%*10+V2%`
	    ~~V1%=2~~ ････Prohibited
	    ~~RETURN~~ ････Prohibited
	  `NEXT`
	`NEXT`

Command	Outline

`WHILE`
conditional expression
`ENDW`
	While condition expression is satisfied, commands between `WHILE` and `ENDW` are repeated. One `WHILE/ENDW` loop can contain another `WHILE/ENDW` loop. Maximum number of nesting level is 4.
	
	Example)
	    `V1% = 0`
	    `WHILE V1% < 10`
	      `. . .`
	      `. . .`
	    `V1% = V1% + 1`
	    `ENDW`

`IF` conditional expression
`ELSEIF`
 conditional expression
`ELSE`
`ENDIF`
	When `IF` conditional expression is satisfied, commands between `IF` and `ELSEIF` is executed.
	When `IF` conditional expression is not satisfied and `ELSEIF` conditional expression is satisfied, commands between `ELSEIF` and `ELSE` is executed.
	When `IF` conditional expression is not satisfied and `ELSEIF` conditional expression is not satisfied, commands between `ELSE` and `ENDIF` is executed.
	Following statement is also available,
	
	Example 1)
	    `IF` conditional expression
	
	    `ENDIF`
	
	Example 2)
	    `IF` conditional expression
	
	    `ELSEIF` conditional expression
	
	    `ENDIF`
	
	Example 3)
	    `IF` conditional expression
	
	    `ELSE`
	
	    `ENDIF`
	
	Example 4)
	    `IF` conditional expression
	
	    `ELSEIF` conditional expression
	
	    `ELSEIF` conditional expression
	
	    `ELSE`
	
	    `ENDIF`
	
	One `IF` – `ENDIF` can contain another `IF` – `ENDIF` Maximum number of nesting level is 4.

Command	Outline

`SWITCH` operation
`CASE` integer constant
`BREAK`
`ENDS`
	Execute only one command between `CASE` and `ENDS` corresponding to the value of integer constant as the result of `SWITCH` operation.
	
	Example)
	`SWITCH V1%`
	`CASE 1`
	・・・
	`BREAK`
	`CASE 2`
	`CASE 3`
	・・・
	`BREAK`
	`CASE`
	・・・
	`BREAK`
	`ENDS`
	
	When `V1%=1`, commands between `CASE 1` and first `BREAK` and executed.
	When `V1%=2`, commands between `CASE 2` and second `BREAK` are executed.
	When `V1%=3`, commands between `CASE 3` and second `BREAK` are executed.
	If `V1%` is not `1` neither `2` and `3`, commands between `CASE` and last `BREAK` s executed.
	
	NOTE)
	`CASE` statement described in advance is estimated in advance. In case of the following Example 1, the commands between first `CASE` and first `BREAK` are executed irrelevant to the value of `V1%`.
	Example 1)
	`SWITCH V1%`
	`CASE`
	・・・
	`BREAK`
	`CASE 1`
	・・・
	`BREAK`
	`CASE 2`
	・・・
	`BREAK`
	`ENDS`
	
	In case of the following Example 2, when `V1%=1`, commands between first `CASE 1` to first `BREAK` is executed.
	Example 2)
	`SWITCH V1%`
	`CASE 1`
	・・・
	`BREAK`
	`CASE 1`
	・・・
	`BREAK`
	`CASE 2`
	・・・
	`BREAK`
	`CASE`
	・・・
	`BREAK`
	`ENDS`

Command	Outline
ON integer variable GOTO line number/label, line number/label...	This transfers control to the line number or label written at the same order as that of the value indicated by the conditional expression. The line numbers and labels are assigned in ascending order by number such as 1, 2, 3 ... from the left. Example) `ON V1% GOTO 100, 200, 300` This example shows the case where control will be transferred to the 100th line when `V1%=1` and to the 200th line when `V1%=2`. A maximum of 10 line numbers or labels can be described
STOP	This stops the execution of the program.
END	The program is ended. When a playback program is executed in the continuous playback mode, it is re-executed from the start of the program. **At least one END instruction must be written per program.**

2.7.5 Command statement

The command statements, such as the `MOVE-X` commands for moving the robot and the `SET` commands for turning ON the output signals, constitute the heart of a program. The robot language programs consist almost entirely of command statements.
These are very important statements, and many instructions are found in them.

Chapter 3 Program editing

This chapter describes how to edit robot language programs. There is a choice of two methods: one involves the use of a text editor which is available on the market and is run in a personal computer, and the other edits directly from the teach pendant. Also described is how to create pose files using the robot.

3.1 Editing using a personal computer

To create or edit a robot language source program, a text editor software in the market can be used on your PC.

3.1.1 Precautions for editing

Filename regulations	Refer to Chapter 1.
Editing method	Refer to the instructions accompanying the personal computer and text editor used.
Size of the files	Ensure that the size of the files is no more than 64KB (65534 bytes).
Precautions for statements, lines and characters	A total of 254 characters may be written on one line, and up to 999 lines can be written in a program. No differentiation is made between upper-case and lower-case letters of the alphabet. Use half-size characters for all text except for comments and character string variables. Also read carefully through the precautions set forth in "Chapter 2 Syntax" before proceeding.
Other	Create files using the shift JIS code for full-size characters and CR+LF for the line feed codes.

3.1.2 Loading a robot language source program into this controller

To copy a robot language source program file to the internal memory of this robot controller, please use (1) or (2). (It is recommended to use USB memory.)

(1) Copying the file using USB memory

Insert a USB memory and then open <Service Utilities> - [7 File Manager] - [1 File Copy] screen to copy the file from the USB memory to the internal memory. For details, refer to the following instruction manual.

"BASIC OPERATIONS MANUAL" Chapter 6

(2) Copying the file using Ethernet and FTP

After connecting this controller and your PC with a Ethernet cable, please transfer the file using a FTP client software on the PC. For details, refer to the following instruction manual.

"SETUP MANUAL" Chapter 8

INFO.

A FTP client software can be purchased in the market etc.

In either way, please do not forget to place the program file in the following folder. The program files not placed in this folder will be ignored by this robot controller.

```
¥Memory¥WORK¥PROGRAM¥
```

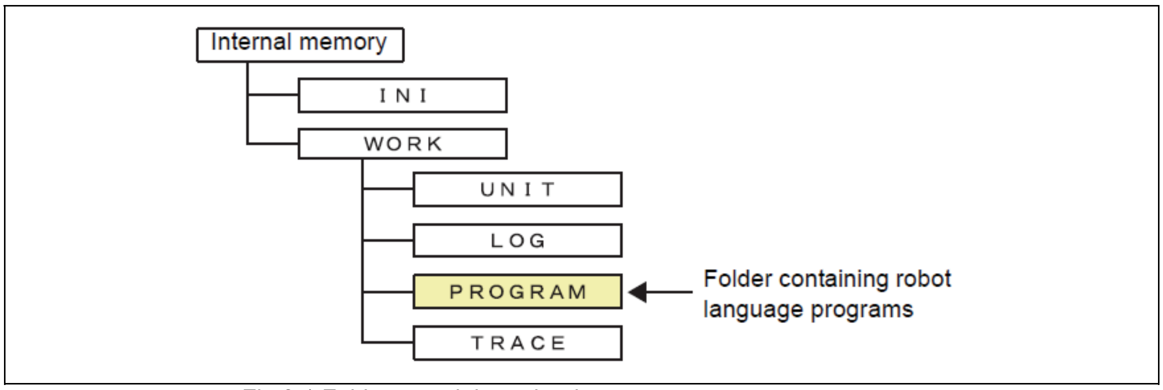


Fig 3.1 Folder containing robot language programs

3.2 Editing using the teach pendant

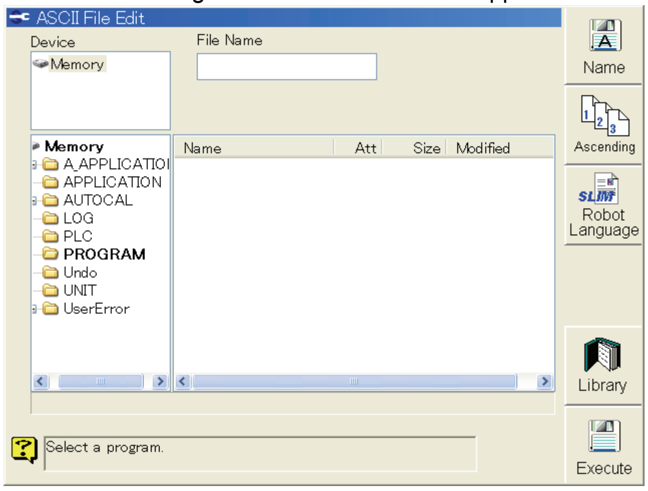
Use of the "ASCII file editing function" enables robot language files to be directly edited using the teach pendant of the AX control unit.
This is ideal for correcting mistakes made in writing the programs and making other simple modifications.

3.2.1 Basic file editing operations

Service Utilities

1 **Press [Service Utilities] f key.**
≫Service menu list will appear.

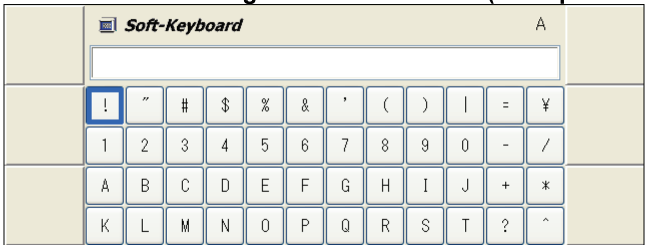
2 **Select [15 ASCII File Edit] and press [Enter].**
≫ASCII file editing screen shown below will appear.



Select a program.

Listed on this screen are only those files which can be edited using the ASCII file editing function (edit-enable files). The term "edit-enable files" refers to robot language files, that is to say, files which have "-A" added onto the end of their filenames to indicate that the files are text files and which have an extension in the form of a number (files such as "`SRA166-1`**`-A`**`.010`"), and also files with the "TXT" extension.

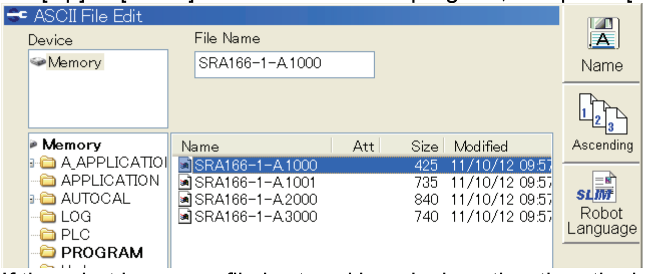
3 **When creating a new file, align the cursor with [File Name], and input the name of the new file to be created.**
Open the soft keyboard by pressing [ENABLE] and [EDIT], and input the entire character string with no omissions. (Example: SRA166-1-A.010)



Complete

4 **Press f12 [Complete] key upon completion of the character input.**
>> Back to [ASCII File Edit] screen.

5 **If a previously created robot language file is to be selected, select the file to be edited by following the steps below.**
If the robot language file is stored in the internal memory, select "Internal memory" for the device and "PROGRAM" for the folder. (In actual fact, this is the ¥WORK¥PROGRAM folder.)
>>A screen listing the edit-enable files such as the one shown below appears. Use the [Up] or [Down] cursor to select the program, and press [Enter].


ASCII File Edit

Device: Memory

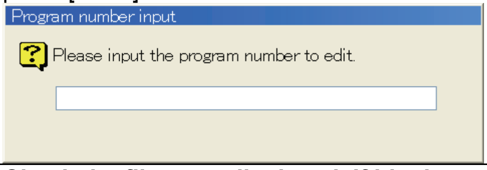
File Name: SRA166-1-A1000

Name	Att	Size	Modified
SRA166-1-A1000		425	11/10/12 09:57
SRA166-1-A1001		735	11/10/12 09:57
SRA166-1-A2000		840	11/10/12 09:57
SRA166-1-A3000		740	11/10/12 09:57

If the robot language file is stored in a device other than the internal memory, select the device concerned and folder, and then select the file to be edited in the same way.

Robot Language

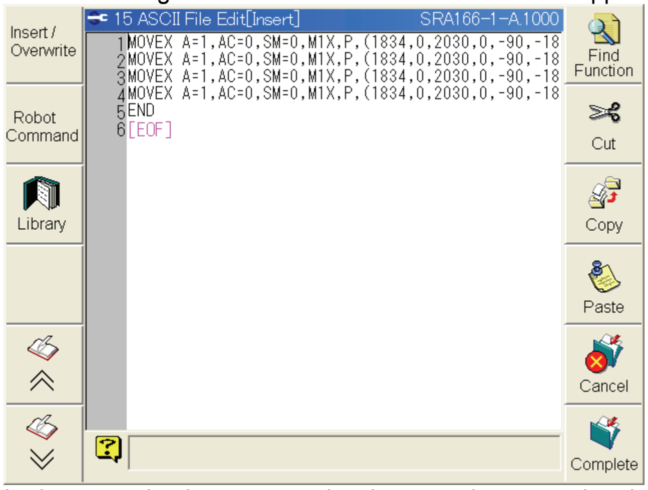
[When selecting a file among the robot language files]
Under the condition of "Memory" for "Device" and "PROGRAM" for "folder, it is possible to select a program for the current unit by inputting a program number.
>>A screen such as the one shown below appears. Input the program number and press [Enter].


Program number input

Please input the program number to edit.

Execute

6 **Check the filename displayed. If it's the correct press the f12 [Execute] key.**
>>A text editing screen such as the one shown below appears.

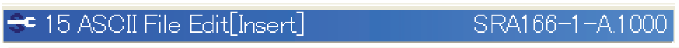

15 ASCII File Edit[Insert] SRA166-1-A1000

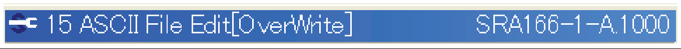
```
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
END
[EOF]
```

In the example shown, a previously created program has been selected. When a new program number has been specified, nothing appears in center editing screen.

Insert / Overwrite

7 **Each time the f1 [Insert/Overwrite] key is pressed, the mode is switched between insert and overwrite.**
>>Whether the insert or overwrite mode has been established is displayed all the time on the taskbar.

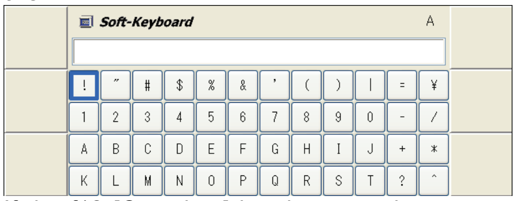
"Insert"

15 ASCII File Edit[Insert] SRA166-1-A1000

"Overwrite"

15 ASCII File Edit[OverWrite] SRA166-1-A1000

8 **Use the [Up], [Down], [Left] and [Right] cursor keys to move the cursor to the position where the editing is to be performed using the cursor keys. The page can be scrolled up or down using [ENABLE] and the [Up] or [Down] cursor key.**

9 **To input numerical values, use the number keys [0] to [9].**

10 **To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the soft keyboard, and then input them.**



If the f12 [Complete] key is pressed upon completion of the character input, the original screen is restored.

11 **When the [BS] key is pressed, the character to the left of the cursor position is deleted.**

12 **When the [DEL] key is pressed, the character to the right of the cursor position is deleted.**

13 **When the [Enter] key is pressed, a line feed (CR+LF) is input.**
>>In actual fact, there is no CR+LF display. The next line appears on the display, and the cursor moves to the next line.

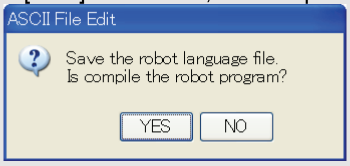
14 **When the [FN] key is pressed, a space can be input.**

15 **When the f5 or f6 key is pressed, the cursor can be made to jump to the start or end of each file.**

16 **Upon completion of the editing, press the f12 [Complete] key.**
>>What has been edited is now saved in the file. The data is saved without checking for syntax errors, etc. Since no backup files are left, care must be taken in executing these operations.
When the file saving process has completed, the [ASCII File Edit] screen will close.

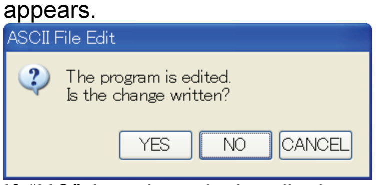
POINT

If a robot language source program file is created or modified in the "¥WORK¥PROGRAM" folder in the [Memory], a confirmation message for compile execution like the one shown as below is displayed when saving the file.
If [YES] is selected, the compilation process will start.


ASCII File Edit

Save the robot language file.
Is compile the robot program?

17 **If the editing is to be canceled and the file is not going to be saved, press the f11 [Cancel] or [RESET/R] key.**
>>A pop-up message requesting confirmation, such as the one shown below, now appears.


ASCII File Edit

The program is edited.
Is the change written?

If "NO" is selected, the display will return to the menu screen of each service without saving the files, destroying all the contents that were edited, and will end editing.
If "CANCEL" is selected, the pop-up message will disappear, returning to the [15 ASCII File Edit] screen, and editing can be continued.

POINT

After creating robot language program with [ASCII file edit], beware to proceed compiling to convert to the executable file with [Service utility]->[9 Program conversion]->[8 Language conversion].
Unless otherwise executing compiling, modified program is never affected to the executable program.

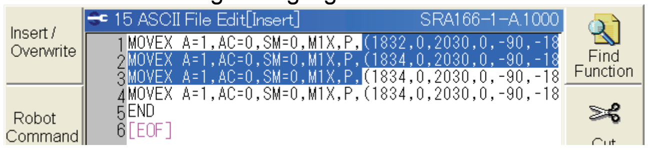
3.2.2 Useful editing functions

Delete multiple lines at one time

Cut

1 **Press the f8 [Cut] key, and select the lines to be deleted using the [Up] or [Down] cursor key.**

>>The selected range is highlighted.



```
MOVEX A=1,AC=0,SM=0,M1X,P,(1832,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
END
[EOF]
```

2 **Select the range that is to be deleted by using the [Up] or [Down] cursor keys and press [Enter].**

>> By using the [Up] or [Down] cursor keys, a multiple number of lines can be selected as the deleting range. The selected range will be highlighted and when [Enter] is pressed, all of those highlighted lines will be deleted. After the lines have been deleted, the lines that were undeleted will replace the deleted lines, without altering its order, and be re-displayed.

To cancel this at any time, press the [RESET/R] key.

Move multiple lines at one time

Paste

1 **After selecting multiple lines in the same way as line deletion, put the cursor to the position right after the target place and press [Paste].**

≫The multiple lines deleted are now inserted immediately before the line with the cursor.

Copy multiple lines at one time

Copy

Paste

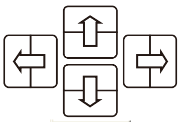
1 **Press f key [Copy] to select the copied range in the same way as line deletion**

2 **Put the cursor to the position right after the target place and press [Paste].**

≫The multiple lines deleted are now inserted immediately before the line with the cursor.

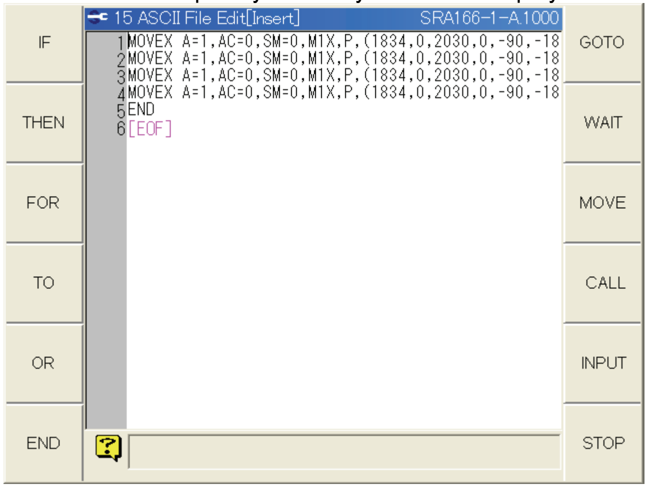
3.2.3 Inputting the basic instructions easily

If the AX control unit is not equipped with a keyboard, it will take some time to input commands. For this reason, provision has been made to enable just the frequently used commands such as `IF, THEN and ELSE` to be recorded easily.
A total of 24 key words can be recorded with a single action using the f keys.

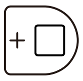


1 Move the cursor to the location where the key word is to be input.

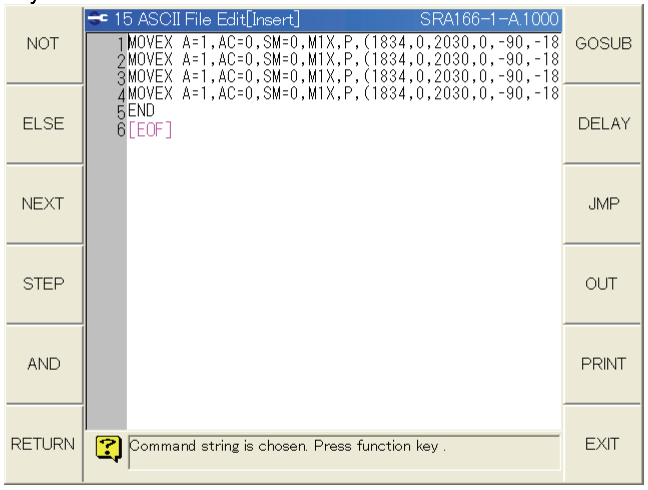
2 Press the f2 [Robot Command] key.
>>The first 12 frequently used key words are displayed.



```
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
END
[EOF]
```



3 Press the [ENABLE] key.
>>The next 12 frequently used key words are displayed. In this way, the display of 24 key words can be selected.



```
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,P,(1834,0,2030,0,-90,-18
END
[EOF]
```

Command string is chosen. Press function key .

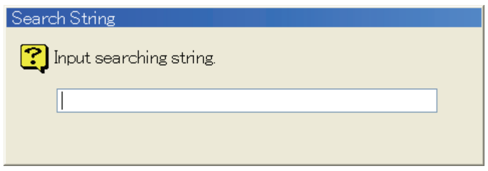
4 Press the desired f key (such as the f1 [IF] key).
>>The f keys return to their original editing screen statuses, and the key word is inserted at the cursor position.
>>The original display screen is restored by the [RESET/R] key.

3.2.4 Searching character strings

Any character string can be searched inside the area of the file being edited.

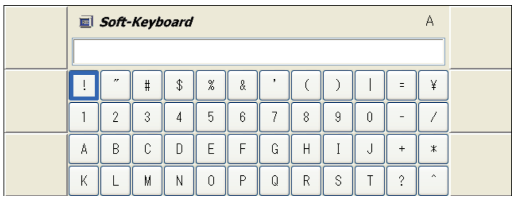
Find Function

1 **Press the f7 [Find Function] key.**
>>A dialog box, such as the one shown below, for inputting the character string to be searched now appears.


Search String
Input searching string.

+ +

2 **Input the character string to be searched.**
If the character string is a numerical value, input it directly using the number keys. To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the soft keyboard, and then input them.


Soft-Keyboard A

```
! ″ # $ % & ' ( ) | = ¥
1 2 3 4 5 6 7 8 9 0 - /
A B C D E F G H I J + *
K L M N O P Q R S T ? ^
```

If the f12 [Complete] key is pressed upon completion of the character input, the screen will go back to the dialog box for inputting the character string to be searched.

Find Function

3 **Once the character string has been entered, press [Enter] key.**
>>The search is commenced. The character string is searched from the beginning of the program, the cursor is positioned at the location where it has been found, and the display is updated. For instance, if the "`MOVEX`" character string is searched, the cursor will move to "`A`" at the head of the first character string which has been found.

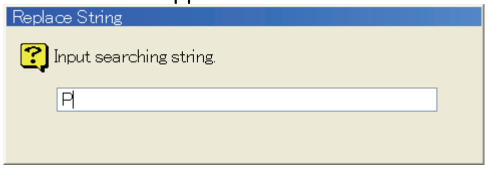
+ + Find Function

4 **To continue with the search, press the [ENABLE] and f7 [Find Function] keys.**
>>The cursor now moves to the next "`MOVEX`" character string which has been found. The display does not return from the end of the program to its beginning.

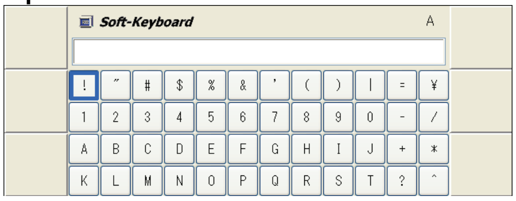
3.2.5 Replacing strings

Any character string can be replaced inside the area of the file being edited.

1 **Press the [ENABLE] + [Replace] key.**
>>A dialog box, such as the one shown below, for inputting the character string to be searched now appears.

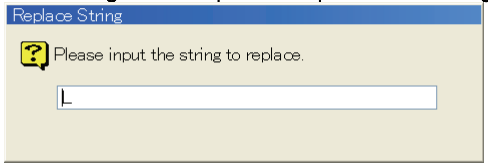


2 **Input the character string to be searched.**
If the character string is a numerical value, input it directly using the number keys. To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the software keyboard, and then input them.



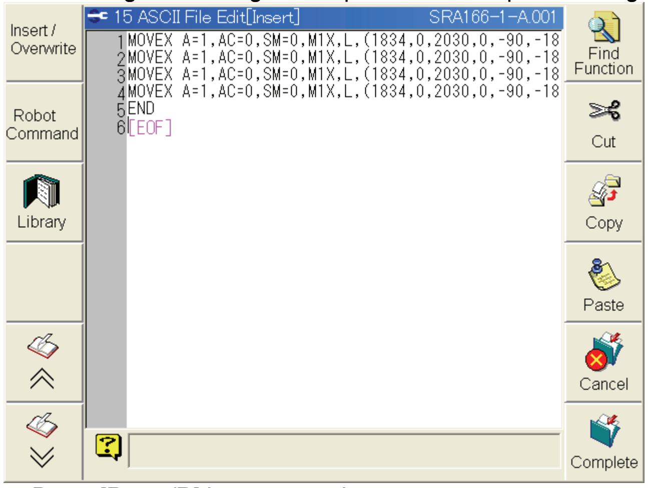
If the f12 [Complete] key is pressed upon completion of the character input, the screen will go back to the dialog box for inputting the character string to be searched.

3 **Once the character string has been entered, press [Enter] key.**
>>A dialog box to input the replacement strings will be displayed.



4 **Input the character string for replacement.**
If the character string is a numerical value, input it directly using the number keys. To input characters such as letters using keys other than number keys, press the [ENABLE] + [EDIT] keys to display the software keyboard, and then input them.
>>Input the string in the same way.

5 **Once the character string has been entered, press [Enter] key.**
>>The designated strings are replaced to the inputted strings.



```
MOVEX A=1,AC=0,SM=0,M1X,L,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,L,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,L,(1834,0,2030,0,-90,-18
MOVEX A=1,AC=0,SM=0,M1X,L,(1834,0,2030,0,-90,-18
END
[EOF]
```

>>Press [Reset/R] key to cancel.

3.3 Creating pose files

3.3.1 Outline of pose files

A "Pose file" is a data file that consists of "Pose variables"(positions and postures).

In robot language, it is possible to write the robot positions or postures in values directly in a `MOVEX` command. However, it is difficult to move the robot accurately following the design because of the installation position error etc. Furthermore, when the position modifications becomes necessary because of the layout change, or, when the same teaching point is used in many steps in one program, the modification work for the all move commands in the original source program would take long time.

To solve this problem, in robot language, it is possible to handle the plural pose variables altogether in one data file. This is a "Pose file". The respective pose variables can be made or modified by moving the actual robot and recording the actual position. And, it is possible to move the robot by selecting the pose file number and calling the pose variable using the `MOVEX` command.

(Example)

```
REM "Selecting a pose file No.10"
USE 10
REM "Call the pose number 1,2, and 3 to move the robot"
MOVEX A=1,M1J,P,P1,R=10.0,H=1,MS
MOVEX A=1,M1J,P,P2,R=10.0,H=1,MS
MOVEX A=1,M1J,P,P3,R=10.0,H=1,MS
```

For example, when the robot type is `SRA166-01`, the pose file is stored in the internal memory under filenames such as the following.
(See the item of file name in the section of "1.1.2 Characteristics and precautions")

`"SRA166-1-P.nnn"` (nnn is the pose file number)

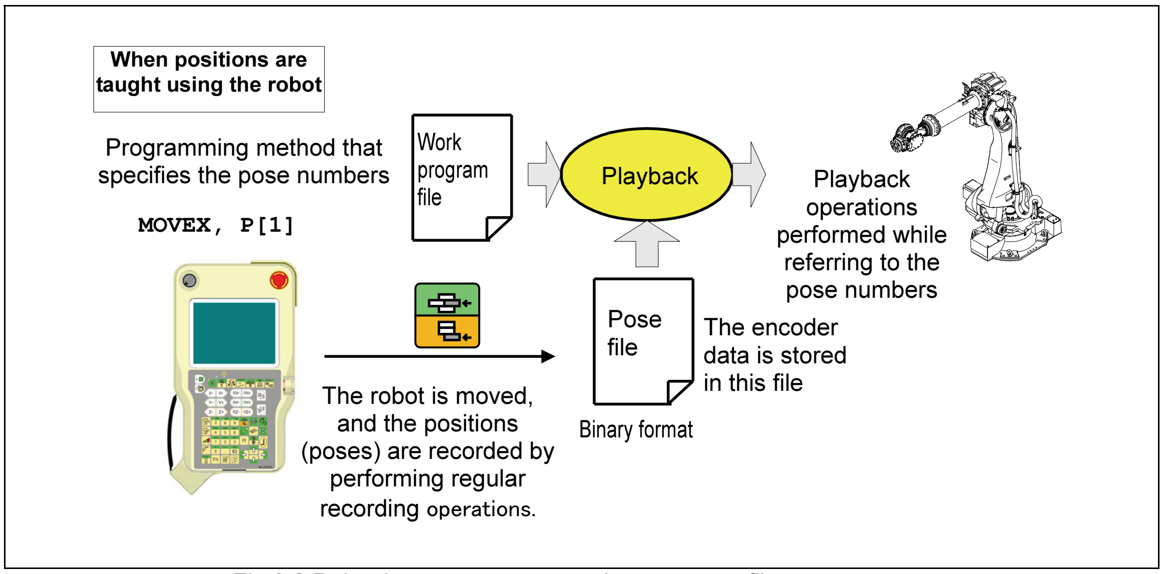

When positions are taught using the robot
Programming method that specifies the pose numbers
MOVEX, P[1]
Work program file
Playback
Playback operations performed while referring to the pose numbers
Pose file
Binary format
The encoder data is stored in this file
The robot is moved, and the positions (poses) are recorded by performing regular recording operations.

Fig 3.2 Robot language programs that use pose files

The pose files are not referenced at the compiling stage. They are referenced during playback operations. There is no need for there to be one pose file for each robot language program and vice versa. Multiple pose files can be prepared to support different types of work and then a playback operation can be performed selecting the desired pose file by the "`USE`" command.

3.3.2 Recording pose files

The operation method used for pose files is in no way different from the one used for regular teaching. All that needs to be done is to "make a pose recording declaration" before starting the recording operation. The programs which are actually created are the same as the execute form programs created by regular teaching.

During a recording operation which has been declared to be a pose recording, it is not possible to record any instructions other than those involved with poses (robot positions), that is to say, any application instructions.

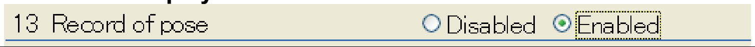
Entering the pose recording mode

1 **Establish the teach mode.**

Teach/Play Condition

2 **Select [1 Teach/Playback Condition] from Service Utilities.**
Alternatively, press the f7 [Teach/Playback Conditions] key.

3 **Align the cursor with [13 Record of pose] on the list of teach/playback conditions displayed.**


13 Record of pose ○ Disabled [✓ Enabled]

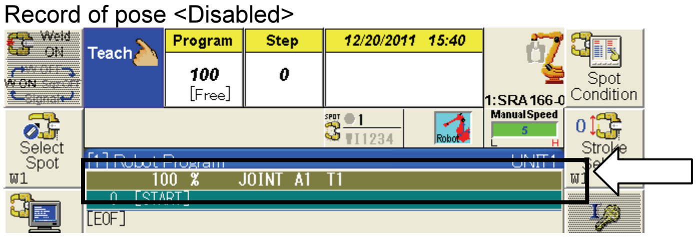
4 **Use the [ENABLE] and [Left] or [Right] cursor key to set pose recording to [Enable].**

Complete

5 **Press the f12 [Complete] key.**
>>In the course of the subsequent recording, only pose files will be recorded.
Regular task programs cannot be taught.
(This state is retained even when the main power of the controller is turned off.)

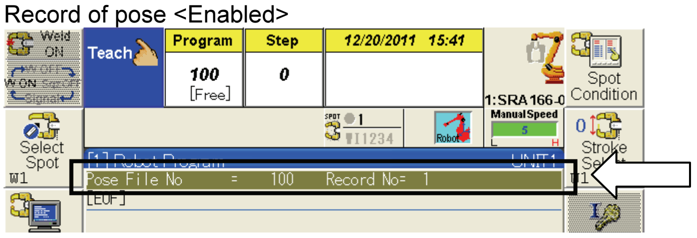
6 **When the display returns to the mode screen, the recording status display area (line below the blue title bar line) on the program monitor screen serves as the display field for the pose file number and recording number. Here, it is possible to determine whether pose recording is enabled or disabled.**

Record of pose <Disabled>


Teach | Program 100 [Free] | Step 0 | 12/20/2011 15:40 | 1:SRA166-0

[1] Robot Program UNIT1
```
    100 %     JOINT  A1  T1
  0  [START]
[EOF]
```

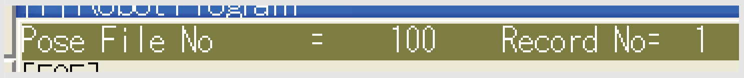
Record of pose <Enabled>


Teach | Program 100 [Free] | Step 0 | 12/20/2011 15:41 | 1:SRA166-0

[1] Robot Program UNIT1
```
Pose File No      =     100     Record No=  1
[EOF]
```

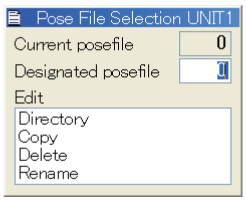
POINT

The "Program" and the "Step" displayed in the status window at the top of the screen (No.100 in the example shown in the figure) has nothing to do with the operation to select the pose files. The current pose file and the current pose number can be confirmed in the line shown as below.


```
[1] Robot Program
Pose File No      =     100     Record No=  1
[EOF]
```

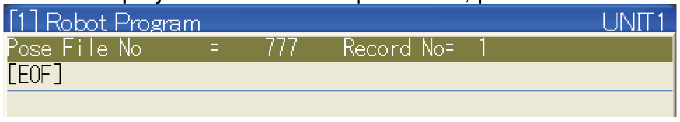
Recording a pose in the pose file (pose recording mode)

1 **Press [ENABLE] and [PROGRAM/STEP] key at the same time.**
≫ Following [pose fie selection] screen will appear.



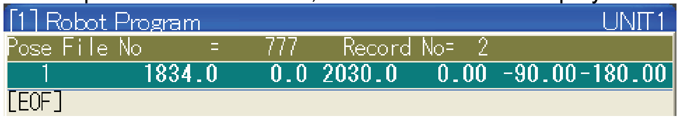
```
Pose File Selection UNIT1
Current posefile        0
Designated posefile     0
Edit
 Directory
 Copy
 Delete
 Rename
```

2 **Put the cursor to ”Designated pose file” and input the pose fie number and press [Enter].**
≫The number of the selected pose file appears in the pose file number and recording number display field. In the example shown, pose file No.777 has been selected.



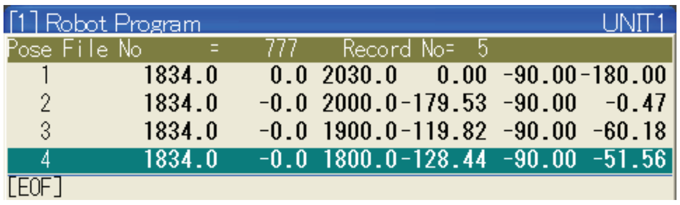
```
[1] Robot Program                                   UNIT1
Pose File No    =    777    Record No=  1
[EOF]
```

3 **Turn on the motor power, move the robot using the axis operation keys in the same way as for regular recording operations, and press the [REC] button.**
>>One pose will be recorded, and its data will be displayed as shown below.



```
[1] Robot Program                                   UNIT1
Pose File No    =    777    Record No=  2
   1         1834.0    0.0  2030.0    0.00  -90.00-180.00
[EOF]
```

4 **Every time the [RECORD] key is pressed, new poses will be recorded one after another.**



```
[1] Robot Program                                   UNIT1
Pose File No    =    777    Record No=  5
   1         1834.0    0.0  2030.0    0.00  -90.00-180.00
   2         1834.0   -0.0  2000.0-179.53  -90.00   -0.47
   3         1834.0   -0.0  1900.0-119.82  -90.00  -60.18
   4         1834.0   -0.0  1800.0-128.44  -90.00  -51.56
[EOF]
```

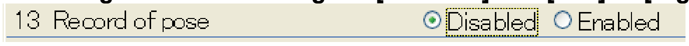
5 **The positions can be checked by selecting the step as with regular tasks and then using CHECK GO/BACK.**

6 **After recording the last pose, just proceed to the steps described in the next section "Ending the pose recording declaration."**
There is no need to record the END or other instructions. (In the pose recording status, it is not possible to record any application instructions.)

Exiting the pose recording mode

Teach/Play Condition

1 **Select [Teach/Playback Condition] and [13 Record of pose], and return pose recording to "disable" using the [ENABLE] and [Left] or [Right] cursor key.**



13 Record of pose [✓ Disabled] Enabled

Complete

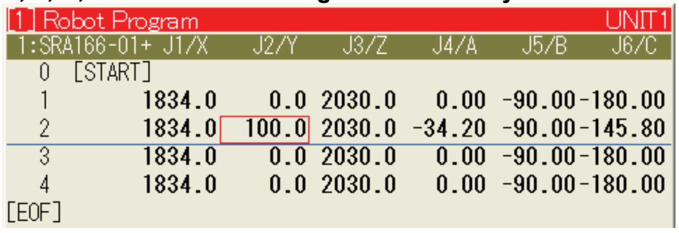
2 **Press the f12 [Complete] key.**
>>From this point on, regular task programs can be taught. Pose files cannot be recorded. This state is retained even when the main power of the controller is turned off.

3.3.3 Modifying pose files

1 **Interim poses can be deleted using [ENABLE] + [DEL].**
However, even when an interim pose is deleted, the subsequent poses will not be adjusted forward. A deleted pose number will become a missing number.

2 **Pose positions can be modified using [ENABLE] + [MOD Position].**

3 **Modifications can be made using manual input on an axis by axis basis by screen editing.**
Press the [EDIT] key, open the screen editing screen, and input directly the X, Y, Z, R, P and Y values using the number keys.



```
[1] Robot Program                                  UNIT1
1:SRA166-01+ J1/X    J2/Y    J3/Z    J4/A   J5/B   J6/C
   0  [START]
   1        1834.0     0.0  2030.0    0.00  -90.00-180.00
   2        1834.0   100.0  2030.0  -34.20  -90.00-145.80
   3        1834.0     0.0  2030.0    0.00  -90.00-180.00
   4        1834.0     0.0  2030.0    0.00  -90.00-180.00
[EOF]
```

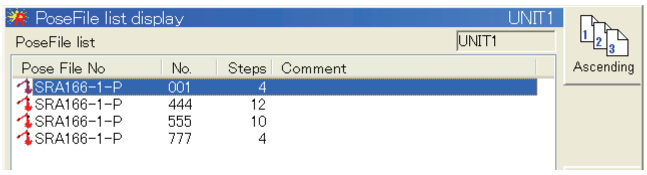
3.3.4 Displaying a list of pose files

A list of only pose files can be displayed.

This is useful for checking the numbers of empty files. It is also possible to select an already recorded pose file from the list.

1 **Set pose recording to "Enable."**

2 **Input the R17 program number list shortcut.**
>>A list of only pose files such as the one shown below appears.



PoseFile list display UNIT1

PoseFile list UNIT1

Pose File No	No.	Steps	Comment
SRA166-1-P	001	4	
SRA166-1-P	444	12	
SRA166-1-P	555	10	
SRA166-1-P	777	4	

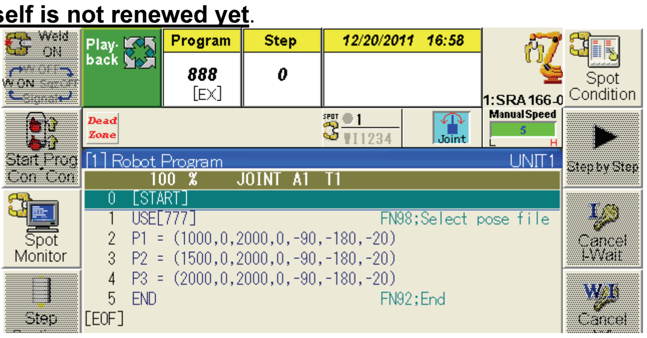
3 **Move the cursor to the designated pose file and press [Enter].**
≫ Pose file selected is displayed. If pressing [Reset] instead of [Enter], then back to the previous teach screen.

3.3.5 Renewing of pose file

This section describes the robot behavior or the pose file access operation when pose calculation or pose assignment command was executed in the pose recording mode (Refer to "3.3.2 Recording pose files").

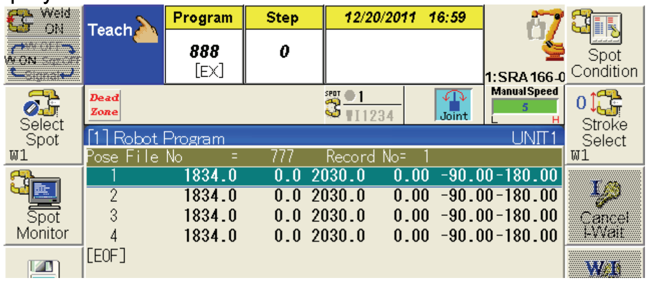
For example, consider about executing "robot program 888" which includes pose assignment command. As the result, "pose 1 to 3" of "pose file 777" **in the work memory** is changed. Next time when executing "pose file 777", robot moves to the new "pose 1 to 3" which pose assignment is completed.

But in this moment, this new "pose 1 to 3" is just stored in the work memory, **so "pose file 777" itself is not renewed yet**.



```
[1] Robot Program                                    UNIT1
        100 %       JOINT  A1   T1
  0  [START]
  1  USE[777]                          FN98;Select pose file
  2  P1 = (1000,0,2000,0,-90,-180,-20)
  3  P2 = (1500,0,2000,0,-90,-180,-20)
  4  P3 = (2000,0,2000,0,-90,-180,-20)
  5  END                               FN92;End
[EOF]
```

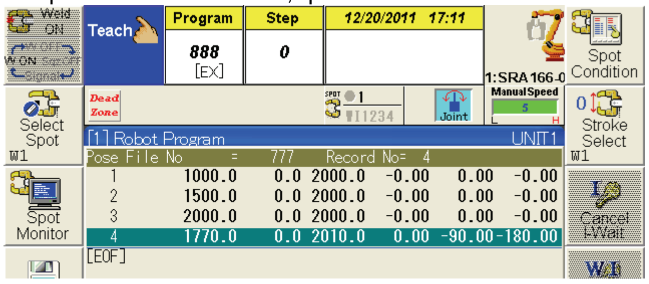
Switch to the teach mode after executing "robot program 888", then "pose file 777" is already displayed in the monitor screen.



```
[1] Robot Program                                    UNIT1
Pose File No      =    777     Record No=   1
  1         1834.0     0.0  2030.0     0.00  -90.00-180.00
  2         1834.0     0.0  2030.0     0.00  -90.00-180.00
  3         1834.0     0.0  2030.0     0.00  -90.00-180.00
  4         1834.0     0.0  2030.0     0.00  -90.00-180.00
[EOF]
```

POINT The result of executing "robot program 888" is just affected to the internal work memory. The displayed pose data on the screen is just the content of the pose file so the data does not show the content in the internal work memory.

However, when pressing [RECORD] key after selecting pose 4, the current robot position will be recorded in "pose 4" and "pose 1 to 3" is renewed to the new data that pose assignment command is completed. At this moment, "pose file 777" is overwritten.



```
[1] Robot Program                                    UNIT1
Pose File No      =    777     Record No=   4
  1         1000.0     0.0  2000.0    -0.00    0.00   -0.00
  2         1500.0     0.0  2000.0    -0.00    0.00   -0.00
  3         2000.0     0.0  2000.0    -0.00    0.00   -0.00
  4         1770.0     0.0  2010.0     0.00  -90.00-180.00
[EOF]
```

POINT Beware that all pose (1 to 4) is overwritten at the same time although only "pose 4" is selected. Because the pose 1- 3 calculated during the playback operation will also be overwritten.

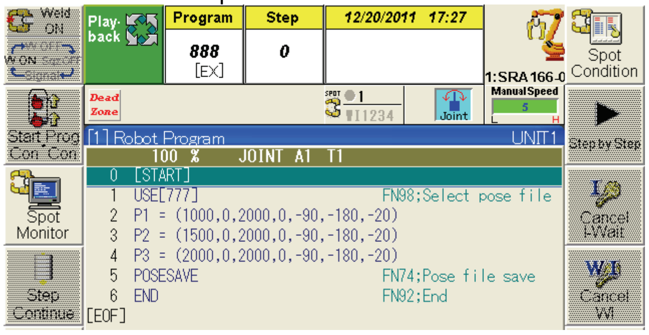
IMPORTANT As described above, when executing robot program while in the pose recording mode, monitor display and real robot position may not match. So while in the pose recording mode, robot program had not better been executed to avoid miss-operation.

3.3.6 Command to save pose file

This command is to save the result of pose calculation or pose assignment onto the pose file that is designated by `USE` command.

Normally the result of pose calculation or pose assignment is not saved onto the pose file (☞ Refer to "3.3.5 Renewing of pose file"), so FN74 `POSESAVE` (Pose File Save) is necessary to save these result to pose file.

Following example shows that “robot program 888” includes the pose save command (FN74 `POSESAVE`) and the pose assignment. As the result of playback “robot program 888”, “pose 1 to 3” data is saved onto “pose file 777” and affected the monitor display data in Teach mode.



```
[1] Robot Program                                UNIT1
        100 %      JOINT  A1   T1
   0  [START]
   1  USE[777]                       FN98;Select pose file
   2  P1 = (1000,0,2000,0,-90,-180,-20)
   3  P2 = (1500,0,2000,0,-90,-180,-20)
   4  P3 = (2000,0,2000,0,-90,-180,-20)
   5  POSESAVE                       FN74;Pose file save
   6  END                            FN92;End
[EOF]
```

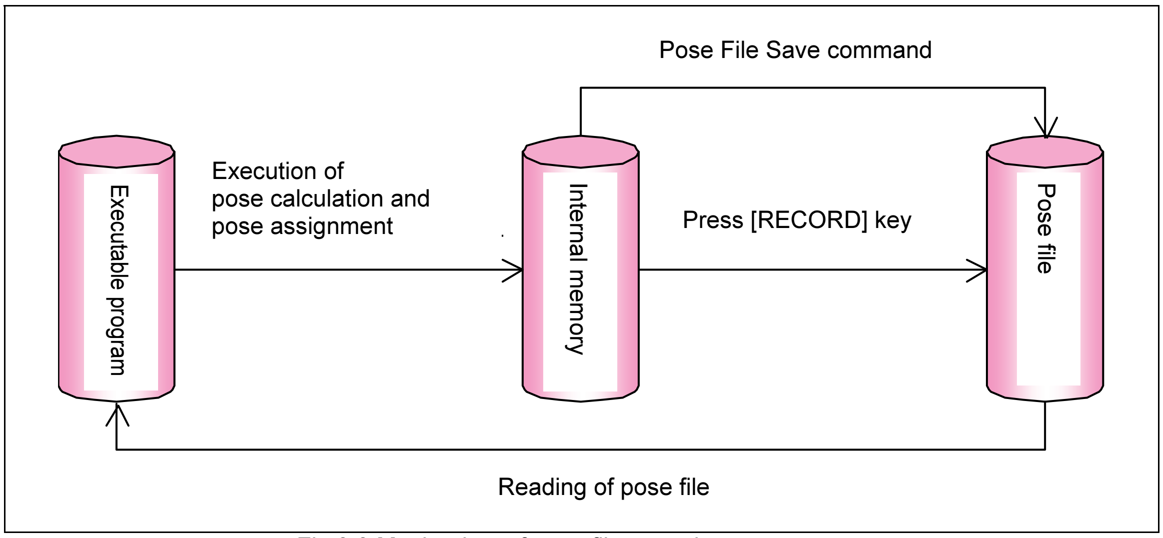


Fig 3.3 Mechanism of pose file renewing

Chapter 4 Compiling programs

This chapter describes how to compile robot language programs. Once compiled, the programs are checked for syntax errors, etc., and they are converted into an executable format to enable them to be played back.

4.1 Compiling

The robot language programs (ASCII files) which have been created cannot be executed unless they are first compiled. The objective of compiling is not only to convert the files into a format which can be executed by the robot but also to find syntax errors. Compiling is accomplished by the following the steps below.

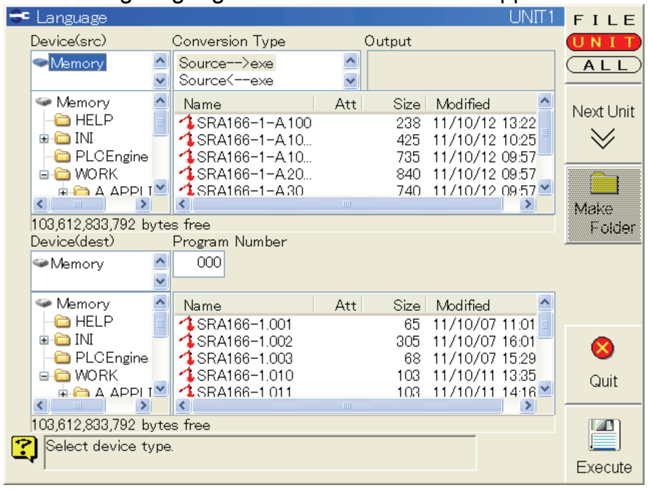
Service Utilities

1 Press [Service Utilities] f key.

≫ Service menu list will appear.

2 Among the service menu list, select [9 Program conversion] -> [8 Language] and press [Enter].

>>Following language conversion screen will appear.



Language UNIT1

Device(src)	Conversion Type	Output
Memory	Source-->exe Source<--exe	

Memory
- HELP
- INI
- PLCEngine
- WORK
- A_APPLI

Name	Att	Size	Modified
SRA166-1-A.100		238	11/10/12 13:22
SRA166-1-A.10...		425	11/10/12 10:25
SRA166-1-A.10...		735	11/10/12 09:57
SRA166-1-A.20...		840	11/10/12 09:57
SRA166-1-A.30		740	11/10/12 09:57

103,612,833,792 bytes free

Device(dest)	Program Number
Memory	000

Memory
- HELP
- INI
- PLCEngine
- WORK
- A_APPLI

Name	Att	Size	Modified
SRA166-1.001		65	11/10/07 11:01
SRA166-1.002		305	11/10/07 16:01
SRA166-1.003		68	11/10/07 15:29
SRA166-1.010		103	11/10/11 13:35
SRA166-1.011		103	11/10/11 14:16

103,612,833,792 bytes free

Select device type.

3 Select "Source → exe" as the conversion type.

4 The output type box is an option selected for reverse compiling ("Source ← exe") and, as such, it is not specified to compile programs.

FILE UNIT ALL

5 Press f7<FILE> to switch the file view so that your desired file will appear.

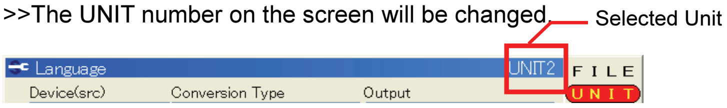
* To switch the file view, the operator qualification of ***EXPERT*** or higher is required.
* This is available only when compiling.

FILE UNIT ALL	The task programs in the selected unit are displayed.
FILE UNIT ALL	All the task programs are displayed. This even enables to display task programs in the unit which is not registered in the current robot controller.

Next Unit

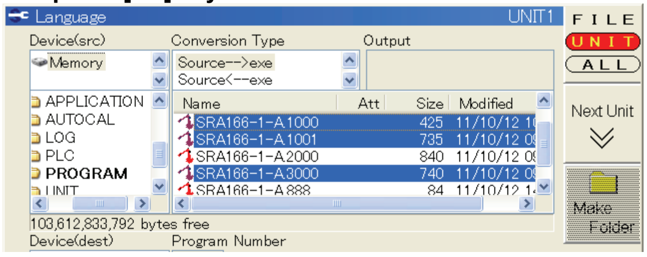
6 Press f8<Next Unit> to switch the UNIT No.
When the selected file view is "UNIT", the file list will be updated.
The UNIT selected here is to be used as the target UNIT in conversion operation.

>>The UNIT number on the screen will be changed.



The unit switched at this point is intended for the manual operation

7 **Specify the file to be compiled. To cancel the selection, select the target file and press [BS] key.**


Language UNIT1

Device(src) Conversion Type Output

Memory | Source-->exe | Source<--exe

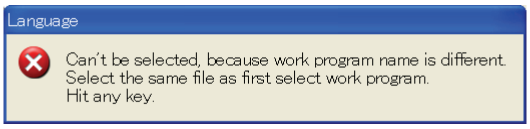
APPLICATION, AUTOCAL, LOG, PLC, PROGRAM, UNIT

Name	Att	Size	Modified
SRA166-1-A1000		425	11/10/12 1
SRA166-1-A1001		735	11/10/12 0
SRA166-1-A2000		840	11/10/12 0
SRA166-1-A3000		740	11/10/12 0
SRA166-1-A888		84	11/10/12 1

103,612,833,792 bytes free

Device(dest) Program Number

However, when the setting is "ALL" and the files like ones shown as below are selected at the same time, a warning message will appear , failing to select those files.

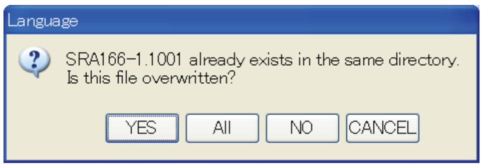

Language

Can't be selected, because work program name is different.
Select the same file as first select work program.
Hit any key.

8 **Next, in the same way, specify the device, folder and file which serve as the compiling destination.**
At the compiling stage, it is possible to change the numbers of the programs. To do so, align the cursor with the program number field of the device (destination), and input the program number using the number keys.
However, it is not possible to change the program number when two or more files have been selected. In this case, the execution file will be created by a program number of the selected file.

Execute

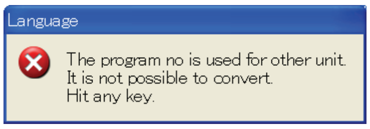
9 **Press the f12 [Execute] key.**
>>Compiling now starts for the selected file.
When a program number already existing in the memory has been selected, a pop-up message such as the one below appears. Select [YES] or [NO] using the [Left] or [Right] cursor key, and press the [Enter] key.


Language

SRA166-1.1001 already exists in the same directory.
Is this file overwritten?

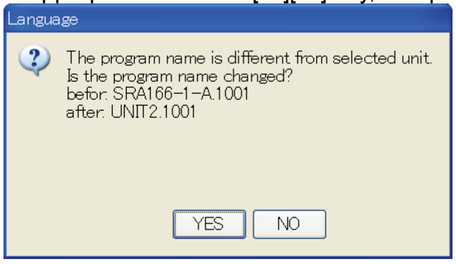
YES	Overwrites.
All	Overwrites all files if already existing. This is displayed when selecting two or more files, while not displayed once after selected.
NO	Does not overwrite, but converts the next file.
CANCEL	Cancels conversion.

When the program number after conversion has been already used in the program in a different unit, the following pop-up message appears, which fails to execute the conversion. In this case, delete the file in advance or change the program number in advance to convert it.


Language

The program no is used for other unit.
It is not possible to convert.
Hit any key.

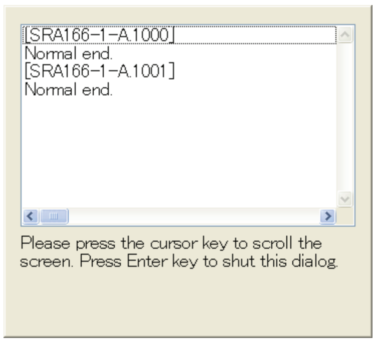
The following pop-up message appears if executing a file of which name differs from the task program in the currently selected unit. In this case, select the appropriate one with [←][→] key, and press [Enter].


Language

The program name is different from selected unit.
Is the program name changed?
befor: SRA166-1-A.1001
after: UNIT2.1001

YES	Changes a name of the created file.
All	Changes a name of all the created files. This is displayed when selecting two or more files, while not displayed once after selected.
CANCEL	Cancels conversion.

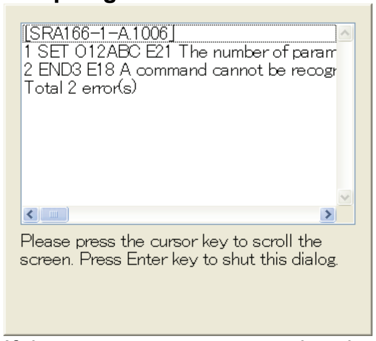
10 **When the compiling has been completed successfully, the "Successfully completed" message appears at the screen center.**
An execute form file which can be played back has now been created.


```
[SRA166-1-A.1000]
Normal end.
[SRA166-1-A.1001]
Normal end.
```
Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

[Displayed item]

[SRA166-1-A.1000]	→Conversion File Name
Normal end.	→Result

11 **If errors have been detected during compiling, all the parts where the compiling errors were detected are displayed at the screen center.**


```
[SRA166-1-A.1006]
1 SET 012ABC E21 The number of param
2 END3 E18 A command cannot be recogr
Total 2 error(s)
```
Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

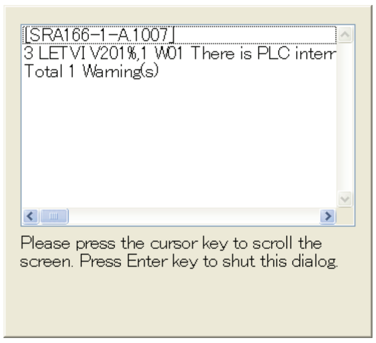
If there are so many errors that they cannot be displayed on one screen, the error displays can be scrolled using the [Up] or [Down] cursor key.

[Example of a compiling error]

[SRA166-1-A.1006]	→Conversion File Name
2 END3	→This indicates the number of the line where the error was found and a description of the error.
E18 A command Cannot be recognized	→This indicates what kind of error has been found. (☞ Refer to Table 4.1.1 Robot language compiling errors)

12 **When an error has been detected, correct the place indicated by editing the ASCII file, and proceed with compiling again.**
An execute form file is not created until the compiling is successfully completed.

13 **If warnings have been detected during compiling, all the parts where the compiling warnings were detected are displayed at the screen center.**



```
[SRA166-1-A.1007]
3 LETVI V201%,1 W01 There is PLC intern
Total 1 Warning(s)
```

Please press the cursor key to scroll the screen. Press Enter key to shut this dialog.

[Example of a compiling warning]

[SRA166-1-A.1007]	→Conversion File Name
3 LETVI V201%,1	→ The content and number of the line where warnings were detected.
W01 There is PLC internal variable access.	→ Displays the content of warning. (Refer to "Table 4.1.2 Robot language compiling warnings".)

14 **An execute form file will be created when only a warning has been detected. Check the indicated part to see whether the created file can be used or not.**

Table 4.1.1 Robot language compiling errors

Error description	Explanation
E01 Too many characters in one line.	Use up to 254 characters per line.
E02 Illegal line number.	Check that the line number or label name is correct.
E03 Syntax error.	Check that there are no syntax errors.
E04 Numerical value outside prescribed range.	The parameter of the application instruction is outside the prescribed range. Alternatively, the FLOAT or INT variable is outside the input range.
E05 Error in description of calculation expression.	There is an error in an assignment expression. • When there is an error in the argument of a general-purpose function • When the right side is missing in an assignment expression (V1%=) • When the expression ends with an operator (V1%=1+)
E06 Too many characters in an identifier.	The maximum number of characters that can be used for a label is 16.
E07 Illegal label.	Check for symbols that cannot be used for labels.
E08 Too many labels.	Reduce the number of labels or divide up the program.
E09 Same label has been defined twice.	Check for a label with the same name.
E10 Program size has been exceeded.	Divide up the program using program call, etc.
E11 Illegal type of operation expression.	Check whether any numerical values have been assigned to character strings.
E12 Missing "(".	Check that "(" and ")" are paired up.
E13 Missing ")".	Check that "(" and ")" are paired up.
E14 Missing "[".	Check that "[" and "]" are paired up.
E15 Missing "]".	Check that "[" and "]" are paired up.
E16 Missing "$".	A character string type is required.
E17 Illegal register.	Check the register numbers.
E18 Cannot recognize instruction.	Check for incorrect command names and function names.
E19 Illegal step number.	Check whether the specified step number exists. Step numbers range from 0 to 999.
E20 Illegal program number.	Program numbers range from 0 to 9999.
E21 Number of parameters do not match.	Check the functions or commands.
E22 No THEN.	Check THEN.
E23 Must be ELSE.	Check for instructions in the IF statements.
E24 Excessively long program on one line.	The instruction on one line is too long due to the functions and parentheses used.
E25 No GOTO.	No GOTO corresponding to ON instruction.
E26 Error in interpolation description.	Check the interpolation parameter settings.

Error description	Explanation
E27 No comma.	Check whether there are not enough multiple parameters.
E28 Missing "=".	There is no part to which the initial value is to be assigned in the FOR statement.
E29 Error in accuracy description.	Check the accuracy number.
E30 Error in speed description.	Check whether the maximum speed is exceeded.
E31 Error in tool description.	Check for errors in the tool numbers.
E32 Too many MOVE instructions (999).	The maximum number of MOVE instructions is 999.
E33 No LET statement in FOR.	There is no part to which the initial value is to be assigned in the FOR statement.
E34 Too many nesting levels for FOR.	Up to 4 nesting levels may be used for FOR.
E35 Number of NEXT's do not match.	Check the number of FOR's and NEXT's.
E36 Illegal input/output device number.	
E37 No TO.	Check that there is a TO in the FOR statement.
E38 Must be STEP.	Check whether a command other than STEP, that indicates the increment, has been described in the FOR statement.
E40 Illegal line number.	Some lines have a line number and some do not.
E41 Instruction which cannot be used in FOR is present.	In the loop between FOR and NEXT, there is a command that jumps out of the loop.
E42 Error in conveyor description.	Check the conveyor register.
E43 Error in CONF description.	Check what is specified for the configuration.
E44 Cannot open temporary file.	A temporarily file could not be opened during conversion.
E45 This speed specification cannot be reverse converted.	An attempt has been made to reverse convert a speed specification existing solely in this controller into the AW format.
E46 This interpolation format cannot be reverse converted.	An attempt has been made to reverse convert an interpolation format existing solely in this controller into the AW format.
E47 Cannot open ASCII file.	The ASCII file to be converted could not be opened.
E48 Cannot open robot program.	The robot program to be converted could not be opened.
E49 Conversion task was forcibly terminated.	The forced termination key was pressed.
E51 Error in acceleration description.	Check what has been specified for the acceleration.
E52 Error in smoothness description.	Check what has been specified for the smoothness.
E53 The label is not found.	
E54 Error is in description of the mechanism.	
E55 Pose calculation failed.	Check the value of pose constant
E56 The error occurred while saving.	
E58 Other compilation processing is executed.	Try to execute after the other ongoing compiling processes are finished.
E59 This interpolation specification is not convertible.	
E60 The translation table overflowed.	The size of conversion table of the INCLUDE command is too large.
E61 A file does no open.	Failed to open the conversion table.
E62 MOVE and MOVEJ are uncorrespondence to seven axis robot.	
E63 ENDW does not match.	Number of WHILE and ENDW does not match. Check them.
E64 Too many nesting levels for WHILE to ENDW.	Up to 4 nesting levels may be used for WHILE to ENDW.
E65 Inadequate usage of flow control statements.	Inadequate command is included.
E66 ENDIF does not match.	Number of IF and ENDIF does not match. Check them.
E67 Too many nesting levels for IF to ENDIF.	Up to 4 nesting levels may be used for IF to ENDIF.
E69 Too many nesting levels for SWITCH to ENDS.	Up to 4 nesting levels may be used for SWITCH to ENDS.
E70 Inadequate usage of SWITCH to ENDS.	This is not allowed to use right after SWITCH command.

Table 4.1.2 Robot language compiling warnings

Warning description	Explanation
W01 There is PLC internal variable access.	The integer and actual variables for the PLC internal variable access are being used. Check whether there are any problems for usage.

4.2 Reverse compiling

The term "reverse compiling" refers to converting an execute form program into a robot language program (ASCII file).

Since plural form are permitted for the movement instructions in robot language programs, so this from must be specified to perform reverse compiling. Except this, all operation is same as compiling.

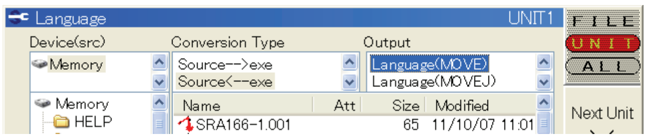
Note that user task programs cannot be reverse compiled.

Automatic indent is not performed when reverse compiling.

Only those operations that differ from compiling are described below.

1 **Select "Source → exe" as the conversion type.**

2 **When the cursor is moved to the output format box, the selection options shown below appear.**
Use the [Up] or [Down] cursor key to align the cursor with the desired movement instruction format, and then press the [Enter] key.



Device(src): Memory
Conversion Type: Source-->exe / Source<--exe
Output: Language(MOVE) / Language(MOVEJ)

Memory
HELP

Name	Att	Size	Modified
SRA166-1.001		65	11/10/07 11:01

Table 4.2.1 Form of move command

Output Form	Position expression
`MOVEX-X`	TCP coordinate (X, Y, Z, r, p, y)
`MOVEX-J`	Angle of each axis (J1, J2, …, J6)
`MOVEX-E`	Axis encoders (E1, E2, …, E6)

For further details, refer to "2.4.3 Pose constant" or the instruction manual "Command Reference".

POINT

Notes for the move command form :

When executing the reverse compiling of executable program that was originally created by robot language, its move command form is the original form that was used in the original robot language, irrelevant to the output form (MOVE/MOVEJ/MOVEX and or so) designated in this screen. Because executable program contains the original move command of robot language in its internal information.

Example :
Original robot language program: was “MOVEX-X”.
 -> Compiling to create the executable program
 -> Reverse compiling to create the robot language program while designating output form as “MOVE-J” in this screen.
 -> Created robot language program is “MOVE-X” irrelevant to the designated output form.

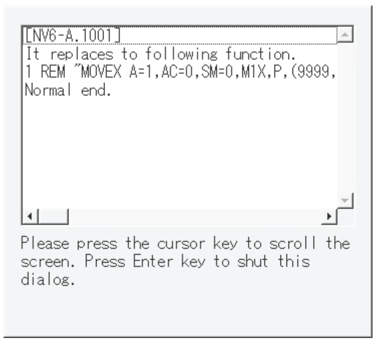
4.3 Force execute language conversion

Even if a particular error (Refer to “Table 4.3.1 Errors of Force conversion”) occurs in compiling, the compiling process can be forcibly continued. The step that causes an error can be replaced with Comment (REM command). Note that this is available only when compiling ([Language format] → [Execution format]).
The following describes the operation different from compiling process only.

1 **Confirm that "Force execute language conversion" setting is ”ON”.**
*** For details, see** Refer to “4.4Force execute of language conversion“

2 **Select a desired file to compile, and press f12<Execute>.**

3 **If the language conversion is forcibly executed in compiling, the message will tell you in the center of screen. However when the other type of errors occur, the file of execution format is not to be created.**



```
[NV6-A.1001]
It replaces to following function.
1 REM "MOVEX A=1,AC=0,SM=0,M1X,P,(9999,
Normal end.
```

```
Please press the cursor key to scroll the
screen. Press Enter key to shut this
dialog.
```

[Display example in force conversion]	
[SRA166-1-A.1006]	→**Conversion File Name**
It replaces to following function.	→Message in force conversion
1 REM “MOVEX A=1,AC=0,SM=0,･･･	→Forcibly converted line number and command after conversion
Normal end.	→Result

Table 4.3.1 Errors of Force conversion

Error description	Explanation
E55 Pose calculation failed.	The value of pose constant is not correct.

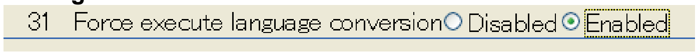
4.4 Force execute of language conversion

This concerns the setting whether to execute the language conversion forcibly when a particular error occurs. (Refer to "Table 4.3.1 Errors of Force conversion") Follow the procedures below.

* To change this setting, the operator qualification of ***EXPERT*** or higher is required.

1 **Press <Constant Setting>.**
>>The setting menu screen will appear.

2 **Select [5 Operation Constants] - [1 Operation condition] in <Constant Setting> menu.**


31 Force execute language conversion Disabled [✓ Enabled]

3 **Align the cursor with "Force execute language conversion", and press [→] with holding [ENABLE] to switch to "Enabled".**

[Setting value]

Disabled	→ Forced conversion is not executed.
Enabled	→ Forced conversion is executed.

4 **Press f12<Complete>.**
>>The setting will be saved.

Chapter 5 Command

"Commands" are the statements such as MOVE to operate (move) robot and SETM to turn ON/OFF the output signals.

Commands of executable program are classified into move commands recorded by using [RECORD] key and application commands recorded by using [FN] key, but in robot language programs all of these are treated as "commands".

Each detail is not described here.

Please refer to the online help of this controller or the instruction manual "Command Reference" for the detail of them. And, please be sure that the contents of these two are the same basically.

NACHI
NACHI-FUJIKOSHI CORP.

http://www.nachi-fujikoshi.co.jp/

JAPAN MAIN OFFICE	Phone: +81-3-5568-5245	Fax: +81-3-5568-5236	Shiodome Sumitomo Bldg. 17F, 1-9-2 Higashi-Shinbashi Minato-ku, TOKYO, 105-0021 JAPAN

NACHI NORTH AMERICA

http://www.nachirobotics.com/

North America Headquarters	Phone: 248-305-6545	Fax: 248-305-6542	22285 Roethel Drive, Novi, Michigan 48375 U.S.A.
Greenville Service Office	Use 248-305-6545	Use 248-305-6542	South Carolina, U.S.A.
San Antonio Service Office	Use 248-305-6545	Use 248-305-6542	Texas, U.S.A.
Kentucky Branch Office	Phone: 502-695-4816	Fax: 502-695-4818	116 Collision Center Drive, Suite A, Frankfort, KY 40601 U.S.A
Training Office	Phone: 248-334-8250	Fax: 248-334-8270	22213 Roethel Drive, Novi, Michigan 48375 U.S.A.
Toronto Branch Office	Phone: 905-760-9542	Fax: 905-760-9477	89 Courtland Avenue, Unit 2, Vaughan, Ontario L4K3T4 CANADA
Mexico Branch Office	Phone : +52-555312-6556	Fax: +52-55-5312-7248	Urbina # 54, Parque Industrial Naucalpan, Naucalpan de Juarez, 53370, Estado de México, MEXICO
Saltillo Service Office	Phone : +52-844416-8053	Fax: +52-844416-8053	Canada 544 Privada Luxemburgo C. P. 25230, Saltillo, Coahuila, MEXICO

NACHI ROBOTIC EUROPE

Germany

http://www.nachi.de/

Nachi Europe GmbH	Phone: +49-(0)2151-65046-0	Fax: +49-(0)2151-65046-90	Bischofstrasse 99, 47809, Krefeld,GERMANY

United Kingdom

http://www.nachi.co.uk/

Nachi U.K. LTD.	Phone: +44-(0)121-250-1895	Fax: +44-(0)121-250-1899	Unit 7, Junction Six Industrial Estate, Electric Avenue, Birmingham B6 7JJ, U.K.

Czech Republic

Nachi Europe	Phone / Fax : +420-321-710-200	Mostni 73, Kolin 4, 28002 CZECH

NACHI ROBOTIC ASIA

Korea

http://www.nachi-korea.co.kr/

Korea	Phone: +82-(0)2-469-2254	Fax: +82-(0)2-469-2264	2F Dongsan Bldg. 276-4, Sungsu 2GA-3DONG, Sungdong-ku, Seoul 133-123, KOREA

Robot Division
1-1-1, FUJIKOSHIHONMACHI, TOYAMA CITY, JAPAN 930-8511
Phone +81-76-423-5137
Fax +81-76-493-5252