YAMAHA

RCX340

Programming Manual

Ver. 1.10

EGR9148110

E135

Introduction

Our sincere thanks for your purchase of this YAMAHA RCX340 robot controller.

This manual describes robot program commands and related information for using YAMAHA RCX340 robot controllers. Be sure to read this manual carefully as well as related manuals and comply with their instructions for using the YAMAHA robot controllers safely and correctly.
For details on how to operate YAMAHA robot controllers, refer to the separate controller user's manual that comes with the YAMAHA robot controller.

Applicable controllers: RCX340

Safety precautions

Be sure to read before using

Before using the YAMAHA robot controller, be sure to read this manual and related manuals, and follow their instructions to use the robot controller safely and correctly.
Warning and caution items listed in this manual relate to YAMAHA robot controllers.
When this robot controller is used in a robot controller system, please take appropriate safety measures as required by the user's individual system.
This manual classifies safety caution items and operating points into the following levels, along with symbols for signal words "CAUTION" and "NOTE".

CAUTION

"CAUTION" indicates a potentially hazardous situation which, if not avoided, could result in minor or moderate injury or damage to the equipment or software.

NOTE

Primarily explains function differences, etc., between software versions.

MEMO

Explains robot operation procedures in a simple and clear manner.

Note that the items classified into "CAUTION" might result in serious injury depending on the situation or environmental conditions. So always comply with CAUTION instructions since these are essential to maintain safety.
Keep this manual carefully so that the operator can refer to it when needed. Also make sure that this manual reaches the end user.

■ System design precautions

CAUTION

When the program execution stops before it is complete, the program re-executes the command that has stopped. Keep this point in mind when re-executing the program, for example, when using an arch motion with the MOVE command, a relative movement command such as the MOVEI or DRIVEI command, or a communication command such as the SEND command.

CONTENTS

CONTENTS

Chapter 1
Writing Programs

1 The YAMAHA Robot Language

The YAMAHA robot language was developed by Yamaha Motor Co., Ltd. IM Company for simple and efficient programming to control YAMAHA industrial robots. The YAMAHA robot language is similar to BASIC (Beginner's All-purpose Symbolic Instruction Code) and makes even complex robot movements easy to program. This manual explains how to write robot control programs with the YAMAHA robot language, including actual examples on how its commands are used.

2 Characters

The characters and symbols used in the YAMAHA robot language are shown below.
Only 1-byte characters can be used.

- **Alphabetic characters**
  A to Z, a to z
- **Numbers**
  0 to 9
- **Symbols**
  ( ) [ ] + - * / ^ = < > & | ~ _ % ! # $: ; , . " ' @ ?
- **katakana (Japanese phonetic characters)**

MEMO

- Katakana (Japanese phonetic characters) cannot be entered from a programming box. Katakana can be used when communicating with a host computer (if it handles katakana).
- Spaces are also counted as characters (1 space = 1 character).

3 Program Basics

NOTE
- For details regarding sub-procedure, refer to "11 CALL" and "105 SUB to END SUB" in Chapter 7.

NOTE
- For details regarding user defined functions, refer to "21 DEF FN" in Chapter 7.

Programs are written in a "1 line = 1 command" format, and every line must contain a command. Blank lines (lines with no command) will cause an error when the program is executed. The program's final line, in particular, must not be blank.

To increase the program's efficiency, processes which are repeated within the program should be written as subroutines or sub-procedures which can be called from the main routine. Moreover, same processing items which occurs in multiple programs should be written as common routines within a program named [COMMON], allowing those processing items to be called from multiple programs.

User functions can be defined for specific calculations. Defined user functions are easily called, allowing even complex calculations to be easily performed.

Multi-task programs can also be used to execute multiple command statements simultaneously in a parallel processing manner.

Using the above functions allows easy creation of programs which perform complex processing.

4 Program Names

Each program to be created in the robot controller must have its own name.
Programs can be named as desired provided that the following conditions are satisfied:

- Program names may contain no more than 32 characters, comprising a combination of alphanumeric characters and underscores (_).
- Each program must have a unique name (no duplications).

The 2 program names shown below are reserved for system operations, and programs with these names have a special meaning.

A) SEQUENCE
B) COMMON

The functions of these programs are explained below.

A) SEQUENCE

Functions Unlike standard robot programs, the RCX340 Controller allows the execution of high-speed processing programs (sequence programs) in response to robot inputs and outputs (DI, DO, MO, LO, TO, SI, SO). Specify a program name of "SEQUENCE" to use this function, thus creating a pseudo PLC within the controller.

When the controller is in the AUTO or MANUAL mode, a SEQUENCE program can be executed in fixed cycles (regardless of the program execution status) in response to dedicated DI10 (sequence control input) input signals, with the cycle being determined by the program capacity. For details, see "Sequence program specifications".
This allows sensors, push-button switches, and solenoid valves, etc., to be monitored and operated by input/output signals.
Moreover, because the sequence programs are written in robot language, they can easily be created without having to use a new and unfamiliar language.

SAMPLE

```
DO(20)=~DI(20)
DO(25)=DI(21) AND DI(22)
MO(26)=DO(26) OR DO(25)
       :
```

REFERENCE For details, see "Sequence function".

B) COMMON

Functions A separate "COMMON" program can be created to perform the same processing in multiple robot programs. The common processing routine which has been written in the COMMON program can be called and executed as required from multiple programs. This enables efficient use of the programming space.

The sample COMMON program shown below contains two processing items (obtaining the distance between 2 points (SUB *DISTANCE), and obtaining the area (*AREA)) which are written as common routines, and these are called from separate programs (SAMPLE 1 and SAMPLE 2).
When SAMPLE1 or SAMPLE2 is executed, the SUB *DISTANCE (A!,B!,C!) and the *AREA routine are executed.

SAMPLE

```
Program name: SAMPLE1
   X!=2.5
   Y!=1.2
   CALL  *DISTANCE(X!,Y!,REF C!)
   GOSUB *AREA
   PRINT C!,Z!
   HALT

Program name: SAMPLE2
   X!=5.5
   Y!=0.2
   CALL  *DISTANCE(X!,Y!,REF C!)
   GOSUB *AREA
   PRINT C!,Z!
   HALT

Program name: COMMON··········· Common routine
   SUB *DISTANCE(A!,B!,C!)
      C!=SQR(A!^2+B!^2)
   END SUB
   *AREA:
      Z!=X!*Y!
   RETURN
```

REFERENCE For details, refer to the command explanations given in this manual.

5 Identifiers

"Identifiers" are a combination of characters and numerals used for label names, variable names, and procedure names. Identifiers can be named as desired provided that the following conditions are satisfied:

- Identifiers must consist only of alphanumeric characters and underscores (_). Special symbols cannot be used, and the identifier must not begin with an underscore (_).
- The identifier length must not exceed 32 characters (all characters beyond the 32th character are ignored).
- The maximum number of usable identifiers varies depending on the length of the identifiers. When all identifier length is 32 characters, the number is at the maximum. Local variables can be used up to 128 (in one program task) and global variables can be used up to 512.
- Variable names must not be the same as a reserved word, or the same as a name defined as a system variable. Moreover, variable name character strings must begin with an alphabetic character. For label names, however, the "*" mark may be immediately followed by a numeric character.

SAMPLE

```
LOOP, SUBROUTINE, GET_DATA
```

REFERENCE For details regarding reserved words, see Chapter 11 "1. Reserved word list".

6 LABEL Statement

Defines a <label> on a program line.

Format

```
*<label> :
```

A <label> must always begin with an asterisk mark (*), and it must be located at the beginning of the line.
Although a colon mark (:) is required at the end of the <label> when defining it, this mark is not required when specifying the <label> as a jump destination in a program.

1. A <label> must begin with an alphabetic or numeric character.
2. Alphanumeric and underbars (_) can be used as the remaining <label> characters. Special symbols, etc., cannot be used.
3. The <label> must not exceed 32 characters (all characters beyond the 32th character are ignored).

SAMPLE

```
*ST:     ························ *ST label is defined.
 MOVE P,P0
 DO(20) = 1
 MOVE P,P1
 DO(20) = 0
GOTO *ST························ Jumps to *ST.
HALT
```

7 Comment

Characters which follow REM or an apostrophe mark (" ' ") are processed as a comment. Comment statements are not executed. Moreover, comments may begin at any point in the line.

SAMPLE

```
REM *** MAIN PROGRAM ***
   (Main program)
' *** SUBROUTINE ***
   (Subroutine)
HALT   ' HALT COMMAND········This comment may begin at any
                             point in the line.
```

8 Command Statement Format

Format

```
[<label>:] <statement> [<operand>]
```

One robot language command must be written on a single line and arranged in the format shown below:

- Items enclosed in [ ] can be omitted. This, however, excludes [ ] that specifies a robot number, point variable, or shift variable. For these, [ ] needs to be written directly.
- Items enclosed in < > must be written in a specific format.
- Items not enclosed in < > should be written directly as shown.
- Items surrounded by | | are selectable.
- The label can be omitted. When using a label, it must always be preceded by an asterisk (*), and it must end with a colon (:) (the colon is unnecessary when a label is used as a branching destination).

For details regarding labels, refer to "6 LABEL Statement" in this chapter.

- Operands may be unnecessary for some commands.
- Programs are executed in order from top to bottom unless a branching instruction is given.

1 line may contain no more than 255 characters.

Chapter 2

Constants

1 Outline

Constants can be divided into two main categories: "numeric types" and "character types". These categories are further divided as shown below.

Category	Type	Details/Range
Numeric type	Integer type	Decimal constants -1,073,741,824 to 1,073,741,823
		Binary constants &B0 to &B11111111
		Hexadecimal constants &H80000000 to &H7FFFFFFF
	Real type	Single-precision real numbers -999,999.9 to +999,999.9
		Exponential format single-precision real numbers $-1.0*10^{38}$ to $+1.0*10^{38}$
Character type	Character string	Alphabetic, numeric, special character, or katakana (Japanese) character string of 255 bytes or less.

2 Numeric constants

2.1 Integer constants

1. **Decimal constants**
   Integers from –1,073,741,824 to 1,073,741,823 may be used.
2. **Binary constants**
   Unsigned binary numbers of 8 bits or less may be used. The prefix "&B" is attached to the number to define it as a binary number.
   Range: &B0 (decimal: 0) to &B11111111 (decimal: 255)
3. **Hexadecimal constants**
   Signed hexadecimal numbers of 32 bits or less may be used. The prefix "&H" is attached to the number to define it as a hexadecimal number.
   Range: &H80000000 (decimal: -2,147,483,648) to &H7FFFFFFF (decimal: 2,147,483,647)

2.2 Real constants

1. **Single-precision real numbers**
   Real numbers from -999999.9 to +999999.9 may be used.
   • 7 digits including integers and decimals. (For example, ".0000001" may be used.)
2. **Single-precision real numbers in exponent form**
   Numbers from $-1.0*10^{38}$ to $+1.0*10^{38}$ may be used.
   • Mantissas should be 7 digits or less, including integers and decimals.

```
Examples: -1. 23456E-12
          3. 14E0
          1. E5
```

MEMO

• An integer constant range of –1,073,741,824 to 1,073,741,823 is expressed in signed hexadecimal number as &HC0000000 to &H3FFFFFFF.

3 Character constants

Character type constants are character string data enclosed in quotation marks ("). The character string must not exceed 255 bytes in length, and it may contain upper-case alphabetic characters, numerals, special characters, or katakana (Japanese) characters.
To include a double quotation mark (") in a string, enter two double quotation marks in succession.

SAMPLE

```
"YAMAHA ROBOT"
"EXAMPLE OF""A"""················ EXAMPLE OF "A"
PRINT "COMPLETED"
"YAMAHA ROBOT"
```

Chapter 3
Variables

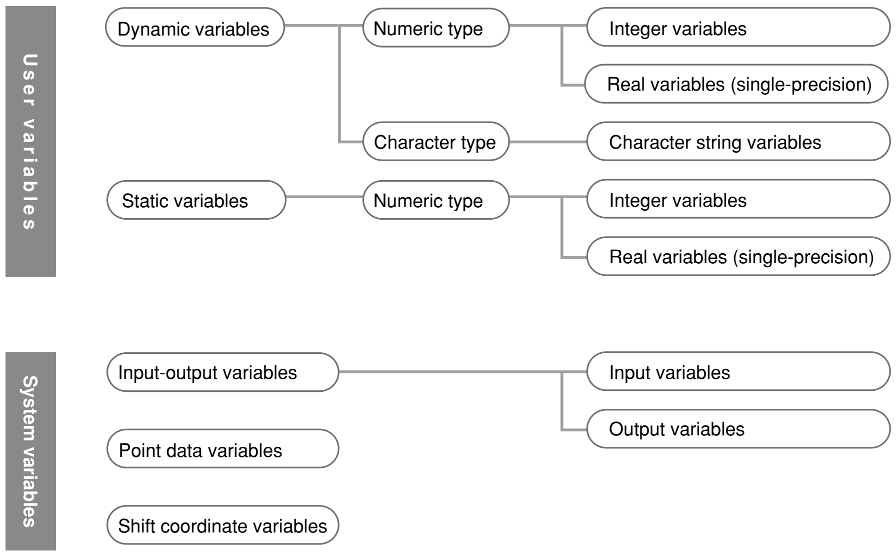
1 Outline

There are "user variables" which can be freely defined, and "system variables" which have pre-defined names and functions.

User variables consist of "dynamic variables" and "static variables". "Dynamic variables" are cleared at program editing, program resets, and program switching. "Static variables" are not cleared unless the memory is cleared. The names of dynamic variables can be freely defined, and array variables can also be used.

Variables can be used simply by specifying the variable name and type in the program. A declaration is not necessarily required. However, array variables must be pre-defined by a DIM statement.

User variables & system variables



33301-R9-00

REFERENCE For details regarding array variables, see "5 Array variables" in this chapter.

2 User Variables & System Variables

2.1 User Variables

Numeric type variables consist of an "integer type" and a "real type", and these two types have different usable numeric value ranges. Moreover, each of these types has different usable variables (character string variables, array variables, etc.), and different data ranges, as shown below.

Category	Variable Type	Details/Range
Dynamic variables	Numeric type	Integer type variables -1,073,741,824 to 1,073,741,823 (Signed hexadecimal constants: &HC0000000 to &H3FFFFFFF)
		Real variables (single-precision) $-1.0*10^{38}$ to $+1.0*10^{38}$
	Character type	Character string variables Alphabetic, numeric, special character, or katakana (Japanese) character string of 255 bytes or less.
Static variables	Numeric type	Integer type variables -1,073,741,824 to 1,073,741,823
		Real variables (single-precision) $-1.0*10^{38}$ to $+1.0*10^{38}$
Array variables	Numeric type	Integer array variables -1,073,741,824 to 1,073,741,823
		Real number array variables (single-precision) $-1.0*10^{38}$ to $+1.0*10^{38}$
	Character type	Character string array variables Alphabetic, numeric, special character, or katakana (Japanese) character string of 255 bytes or less.

NOTE
- Array variables are dynamic variables.

2.2 System Variables

As shown below, system variables have pre-defined names which cannot be changed.

Category	Type	Details	Specific Examples
Input/output variables	Input variable	External signal / status inputs	DI, SI, SIW, SID
	Output variable	External signal / status outputs	DO, SO, SOW, SOD
Point variable		Handles point data	Pnnnn
Shift variable		Specifies the shift coordinate No. as a numeric constant or expression.	Sn

REFERENCE For details, see "9 System Variables" in this chapter.

3 Variable Names

3.1 Dynamic Variable Names

Dynamic variables can be named as desired, provided that the following conditions are satisfied:

- The name must consist only of alphanumeric characters and underscores (_). Special symbols cannot be used.
- The name must not exceed 32 characters (all characters beyond the 32th character are ignored).
- The name must begin with an alphabetic character.

SAMPLE

```
COUNT    ............................ ○ Use is permitted
COUNT123 ............................ ○ Use is permitted
2COUNT   ............................ × Use is not permitted
```

- Variable names must not be the same as a reserved word.
- Variable names must not begin with characters used for system variable names (pre-defined variables). These characters include the following: FN, DIn, DOn, MOn, LOn, TOn, SIn, SOn, Pn, Sn, Hn ("n" denotes a numeric value).

SAMPLE

```
COUNT    ............................ ○ Use is permitted
ABS      ............................ × (Reserved word)
FNAME    ............................ × (FN: pre-defined variable)
S91      ............................ × (Sn: pre-defined variable)
```

REFERENCE For details regarding reserved words, see Chapter 11 "1 Reserved word list".

3.2 Static Variable Names

Static variable names are determined as shown below, and these names cannot be changed.

Variable Type	Variable Name
Integer variable	SGIn (n: 0 to 31)
Real variable	SGRn (n: 0 to 31)

Static variables are cleared only when initializing is executed by a SYSTEM mode or online command.

REFERENCE For details regarding clearing of static variables, see "12 Clearing variables" in this chapter.

4 Variable Types

The type of variable is specified by the type declaration character attached at the end of the variable name.
However, because the names of static variables are determined based on their type, no type declaration statement is required.

Type Declaration Character	Variable Type	Specific Examples
$	Character type variables	STR1$
%	Integer type variables	CONT0%, ACT%(1)
!	Real type variables	CNT1!, CNT1

MEMO

- If no type declaration character is attached, the variable is viewed as a real type.
- Variables using the same identifier are recognized to be different from each other by the type of each variable.
  - ASP_DEF% .... Integer variable
  - ASP_DEF .... Real variable

  → ASP_DEF% and ASP_DEF are different variables.
  - ASP_DEF! .... Real variable
  - ASP_DEF .... Real variable

  → ASP_DEF! and ASP_DEF are the same variables.

4.1 Numeric variables

NOTE

- When a real number is assigned to an integer type variable, the decimal value is rounded off to the nearest whole number. For details, refer to Chapter 4 "1.5 Data format conversion".

Integer variables

Integer variables and integer array elements can handle an integer from –1,073,741,824 to 1,073,741,823 (in signed hexadecimal, this range is expressed as &HC0000000 to &H3FFFFFFF).

```
Examples: R1% = 10
          R2%(2) = R1% + 10000
```

NOTE

- The "!" used in real variables may be omitted .

Real variables

Real variables and real array elements can handle a real number from –1.0*10^{38} to 1.0*10^{38}.

```
Examples: R1!    = 10.31
          R2!(2)= R1% + 1.98E3
```

4.2 Character variables

Character variables and character array elements can handle a character string of up to 255 characters.
Character strings may include alphabetic characters, numbers, symbols and katakana (Japanese phonetic characters).

```
Examples: R1$ = "YAMAHA"
          R2$(2) = R1$ + "MOTOR" "YAMAHA MOTOR"
```

5 Array variables

Both numeric and character type arrays can be used at dynamic variables.
Using an array allows multiple same-type continuous data to be handled together.
Each of the array elements is referenced in accordance with the parenthesized subscript which appears after each variable name. Subscripts may include integers or <expressions> in up to 3 dimensions.
In order to use an array, a DIM statement must be declared in advance, and the maximum number of elements which can be used is the declared subscripts + 1 (0 ~ number of declared subscripts).

MEMO

- All array variables are dynamic variables. (For details regarding dynamic variables, see "11 Valid range of variables" in this Chapter.)
- The length of an array variable that can be declared with the DIM statement depends on the program size.

Format

```
<variable name>[| % |](<expression>, [<expression>, [<expression>]])
               | ! |
               | $ |
```

SAMPLE

```
A%(1)     ............................ Integer array variable
DATA!(1,10,3)  ..................... Single-precision real number array
                                     variable (3-dimension array)
STRING$(10) ........................ Character array variable
```

6 Value Assignments

An assignment statement (LET) can also be used to assign a value to a variable.

MEMO

- "LET" directly specifies an assignment statement, and it can always be omitted.

Format

```
[LET] <variable> = <expression>
```

Write the value assignment target variable on the left side, and write the assignment value or the <expression> on the right side. The <expression> may be a constant, a variable, or an arithmetic expression, etc.

REFERENCE For details, refer to Chapter 7 "51 LET (Assignment Statement)"

7 Type Conversions

When different-type values are assigned to variables, the data type is converted as described below.

- When a real number is assigned to an integer type:
  The decimal value is rounded off to the nearest whole number.
- When an integer is assigned to a real type:
  The integer is assigned as it is, and is handled as a real number.
- When a numeric value is assigned to a character string type:
  The numeric value is automatically converted to a character string which is then assigned.
- When a character string is assigned to numeric type:
  This assignment is not possible,and an error will occur at the program is execution. Use the "VAL" command to convert the character string to a numeric value, and that value is then assigned.

8 Value Pass-Along & Reference Pass-Along

A variable can be passed along when a sub-procedure is called by a CALL statement. This pass-along can occur in either of two ways: as a value pass-along, or as a reference pass-along.

Value pass-along

With this method, the variable's value is passed along to the sub-procedure. Even if this value is changed within the sub-procedure, **the content of the call source variable is not changed.**
A value pass-along occurs when the CALL statement's actual argument specifies a constant, an expression, a variable, or an array element (array name followed by (<subscript>)).

Reference pass-along

With this method, the variable's reference (address in memory) is passed along to the sub-procedure. If this value is changed within the sub-procedure, **the content of the call source variable is also changed.**
A reference pass-along occurs when the CALL statement's actual argument specifies an entire array (an array named followed by parenthetical content), or when the actual argument is preceded by "REF".

Value pass-along & reference pass-along

Value pass-along

```
X%=5
CALL *TEST( X% )
PRINT X%
HALT
’ SUB ROUTINE
SUB *TEST( A% )
   A%=A%*10
END SUB
```

Execution result: the X% value remains as "5".

Reference pass-along

```
X%=5
CALL *TEST( REF X% )
PRINT X%
HALT
’ SUB ROUTINE
SUB *TEST( A% )
   A%=A%*10
END SUB
```

Execution result: the X% value becomes "50".

33302-R7-00

9 System Variables

The following system variables are pre-defined, and other variable names must not begin with the characters used for these system variable names.

Variable Type	Format	Meaning
Point variable	Pnnn / P " ["<expression>"] "	Specifies a point number.
Shift variable	Sn / S " ["<expression>"] "	Specifies the shift number as a constant or as an expression.
Parallel input variable	DI(mb), DIm(b)	Parallel input signal status.
Parallel output variable	DO(mb), DOm(b)	Parallel output signal setting and status.
Internal output variable	MO(mb), MOm(b)	Controller's internal output signal setting and status
Arm lock output variable	LO(mb), LOm(b)	Axis-specific movement prohibit.
Timer output variable	TO(mb), TOm(b)	For sequence program's timer function.
Serial input variable	SI(mb), SIm(b)	Serial input signal status.
Serial output variable	SO(mb), SOm(b)	Serial output signal setting and status.
Serial word input	SIW(m)	Serial input's word information status
Serial double-word input	SID(m)	Serial input's double-word information status.
Serial word output	SOW(m)	Serial output's word information status
Serial double-word output	SOD(m)	Serial output's double-word information status.

9.1 Point data variable

This variable specifies a point data number with a numeric constant or expression.

Format

```
Pnnnnn or P" ["<expression>"]"
```

Values n: Point number .... 0 to 9

Each bracket in quotation marks ("[" "]") must be written. Brackets are not used to indicate an item that may be omitted.

Functions A point data number is expressed with a 'P' followed by a number of 5 digits or less, or an expression surrounded by brackets ("[" <expression> "]").

Point numbers from 0 to 29999 can be specified with point variables.

```
Examples: P0
          P110
          P[A]
          P[START_POINT]
          P[A(10)]
```

9.2 Shift coordinate variable

This variable specifies a shift coordinate number with a numeric constant or expression.

Format

```
Snn or S "["<expression>"]"
```

Values n: Shift number .... 0 to 9
Each bracket in quotation marks ("[" "]") must be written. Brackets are not used to indicate an item that may be omitted.

Functions A shift number is expressed with an 'S' followed by a 2-digits number or an expression surrounded by brackets ("[" <expression> "]"). As a shift number, 0 to 39 can be specified.

```
Examples: S1
          S[A]
          S[BASE]
          S[A(10)]
```

MEMO
- The "shift coordinate range" for each shift number can be changed from the programming box.

9.3 Parallel input variable

This variable is used to indicate the status of parallel input signals.

Format 1

```
DIm([b,···,b])
```

Format 2

```
DI(mb,···,mb)
```

Values m : port number .... 0 to 7, 10 to 17, 20 to 27
b : bit definition .... 0 to 7
If the bit definition is omitted, bits 0 to 7 are all selected.

```
Examples: A%=DI1()
           →Input status of ports DI(17) to DI(10)
            is assigned to variable A%.
           A 0 to 255 integer can be assigned to A%.
          A%=DI5(7,4,0)
           →Input status of DI(57), DI(54) and
            DI(50) is assigned to variable A%.
           (If all above signals are 1(ON), then A%=7.)
          A%=DI(27,15,10)
           →Input status of DI(27), DI(15) and
            DI(10) is assigned to variable A%.
           (If all above signals except DI(10) are 1 (ON), then A%=6.)
          WAIT DI(21)=1
           →Waits for DI(21) to change to 1(ON).
```

MEMO

- When specifying multiple bits, specify them from left to right in descending order (large to small).
- A '0' is entered if there is no actual input board.

9.4 Parallel output variable

Specifies the parallel output signal or indicates the output status.

Format 1

```
DOm([b,・・・,b])
```

Format 2

```
DO(mb,・・・,mb)
```

Values

m : port number	0 to 7, 10 to 17, 20 to 27
b : bit definition	0 to 7

If the bit definition is omitted, bits 0 to 7 are all selected.

```
Examples: A%=DO2()
           →Output status of DO(27) to DO(20) is
             assigned to variable A%.
          A%=DO5(7,4,0)
           →Output status of DO(57), DO(54) and
             DO(50) is assigned to variable A%.
            (If all above signals are 1(ON), then A%=7.)
          A%=DO(37,25,20)
           →Output status of DO(37), DO(25) and
             DO(20) is assigned to variable A%.
            (If all above signals except DO(20) are 1
            (ON), then A%=6.)
          DO3()=B%
           →Changes to a status in which the DO(37)
             to DO(30) output can be indicated by B%.
           For example, if B% is "123": If a binary
           number is used, "123" will become
           "01111011", DO(37) and DO(32) will become
           "0", and the other bits will become "1".
          DO4(5,4,0)=&B101
           →DO(45) and DO(40) become "1", and DO(44) becomes "0".
```

MEMO

- When specifying multiple bits, specify them from left to right in descending order (large to small).
- A '0' is entered if there is no actual input board.

9.5 Internal output variable

Specifies the controller's internal output signals and indicates the signal status.

Format 1

```
MOm([b,・・・,b])
```

Format 2

```
MO(mb,・・・,mb)
```

Values m : port number .... 0 to 7, 10 to 17, 20 to 27, 30 to 33
b : bit definition .... 0 to 7
• If the bit definition is omitted, bits 0 to 7 are all selected.

Functions Internal output variables which are used only in the controller, can be changed and referenced. These variables are used for signal communications, etc., with the sequence program.
Ports 30 to 33 are for dedicated internal output variables which can only be referenced (they cannot be changed).

1. **Port 30 indicates the status of origin sensors for axes 1 to 8 (in order from bit 0). Port 31 indicates the status of origin sensors for axes 9 to 16 (in order from bit 0).**
   Each bit sets to '1' when the origin sensor turns ON, and to '0' when OFF.
2. **Port 34 indicates the HOLD status of axes 1 to 8 (in order from bit 0). Port 35 indicates the HOLD status of axes 9 to 16 (in order from bit 0).**
   Each bit sets to '1' when the axis is in HOLD status, and to '0' when not.

Bit	7	6	5	4	3	2	1	0
Port 30	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1
Port 31	Axis 16	Axis 15	Axis 14	Axis 13	Axis 12	Axis 11	Axis 10	Axis 9
	Origin sensor statuses 0: OFF / 1: ON (Axis 1 is not connected)							
Port 34	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1
Port 35	Axis 16	Axis 15	Axis 14	Axis 13	Axis 12	Axis 11	Axis 10	Axis 9
	Hold status 0: RELEASE / 1: HOLD (Axis 1 is not connected)							

MEMO

- Axes where no origin sensor is connected are always ON.
- Being in HOLD status means that the axis movement is stopped and positioned within the target point tolerance while the servo is still turned ON.
- When the servo turns OFF, the HOLD status is released.
- Axes not being used are set to '1'.
- The status of each axis in order from the smallest axis number used by robot 1 is maintained. Example) In the case of a configuration where robot 1 has 5 axes and robot 2 has 4 axes, bits 0 to 4 of port 30 indicate the status of axes 1 to 5 of robot 1, bits 5 to 7 of port 30 indicate the status of axes 1 to 3 of robot 2, and bit 0 of port 31 indicates the status of axis 4 of robot 2.

```
Examples: A%=MO2 ()
          →Internal output status of MO(27) to
           MO(20) is assigned to variable A%.
         A%=MO5(7,4,0)
          →Internal output status of MO(57), MO(54)
           and MO(50) is assigned to variable A%.
          (If all above signals are 1 (ON), then A%=7.)
         A%=MO(37,25,20)
          →Internal output status of MO(37), MO(25)
           and MO(20) is assigned to variable A%.
          (If all above signals except MO(25) are 1 (ON), then A%=5.)
```

MEMO
- When specifying multiple bits, specify them from left to right in descending order (large to small).

9.6 Arm lock output variable

Specifies axis-specific movement prohibit settings.

Format 1

```
LOm([b,···,b])
```

Format 2

```
LO(mb,···,mb)
```

Values

m : port number	0, 1
b : bit definition	0 to 7

- If the bit definition is omitted, bits 0 to 7 are all selected.

Functions The contents of this variable can be output and referred to as needed.
Of Port 0, bits 0 to 7 respectively correspond to axes 1 to 8, and of port 1, bits 0 to respectively correspond to axes 9 to 16.
When this bit is ON, movement on the corresponding axis is prohibited.

```
Examples: A%=LO0()
          →Arm lock status of LO(07) to LO(00) is
           assigned to variable A%.
          A%=LO0(7,4,0)
          →Arm lock status of LO(07), LO(04) and
           LO(00) is assigned to variable A%.
          (If all above signals are 1 (ON), then A%=7.)
         A%=LO0(06,04,01)
          →Arm lock status of LO(06), LO(04) and
           LO(01) is assigned to variable A%.
          (If all above signals except LO(01) are 1
          (ON), then A%=6.)
```

MEMO

- When specifying multiple bits, specify them from left to right in descending order (large to small).
- Servo OFF to ON switching is disabled if an arm lock is in effect at even 1 axis.
- When performing JOG movement in the MANUAL mode, axis movement is possible at axes where an arm lock status is not in effect, even if an arm lock status is in effect at another axis.
- When executing movement commands from the program, etc., the "12.3 XX.Arm lock" error will occur if an arm lock status is in effect at the axis in question.
- Arm locks sequentially correspond to axes in order from the axis with the smallest axis number used by robot 1.
  Example) In the case of a configuration where robot 1 has 5 axes and robot 2 has 4 axes, the status of axes 1 to 5 of robot 1 is set by bits 0 to 4 of port 0, the status of axes 1 to 3 of robot 2 is set by bits 5 to 7 of port 0, and the prohibition of motion of axis 4 of robot 2 is set by bit 0 of port 1.

9.7 Timer output variable

This variable is used in the timer function of a sequence program.

Format 1

```
TOm([b,・・・,b])
```

Format 2

```
TO(mb,・・・,mb)
```

Values m : port number 0, 1
b : bit definition .... 0 to 7
- If the bit definition is omitted, bits 0 to 7 are all selected.

Functions The contents of this variable can be changed and referred to as needed.
Timer function can be used only in the sequence program. If this variable is output in a normal program, it is an internal output.

For details regarding sequence program usage examples, refer to the timer usage examples given in "Input/output variables".

```
Examples:A%=TO0()
          →Status of TO(07) to TO(00) is assigned
            to variable A%.
         A%=TO0(7,4,0)
          →Status of TO(07), TO(04) and TO(00) is
            assigned to variable A%.
          (If all above signals are 1 (ON), then A%=7.)
         A%=TO(06,04,01)
          →Status of TO(06), TO(04) and TO(01) is
            assigned to variable A%.
          (If all above signals except TO(01) are 1
          (ON), then A%=6.)
```

MEMO

- When specifying multiple bits, specify them from left to right in descending order (large to small).

9.8 Serial input variable

This variable is used to indicate the status of serial input signals.

Format 1

```
SIm([b,···,b])
```

Format 2

```
SI(mb,···,mb)
```

Values

m : port number	....	0 to 7, 10 to 17, 20 to 27
b : bit definition	....	0 to 7

• If the bit definition is omitted, bits 0 to 7 are all selected.

```
Examples:A%=SI1()
           →Input status of ports SI(17) to SI(10)
             is assigned to variable A%.
         A%=SI5(7,4,0)
           →Input status of SI(57), SI(54) and
             SI(50) is assigned to variable A%.
           (If all above signals are 1(ON), then A%=7.)
         A%=SI(27,15,10)
           →Input status of SI(27), SI(15) and
             SI(10) is assigned to variable A%.
           (If all above signals except SI(10) are 1
           (ON), then A%=6.)
         WAIT SI(21)=1
           →Waits until SI(21) sets to 1 (ON).
```

MEMO

- When specifying multiple bits, specify them from left to right in descending order (large to small).
- A '0' is entered if there is no actual serial board.

9.9 Serial output variable

This variable is used to define the serial output signals and indicate the output status.

Format 1

```
SOm([b,・・・,b])
```

Format 2

```
SO(mb,・・・,mb)
```

Values

m : port number	0 to 7, 10 to 17, 20 to 27
b : bit definition	0 to 7

• If the bit definition is omitted, bits 0 to 7 are all selected.

```
Examples:A%=SO2()
          →Output status of SO(27) to SO(20) is
           assigned to variable A%.
         A%=SO5(7,4,0)
          →Output status of SO(57), SO(54) and
           SO(50) is assigned to variable A%.
          (If all above signals are 1(ON), then A%=7.)
         A%=SO(37,25,20)
          →Output status of SO(37), SO(25) and
           SO(20) is assigned to variable A%.
          (If all above signals except SO(25) are 1
          (ON), then A%=5.)
         SO3()=B%
          →Changes to a status in which the DO(37)
           to DO(30) output can be indicated by B%.
          For example, if B% is "123": If a binary
          number is used, "123" will become
          "01111011", DO(37) and DO(32) will become
          "0", and the other bits will become "1".
         SO4(5,4,0)=&B101
          →DO(45) and DO(40) become "1", and DO(44) becomes "0".
```

MEMO

• When specifying multiple bits, specify them from left to right in descending order (large to small).
• External output is unavailable if the serial port does not actually exist.

9.10 Serial word input

This variable indicates the status of the serial input word information.

Format

```
SIW(m)
```

Values m : Port No. 2 to 15
The acquisition range is 0 (&H0000) to 65535 (&HFFFF).

```
Examples: A%=SIW(2)
           →The input state from SIW (2) is
             assigned to variable A%.
          A%=SIW(15)
           →The input state from SIW (15) is
             assigned to variable A%.
```

MEMO

- The information is handled as unsigned word data.
- '0' is input if the serial port does not actually exist.

9.11 Serial double word input

This variable indicates the state of the serial input word information as a double word.

Format

```
SID(m)
```

Values m : Port No. 2, 4, 6, 8, 10, 12, 14
The acquisition range is -1073741824 (&HC0000000) to 1073741823 (&H3FFFFFFF).

```
Examples: A%=SID(2)
           →The input state from SIW (2) , SIW (3)
             is assigned to variable A%.
          A%=SID(14)
           →The input state from SIW (14), SIW (15)
             is assigned to variable A%.
```

MEMO

- The information is handled as signed double word data.
- '0' is input if the serial port does not actually exist.
- An error will occur if the value is not within the acquisition range (&H80000000 to &HBFFFFFFF, &H40000000 to &H7FFFFFFF.)
- The lower port number data is placed at the lower address.
  For example, if SIW(2) =&H2345,SIW(3) =&H0001, then SID(2) =&H000123245.

9.12 Serial word output

Outputs to the serial output word information or indicates the output status.

Format

```
SOW(m)
```

Values m : Port No. 2 to 15

The output range is 0 (&H0000) to 65535 (&HFFFF).

Note that if a negative value is output, the low-order word information will be output after being converted to hexadecimal.

```
Examples:A%=SOW(2)
          →The output status from SOW (2) is
           assigned to variable A%.
         SOW(15)=A%
          →The contents of variable A% are
           assigned in SOW (15).
          If the variable A% value exceeds the output range,
          the low-order word information will be assigned.
         SOW(15)=-255
          →The contents of -255 (&HFFFFFF01) are
           assigned to SOW (15).
          -255 is a negative value, so the low-order
          word information (&HFF01) will be assigned.
```

MEMO

- The information is handled as unsigned word data.
- If a serial board does not actually exist, the information is not output externally.
- If a value exceeding the output range is assigned, the low-order 2-byte information is output.

9.13 Serial double word output

Output the status of serial output word information in a double word, or indicates the output status.

Format

```
SOD(m)
```

Values m : Port No. 2, 4, 6, 8, 10, 12, 14

The output range is -1073741824 (&HC0000000) to 1073741823 (&H3FFFFFFF).

```
Examples: A%=SOD(2)
          →The input status from SOW (2) is assigned to variable A%.
         SOD(14)=A%
          →The contents of variable A% are assigned in SOD (14).
```

- The information is handled as signed double word data.
- If a serial board does not actually exist, the information is not output externally.
- An error will occur if the value is not within the output range (&H80000000 to &HBFFFFFFF, &H40000000 to &H7FFFFFFF.)
- The lower port number data is placed at the lower address.
  For example, if SOW(2) =&H2345,SOW(3) =&H0001, then SOD(2) =&H000123245.

10 Bit Settings

Bits can be specified for input/output variables by any of the following methods.

1. Single bit

To specify only 1 of the bits, the target port number and bit number are specified in parentheses. The port number may also be specified outside the parentheses.

Programming example: DOm(b)DOm(b)
```
Example: DO(25)     Specifies bit 5 of port 2.
         DO2(5)
```

2. Same-port multiple bits

To specify multiple bits at the same port, those bit numbers are specified in parentheses (separated by commas) following the port number.
The port number may also be specified in parentheses.

Programming example: DOm(b,b,...,b) DO(mb,mb,...,mb)
```
Example: DO2(7,5,3) Specifies DO(27), DO(25), DO(23)
         DO(27,25,23)
```

3. Different-port multiple bits

To specify multiple bits at different ports, the port number and the 2-digit bit number must be specified in parentheses and must be separated by commas.

Programming example: DO(mb,mb,...,mb)
```
Example: DO(37,25,20) Specifies DO(37), DO(25), DO(20).
```

4. All bits of 1 port

To specify all bits of a single port, use parentheses after the port number. Methods 2 and 3 shown above can also be used.

Programming example: DOm()
```
Example: DO2() Specifies all the DO(27) to DO(20) bits
        →The same result can be obtained by the following:
         DO(27,26,25,24,23,22,21,20)
         or,
         DO2(7,6,5,4,3,2,1,0)
```

11 Valid range of variables

Variable branching occurs as shown below.

11.1 Valid range of dynamic variables

Dynamic variables are divided into global variables and local variables, according to their declaration position in the program. Global and local variables have different valid ranges.

Variable Type	Explanation
Global variables	Variables are declared outside of sub-procedures (outside of program areas enclosed by a SUB statement and END SUB statement). These variables are valid throughout the entire program.
Local variables	Variables are declared within sub-procedures and are valid only in these sub-procedures.

11.2 Valid range of static variables

Static variable data is not cleared when a program reset occurs. Moreover, variable data can be changed and referenced from any program.
The variable names are determined as shown below (they cannot be named as desired).

Variable type	Variable name
Integer variable	SGIn (n: 0 to 31)
Real variable	SGRn (n: 0 to 31)

11.3 Valid range of dynamic array variables

Dynamic array variables are classified into global array variables and local array variables according to their declaration position in the program.

Variable Type	Explanation
Global variables	Variables are declared outside of sub-procedures (outside of program areas enclosed by a SUB statement and END SUB statement). These variables are valid throughout the entire program.
Local variables	Variables are declared within sub-procedures and are valid only in these sub-procedures.

MEMO

- For details regarding arrays, refer to Chapter 3 "5 Array variables".
- A variable declared at the program level can be referenced from a sub-procedure without being passed along as a dummy argument, by using the SHARED statement (for details, refer to Chapter 7 "96 SHARED").

12 Clearing variables

12.1 Clearing dynamic variables

In the cases below, numeric variables are cleared to zero, and character variables are cleared to a null string. The variable array is cleared in the same manner.

- When a program is edited.
- When program switching occurs (including SWI command execution).
- When program compiling occurs.
- When a program reset occurs.
- When dedicated input signal DI15 (program reset input) was turned on while the program was stopped in AUTO mode.
- When either of the following is initialized by an initialization operation.
    1. Program memory
    2. Entire memory
- When any of the following online commands was executed.
  @RESET, @INIT PGM, @INIT MEM, @INIT ALL, @SWI
- When the HALT statement was executed in the program.

12.2 Clearing static variables

In the cases below, integer variables and real variables are cleared to zero.

- When the following is initialized by an initialization operation.
  Entire memory
- When any of the following online commands was executed.
  @INIT MEM, @INIT ALL

MEMO

- Static variable values are not cleared even if the program is edited.

Chapter 4
Expressions and Operations

1 Arithmetic operations

1.1 Arithmetic operators

Operators	Usage Example	Meaning
+	A+B	Adds A to B
-	A-B	Subtracts B from A
*	A*B	Multiplies A by B
/	A/B	Divides A by B
^	A^B	Obtains the B exponent of A (exponent operation)
-	-A	Reverses the sign of A
MOD	A MOD B	Obtains the remainder A divided by B

When a "remainder" (MOD) operation involves real numbers, **the decimal value is rounded off to the nearest whole number which is then converted to an integer** before the calculation is executed. The result represents the remainder of an integer division operation.

```
Examples: A=15 MOD 2          →   A=1(15/2=7....1)
          A=17.34 MOD 5.98    →   A=2(17/5=3....2)
```

1.2 Relational operators

Relational operators are used to compare 2 values. If the result is "true", a "-1" is obtained. If it is "false", a "0" is obtained.

Operators	Usage Example	Meaning
=	A=B	"-1" if A and B are equal, "0" if not.
<>, ><	A<>B	"-1" if A and B are unequal, "0" if not.
<	A<B	"-1" if A is smaller than B, "0" if not.
>	A>B	"-1" if A is larger than B, "0" if not.
<=, =<	A<=B	"-1" if A is equal to or smaller than B, "0" if not.
>=, =>	A>=B	"-1" if A is equal to or larger than B, "0" if not.

```
Examples: A=10>5     →    Since 10 > 5 is "true", A = -1.
```

MEMO

- When using equivalence relational operators with real variables and real arrays, the desired result may not be obtained due to the round-off error.
  Examples: ....A=2
  B=SQR(A!)
  IF A!=B!*B! THEN...
  → In this case, A! will be unequal to B!*B!.

1.3 Logic operations

Logic operators are used to manipulate 1 or 2 values bit by bit. For example, the status of an I/O port can be manipulated.

- Depending on the logic operation performed, the results generated are either 0 or 1.
- Logic operations with real numbers convert the values into integers before they are executed.

Operators	Functions	Meaning
NOT, ~	Logical NOT	Reverses the bits.
AND, &	Logical AND	Becomes "1" when both bits are "1".
OR, |	Logical OR	Becomes "1" when either of the bits is "1".
XOR	Exclusive OR	Becomes "1" when both bits are different.
EQV	Logical equivalence operator	Becomes "1" when both bits are equal.
IMP	Logical implication operator	Becomes "0" when the first bit is "1" and the second bit is "0".

Examples: A%=NOT 13.05 → "-14" is assigned to A% (reversed after being rounded off to 13).

Bit	7	6	5	4	3	2	1	0
13	0	0	0	0	1	1	0	1
NOT 13=-14	1	1	1	1	0	0	1	0

Examples: A%=3 AND 10 → "2" is assigned to A%

Bit	7	6	5	4	3	2	1	0
3	0	0	0	0	0	0	**1**	1
10	0	0	0	0	1	0	**1**	0
3 AND 10 = 2	0	0	0	0	0	0	**1**	0

Examples: A%=3 OR 10 → "11" is assigned to A%

Bit	7	6	5	4	3	2	1	0
3	0	0	0	0	**0**	0	**1**	**1**
10	0	0	0	0	**1**	0	**1**	**0**
3 OR 10 = 11	0	0	0	0	**1**	0	**1**	**1**

Examples: A%=3 XOR 10 → "9" is assigned to A%

Bit	7	6	5	4	3	2	1	0
3	0	0	0	0	**0**	0	1	**1**
10	0	0	0	0	**1**	0	1	**0**
3 XOR 10 = 9	0	0	0	0	**1**	0	0	**1**

1.4 Priority of arithmetic operation

Operations are performed in the following order of priority. When two operations of equal priority appear in the same statement, the operations are executed in order from left to right.

Priority Rank	Arithmetic Operation
1	Expressions included in parentheses
2	Functions, variables
3	^ (exponents)
4	Independent "+" and "-" signs (monominal operators)
5	* (multiplication), / (division)
6	MOD
7	+ (addition), - (subtraction)
8	Relational operators
9	NOT, ~ (Logical NOT)
10	AND, & (logical AND)
11	OR, |, XOR (Logical OR, exclusive OR)

1.5 Data format conversion

Data format is converted in cases where two values of different formats are involved in the same operation.

1. **When a real number is assigned to an integer, decimal places are rounded off.**

```
Examples: A%=125.67          →     A%=126
```

2. **When integers and real numbers are involved in the same operation, the result becomes a real number.**

```
Examples: A(0)=125 * 0.25    →     A(0)=31.25
```

3. **When an integer is divided by an integer, the result is an integer with the remainder discarded.**

```
Examples: A(0)=100/3         →     A(0)=33
```

2 Character string operations

2.1 Character string connection

Character strings may be combined by using the "+" sign.

SAMPLE

```
A$="YAMAHA"
B$="ROBOT"
C$="LANGUAGE"
D$="MOUNTER"
E$=A$+" "+B$+" "+C$
F$=A$+" "+D$
PRINT E$
PRINT F$

Results: YAMAHA ROBOT LANGUAGE
         YAMAHA MOUNTER
```

2.2 Character string comparison

Characters can be compared with the same relational operators as used for numeric values. Character string comparison can be used to find out the contents of character strings, or to sort character strings into alphabetical order.

- In the case of character strings, the comparison is performed from the beginning of each string, character by character.
- If all characters match in both strings, they are considered to be equal.
- Even if only one character in the string differs from its corresponding character in the other string, then the string with the larger (higher) character code is treated as the larger string.
- When the character string lengths differ, the longer of the character strings is judged to be the greater value string.

All examples below are "true".

```
Examples: "AA"<"AB"
          "X&">"X#"
          "DESK"<"DESKS"
```

3 Point data format

NOTE
- The data format is common for axes 1 to 6 for both the joint coordinate format and the Cartesian coordinate format.
- Plus (+) signs can be omitted.
- The first arm and the second arm rotation information is not available on any robot model except the YK500TW.

There are two types of point data formats: joint coordinate format and Cartesian coordinate format. Point numbers are in the range of 0 to 29999.

Coordinate Format	Data Format	Explanation
Joint coordinate format	± nnnnnnn	This is a decimal integer constant of 7 digits or less with a plus or minus sign, and can be specified from –6144000 to 6144000. Unit: [pulses]
Cartesian coordinate format	± nnn.nn to ± nnnnnnn	This is a decimal fraction of a total of 7 digits including 3 or less decimal places. Unit: [mm] or [degrees]

When setting an extended hand system flag for SCARA robots, set either 1 or 2 at the end of the data. If a value other than 1 or 2 is set, or if no value is designated, 0 will be set to indicate that no hand system flag is set.

Hand System	Data Value
RIGHTY (right-handed system)	1
LEFTY (left-handed system)	2

On the YK500TW model robot, the first arm and the second arm movement range is extended beyond 360 degrees (The working envelope for both the first arm and second arm is -225° to +225°). Therefore, attempts to convert Cartesian coordinate data ("mm" units) to joint coordinate data (pulse units) will result in multiple solutions, making the position impossible to determine.
In order to obtain the correct robot position and arm posture when converting to joint coordinates, the first arm and the second arm rotation information is added after the "mm" units point data's extended hand system flag.
The Cartesian coordinate data ("mm" units) is then converted to joint coordinate data (pulse units) according to the specified the first arm and the second arm rotation information.

To set extended the first arm and the second arm rotation information at the YK500TW model robot, a "-1", "0", or "1" value must be specified after the hand system flag. Any other value, or no value, will be processed as "0".

Arm rotation information	Data Value
"mm" $\rightarrow$ pulse converted angle data x (*1) range: -180° < x <= 180°	0
"mm" $\rightarrow$ pulse converted angle data x (*1) range: 180° < x <= 540°	1
"mm" $\rightarrow$ pulse converted angle data x (*1) range: -540° < x <= -180°	-1

*1: The joint-coordinates-converted pulse data represents each arm's distance (converted to angular data) from its mechanical origin point.

4 DI/DO conditional expressions

DI/DO conditional expressions may be used to set conditions for WAIT statements and STOPON options in MOVE statements.
Numeric constants, variables and arithmetic operators that may be used with DI/DO conditional expressions are shown below.

- Constant
  Decimal integer constant, binary integer constant, hexadecimal integer constant
- Variables
  Global integer type, global real number type, input/output type
- Operators
  Relational operators, logic operators
- Operation priority
  1. Relational operators
  2. NOT, ~
  3. AND, &
  4. OR, |, XOR

```
Examples: WAIT DI(31)=1 OR DI(34)=1
          → The program waits until either DI31 or
            DI34 turns ON.
```

Chapter 5
Multiple Robot Control

1 Overview

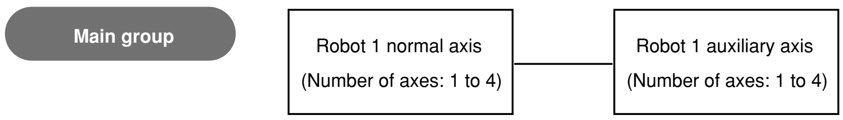
RCX340 can be used to control multiple robots (up to 4).
The multitask function also enables multiple robots to move asynchronously.
To use this function, settings for multiple robots or settings for auxiliary axes must be made in the system prior to shipment.
The following settings are possible to the axes of robots.

- Robot 1 (4 axes)
- Robot 1 (1 axis) + robot 2 (1 axis) + robot 3 (1 axis) + robot 4 (1 axis)
- Robot 1 (6 axes) + robot 2 (2 axes) (when using the YC-LINK/E option)
- Robot 1 (4 axes) + robot 2 (4 axes) (when using the YC-LINK/E option)
- Robot 1 (2 axes) + robot 2 (2 axes)
- Robot 1 (4 axes) + robot 2 (4 axes) + robot 3 (4 axes) + robot 4 (4 axes)
  (when using the YC-LINK/E option)

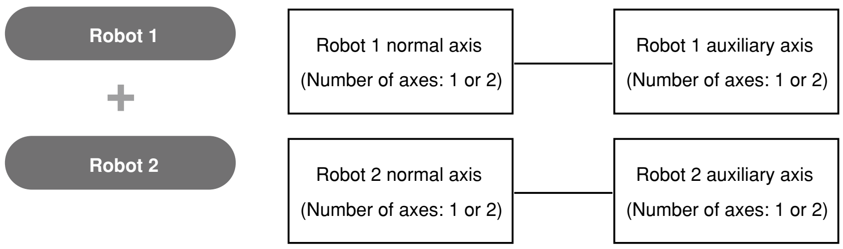
Each robot consists of normal axes and auxiliary axes.
When using one robot without auxiliary axes, the setting is made only to normal axes.

Axes configuration

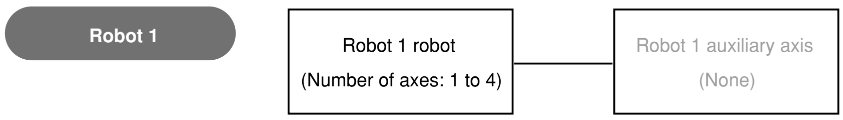
1. For robot 1



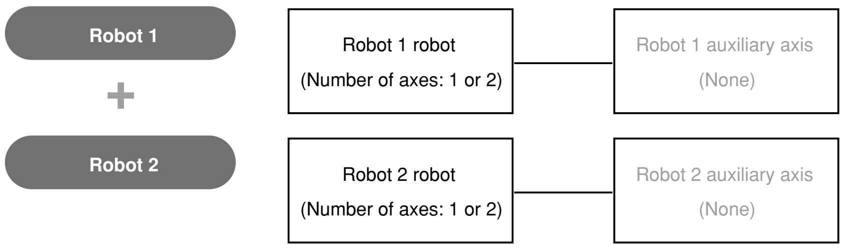
2. For robot 1 and robot 2



3. For 1 robot with no additional axes used



4. When no auxiliary axes are set to two robots



33501-R9-00

2 Command list with a robot specified

The special commands and functions for robot movements and coordinate control are common for all robots. A robot can be specified with an option of a command. Main commands are shown below.

Operator	Command name	
Robot movement	DRIVE MOVE PMOVE WAIT ARM	DRIVEI MOVEI SERVO
Coordinate control	CHANGE LEFTY SHIFT	HAND RIGHTY
Status change	ACCEL ASPEED DECEL OUTPOS TOLE	ARCHP1 AXWEIGHT ORGORD SPEED WEIGHT
Point operation	JTOXY XYTOJ	WHERE WHRXY
Parameter reference	ACCEL AXWEIGHT ORGORD TOLE	ARCHP1 DECEL OUTPOS WEIGHT
Status reference	ABSRPOS ARMSEL	ARMCND MCHREF
Torque control	DRIVE (with torque limit setting option) TORQUE TRQTIME	 TRQSTS CURTRQ

- An axis specified as an auxiliary axis cannot be moved with the MOVE, MOVEI and PMOVE commands. Use the DRIVE or DRIVEI command to move it.

Chapter 6

Multi-tasking

1 Outline

The multi-task function performs multiple processing simultaneously in a parallel manner, and can be used to create programs of higher complexity. Before using the multi-tasking function, read this section thoroughly and make sure that you fully understand its contents.
Multi-tasking allows executing two or more tasks in parallel. However, this does not mean that multiple tasks are executed simultaneously because the controller has only one CPU to execute the tasks. In multi-tasking, the CPU time is shared among multiple tasks by assigning a priority to each task so that they can be executed efficiently.

- A maximum of 16 tasks (task 1 to task 16) can be executed in one program.
- Tasks can be prioritized and executed in their priority order (higher priority tasks are executed first).
- The priority level can be set to any level between 17 and 47.
- Smaller values have higher priority, and larger values have lower priority
  (High priority: 17 ⇔ 47: low priority).

2 Task definition

A task is a set of instructions which are executed as a single sequence. As explained below, a task is defined by assigning a label to it.

1. Create one program that describes a block of the command which is to be defined as a task.
2. In the START statement of the program that will be a main task, specify the program created at step 1 above. Task Nos. are then assigned, and the program starts.

SAMPLE

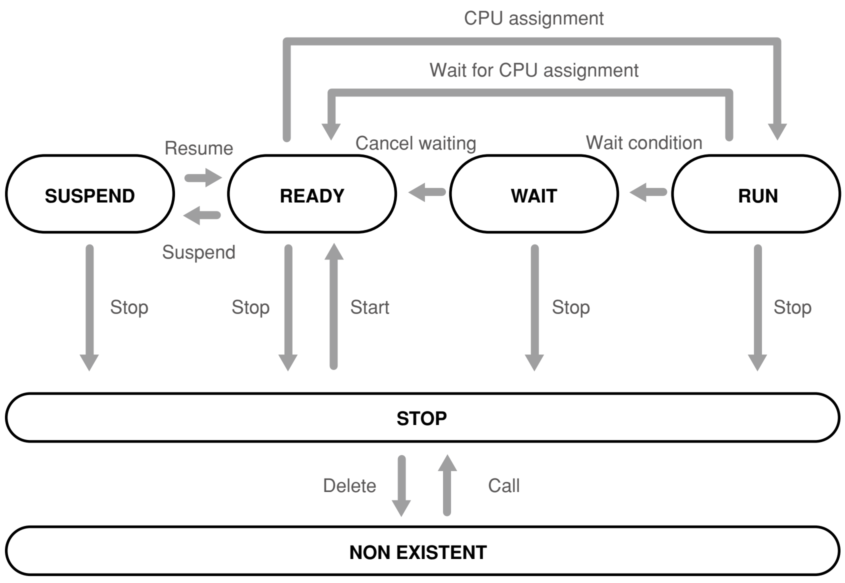
```
' MAIN TASK(TASK1)
START *IOTASK,T2 ················ *IOTASK is started as Task 2
*ST1:
MOVE P,P1,P0
   IF DI(20)= 1 THEN
      HALT
   ENDIF
GOTO *ST
HALT
Program name:SUB_PGM
*SUBPGM:
' SUB TASK(TASK2)
*IOTASK: ···························· Task 2 begins from here
   IF DI(21)=1 THEN
      DO(30)=1
   ELSE
      DO(30)=0
   ENDIF
GOTO *SUBPGM ····················· Task 2 processing ends here
EXIT TASK
```

3 Task status and transition

There are 6 types of task status.

1. **STOP status**
   A task is present but the task processing is stopped.
2. **RUN status**
   A task is present and the task processing is being executed by the CPU.
3. **READY status**
   A task is present and ready to be allocated to the CPU for task processing.
4. **WAIT status**
   A task is present and waiting for an event to begin the task processing.
5. **SUSPEND status**
   A task is present but suspended while waiting to begin the task processing.
6. **NON EXISTENT status**
   No tasks exist in the program. (The START command is used to perform a call).

Task state transition



33601-R9-00

3.1 Starting tasks

When the program is being executed in the AUTO mode, Task 1 (main task) is automatically selected and placed in a RUN status when the program begins. Therefore, the delete, forced wait, forced end commands, etc., cannot be executed for Task 1.
Other tasks (2 to 8 subtasks) will not be called simply by executing the program. The START command must be used at Task 1 in order to call, start, and place these tasks in a READY status.

MEMO

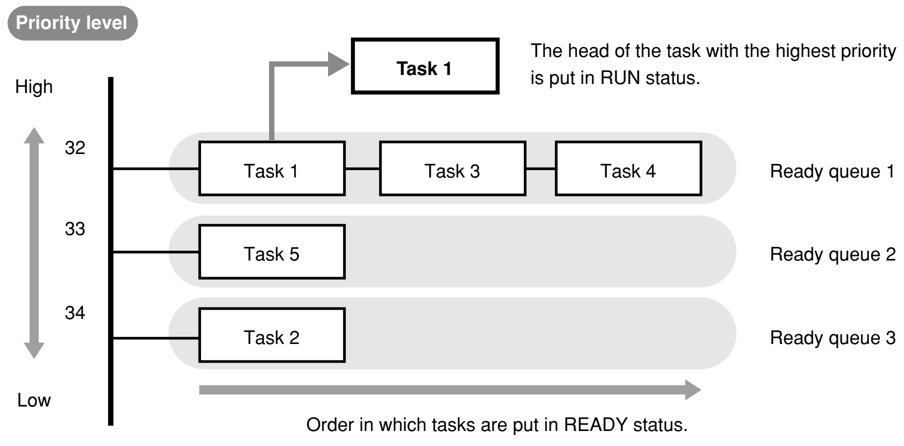
- The RESTART, SUSPEND, EXIT TASK, and CUT commands cannot be executed at Task 1.

3.2 Task scheduling

Task scheduling determines the priority to be used in allocating tasks in the READY(execution enabled) status to the CPU and executing them.
When there are two or more tasks which are put in the READY status, ready queues for CPU allocation are used to determine the priority for executing the tasks. One of these READY status tasks is then selected and executed (RUN status).

Only tasks with the same priority ranking are assigned to a given ready queue. Therefore, where several tasks with differing priority rankings exist, a corresponding number of ready queues are created. Tasks within a given ready queue are handled on a first come first serve (FCFS) basis. The task where a READY status is first established has priority. The smaller the number, the higher the task priority level.

Task scheduling


Priority level
High
Low
32
33
34
Task 1
The head of the task with the highest priority is put in RUN status.
Task 1
Task 3
Task 4
Ready queue 1
Task 5
Ready queue 2
Task 2
Ready queue 3
Order in which tasks are put in READY status.

33602-R7-00

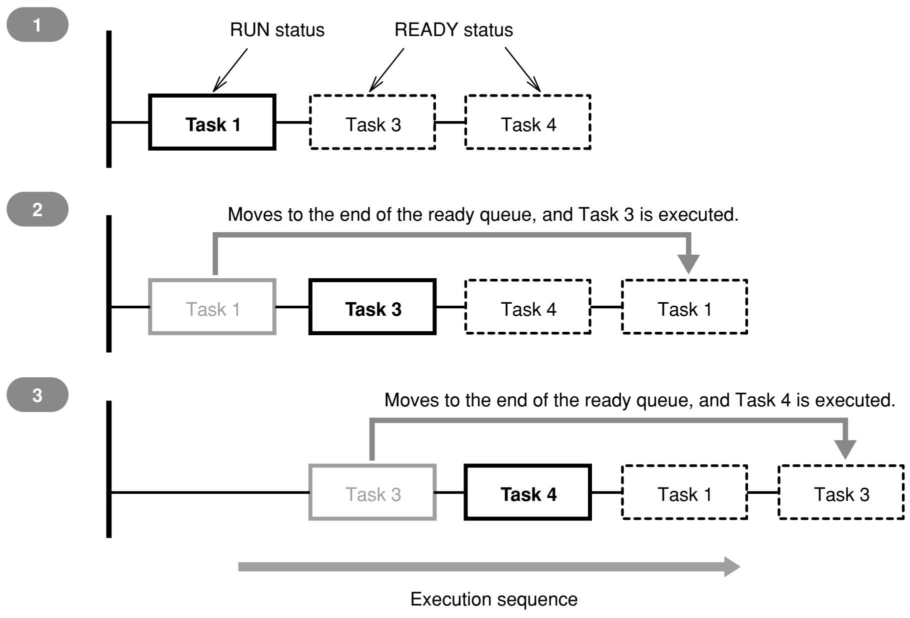
A RUN status task will be moved to the end of the ready queue if placed in a READY status by any of the following causes:

1) A WAIT status command was executed.
2) The CPU occupation time exceeds a specified time.
3) A task with a higher priority level is put in READY status.

NOTE

- When the prescribed CPU occupation time elapses, the active command is ended, and processing moves to the next task. However, if there are no other tasks of the same or higher priority (same or higher ready queue), the same task will be executed again.

Ready queue


1
RUN status
READY status
Task 1
Task 3
Task 4
2
Moves to the end of the ready queue, and Task 3 is executed.
Task 1
Task 3
Task 4
Task 1
3
Moves to the end of the ready queue, and Task 4 is executed.
Task 3
Task 4
Task 1
Task 3
Execution sequence

33603-R7-00

3.3 Condition wait in task

A task is put in the WAIT status (waiting for an event) when a command causing a wait status is executed for that task. At this time, the transition to READY status does not take place until the wait condition is canceled.

1. **When a command causing a wait status is executed, the following transition happens.**
   - Task for which a command causing a wait status is executed → WAIT status
   - Task at the head of the ready queue with higher priority → RUN status

MEMO

- For example, when a MOVE statement (a command that establishes a WAIT status) is executed, the CPU sends a "MOVE" instruction to the driver, and then waits for a "MOVE COMPLETED" reply from the driver. This is a "waiting for an event" status. In this case, a WAIT status is established at the task which executed the MOVE command, and that task is moved to the end of the ready queue. A RUN status is then established at the next task.

NOTE

- If multiple tasks are in WAIT status awaiting the same condition event, or different condition events occur simultaneously, all tasks for which the waited events occur are put in READY status.

2. **When an event waited by the task in the WAIT status occurs, the following status transition takes place by task scheduling.**
   - Task in the WAIT status for which the awaited event occurred → READY status

   However, if the task put in the READY status was at the head of the ready queue with the highest priority, the following transition takes place.
   1) Task that is currently in RUN status → READY status
   2) Task at the head of the ready queue with higher priority → RUN status

MEMO

- In the above MOVE statement example, the task is moved to the end of the ready queue. Then, when a "MOVE COMPLETED" reply is received, this task is placed in READY status.

Tasks are put in WAIT status by the following commands.

Event		Command			
Wait for axis movement to complete	Axis movement command	MOVE PMOVE	MOVEI SERVO	DRIVE WAIT ARM	DRIVEI
	Parameter command	ACCEL DECEL WEIGHT	ARCHP1 OUTPOS	ARCHP2 TOLE	AXWEIGHT ORGORD
	Robot status change command	CHANGE ASPEED	SHIFT SPEED	LEFTY	RIGHTY
Wait for time to elapse		DELAY, SET (Time should be specified.), WAIT ARM (Time should be specified.)			
Wait for condition to be met		WAIT			
Wait for data to send or to be received		SEND			
Wait for print buffer to become empty		PRINT			
Wait for key input		INPUT			

MEMO

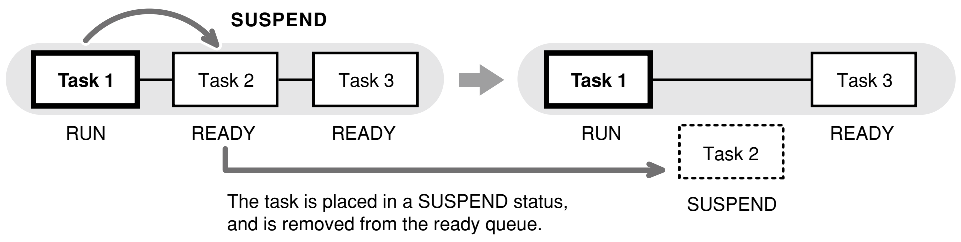
- The tasks are not put in WAIT status if the event has been established before the above commands are executed.

3.4 Suspending tasks (SUSPEND)

The SUSPEND command temporarily stops tasks other than task 1 and places them in SUSPEND status. The SUSPEND command cannot be used for task 1.
When the SUSPEND command is executed, the status transition takes place as follows.

- Task that executed the SUSPEND command → RUN status
- Specified task → SUSPEND status

Suspending tasks (SUSPEND)


SUSPEND
Task 1
Task 2
Task 3
RUN
READY
READY
Task 1
Task 3
RUN
Task 2
READY
SUSPEND
The task is placed in a SUSPEND status, and is removed from the ready queue.

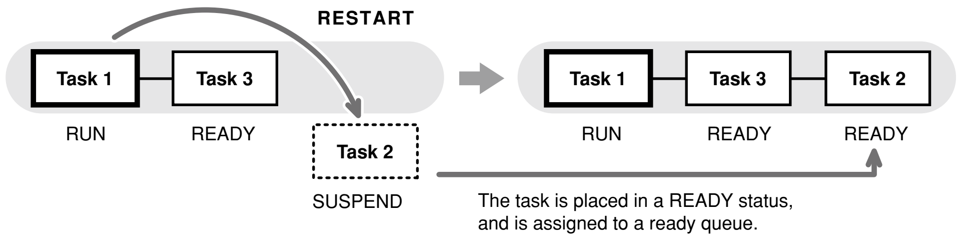
33604-R7-00

3.5 Restarting tasks (RESTART)

Tasks in the SUSPEND status can be restarted with the RESTART command. However, the RESTART command cannot be used for task 1.
When the RESTART command is executed, the status transition takes place as follows.

- Task for which the RESTART command was executed → RUN status
- Specified task → READY status

Restarting tasks (RESTART)


RESTART
Task 1
Task 3
RUN
READY
Task 2
SUSPEND
Task 1
Task 3
Task 2
RUN
READY
READY
The task is placed in a READY status, and is assigned to a ready queue.

33605-R7-00

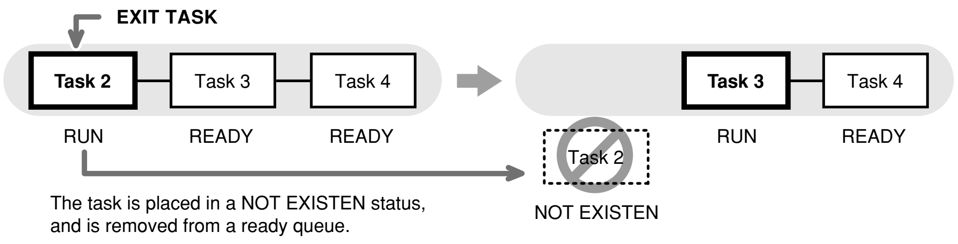
3.6 Deleting tasks

Task self-delete (EXIT TASK)

Tasks can delete themselves by using the EXIT TASK command and set to the NON EXISTEN (no task registration) status. The EXIT TASK command cannot be used for task 1.
When the EXIT TASK command is executed, the status transition takes place as follows.

■ Task that executed the EXIT TASK command	→ NON EXISTEN status
■ Task at the head of the ready queue with higher priority	→ RUN status

Task self-delete (EXIT TASK)


EXIT TASK
Task 2
Task 3
Task 4
RUN
READY
READY
Task 3
Task 4
RUN
READY
Task 2
NOT EXISTEN
The task is placed in a NOT EXISTEN status, and is removed from a ready queue.

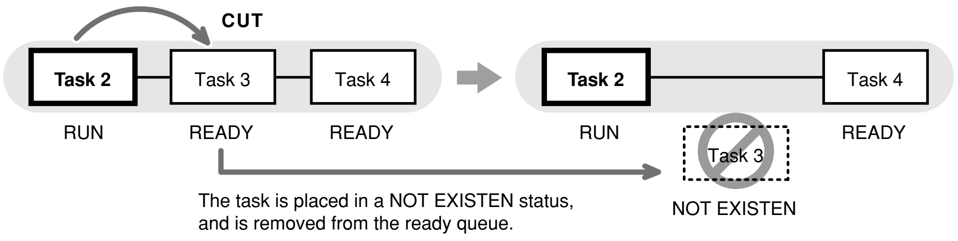
33606-R7-00

Other-task delete (CUT)

A task can also be deleted and put in the NON EXISTEN (no task registration) status by the other tasks using the CUT command. The CUT command cannot be used for task 1.
When the CUT command is executed, the status transition takes place as follows.

■ Task that executed the CUT command	→ RUN
■ Specified task	→ NON EXISTEN

Other-task delete (CUT)


CUT
Task 2
Task 3
Task 4
RUN
READY
READY
Task 2
Task 4
RUN
READY
Task 3
NOT EXISTEN
The task is placed in a NOT EXISTEN status, and is removed from the ready queue.

33607-R7-00

MEMO
- If a SUSPEND command is executed for a WAIT-status task, the commands being executed by that task are ended.
- None of these commands can be executed for Task 1.

3.7 Stopping tasks

All tasks stop if any of the following cases occurs.

1. **HALT command is executed.** (stop & reset)
   The program is reset and all tasks other than task 1 are put in the NON EXISTEN status. Task 1 is put in the STOP status.
2. **HOLD command is executed.** (temporary stop)
   All tasks are put in the STOP status. When the program is restarted, the tasks in the STOP status set to the READY or SUSPEND status.
3. **STOP key on the programming box is pressed or the interlock signal is cut off.**
   Just as in the case where the HOLD command is executed, all tasks are put in the STOP status. When the program is restarted, the tasks in the STOP status set to the READY status (or, the task is placed in a SUSPEND status after being placed in a READY status).
4. **When the emergency stop switch on the programming box is pressed or the emergency stop signal is cut off.**
   All tasks are put in STOP status. At this point, the power to the robot is shut off and the servo sets to the non-hold state.
   After the canceling emergency stop, when the program is restarted, the tasks in STOP status are set to the READY or SUSPEND status. However, a servo ON is required in order to restart the robot power supply.

MEMO

- When the program is restarted without being reset after the tasks have been stopped by a cause other than 1., then each task is processed from the status in which the task stopped. This holds true when the power to the controller is turned off and then turned on.

4 Multi-task program example

Tasks are executed in their scheduled order. An example of a multi-task program is shown below.

SAMPLE

```
' TASK1
START *ST2,T2
START *ST3,T3
*ST1:
   DO(20) = 1
   WAIT MO(20) = 1
   MOVE P,P1,P2,Z=0
   IF MO(21)=1 THEN *FIN
GOTO *ST1
*FIN:
CUT T2
HALT
' TASK2
*ST2:    ............................ Task 2 begins here.
   IF DI(20) = 1
      MO(20) = 1
      DELAY 100
   ELSE
      MO(20) = 0
   ENDIF
GOTO *ST2
EXIT TASK ........................... Ends here.
' TASK3  ............................ Task 3 begins here.
*ST3:
   IF DI(21) = 0 THEN *ST3
   IF DI(30) = 0 THEN *ST3
   IF DI(33) = 0 THEN *ST3
   MO(21) = 1
EXIT TASK ........................... Ends here.
```

5 Sharing the data

Point data, shift coordinate definition data, hand definition data, pallet definition data, all global variables and other variables are shared between all tasks.
Execution of each task can be controlled while using the same variables and data shared with the other tasks.

MEMO

- In this case, however, use sufficient caution when rewriting the variable and data because improper changes may cause trouble in the task processing.

6 Cautionary Items

A silence stop may occur if subtasks are continuously started (START command) and ended (EXIT TASK command) by a main task in an alternating manner.
This occurs for the following reason: if the main task and subtask priority levels are the same, a task transition to the main task occurs during subtask END processing, and an illegal task status then occurs when the main task attempts to start a subtask.
Therefore, in order to properly execute the program, the subtask priority level must be set higher than that of the main task. This prevents a task transition condition from occurring during execution of the EXIT TASK command.
In the sample program shown below, the priority level of task 1 (main task) is set as 32, and the priority level of task 2 is set as 31 (the lower the value, the higher the priority).

SAMPLE

```
FLAG1 = 0
*MAIN_TASK:
   IF FLAG1=0 THEN
      FLAG1 = 1
      START *TASK2,T2,31······ Task 2 (*TASK2) is started at the
                               priority level of 31.
   ENDIF
GOTO *MAIN_TASK
'==============
'     TASK2
'==============
*TASK2:
   DRIVE(1,P1)
   WAIT ARM(1)
   DRIVE(1,P2)
   WAIT ARM(1)
   FLAG1 = 0
EXIT TASK
HALT
```

Chapter 7

Robot Language Lists

How to read the robot language table

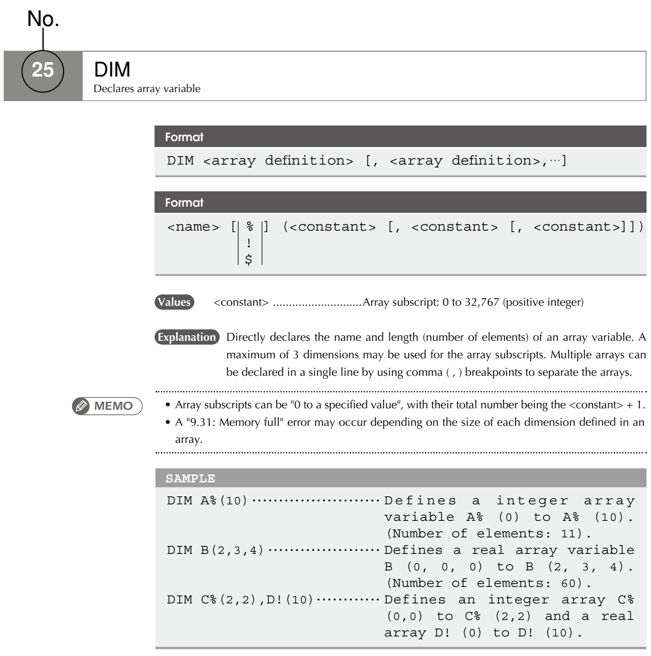
The key to reading the following robot language table is explained below.

	(1)	(2)	(3)	(4)
	No.	Function	Online	Type
DIM	25	Declares array variable	×	Command

(1) No.

Indicates the Item No. where this robot language is explained in detail.

Example of "No." column



No.

25 DIM
Declares array variable

Format
```
DIM <array definition> [, <array definition>,…]
```

Format
```
<name> [| % |] (<constant> [, <constant> [, <constant>]])
        | ! |
        | $ |
```

Values <constant> ....Array subscript: 0 to 32,767 (positive integer)

Explanation Directly declares the name and length (number of elements) of an array variable. A maximum of 3 dimensions may be used for the array subscripts. Multiple arrays can be declared in a single line by using comma (,) breakpoints to separate the arrays.

MEMO
- Array subscripts can be "0 to a specified value", with their total number being the <constant> + 1.
- A "9.31: Memory full" error may occur depending on the size of each dimension defined in an array.

SAMPLE
```
DIM A%(10) ························ Defines a integer array
                                     variable A% (0) to A% (10).
                                     (Number of elements: 11).
DIM B(2,3,4) ······················ Defines a real array variable
                                     B (0, 0, 0) to B (2, 3, 4).
                                     (Number of elements: 60).
DIM C%(2,2),D!(10) ················ Defines an integer array C%
                                     (0,0) to C% (2,2) and a real
                                     array D! (0) to D! (10).
```

(2) Function

Explains the function of the robot language.

(3) Online

If "◯" is indicated at this item, online commands can be used.

If "△" is indicated at this item, commands containing operands that cannot partially be executed by online command.

(4) Type

Indicates the robot language type as "Command" or "Function".

When a command is used as both a "Command" and "Function", this is expressed as follows: Command/Function

Command list in alphabetic order

No.	Command	Function	Online	Type
A				
1	ABS	Acquires the absolute value of a specified value.	-	Command Statements
2	ABSRPOS	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)	-	Command Statements/ Functions
3	ACCEL	Specifies/acquires the acceleration coefficient parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP1	Specifies/acquires the arch position 1 parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP2	Specifies/acquires the arch position 2 parameter of a specified robot.	○	Command Statements/ Functions
5	ARMCND	Acquires the current arm status of a specified robot.	-	Functions
6	ARMSEL	Acquires the current “hand system” setting of a specified robot.	-	Functions
7	ARMTYP	Acquires the “hand system” setting of a specified robot.	-	Functions
8	ASPEED	Specifies/acquires the AUTO movement speed of a specified robot.	○	Command Statements/ Functions
9	ATN	Acquires the arctangent of the specified value.	-	Functions
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.	-	Functions
10	AXWGHT	Specifies/acquires the axis tip weight parameter of a specified robot.	○	Command Statements/ Functions
C				
11	CALL	Calls a sub-procedure.	×	Command Statements
12	CHANGE	Switches the hand of a specified robot.	○	Command Statements
13	CHGPRI	Changes the priority ranking of a specified task.	○	Command Statements
14	CHR$	Acquires a character with the specified character code.	-	Functions
15	COS	Acquires the cosine value of a specified value.	-	Functions
16	CURTQST	Acquires the current torque against the rated torque of a specified axis.	-	Functions
17	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.	-	Functions
18	CUT	Terminates a task currently being executed or temporarily stopped.	○	Command Statements
D				
19	DATE$	Acquires the date as a "yy/mm/dd" format character string.	-	Functions
20	DECEL	Specifies/acquires the deceleration rate parameter of a specified robot.	○	Command Statements/ Functions
21	DEF FN	Defines the functions that can be used by the user.	×	Command Statements
22	DEGRAD	Converts a specified value to radians (↔RADDEG).	-	Functions
23	DELAY	Waits for the specified period (units: ms).	×	Command Statements
24	DI	Acquires the input from the parallel port.	-	Functions
25	DIM	Declares the array variable name and the number of elements.	×	Command Statements
26	DIST	Acquires the distance between 2 specified points.	-	Functions
27	DO	Outputs a specified value to the DO port.	○	Command Statements
28	DRIVE	Moves a specified axis of a specified robot to an absolute position.	○	Command Statements
28	DRIVE	(With T-option) Executes an absolute movement command for a specified axis.	○	Command Statements
29	DRIVEI	Moves a specified axis of a specified robot to a relative position.	○	Command Statements

No.	Command	Function	Online	Type
E				
30	END SELECT	Terminates the SELECT CASE statement.	×	Command Statements
31	END SUB	Terminates the sub-procedure definition.	×	Command Statements
32	ERL	Gives the line No. where an error occurred.	-	Functions
32	ERR	Gives the error code number of an error which has occurred.	-	Functions
33	EXIT FOR	Terminates the FOR to NEXT statement loop.	×	Command Statements
34	EXIT SUB	Terminates the sub-procedure defined in SUB to END.	×	Command Statements
35	EXIT TASK	Terminates its own task which is in progress.	×	Command Statements
F				
36	FOR to NEXT	Controls repetitive operations. Executes the FOR to NEXT statement repeatedly until a specified value is exceeded.	×	Command Statements
G				
37	GOSUB to RETURN	Jumps to a subroutine with the label specified by a GOSUB statement, and executes that subroutine.	×	Command Statements
38	GOTO	Unconditionally jumps to the line specified by a label.	×	Command Statements
H				
39	HALT	Stops the program and performs a reset.	×	Command Statements
40	HALTALL	Stops and resets all programs.	×	Command Statements
41	HAND	Defines the hand of a specified robot.	○	Command Statements
42	HOLD	Temporarily stops the program.	×	Command Statements
43	HOLDALL	Temporarily stops all programs.	×	Command Statements
I				
44	IF	Allows control flow to branch according to conditions.	×	Command Statements
45	INPUT	Assigns a value to a variable specified from the programming box.	○	Command Statements
46	INT	Acquires an integer for a specified value by truncating all decimal fractions.	-	Functions
J				
47	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)	-	Functions
L				
48	LEFT$	Extracts a character string comprising a specified number of digits from the left end of a specified character string.	-	Functions
49	LEFTY	Sets the hand system of a specified robot to "Left."	○	Command Statements
50	LEN	Acquires the length (number of bytes) of a specified character string.	-	Functions
51	LET	Executes a specified assignment statement.	○	Command Statements
52	LO	Outputs a specified value to the LO port to enable/disable axis movement.	○	Command Statements
53	LOCx	Specifies/acquires point data for a specified axis or shift data for a specified element.	-	Command Statements/ Functions
54	LSHIFT	Shifts a value to the left by the specified number of bits. (↔RSHIFT)	-	Functions
M				
55	MCHREF	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.	-	Functions
56	MID$	Extracts a character string of a desired length from a specified character string.	-	Functions
57	MO	Outputs a specified value to the MO port.	○	Command Statements
58	MOTOR	Controls the motor power status.	○	Command Statements
59	MOVE	Performs absolute movement of all axes of a specified robot.	○	Command Statements

No.	Command	Function	Online	Type
60	MOVEI	Performs relative movement of all axes of a specified robot.	○	Command Statements
O				
61	OFFLINE	Sets a specified communication port to the "offline" mode.	○	Command Statements
62	ON ERROR GOTO	If an error occurs during program execution, this command allows the program to jump to the error processing routine specified by the label without stopping the program, or it stops the program and displays the error message.	×	Command Statements
63	ON to GOSUB	Jumps to a subroutine with labels specified by a GOSUB statement in accordance with the conditions, and executes that subroutine.	×	Command Statements
64	ON to GOTO	Jumps to label-specified lines in accordance with the conditions.	×	Command Statements
65	ONLINE	Sets the specified communication port to the "online" mode.	○	Command Statements
66	ORD	Acquires the character code of the first character in a specified character string.	-	Functions
67	ORGORD	Specifies/acquires the axis sequence parameter for performing return-to-origin and an absolute search operation in a specified robot.	○	Command Statements/ Functions
68	ORIGIN	Performs a return-to-origin.	○	Command Statements
69	OUT	Turns ON the bits of the specified output ports and the command statement ends.	×	Command Statements
70	OUTPOS	Specifies/acquires the OUT enable position parameter of a specified robot.	○	Command Statements/ Functions
P				
71	PDEF	Defines the pallet used to execute pallet movement commands.	○	Command Statements
72	PMOVE	Executes the pallet movement command of a specified robot.	○	Command Statements
73	Pn	Defines points within a program.	○	Command Statements
74	PPNT	Creates point data specified by a pallet definition number and pallet position number.	-	Functions
75	PRINT	Displays a character string at the programming box screen.	○	Command Statements
76	PSHFRC	Specifies/acquires the pushing thrust parameter.	○	Command Statements/ Functions
77	PSHJGSP	Specifies/acquires the pushing check speed threshold parameter.	○	Command Statements/ Functions
78	PSHMTD	Specifies/acquires the pushing method parameter.	○	Command Statements/ Functions
79	PSHRSLT	Acquires the status at the end of the PUSH statement.	-	Functions
80	PSHSPD	Specifies/acquires the pushing movement speed parameter.	○	Functions
81	PSHTIME	Specifies/acquires the pushing time parameter.	○	Functions
82	PUSH	Executes a pushing operation in the axis unit.	○	Command Statements
R				
83	RADDEG	Converts a specified value to degrees. (↔DEGRAD)	-	Functions
84	REM	Expresses a comment statement.	×	Command Statements
85	RESET	Turns the bit of a specified output port OFF.	○	Command Statements
86	RESTART	Restarts another task during a temporary stop.	○	Command Statements
87	RESUME	Resumes program execution after error recovery processing.	×	Command Statements
88	RETURN	Returns the processing branching with GOSUB to the next line of GOSUB.	×	Command Statements
89	RIGHT$	Extracts a character string comprising a specified number of digits from the right end of a specified character string.	-	Functions
90	RIGHTY	Sets the hand system of a specified robot to “Right.”	○	Command Statements
91	RSHIFT	Shifts a value to the right by the specified number of bits. (↔LSHIFT)	-	Functions

No.	Command	Function	Online	Type
S				
92	SELECT CASE to END SELECT	Allows control flow to branch according to conditions.	×	Command Statements
93	SEND	Sends a file.	○	Command Statements
94	SERVO	Controls the servo ON/OFF of a specified axis or all axes of a specified robot.	○	Command Statements
95	SET	Turns the bit at the specified output port ON.	△	Command Statements
96	SHARED	Enables reference with a sub-procedure without transferring a variable.	×	Command Statements
97	SHIFT	Sets the shift coordinate for a specified robot by using the shift data specified by a shift variable.	○	Command Statements
98	SIN	Acquires the sine value for a specified value.	-	Functions
99	Sn	Defines the shift coordinates within the program.	○	Command Statements
100	SO	Outputs a specified value to the SO port.	○	Command Statements
101	SPEED	Changes the program movement speed of a specified robot.	○	Command Statements
102	SQR	Acquires the square root of a specified value.	-	Functions
103	START	Specifies the task number and priority ranking of a specified program, and starts that program.	○	Command Statements
104	STR$	Converts a specified value to a character string (↔VAL)	-	Functions
105	SUB to END SUB	Defines a sub-procedure.	×	Command Statements
106	SUSPEND	Temporarily stops another task which is being executed.	×	Command Statements
107	SWI	Switches the program being executed, then begins execution from the first line.	×	Command Statements
T				
108	TAN	Acquires the tangent value for a specified value.	-	Functions
109	TCOUNTER	Outputs count-up values at 10ms intervals starting from the point when the TCOUNTER variable is reset.	-	Functions
110	TIME$	Acquires the current time as an "hh:mm:ss" format character string.	-	Functions
111	TIMER	Acquires the current time in seconds, counting from 12:00 midnight.	-	Functions
112	TO	Outputs a specified value to the TO port.	○	Command Statements
113	TOLE	Specifies/acquires the tolerance parameter of a specified robot.	○	Command Statements/ Functions
114	TORQUE	Specifies/acquires the maximum torque command value which can be set for a specified axis of a specified robot.	○	Command Statements/ Functions
V				
115	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)	-	Functions
W				
116	WAIT	Waits until the conditions of the DI/DO conditional expression are met (with time-out).	×	Command Statements
117	WAIT ARM	Waits until the axis operation of a specified robot is completed.	×	Command Statements
118	WEIGHT	Specifies/acquires the tip weight parameter of a specified robot.	○	Command Statements/ Functions
119	WEND	Terminates the command block of the WHILE statement.	×	Command Statements
120	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).	-	Functions
121	WHILE to WEND	Controls repeated operations.	×	Command Statements
122	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).	-	Functions
X				
123	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).	-	Functions

Function Specific

Program commands

General commands

No.	Command	Function	Online	Type
25	DIM	Declares the array variable name and the number of elements.	×	Command Statements
51	LET	Executes a specified assignment statement.	○	Command Statements
84	REM	Expresses a comment statement.	×	Command Statements

Arithmetic commands

No.	Command	Function	Online	Type
1	ABS	Acquires the absolute value of a specified value.	-	Command Statements
9	ATN	Acquires the arctangent of the specified value.	-	Functions
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.	-	Functions
15	COS	Acquires the cosine value of a specified value.	-	Functions
22	DEGRAD	Converts a specified value to radians (↔RADDEG).	-	Functions
26	DIST	Acquires the distance between 2 specified points.	-	Functions
46	INT	Acquires an integer for a specified value by truncating all decimal fractions.	-	Functions
54	LSHIFT	Shifts a value to the left by the specified number of bits. (↔RSHIFT)	-	Functions
83	RADDEG	Converts a specified value to degrees. (↔DEGRAD)	-	Functions
91	RSHIFT	Shifts a value to the right by the specified number of bits. (↔LSHIFT)	-	Functions
98	SIN	Acquires the sine value for a specified value.	-	Functions
102	SQR	Acquires the square root of a specified value.	-	Functions
108	TAN	Acquires the tangent value for a specified value.	-	Functions

Date / time

No.	Command	Function	Online	Type
19	DATE $	Acquires the date as a "yy/mm/dd" format character string.	-	Functions
109	TCOUNTER	Outputs count-up values at 10ms intervals starting from the point when the TCOUNTER variable is reset.	-	Functions
110	TIME $	Acquires the current time as an "hh:mm:ss" format character string.	-	Functions
111	TIMER	Acquires the current time in seconds, counting from 12:00 midnight.	-	Functions

Character string operation

No.	Command	Function	Online	Type
14	CHR $	Acquires a character with the specified character code.	-	Functions
48	LEFT $	Extracts a character string comprising a specified number of digits from the left end of a specified character string.	-	Functions
50	LEN	Acquires the length (number of bytes) of a specified character string.	-	Functions
56	MID $	Extracts a character string of a desired length from a specified character string.	-	Functions
66	ORD	Acquires the character code of the first character in a specified character string.	-	Functions
89	RIGHT $	Extracts a character string comprising a specified number of digits from the right end of a specified character string.	-	Functions
104	STR $	Converts a specified value to a character string (↔VAL)	-	Functions
115	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)	-	Functions

Point, coordinates, shift coordinates

No.	Command	Function	Online	Type
12	CHANGE	Switches the hand of a specified robot.	○	Command Statements
41	HAND	Defines the hand of a specified robot.	○	Command Statements
47	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)	-	Functions
49	LEFTY	Sets the hand system of a specified robot to “Left.”	○	Command Statements
53	LOCx	Specifies/acquires point data for a specified axis or shift data for a specified element.	-	Command Statements/ Functions
73	Pn	Defines points within a program.	○	Command Statements
74	PPNT	Creates point data specified by a pallet definition number and pallet position number.	-	Functions
90	RIGHTY	Sets the hand system of a specified robot to “Right.”	○	Command Statements
99	Sn	Defines the shift coordinates in the program.	○	Command Statements
97	SHIFT	Sets the shift coordinate for a specified robot by using the shift data specified by a shift variable.	○	Command Statements
123	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).	-	Functions

Branching commands

No.	Command	Function	Online	Type
33	EXIT FOR	Terminates the FOR to NEXT statement loop.	×	Command Statements
36	FOR to NEXT	Controls repetitive operations. Executes the FOR to NEXT statement repeatedly until a specified value is exceeded.	×	Command Statements
37	GOSUB to RETURN	Jumps to a subroutine with the label specified by a GOSUB statement, and executes that subroutine.	×	Command Statements
38	GOTO	Unconditionally jumps to the line specified by a label.	×	Command Statements
44	IF	Allows control flow to branch according to conditions.	×	Command Statements
63	ON to GOSUB	Jumps to a subroutine with labels specified by a GOSUB statement in accordance with the conditions, and executes that subroutine.	×	Command Statements
64	ON to GOTO	Jumps to label-specified lines in accordance with the conditions.	×	Command Statements
92	SELECT CASE to END SELECT	Allows control flow to branch according to conditions.	×	Command Statements
121	WHILE to WEND	Controls repeated operations.	×	Command Statements

Error control

No.	Command	Function	Online	Type
62	ON ERROR GOTO	If an error occurs during program execution, this command allows the program to jump to the error processing routine specified by the label without stopping the program, or it stops the program and displays the error message.	×	Command Statements
87	RESUME	Resumes program execution after error recovery processing.	×	Command Statements
32	ERL	Gives the line No. where an error occurred.	-	Functions
32	ERR	Gives the error code number of an error which has occurred.	-	Functions

Program & task control

Program control

No.	Command	Function	Online	Type
11	CALL	Calls a sub-procedure.	×	Command Statements
39	HALT	Stops the program and performs a reset.	×	Command Statements
40	HALTALL	Stops all programs, resets task 1, and terminates all other tasks.	×	Command Statements
42	HOLD	Temporarily stops the program.	×	Command Statements
43	HOLDALL	Temporarily stops all programs.	×	Command Statements
107	SWI	Switches the program being executed, performs compiling, then begins execution from the first line.	×	Command Statements

Task control

No.	Command	Function	Online	Type
13	CHGPRI	Changes the priority ranking of a specified task.	-	Command Statements
18	CUT	Terminates a task currently being executed or temporarily stopped.	○	Command Statements
35	EXIT TASK	Terminates its own task which is in progress.	×	Command Statements
82	PUSH	Executes a pushing operation in the axis unit.	○	Command Statements
86	RESTART	Restarts another task during a temporary stop.	○	Command Statements
103	START	Specifies the task number and priority ranking of a specified task, and starts that task.	○	Command Statements
106	SUSPEND	Temporarily stops another task which is being executed.	×	Command Statements

Robot control

Robot operations

No.	Command	Function	Online	Type
12	CHANGE	Switches the hand of a specified robot.	○	Command Statements
28	DRIVE	Moves a specified axis of a specified robot to an absolute position.	○	Command Statements
29	DRIVEI	Moves a specified axis of a specified robot to a relative position.	○	Command Statements
41	HAND	Defines the hand of a specified robot.	○	Command Statements
49	LEFTY	Sets the hand system of a specified robot to “Left.”	○	Command Statements
58	MOTOR	Controls the motor power status.	○	Command Statements
59	MOVE	Performs absolute movement of all axes of a specified robot.	○	Command Statements
60	MOVEI	Performs relative movement of all axes of a specified robot.	○	Command Statements
68	ORIGIN	Performs a return-to-origin.	○	Command Statements
72	PMOVE	Executes the pallet movement command of a specified robot.	○	Command Statements
90	RIGHTY	Sets the hand system of a specified robot to “Right.”	○	Command Statements
94	SERVO	Controls the servo ON/OFF of a specified axis or all axes of a specified robot.	○	Command Statements

Status acquisition

No.	Command	Function	Online	Type
2	ABSRPOS	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)	-	Command Statements/ Functions
5	ARMCND	Acquires the current arm status of a specified robot.	-	Functions
6	ARMSEL	Acquires the current “hand system” setting of a specified robot.	-	Functions
7	ARMTYP	Acquires the “hand system” setting of a specified robot.	-	Functions
16	CURTQST	Acquires the current torque against the rated torque of a specified axis.	-	Functions
55	MCHREF	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.	-	Functions
79	PSHRSLT	Acquires the status at the end of the PUSH statement.	-	Functions
80	PSHSPD	Specifies/acquires the pushing movement speed parameter.	○	Command Statements/ Functions
81	PSHTIME	Specifies/acquires the pushing time parameter.	○	Command Statements/ Functions
117	WAIT ARM	Waits until the axis operation of a specified robot is completed.	×	Command Statements
120	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).	-	Functions
122	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).	-	Functions

Status change

No.	Command	Function	Online	Type
3	ACCEL	Specifies/acquires the acceleration coefficient parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP1	Specifies/acquires the arch position 1 parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP2	Specifies/acquires the arch position 2 parameter of a specified robot.	○	Command Statements/ Functions

No.	Command	Function	Online	Type
8	ASPEED	Specifies/acquires the AUTO movement speed of a specified robot.	○	Command Statements/ Functions
10	AXWGHT	Specifies/acquires the axis tip weight parameter of a specified robot.	○	Command Statements/ Functions
20	DECEL	Specifies/acquires the deceleration rate parameter of a specified robot.	○	Command Statements/ Functions
67	ORGORD	Specifies/acquires the axis sequence parameter for performing return-to-origin and an absolute search operation in a specified robot.	○	Command Statements/ Functions
70	OUTPOS	Specifies/acquires the OUT enable position parameter of a specified robot.	○	Command Statements/ Functions
71	PDEF	Defines the pallet used to execute pallet movement commands.	○	Command Statements
76	PSHFRC	Specifies/acquires the pushing thrust parameter.	○	Command Statements/ Functions
77	PSHJGSP	Specifies/acquires the pushing check speed threshold parameter.	○	Command Statements/ Functions
78	PSHMTD	Specifies/acquires the pushing method parameter.	○	Command Statements/ Functions
101	SPEED	Changes the program movement speed of a specified robot.	○	Command Statements
113	TOLE	Specifies/acquires the tolerance parameter of a specified robot.	○	Command Statements/ Functions
118	WEIGHT	Specifies/acquires the tip weight parameter of a specified robot.	○	Command Statements/ Functions

Input/output & communication control

Input/output control

No.	Command	Function	Online	Type
23	DELAY	Waits for the specified period (units: ms).	×	Command Statements
27	DO	Outputs a specified value to the DO port.	○	Command Statements
52	LO	Outputs a specified value to the LO port to enable/disable axis movement.	○	Command Statements
57	MO	Outputs a specified value to the MO port.	○	Command Statements
69	OUT	Turns ON the bits of the specified output ports and the command statement ends.	×	Command Statements
85	RESET	Turns the bit of a specified output port OFF.	○	Command Statements
95	SET	Turns the bit at the specified output port ON.	△	Command Statements
100	SO	Outputs a specified value to the SO port.	○	Command Statements
112	TO	Outputs a specified value to the TO port.	○	Command Statements
116	WAIT	Waits until the conditions of the DI/DO conditional expression are met (with time-out).	×	Command Statements

Communication control

No.	Command	Function	Online	Type
65	ONLINE	Sets the specified communication port to the "online" mode.	○	Command Statements
61	OFFLINE	Sets a specified communication port to the "offline" mode.	○	Command Statements
93	SEND	Sends a file.	○	Command Statements

Functions: in alphabetic order

No.	Function	Type	Function
A			
1	ABS	Arithmetic function	Acquires the absolute value of a specified value.
2	ABSRPOS	Arithmetic function	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)
3	ACCEL	Arithmetic function	Acquires the acceleration coefficient parameter of a specified robot.
4	ARCHP1	Arithmetic function	Acquires the arch position 1 parameter of a specified robot.
4	ARCHP2	Arithmetic function	Acquires the arch position 2 parameter of a specified robot.
5	ARMCND	Arithmetic function	Acquires the current arm status of a specified robot.
6	ARMSEL	Arithmetic function	Acquires the current “hand system” setting of a specified robot.
7	ARMTYP	Arithmetic function	Acquires the “hand system” setting of a specified robot.
8	ASPEED	Arithmetic function	Sets the automatic movement speed.
9	ATN	Arithmetic function	Acquires the arctangent of the specified value.
9	ATN2	Arithmetic function	Acquires the arctangent of the specified X-Y coordinates.
10	AXWGHT	Arithmetic function	Acquires the axis tip weight parameter of a specified robot.
C			
14	CHR$	Character string function	Acquires a character with the specified character code.
15	COS	Arithmetic function	Acquires the cosine value of a specified value.
16	CURTQST	Arithmetic function	Acquires the current torque against the rated torque of a specified axis.
17	CURTRQ	Arithmetic function	Acquires the current torque value of the specified axis of a specified robot.
D			
19	DATE$	Character string function	Acquires the date as a "yy/mm/dd" format character string.
20	DECEL	Arithmetic function	Acquires the deceleration rate parameter of a specified robot.
22	DEGRAD	Arithmetic function	Converts a specified value to radians (↔RADDEG).
26	DIST	Arithmetic function	Acquires the distance between 2 specified points.
E			
32	ERL	Arithmetic function	Gives the line No. where an error occurred.
32	ERR	Arithmetic function	Gives the error code number of an error which has occurred.
I			
46	INT	Arithmetic function	Acquires an integer for a specified value by truncating all decimal fractions.
J			
47	JTOXY	Point function	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)
L			
48	LEFT$	Character string function	Extracts a character string comprising a specified number of digits from the left end of a specified character string.
50	LEN	Arithmetic function	Acquires the length (number of bytes) of a specified character string.
53	LOCx	Point function	Specifies/acquires point data for a specified axis or shift data for a specified element.
54	LSHIFT	Arithmetic function	Shifts a value to the left by the specified number of bits. (↔RSHIFT)

No.	Function	Type	Function
M			
55	MCHREF	Arithmetic function	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.
56	MID$	Character string function	Extracts a character string of a desired length from a specified character string.
O			
66	ORD	Arithmetic function	Acquires the character code of the first character in a specified character string.
67	ORGORD	Arithmetic function	Acquires the axis sequence parameter for performing return-to-origin and an absolute search operation of a specified robot.
70	OUTPOS	Arithmetic function	Acquires the OUT enable position parameter of a specified robot.
P			
74	PPNT	Point function	Creates point data specified by a pallet definition number and pallet position number.
76	PSHFRC	Arithmetic function	Specifies/acquires a pushing thrust parameter.
77	PSHJGSP	Arithmetic function	Specifies/acquires a pushing detection speed threshold parameter.
78	PSHMTD	Arithmetic function	Specifies/acquires a pushing type parameter.
79	PSHRSLT	Arithmetic function	Acquires the status when PUSH statement ends.
80	PSHSPD	Arithmetic function	Specifies/acquires the pushing movement speed parameter.
81	PSHTIME	Arithmetic function	Acquires the status at the end of the PUSH statement.
R			
83	RADDEG	Arithmetic function	Converts a specified value to degrees. (↔DEGRAD)
89	RIGHT$	Character string function	Extracts a character string comprising a specified number of digits from the right end of a specified character string.
91	RSHIFT	Arithmetic function	Shifts a value to the right by the specified number of bits. (↔LSHIFT)
S			
98	SIN	Arithmetic function	Acquires the sine value for a specified value.
102	SQR	Arithmetic function	Acquires the square root of a specified value.
104	STR$	Character string function	Converts a specified value to a character string (↔VAL)
T			
108	TAN	Arithmetic function	Acquires the tangent value for a specified value.
109	TCOUNTER	Arithmetic function	Outputs count-up values at 10ms intervals starting from the point when the TCOUNTER variable is reset.
110	TIME$	Character string function	Acquires the current time as an "hh:mm:ss" format character string.
111	TIMER	Arithmetic function	Acquires the current time in seconds, counting from 12:00 midnight.
113	TOLE	Arithmetic function	Acquires the tolerance parameter of a specified robot.
114	TORQUE	Arithmetic function	Acquires the maximum torque command value which can be set for a specified axis of a specified robot.
V			
115	VAL	Arithmetic function	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)
W			
118	WEIGHT	Arithmetic function	Acquires the tip weight parameter of a specified robot.
120	WHERE	Point function	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).

No.	Function	Type	Function
122	WHRXY	Point function	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).
X			
123	XYTOJ	Point function	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).

Functions: operation-specific

Point related functions

No.	Function name	Function
47	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)
53	LOCx	Acquires point data for a specified axis or shift data for a specified element.
74	PPNT	Creates point data specified by a pallet definition number and pallet position number.
120	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).
122	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).
123	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).

Parameter related functions

No.	Function name	Function
2	ABSRPOS	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)
3	ACCEL	Acquires the acceleration coefficient parameter of a specified robot.
4	ARCHP1	Acquires the arch position 1 parameter of a specified robot.
4	ARCHP2	Acquires the arch position 2 parameter of a specified robot.
5	ARMCND	Acquires the current arm status of a specified robot.
6	ARMSEL	Acquires the current “hand system” setting of a specified robot.
7	ARMTYP	Acquires the “hand system” setting of a specified robot.
10	AXWGHT	Acquires the axis tip weight parameter of a specified robot.
16	CURTQST	Acquires the current torque against the rated torque of a specified axis.
17	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.
20	DECEL	Acquires the deceleration rate parameter of a specified robot.
50	LEN	Acquires the length (number of bytes) of a specified character string.
55	MCHREF	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.
66	ORD	Acquires the character code of the first character in a specified character string.
67	ORGORD	Acquires the axis sequence parameter for performing return-to-origin and an absolute search operation of a specified robot.
70	OUTPOS	Acquires the OUT enable position parameter of a specified robot.
79	PSHRSLT	Acquires the status at the end of the PUSH statement.
113	TOLE	Acquires the tolerance parameter of a specified robot.
114	TORQUE	Acquires the maximum torque command value which can be set for a specified axis of a specified robot.
118	WEIGHT	Acquires the tip weight parameter of a specified robot.

Numeric calculation related functions

No.	Function name	Function
1	ABS	Acquires the absolute value of a specified value.
9	ATN	Acquires the arctangent of the specified value.
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.
15	COS	Acquires the cosine value of a specified value.
22	DEGRAD	Converts a specified value to radians (↔RADDEG).
26	DIST	Acquires the distance between 2 specified points.
46	INT	Acquires an integer for a specified value by truncating all decimal fractions.
54	LSHIFT	Shifts a value to the left by the specified number of bits. (↔RSHIFT)
83	RADDEG	Converts a specified value to degrees. (↔DEGRAD)
91	RSHIFT	Shifts a value to the right by the specified number of bits. (↔LSHIFT)
98	SIN	Acquires the sine value for a specified value.
102	SQR	Acquires the square root of a specified value.
108	TAN	Acquires the tangent value for a specified value.
115	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)

Character string calculation related functions

No.	Function name	Function
14	CHR $	Acquires a character with the specified character code.
19	DATE $	Acquires the date as a "yy/mm/dd" format character string.
48	LEFT $	Extracts a character string comprising a specified number of digits from the left end of a specified character string.
56	MID $	Extracts a character string of a desired length from a specified character string.
89	RIGHT $	Extracts a character string comprising a specified number of digits from the right end of a specified character string.
104	STR $	Converts a specified value to a character string (↔VAL)

Parameter related functions

No.	Function name	Function
32	ERL	Gives the line No. where an error occurred.
32	ERR	Gives the error code number of an error which has occurred.
109	TCOUNTER	Outputs count-up values at 1ms intervals starting from the point when the TCOUNTER variable is reset.
110	TIME $	Acquires the current time as an "hh:mm:ss" format character string.
111	TIMER	Acquires the current time in seconds, counting from 12:00 midnight.

1 ABS

Acquires absolute values

Format

```
ABS (<expression>)
```

Explanation Returns a value specified by an <expression> as an absolute value.

SAMPLE

```
A=ABS(-326.55) ················· The absolute value of -362.54
                                 (=362.54) is assigned to
                                 variable A.
```

2 ABSRPOS

Acquires a machine reference

Format

```
ABSRPOS [<robot number>] (<axis number>)
```

Values

<robot number>.....................	1 to 4
<axis number>.......................	1 to 6

Explanation The machine reference value for a specified by the <axis number> of the robot specified by the <robot number> is acquired (unit: %). The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This function is valid only for axes where the return-to-origin method is set as "mark method". It is not valid at axes where the return-to-origin method is set as "sensor" or "stroke end".

MEMO

- At axes where return-to-origin method is set to "mark" method, absolute reset is possible when the machine reference value is in a 44 to 56% range.

SAMPLE

```
A=ABSRPOS(4)····················· The machine reference value
                                  for axis 4 of robot 1 is
                                  assigned to variable A.
```

3 ACCEL

Specifies/acquires the acceleration coefficient parameter

Format

```
1. ACCEL [<robot number>] <expression>
2. ACCEL [<robot number>] (<axis number>)=<expression>
```

Values	
<robot number>	1 to 4
<axis number>	1 to 6
<expression>	1 to 100 (units: %)

Explanation Directly changes the acceleration coefficient parameter of the robot axis specified by the <robot number> to the value specified by the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
In format 1, the change occurs at all axes specified with a specified robot.
In format 2, the change occurs at the axis specified in <axis number>.

MEMO

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and command execution is stopped.
- Changes the value which has been set at an operation terminal such as a pendant box. Program declared values have priority.

Functions

Format

```
ACCEL [<robot number>] (<axis number>)
```

Values	
<robot number>	1 to 4
<axis number>	1 to 6

Explanation The acceleration parameter value is acquired for the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=50
ACCEL A  ························ The acceleration coefficient of
                                   all axes of robot 1 becomes 50%.
ACCEL(3)=100 ···················· Only axis 3 of robot 1 becomes 100%.
' CYCLE WITH INCREASING ACCELERATION
FOR A=10  TO 100 STEP 10··· The acceleration coefficient
                                   parameter is increased from
                                   10% to 100% in 10% increments.
   ACCEL A
   MOVE P,P0
   MOVE P,P1
NEXT A
A=ACCEL(3) ······················ The acceleration coefficient
                                   parameter of axis 3 of robot 1
                                   is assigned to variable A.
HALT "END TEST"
```

4 ARCHP1 / ARCHP2

Specifies/acquires the acceleration coefficient parameter

Format

```
1. ARCHP1 [<robot number>] <expression>
2. ARCHP1 [<robot number>] (<axis number>)=<expression>
```

Format

```
1. ARCHP2 [<robot number>] <expression>
2. ARCHP2 [<robot number>] (<axis number>)=<expression>
```

Values

<robot number>.....................	1 to 4
<axis number>.......................	1 to 6
<expression>..........................	1 to 6144000 (Unit: pulses)

Explanation ARCHP1 corresponds to the arch position parameter; ARCHP2 corresponds to the arch position 2 parameter, respectively. Changes the parameter's arch position to the value indicated in the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
In format 1, the change occurs at all axes specified with a specified robot.
In format 2, the change occurs at the arch position parameter for the axis specified in <axis number> to the value specified in <expression>.

MEMO

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and command execution is stopped.

Functions

Format

```
ARCHP1 [<robot number>] (<axis number>)
```

Format

```
ARCHP2 [<robot number>] (<axis number>)
```

Values

<robot number>.....................	1 to 4
<axis number>.......................	1 to 6

Explanation ARCHP1 corresponds to the arch position parameter; ARCHP2 corresponds to the arch position 2 parameter, respectively.
Acquires the arch position parameter value of the axis specified at <axis number>.
The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
DIM SAV (3)
DIM SAV2 (3)
GOSUB *SAVE_ARCH
FOR A=1000  TO 10000  STEP 1000
   GOSUB *CHANGE_ARCH
   MOVE P,P0,A3=0
   DO3(0)=1······················ Chuck CLOSE
   MOVE P,P1,A3=0
   DO3(0)=0······················ Chuck OPEN
NEXT A
GOSUB *RESTORE_ARCH
HALT
*CHANGE_ARCH:
FOR B=1 TO 4
   ARCHP1 (B) =A
   ARCHP2 (B) =A
NEXT B
RETURN
*SAVE_ARCH:
FOR B=1 TO 4
   SAV (B-1) =ARCHP1 (B) ······ The  arch  position  parameters
                                ARCHP1(1) to (4) are assigned to
                                array variables SAV(0) to (3).
   SAV2 (B-1) =ARCHP2 (B)
NEXT B
RETURN
*RESTORE_ARCH:
FOR B=1 TO 4
   ARCHP1 (B) =SAV (B-1)
   ARCHP2 (B) =SAV2 (B-1)
NEXT B
RETURN
```

5 ARMCND

Arm status acquisition

Format

```
ARMCND [<robot number>]
```

Values <robot number>......................1 to 4

Explanation This function acquires the current arm status of the SCARA robot. The robot to acquire an arm status is specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

The arm status is "2" for a left-handed system and "1" for a right-handed system.

This function is enabled only when a SCARA robot is used.

SAMPLE

```
A=ARMCND ························· The current arm status
                                   of robot 1 is assigned to
                                   variable A.
IF A=0 THEN ······················ Right-handed system status.
 MOVE P, P100, Z=0
ELSE     ························· Left-handed system status.
 MOVE P, P200, Z=0
ENDIF
```

6 ARMSEL

Sets/acquires the current hand system selection.

Format

```
ARMSEL [<robot number>] <expression>
```

Values <robot number>.....................1 to 4
<expression>..........................1: right hand system; 2: left hand system

Explanation This function sets the current hand system selection of the SCARA robot. A robot to set a hand system is specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This function is enabled only when a SCARA robot is used.

SAMPLE

```
ARMSEL[2] 2 ···················· Sets the left-handed system
                                 for the hand system selection
                                 of the robot 2.
```

Functions

Format

```
ARMSEL [<robot number>]
```

Values <robot number>.....................1 to 4

Explanation This function acquires the hand system currently selected for the SCARA robot. The robot to acquire a hand system is specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
The arm type is "2" for a left-handed system, and "1" for a right-handed system. This function is enabled only when a SCARA robot is used.

SAMPLE

```
A=ARMSEL ························ The arm type value of robot 1
                                 is assigned.
IF A=1 THEN ····················· The arm type is a right-handed system.
 MOVE P,P100,Z=0
ELSE     ························ The arm type is a left-handed system.
 MOVE P,P200,Z=0
ENDIF
```

7 ARMTYP

Sets/acquires the hand system selection during program reset.

Format

```
ARMTYP [<robot number>] <expression>
```

Values

<robot number>	1 to 4
<expression>	1: right hand system; 2: left hand system

Explanation This function sets the hand system at program reset of the SCARA robot. A robot to set a hand system selection is specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This function is enabled only when a SCARA robot is used.

SAMPLE

```
ARMTYP[2] 2 ····················· Sets the left-handed system
                                  for the hand system of the
                                  robot 2.
```

Functions

Format

```
ARMTYP [<robot number>]
```

Values

<robot number>	1 to 4

Explanation This function provides the hand system at program reset of the SCARA robot. The robot to acquire a hand system is specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
The arm type is "2" for a left-handed system, and "1" for a right-handed system. This function is enabled only when a SCARA robot is used.

SAMPLE

```
A=ARMTYP ························ The arm type value of robot 1
                                  is assigned.
IF A=1 THEN ····················· The arm type is a right-handed system.
 MOVE P,P100,Z=0
ELSE      ······················· The arm type is a left-handed system.
 MOVE P,P200,Z=0
ENDIF
HALTALL  ························ Program reset
```

8 ASPEED

Sets/acquires the AUTO movement speed of a specified robot.

Format

```
ASPEED [<robot number>] <expression>
```

Values <robot number>.....................1 to 4
<expression>..........................1 to 100 (units: %)

NOTE

- Automatic movement speed
  Specified by programming box operation or by the ASPEED command.
- Program movement speed
  Specified by SPEED commands or MOVE, DRIVE speed settings.

Explanation Directly changes the automatic movement speed of the robot specified by the <robot number> to the value indicated in the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This speed change applies to all the robot axes and auxiliary axes. The operation speed is determined by the product of the automatic movement speed (specified by programming box operation and by the ASPEED command), and the program movement speed (specified by SPEED command, etc.).
Operation speed = automatic movement speed x program movement speed.

Example:

Automatic movement speed	80%
Program movement speed	50%

Movement speed = 40% (80% × 50%)

Functions

Format

```
ASPEED [<robot number>]
```

Values <robot number>.....................1 to 4

Explanation Acquire the automatic movement speed value of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
SPEED 70
ASPEED 100
MOVE P,P0 ························Movement from the current
                                  position to P0 occurs at 70%
                                  speed (=100 * 70).
ASPEED 50
MOVE P,P1 ························Movement from the current
                                  position to P1 occurs at 35%
                                  speed (=50 * 70).
MOVE P,P2,S=10····················Movement from the current
                                  position to P2 occurs at 5%
                                  speed (=50 * 10).
HALT
```

Related commands	SPEED

9 ATN / ATN2

Acquires the arctangent of the specified value

Format

```
ATN (<expression>)
```

Format

```
ATN2 (<expression 1>) (<expression 2>)
```

Explanation

ATN:	Acquires the arctangent values of the specified <expression> values. The acquired values are radians within the following range: -π / 2 to +π / 2
ATN2:	Acquires the arctangent values of the specified <expression 1> and <expression 2> X-Y coordinates. The acquired values are radians within the following range: -π to +π

SAMPLE

```
A(0)=A*ATN(Y/X) ··············· The product of the expression
                                (Y/X) arctangent value and
                                variable A is assigned to
                                array A (0).
A(0)=ATN(0.5) ················· The 0.5 arctangent value is
                                assigned to array A (0).
A(0)=ATN2(B,C)-D ·············· The difference between the X-Y
                                coordinates (B,C) arctangent
                                value and variable D is
                                assigned to array A (0).
A(1)=RADDEG(ATN2(B,C)) ········ The X-Y coordinates (B,C)
                                arctangent value is converted
                                to degrees, and is then
                                assigned to array A (1).
```

Related commands	COS, DEGRAD, RADDEG, SIN, TAN

10 AXWGHT

Sets/acquires the axis tip weight

Format

```
AXWGHT [<robot number>] (<axis number>)=<expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	Varies according to the specified robot.

Explanation Directly changes the axis tip weight parameter for the axis specified by the <axis number> among the robot axes specified by the <robot number> to the <expression> value. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This statement is valid in systems with "MULTI" axes and auxiliary axes (the robot type and auxiliary axes are factory set prior to shipment).

Functions

Format

```
AXWGHT [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the axis tip weight parameter value for the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This statement is valid in systems with "MULTI" axes and auxiliary axes.

SAMPLE

```
A=5
B=0
C=AXWGHT(1) ·····················Axis tip weight value is
                                 acquired (the current value
                                 is saved to variable C).
AXWGHT(1)=A
DRIVE(1,P0)
AXWGHT(1)=B
DRIVE(1,P1)
AXWGHT(1)=C ·····················The axis tip weight value is set again.
HALT
```

Related commands	WEIGHT

11 CALL

Calls a sub-procedure

NOTE

- When a value is passed on to a sub-procedure, the original value of the actual argument will not be changed even if it is changed in the sub-procedure.
- When a reference is passed on to a sub-procedure, the original value of the actual argument will also be changed if it is changed in the sub-procedure.
- For details, see Chapter 3 "8 Value Pass-Along & Reference Pass-Along".

Format

```
CALL <label> [(<actual argument> [, <actual argument>…])]
```

Explanation This statement calls up sub-procedures defined by the SUB to END SUB statements. The <label> specifies the same name as that defined by the SUB statement.

1. When a constant or expression is specified as an actual argument, its value is passed on to the sub-procedure.
2. When a variable or array element is specified as an actual argument, its value is passed on to the sub-procedure. It will be passed on as a reference if "REF" is added at the head of the actual argument.
3. When an entire array (array name followed by parentheses) is specified as an actual argument, it is passed along as a reference.

MEMO

- CALL statements can be used up to 120 times in succession. Note that this number is reduced if commands which use stacks such as an FOR statement or GOSUB statement are used, or depending on the use status of identifiers.
- Always use the END SUB or EXT SIB statement to end a sub-procedure which has been called with the CALL statement. If another statement such as GOTO is used to jump out of the sub-routine, a "5.12: Stack overflow" error, etc., may occur.

SAMPLE 1

```
X%=4
Y%=5
CALL *COMPARE ( REF X%, REF Y% )
HALT
' SUB ROUTINE:  COMPARE
SUB *COMPARE ( A%,  B% )
   IF A% < B% THEN
      TEMP%=A%
      A%=B%
      B%=TEMP%
   ENDIF
END SUB
```

SAMPLE 2

```
I = 1
CALL  *TEST( I )
HALT
' SUB ROUTINE:  TEST
SUB *TEST
   X = X + 1
   IF X < 15 THEN
      CALL *TEST( X )
   ENDIF
END SUB
```

Related commands	SUB, END SUB, CALL, EXIT SUB, SHARED

12 CHANGE

Switches the hand

Format

```
CHANGE [<robot number>] | Hn  |
                        | OFF |
```

Values	
<robot number>	1 to 4
n: hand No.	0 to 31

Explanation CHANGE is used to switch the robot hand specified by the <robot number>. If OFF is specified, the hand setting is not enabled. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

Before hand switching can occur, the hands must be defined at the HAND statement. For details, see section "41 HAND". If hand data defined with another robot specified is specified, "Hand data mismatch error" occurs.

SAMPLE

```
HAND H1=      0   150.0     0.0
HAND H2=  -5000   20.00     0.0
P1=150.00 300.00 0.00 0.00 0.00 0.00
CHANGE H2 ························ Changes to hand 2.
MOVE P,P1 ························ Moves the hand 2 tip to P1 (1).
CHANGE H1 ························ Changes to hand 1.
MOVE P,P1 ························ Moves the hand 1 tip to P1 (2).
HALT
```

13 CHGPRI

Changes the priority ranking of a specified task

Format

```
CHGPRI | Tn                        | ,p
       | “< “<program name>” >”    |
       | PGm                       |
```

Values

m: Program number	0 to 99
n: Task No	1 to 16
p: Task priority ranking	1 to 64

Explanation Directly changes the priority ranking of the specified task ("n") to "p".
The smaller the priority number, the higher the priority (high priority: 1 ⇔ low priority: 64).
When a READY status occurs at a task with higher priority, all tasks with lower priority also remain in a READY status.

SAMPLE

```
START <SUB_PGM>,T2,33
*ST:
   MOVE P,P0,P1
   IF DI(20) = 1 THEN
      CHGPRI T2,32
   ELSE
      CHGPRI T2,33
   ENDIF
GOTO *ST
HALT
Program name:SUB_PGM
*SUBPGM:
’ SUBTASK ROUTINE
*SUBTASK:
   IF LOC3(WHERE) > 10000 THEN
      DO(20) = 1
      GOTO *SUBPGM
   ENDIF
   DO(20) = 0
GOTO *SUBPGM
EXIT TASK
```

Related commands	CUT, EXIT TASK, RESTART, SUSPEND, START

14 CHR$

Acquires a character with the specified character code

Format

```
CHR$ (<expression>)
```

Values <expression>..........................0 to 255

Explanation Acquires a character with the specified character code. An error occurs if the <expression> value is outside the 0 to 255 range.

SAMPLE

```
A$=CHR$(65) ····················· "A" is assigned to A$.
```

Related commands	ORD

15 COS

Acquires the cosine value of a specified value

Format

```
COS (<expression>)
```

Values <expression>..........................Angle (units: radians)

Explanation Acquires a cosine value for the <expression> value.

SAMPLE

```
A(0)=B*COS(C) ··················· The product of the C42 variable's
                                  cosine value and variable B is
                                  assigned to array A (0).
A(1)=COS(DEGRAD(20)) ········ The 20.0 ° cosine value is
                                  assigned to array A (1).
```

Related commands	ATN, DEGRAD, RADDEG, SIN, TAN

16 CURTQST

Acquires the current torque against the rated torque of a specified axis

Format

```
CURTQST [<robot number>](<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the current torque value (-1000 to 1000) against the rated torque of the axis specified by the <axis number> of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A = CURTQST(3) ················The current torque value
                               against the rated torque
                               of the axis 3 of robot 1 is
                               assigned to variable A.
```

17 CURTRQ

Acquires the current torque value of the specified axis

Format

```
CURTRQ [<robot number>] (<expression>)
```

Values

<robot number>	1 to 4
<expression>	1 to 6

Explanation Acquires the current torque value (-100 to 100) of the axis specified by the <expression> among the robot axes specified by the <robot number> is acquired. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

The current torque value is expressed as a percentage of the maximum torque command value. Plus/minus signs indicate the direction.

NOTE

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and command execution is stopped.

SAMPLE

```
A = CURTRQ(3) ··················· The current torque value
                                  of the axis 3 of robot 1 is
                                  assigned to variable A.
```

18 CUT

Terminates another task which is currently being executed

Format

```
CUT | Tn                       |
    | "< "<program name>" >"   |
    | PGm                      |
```

Values

m: Program number	0 to 99
n: Task No	1 to 16

Explanation Directly terminates another task which is currently being executed or which is temporarily stopped. A task can be specified by the name or the number of a program in execution.
This statement cannot terminate its own task.

MEMO

- If a task (program) not active is specified for the execution, an error occurs.

SAMPLE

```
' TASK1 ROUTINE
*ST:
   MO(20) = 0
   START <SUB_PGM>,T2
   MOVE P,P0
   MOVE P,P1
   WAIT MO(20) = 1
   CUT T2
GOTO *ST
HALT
Program name:SUB_PGM
*SUBPGM:
' TASK2 ROUTINE
*SUBTASK2:
   P100=JTOXY(WHERE)
   IF LOC3(P100) >= 100.0 THEN
      MO(20) = 1
   ELSE
      DELAY 100
   ENDIF
GOTO *SUBPGM
EXIT TASK
```

Related commands	EXIT TASK, CUT, RESTART, START, SUSPEND

19 DATE$

Acquires the date

Format

```
DATE$
```

Explanation Acquires the date as a "yyyy/mm/dd" format character string.
"yyyy" indicates the year, "mm" indicates the month, and "dd" indicates the day.
Date setting is performed from an operation terminal such as a pendant.

SAMPLE

```
A$=DATE$
PRINT DATE$
HALT
```

Related commands	TIME$

20 DECEL

Specifies/acquires the deceleration rate parameter

Format

```
1. DECEL [<robot number>] <expression>
2. DECEL [<robot number>] (<axis number>)=<expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	1 to 100 (units: %)

Explanation Change the deceleration rate parameter of a robot axis specified by the <robot number> to the <expression> value. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
In format 1, the change occurs at all axes of a specified robot.
In format 2, the change occurs at the axis specified in <axis number>.

MEMO

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and command execution is stopped.
- Command statement ACCEL can be used to change the acceleration parameter.

Functions

Format

```
DECEL [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the deceleration rate parameter value for the axis specified by the <axis number> among the robot axis specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A =50
DECEL A
DECEL(3)=100
' CYCLE WITH INCREASING DECELERATION
FOR A =10 TO 100 STEP 10
   DECEL A
   MOVE P ,P0
   MOVE P ,P1
NEXT A
A=DECEL(3)······················The deceleration rate
                                parameter for the axis 3
                                of robot 1 is assigned to
                                variable A.
HALT "END TEST "
```

21 DEF FN

Defines functions which can be used by the user

Format

```
DEF FN <name> [|%|] [(<dummy argument>, [<dummy argument>···])] = <function definition expression>
               |! |
               |$ |
```

Values	<name>	Function name. Max. of 16 chars., including "FN".
	<dummy argument>	Numeric or character string variable.

Explanation Defines the functions which can be used by the user. Defined functions are called in the FN <name> (<variable>) format.

MEMO

- The <dummy argument> names are the same as the variable names used in the <function definition expression>. The names of these variables are valid only when the <function definition expression> is evaluated. There may be other variables with the same name in the program.
- When calling a function that uses a <dummy argument>, specify the constant, variable, or expression type which is the same as the <dummy argument> type. The <dummy argument> can be omitted.
- If a variable used in the <function definition expression> is not included in the <dummy argument> list, the current value of that particular variable is used for the calculation.
- A space must be entered between "DEF" and "FN". If no space is entered, DEFFN will be handled as a variable.
- The DEF FN statement cannot be used in sub-procedures.
- Definition by the DEF FN statement must be declared before statements which use functions.

SAMPLE

```
DEF FNPAI=3.141592
DEF FNASIN(X)=ATN(X/SQR(-X^2+1))
          ························ Both the <dummy argument>
                                   and <function definition
                                   expression> use "X".
   .
   .
A=FNASIN(B)*10················· "X" is not required for calling.
```

22 DEGRAD

Angle conversion (angle → radian)

Format

```
DEGRAD (<expression>)
```

Values <expression>..........................Angle (units: degrees)

Explanation The <expression> value is converted to radians.
To convert radians to degrees, use RADDEG.

SAMPLE

```
A=COS(DEGRAD(30)) ············· A 30° cosine value is assigned
                                to variable A.
```

Related commands	ATN, COS, RADDEG, SIN, TAN

23 DELAY

Program execution waits for a specified period of time

Format

```
DELAY <expression>
```

Values <expression>..........................1 to 3600000 (units: ms)

Explanation A "program wait" status is established for the period of time specified by the <expression>. The minimum wait period is 1ms.

SAMPLE

```
DELAY 3500 ························ 3,500ms (3.5 secs) wait
DELAY A*10
```

24 DI

Acquires the input status from the parallel port

Format

```
1. [LET] <expression> = DIm([b,···,b])
2. [LET] <expression> = DI(mb,···,mb)
```

Values

m	Port No.: 0 to 7, 10 to 17, 20 to 27
b	Bit definition: 0 to 7

Explanation Indicates the parallel input signal status.
If multiple bits are specified, they are expressed from the left in descending order (large to small).
Enter "0" if no input port exists.
If the [b,…,b] data is omitted, all 8 bits are processed.

SAMPLE

```
A%=DI2() ························ The input status from DI (27)
                                  to DI (20) is assigned to
                                  variable A%.
A%=DI5(7,4,0) ·················· The DI (57), DI (54), DI (50)
                                  input status is assigned to
                                  variable A% (when all the above
                                  signals are "1" (ON), A% = 7).
A%=DI(37,25,20) ················ The DI (37), DI (25), DI (20)
                                  input status is assigned to
                                  variable A% (when all the
                                  above signals except DI (20)
                                  are "1" (ON), A% = 6).
```

Reference For details, refer to Chapter 3 "9.3 Parallel input variable".

25 DIM

Declares array variable

Format

```
DIM <array definition> [, <array definition>,…]
```

Array definition

```
<name> [| % |] (<constant> [, <constant> [, <constant>]])
        | ! |
        | $ |
```

Values <constant> ....Array subscript: 0 to 32,767 (positive integer)

Explanation Directly declares the name and length (number of elements) of an array variable. A maximum of 3 dimensions may be used for the array subscripts. Multiple arrays can be declared in a single line by using comma (,) breakpoints to separate the arrays.

MEMO

- Array subscripts can be "0 to a specified value", with their total number being the <constant> + 1.
- A "9.31: Memory full" error may occur depending on the size of each dimension defined in an array.

SAMPLE

```
DIM A%(10) ························ Defines a integer array
                                    variable A% (0) to A% (10).
                                    (Number of elements: 11).
DIM B(2,3,4) ······················ Defines a real array variable
                                    B (0, 0, 0) to B (2, 3, 4).
                                    (Number of elements: 60).
DIM C%(2,2),D!(10)·········· Defines an integer array C%
                                    (0,0) to C% (2,2) and a real
                                    array D! (0) to D! (10).
```

26 DIST

Acquires the distance between 2 specified points

Format

```
DIST (<point expression 1>,<point expression 2>)
```

Values

<point expression 1>.............	Cartesian coordinate system point
<point expression 2>.............	Cartesian coordinate system point

Explanation Acquires the distance (X,Y,Z)between the 2 points specified by <point expression 1> and <point expression 2>. An error occurs if the 2 points specified by each <point expression> do not have a Cartesian coordinates.

SAMPLE

```
A=DIST(P0,P1) ··················· The distance between P0 and P1
                                  is assigned to variable A.
```

27 DO

Outputs to parallel port

Format

```
1. [LET]  DOm ([b,···,b])  = <expression>
2. [LET]  DO (mb,···,mb)   = <expression>
```

Values m: Port No. ....2 to 7, 10 to 17, 20 to 27

b: Bit definition ....0 to 7

The output value is the lower left-side bit of the integer-converted <expression> value.

Explanation Directly outputs the specified value to the DO port.

If multiple bits are specified, they are expressed from the left in descending order (large to small).

No output will occur if a nonexistent DO port is specified.

If the [b,…,b] data is omitted, all 8 bits are processed.

Outputs are not possible to DO0() and DO1(). These ports are for referencing only.

SAMPLE

```
DO2() = &B10111000 ··········· DO (27, 25, 24, 23) are turned
                               ON, and DO (26, 22, 21, 20)
                               are turned OFF.
DO2(6,5,1) = &B010 ··········· DO (25) are turned ON, and DO
                               (26, 21) are turned OFF.
DO3() = 15 ··················· DO (33, 32, 31, 30) are turned
                               ON, and DO (37, 36, 35, 34)
                               are turned OFF.
DO(37,35,27,20) = A ·········· The contents of the 4 lower
                               bits acquired when variable
                               A is converted to an integer
                               are output to DO (37, 35, 27,
                               20) respectively.
```

Related commands	RESET, SET

28 DRIVE

Executes absolute movement of specified axes

Format

```
DRIVE [<robot number>] (<axis number>, <expression>)
[,(<axis number>, <expression>)...] [, option]
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	Motor position (mm, degrees, pulses) or point expression

Explanation Executes absolute movement command for specified axis of the robot specified by the <robot number>.
The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This command is also used in the same way for the auxiliary axes.

- Movement type: PTP movement of specified axis.
- Point setting method: By direct numeric value input and point definition.
- Options: Speed setting, STOPON conditions setting, XY setting.movement direction setting. Multiple options can be specified.

Movement type

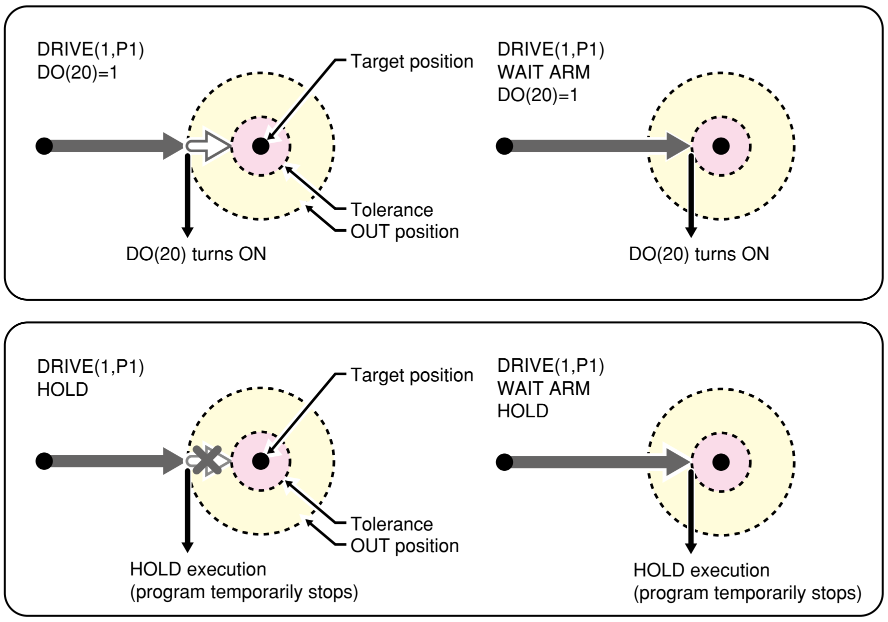
● PTP (Point to Point) movement of specified axis:

PTP movement begins after positioning of all axes specified at <axis number> is complete (within the tolerance range), and the command terminates when the specified axes enter the OUT position range. When two or more axes are specified, they will reach their target positions simultaneously.
If the next command following the DRIVE command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position tolerance range.
Example:

Signal output (DO, etc.)	Signal is output when axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when axis enters the OUT position range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	All programs in execution stop when axis enters the OUT position range, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops when axis enters the OUT position range. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop when axis enters the OUT position range. Therefore, the movement also stops.
WAIT	WAIT command is executed when axis enters the OUT position range.

The WAIT ARM statement is used to execute the next command after the axis enters the tolerance range.

DRIVE command



33819-R7-00

SAMPLE

```
DRIVE(1,P0)··················Axis 1 moves from its current
                             position to the position
                             specified by P0.
```

Point data setting types

● Direct numeric value input

The motor position is specified directly in <expression>.

If the motor position's numeric value is an integer, this is interpreted as a "pulse" units. If the motor position's numeric value is a real number, this is interpreted as a "mm/degrees" units, and each axis will move from the 0-pulse position to a pulse-converted position.

However, when using the optional XY setting, movement occurs from the coordinate origin position.

SAMPLE

```
DRIVE(1,10000) ·············Axis 1 of robot 1 moves from
                             its current position to the
                             10000 pulses position.
DRIVE[2](2,90.00) ·········Axis 2 of robot 2 moves from
                             its current position to a
                             position which is 90 ° in
                             the plus-direction from the
                             0-pulse position.
```

- **Point definition**

Point data is specified in <expressions>. The axis data specified by the <axis number> is used. If the point expression is in "mm/degrees" units, movement for each axis occurs from the 0-pulse position to the pulse-converted position.
However, when using the optional XY setting, movement occurs from the coordinate origin position.

NOTE

- If point data is specified with both integers and real numbers in the same statement, all values are handled in "mm/degrees" units.

SAMPLE

```
DRIVE(1,P1)  ················· Axis 1 of robot 1 moves from its current
                               position to the position specified by P1.
DRIVE(4,P90) ················· Axis 4 moves from its current position to the
                               position specified by P90 (deg) relative to the 0
                               pulse position. (When axis 4 is a rotating axis.)
```

Option types

- **Speed setting**

Format

```
1.  SPEED =<expression>
2.  S =<expression>
```

NOTE

- This defines the maximum speed, and does not guarantee that all movement will occur at specified speed.

Values <expression>..........................1 to 100 (units: %)

Explanation The program's movement speed is specified as an <expression>.
The actual speed is determined as shown below.

- Robot's max. speed (mm/sec, or deg/sec) × automatic movement speed (%) × program movement speed (%).

This option is enabled only for the specified DRIVE statement.

SAMPLE

```
DRIVE[2](1,10000),S=10 ··· Axis 1 of robot 2 moves from its current
                           position to the 10000 pulses position at
                           10% of the automatic movement speed.
```

Format

```
1.  DSPEED =<expression>
2.  DS =<expression>
```

NOTE

- SPEED option and DSPEED option cannot be used together

Values <expression>..........................0.01 to 100.00 (units: %)

Explanation The axis movement speed is specified in <expression>.
The actual speed is determined as shown below.

- Robot's max. speed (mm/sec, or deg/sec) × axis movement speed (%).

This option is enabled only for the specified DRIVE statement.

- Movement always occurs at the DSPEED <expression> value (%) without being affected by the automatic movement speed value (%).

SAMPLE

```
DRIVE[2](1,10000),DS=0.1 ··· Axis 1 of robot 2 moves from its current
                             position to the 10000 pulses position at
                             0.1% of the maximum speed.
```

● STOPON conditions setting

Format

```
STOPON <conditional expression>
```

Explanation Stops movement when the conditions specified by the conditional expression are met. **Because this is a deceleration type stop, there will be some movement (during deceleration) after the conditions are met.**
If the conditions are already met before movement begins, no movement occurs, and the command is terminated.
This option is enabled only by program execution.

SAMPLE

```
DRIVE(1,10000),STOPON DI(20)=1
       ·············· Axis 1 moves from its current
                      position toward the "10000
                      pulses" position and stops at an
                      intermediate point if the "DI (20) =
                      1" condition is met. The next step
                      is then executed.
```

MEMO
• When the conditional expression used to designate the STOPON condition is a numeric expression, expression value other than "0" indicates a TRUE status, and "0" indicates a FALSE status.

● XY setting

Format

```
XY
```

Explanation Moves multiple specified axes to a position specified by Cartesian coordinates.
All the specified axes arrive at the target position at the same time.
If all axes which can be moved by MOVE statement have been specified, operation is identical to that which occurs when using MOVE statement.
The following restrictions apply to this command:
1. Axes specified by <axis number> must include the axis 1 and 2.
2. This command can be specified at SCARA robots and XY robots.
3. Point settings must be in "mm" or "deg" units (real number setting).

SAMPLE

```
DRIVE(1,P100),(2,P100),(4,P100),XY
       ·············· The axis 1, 2 and 4 move from their
                      current positions to the Cartesian
                      coordinates position specified by P100.
```

● Movement direction setting

Format

```
PLS
MNS
```

Explanation <With a "movement direction setting">

- Movement occurs in the specified direction. A PLS setting always results in plus-direction movement, and a MNS setting always results in minus-direction movement.
- If the target position and the current position are the same, a 1-cycle movement amount occurs in the specified direction, then operation stops.

<Without a "movement direction setting">

- Movement occurs in the direction in which the movement distance is shortest.
- If the target position and the current position are the same, no movement occurs.

• Cautions

1. When using this option, the maximum movement distance per operation is the distance equivalent to 1 cycle (360°). If movement which exceeds the 1-cycle distance is desired, the movement must be divided into 2 or more operations.
2. When using this option, the DRIVE statement's soft limit values are as shown below.

Plus-direction soft limit:	67,000,000 [pulse]
Minus-direction soft limit:	-67,000,000 [pulse]

• Restrictions

1. Only the axis of a single-axis rotary type robot can be specified.
2. Simultaneous movement of multiple axes is not possible when a movement direction has been specified. If such movement is attempted, the "5.37: Specification mismatch" error will occur (see below).
   Example: DRIVE (3,P1), (4,P1), PLS
3. The PLS and MNS options cannot both be specified simultaneously.
4. Attempting to use this option for a "limitless motion INVALID" axis will result in the "2.29: Cannot move without the limit" error.
5. If a stop is executed by pressing the [STOP] key, etc., during movement which uses this option (including during a deceleration), the movement distance when restarted will be equivalent to a 1-cycle distance (360°).

28 DRIVE

SAMPLE

```
DRIVE (4,270.00), PLS
     .......When the robot current position is 260°:
            Moves 10° in the plus direction from the
            current position.
            When the robot current position is 280°:
            Moves 350° in the plus direction from the
            current position.
DRIVE (4,270.00), MNS
     .......When the robot current position is 260°:
            Moves 350° in the minus direction from the
            current position.
            When the robot current position is 280°:
            Moves 10° in the minus direction from the
            current position.
DRIVE (4,270.00)
     .......When the robot current position is 260°:
            Moves 10° in the plus direction from the
            current position.
            When the robot current position is 280°:
            Moves 10° in the minus direction from the
            current position.
DRIVE[2] (3,270.00), PLS
     .......When the robot current position is 260°:
            Moves 10° in the plus direction from the
            current position.
            When the robot current position is 280°:
            Moves 350° in the plus direction from the
            current position.
DRIVE[2] (3,270.00), MNS
     .......When the robot current position is 260°:
            Moves 350° in the minus direction from the
            current position.
            When the robot current position is 280°:
            Moves 10° in the minus direction from the
            current position.
DRIVE[2] (3,270.00)
     .......When the robot current position is 260°:
            Moves 10° in the plus direction from the
            current position.
            When the robot current position is 280°:
            Moves 10° in the minus direction from the
            current position.
```

29 DRIVEI

Moves the specified robot axes in a relative manner

Format

```
DRIVEI(<axis number>, <expression>)[,(<axis number>,
<expression>)...][,option]
```

Values	<robot number>	1 to 4
	<axis number>	1 to 6
	<expression>	Motor position (mm, deg, pulses) or point expression.

Explanation Directly executes relative movement of each axis of a group, including the auxiliary axes.

• Movement type :	PTP movement of a specified axis
• Point data setting :	Direct coordinate data input, point definition
• Options :	Speed setting, STOPON conditions setting multiple options specifiable

MEMO

- When DRIVEI motion to the original target position is interrupted and then restarted, the target position for the resumed movement can be selected as the "MOVEI/DRIVEI start position" in the controller's "other parameters". For details, refer to the controller manual.

1) KEEP (default setting)	Continues the previous (before interruption) movement. The original target position remains unchanged.
2) RESET	Relative movement begins anew from the current position. The new target position is different from the original one (before interruption).

Movement type

● PTP (point-to-point) of specified axis

PTP movement begins after positioning of all axes specified at <axis number> is complete (within the tolerance range), and the command terminates when the specified axes enter the OUT position range. When two or more axes are specified, they will reach their target positions simultaneously.

If the next command following the DRIVEI command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position tolerance range.

Example:

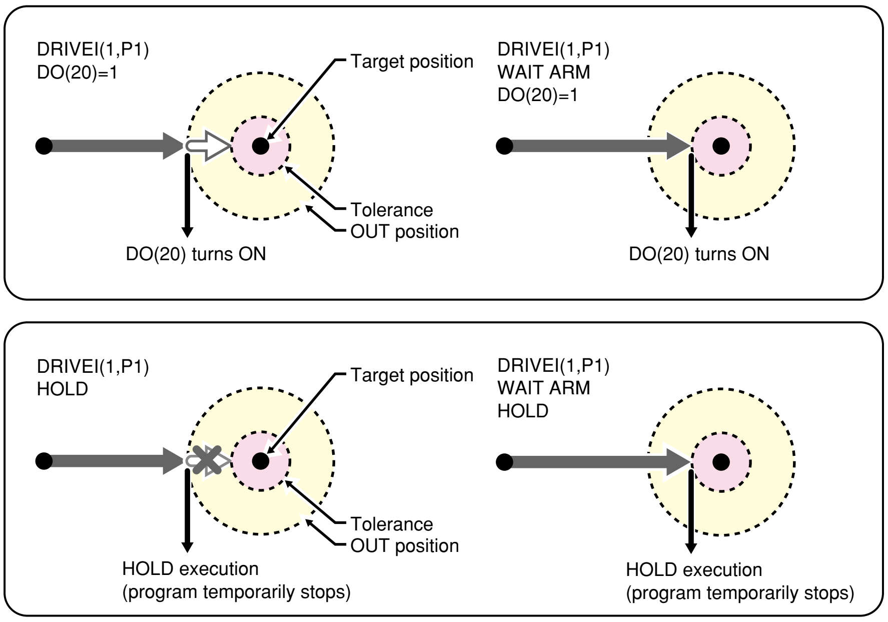
Signal output (DO, etc.)	Signal is output when axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when axis enters the OUT position range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	All programs in execution stop when axis enters the OUT position range, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops when axis enters the OUT position range. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop when axis enters the OUT position range. Therefore, the movement also stops.
WAIT	WAIT command is executed when axis enters the OUT position range.

29 DRIVEI

The WAIT ARM statement are used to execute the next command after the axis enters the tolerance range.

DRIVEI command

WAIT ARM statement



33820-R7-00

Limitless motion related cautions

• When the "limitless motion" parameter is enabled, the DRIVEI statement soft limit check values are as follows:

Plus-direction soft limit:	67,000,000 [pulse]
Minus-direction soft limit:	-67,000,000 [pulse]

•When using the DRIVEI statement, the above values represent the maximum movement distance per operation.

SAMPLE

```
DRIVEI(1,P0)····················The axis 1 moves from its
                                current position to the
                                position specified by P0.
```

Point data setting types

● Direct numeric value input

The motor position is specified directly in <expression>.
If the motor position's numeric value is a real number, this is interpreted as a "mm / deg" units, and each axis will move from the 0-pulse position to a pulse-converted position.

SAMPLE

```
DRIVEI(1,10000)  ···········The axis 1 moves from its
                            current position to the
                            "+10000 pulses" position.
DRIVEI(4,90.00)  ···········The axis 4 moves from its current
                            position to the +90° position
                            (when axis 4 is a rotating axis).
```

NOTE

• If point data is specified with both integers and real numbers in the same statement, all values are handled in "mm/degrees" units.

● Point definition

Point data is specified in <expression>. The axis data specified by the <axis number> is used. The motor position is determined in accordance with the point data defined by the point expression. If the point expression is in "mm/degrees" units, movement for each axis occurs from the 0-pulse position to the pulse-converted position.

SAMPLE

```
DRIVEI(1,P1)  ···············The axis 1 moves from its
                             current position the distance
                             specified by P1.
DRIVEI(4,P90)················The axis 4 moves from its
                             current position the number of
                             degrees specified by P90 (when
                             axis 4 is a rotating axis).
```

29 DRIVEI

Option types

● Speed setting

Format

```
1.   SPEED=<expression>
2.   S=<expression>
```

NOTE
- This defines the maximum speed, and does not guarantee that all movement will occur at specified speed.

Values <expression>..........................1 to 100 (units: %)

Explanation The program's movement speed is specified by the <expression>.
The actual speed is as follows:
- Robot's max. speed (mm/sec, or deg/sec) × automatic movement speed (%) × program movement speed (%).

This option is enabled only for the specified DRIVEI statement.

SAMPLE

```
DRIVEI(1,10000),S=10 ····· The axis 1 moves from its
                           current position to the +10000
                           pulses position at 10% of the
                           automatic movement speed.
```

Format

```
1.   DSPEED=<expression>
2.   DS=<expression>
```

NOTE
- SPEED option and DSPEED option cannot be used together.

Values <expression>..........................0.01 to 100.00 (units: %)

Explanation The axis movement speed is specified as an <expression>.
The actual speed is determined as shown below.
- Robot's max. speed (mm/sec, or deg/sec) × axis movement speed (%).

This option is enabled only for the specified DRIVEI statement.
- Movement always occurs at the DSPEED <expression> value (%) without being affected by the automatic movement speed value (%).

SAMPLE

```
DRIVEI(1,10000),DS=0.1 ·· The axis 1 moves from its
                          current position to the +10000
                          pulses position at 0.1% of the
                          automatic movement speed.
```

● STOPON conditions setting

Format

```
STOPON <conditional expression>
```

Explanation Stops movement when the conditions specified by the conditional expression are met. **Because this is a deceleration type stop, there will be some movement (during deceleration) after the conditions are satisfied.**
If the conditions are already satisfied before movement begins, no movement occurs, and the command is terminated.
This option is enabled only by program execution.

MEMO

- When the conditional expression used to designate the STOPON condition is a numeric expression, expression value other than "0" indicates a TRUE status, and "0" indicates a FALSE status.

SAMPLE

```
DRIVEI(1,10000),STOPON DI(20)=1
           ..............Axis 1 moves from its current
                         position toward the "+10000
                         pulses" position and stops at an
                         intermediate point if the "DI (20)
                         = 1" condition become satisfied. The
                         next step is then executed.
```

30 END SELECT

Ends the SELECT CASE statement

Format

```
SELECT [CASE] <expression>
   CASE <expression's list 1>
      [command block 1]
   CASE <expression's list 2>
      [command block 2]
      :
   [CASE ELSE
      [command block n]
END SELECT
```

Explanation Directly ends the SELECT CASE command block.
For details, see section "92 SELECT CASE to END SELECT".

SAMPLE

```
WHILE -1
SELECT CASE DI3()
   CASE 1,2,3
      CALL *EXEC(1,10)
   CASE 4,5,6,7,8,9,10
      CALL *EXEC(11,20)
   CASE ELSE
      CALL *EXEC(21,30)
END SELECT
WEND
HALT
```

Related commands	SELECT CASE

31 END SUB

Ends the sub-procedure definition

Format

```
SUB <label> [(<dummy argument> [, <dummy argument>…])]
   <command block>
END SUB
```

Explanation Ends the sub-procedure definition which begins at the SUB statement.
For details, see section "105 SUB to END SUB".

SAMPLE 1

```
I=1
CALL  *TEST
PRINT I
HALT
' SUB ROUTINE: TEST
SUB *TEST
  I=50
END SUB
```

Related commands	CALL, EXIT SUB, SUB to END SUB

32 ERR / ERL

Acquires the error code / error line No

Format

```
ERR(<task number>)
ERL(<task number>)
```

Values <task number>.......................1 to 4

Explanation Variables ERR and ERL are used in error processing routines specified by the ON ERROR GOTO statement.
ERR of the task specified by the <task number> gives the error code of the error that has occurred and ERL gives the line number in which the error occurred.

SAMPLE 1

```
IF ERR (1) <> &H604 THEN HALT
IF ERL (1) =20 THEN RESUME NEXT
```

Related commands	ON ERROR GOTO, RESUME

33 EXIT FOR

Terminates the FOR to NEXT statement loop

Format

```
EXIT FOR
```

Explanation Directly terminates the FOR to NEXT statement loop, then jumps to the command which follows the NEXT statement.
This statement is valid only between the FOR to NEXT statements.

MEMO

- The FOR to NEXT statement loop will end when the FOR statement condition is satisfied or when the EXIT FOR statement is executed. A "5.12: Stack overflow" error, etc., will occur if another statement such as GOTO is used to jump out of the loop.

SAMPLE

```
*ST:
WAIT DI(20)=1
FOR  A%=101 TO 109
   MOVE  P,P100,Z=0
   DO(20)=1
   MOVE P,P[A%],Z=0
   DO(20)=0
   IF DI(20)=0 THEN EXIT FOR
NEXT A%
GOTO  *ST
HALT
```

Related commands	FOR, NEXT

34 EXIT SUB

Terminates the sub-procedure defined by SUB to END

Format

```
EXIT SUB
```

Explanation The EXIT SUB statement terminates the sub-procedure defined by the SUB to END SUB statements, then jumps to the next command in the CALL statement that called up the sub-procedure.
This statement is valid only within the sub-procedure defined by the SUB to END SUB statements.

MEMO

- To end the sub-procedure defined by the SUB to END SUB statements, use the END SUB statement or EXIT SUB statement. A "5.12: Stack overflow" error, etc., will occur if another statement such as GOTO is used to jump out of the loop.

SAMPLE

```
' MAIN ROUTINE
CALL  *SORT2(REF X%,REF Y%)
HALT
' SUB ROUTINE: SORT
SUB *SORT2(X%, Y%)
   IF X%>=Y% THEN EXIT SUB
   TMP%=Y%
   Y%=X%
   X%=TMP%
END SUB
```

Related commands	CALL, SUB to END SUB, END SUB

35 EXIT TASK

Terminates its own task which is in progress

Format

```
EXIT TASK
```

Explanation Terminates its own task which is currently being executed.

SAMPLE

```
' TASK1 ROUTINE
*ST:
   MO(20)=0
   START <SUB_PGM>,T2
   MOVE P,P0,P1
   WAIT MO(20)=1
   GOTO *ST
HALT
Program name:SUB_PGM
*SUBPGM:
' TASK2 ROUTINE
*SUBTASK2:
   P100=JTOXY(WHERE)
   IF LOCZ(P100)>=100.0 THEN
      MO(20)=1
      EXIT TASK
   ENDIF
   DELAY 100
GOTO *SUBPGM
EXIT TASK
```

Related commands	CUT, RESTART, START, SUSPEND, CHGPRI

36 FOR to NEXT

Performs loop processing until the variable-specified value is exceeded

Format

```
FOR<control variable> = <start value> TO <end value> [STEP] <step>]
   <command block>
NEXT [<control variable>]
```

Explanation These direct statements repeatedly execute commands between the FOR to NEXT statements for the <start value> to <end value> number of times, while changing the <control variable> value in steps specified by <STEP>. If <STEP> is omitted, its value becomes "1".

The <STEP> value may be either positive or negative.

The <control variable> must be a numeric <simple variable> or <array variable>.

The FOR and NEXT statements are always used as a set.

SAMPLE

```
'CYCLE WITH CYCLE NUMBER OUTPUT TO DISPLAY
FOR A=1 TO 10
   MOVE P,P0
   MOVE P,P1
   MOVE P,P2
   PRINT"CYCLE NUMBER=";A
NEXT A
HALT
```

Related commands	EXIT FOR

37 GOSUB to RETURN

Jumps to a sub-routine

Format

```
GOSUB <label>      * GOSUB can also be expressed as "GO SUB".
  :
<label>:
  :
RETURN
```

Explanation Jumps to the <label> sub-routine specified by the GOSUB statement.
A RETURN statement within the sub-routine causes a jump to the next line of the GOSUB statement.

MEMO

- The GOSUB statement can be used up to 120 times in succession. Note that this number of times is reduced if commands containing a stack such as an FOR statement or CALL statement are used.
- When a jump to a subroutine was made with the GOSUB statement, always use the RETURN statement to end the subroutine. If another statement such as GOTO is used to jump out of the subroutine, an error such as "5.12: Stack overflow" may occur.

SAMPLE

```
*ST:
MOVE P,P0
GOSUB *CLOSEHAND
MOVE P,P1
GOSUB *OPENHAND
GOTO *ST
HALT
' SUB ROUTINE
*CLOSEHAND:
   DO(20) = 1
RETURN
*OPENHAND:
   DO(20) = 0
RETURN
```

Related commands	RETURN

A B C D E F G H I J K L M

38 GOTO

Executes an unconditional jump to the specified line

Format

```
GOTO <label>       * GOTO can also be expressed as "GO TO".
```

Explanation Executes an unconditional jump to the line specified by <label>.
To select a conditional jump destination, use the ON to GOTO, or IF statements.

SAMPLE

```
' MAIN ROUTINE
*ST:
  MOVE P,P0,P1
  IF DI(20) = 1 THEN
    GOTO *FIN
  ENDIF
GOTO *ST
*FIN:
HALT
```

39 HALT

Stops the program and performs a reset

Format

```
HALT[|    <expression>    |]
     |<character string>|
```

Explanation Directly stops the program and resets it. If restarted after a HALT, the program runs from its beginning.

If an <expression> or <character string> is written in the statement, the contents of the <expression> or <character string> are displayed on the programming box screen.

MEMO

- HALT is effective only in the task executed. The programs executed in other tasks continue execution.

SAMPLE

```
 ' MAIN ROUTINE
*ST:
  MOVE P,P0,P1
  IF DI(20) = 1 THEN
    GOTO *FIN
  ENDIF
GOTO *ST
*FIN:
HALT "PROGRAM FIN"
```

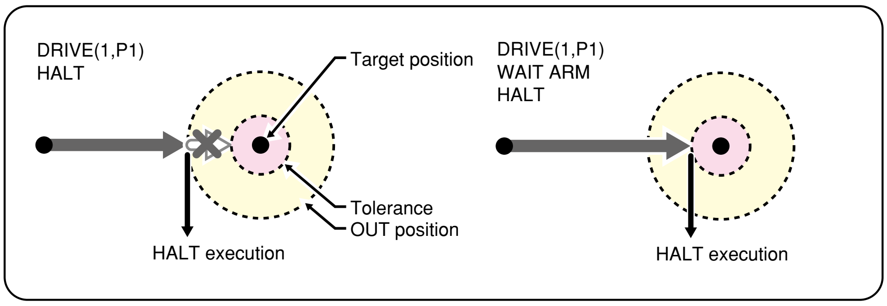
In PTP movement specified by movement commands such as MOVE and DRIVE, the next line's command is executed when the axis enters the OUT position range.

Therefore, if a HALT command exists immediately after a PTP movement command, that HALT command is executed before the axis arrives in the target position tolerance range.

Likewise, in interpolation movement during MOVE command, the next command is executed immediately after movement starts. Therefore, if a HALT command exists immediately after the interpolation movement command during MOVE, that HALT command is executed immediately after movement starts.

In either of the above cases, use the WAIT ARM command if desiring to execute the HALT command after the axis arrives within the target position tolerance range.

HALT command



33821-R7-00

40 HALTALL

Stops all programs and performs reset.

Format

```
HALT[|    <expression>    |]
     |<character string>|
```

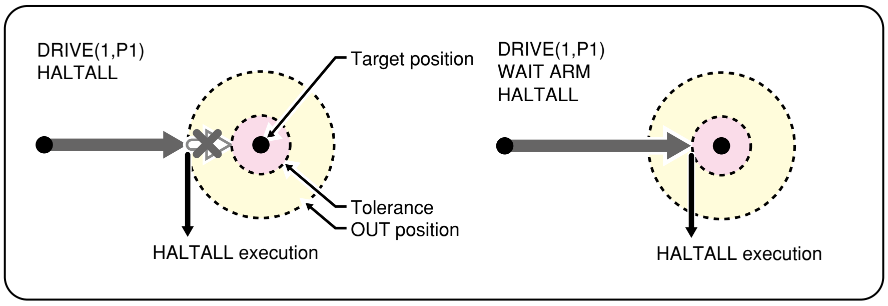
Explanation Directly stops all programs and reset them. If restarted after a HALTALL, the program runs from its beginning of the main program or the program that is executed lastly in task 1. If an <expression> or <character string> is written in the statement, the contents of the <expression> or <character string> are displayed on the programming box screen.

SAMPLE

```
’ MAIN ROUTINE
*ST:
  MOVE P,P0,P1
  IF DI(20) = 1 THEN
    GOTO *FIN
  ENDIF
GOTO *ST
*FIN:
HALT "PROGRAM FIN"
```

In PTP movement specified by movement commands such as MOVE and DRIVE, the next line's command is executed when the axis enters the OUT position range.
Therefore, if a HALTALL command exists immediately after a PTP movement command, that HALTALL command is executed before the axis arrives in the target position tolerance range.
Likewise, if the CONT option is specified in the interpolation movement during MOVE command, the next command is executed immediately after movement starts. Therefore, if a HALTALL command exists immediately after the interpolation movement command with the CONT option specified during MOVE, that HALTALL command is executed immediately after movement starts.
In either of the above cases, use the WAIT ARM command if desiring to execute the HALTALL command after the axis arrives within the target position tolerance range.

HALTALL command



33701-R9-00

41 HAND

Defines the hand

Format

```
Definition statement:
 HAND[<robot number>] Hn = <1st parameter> <2nd parameter> <3rd parameter> [R]
Selection statement:
 CHANGE [<robot number>] Hn
```

Values	<robot number>.....................1 to 4
	n: hand No.............................0 to 31

Explanation The HAND statement only defines the hand. To actually change hands, the CHANGE statement must be used.
For CHANGE statement details, see section "12 CHANGE".

MEMO

- If a power OFF occurs during execution of the hand definition statement, the "9.7 Hand check-sum error" may occur.

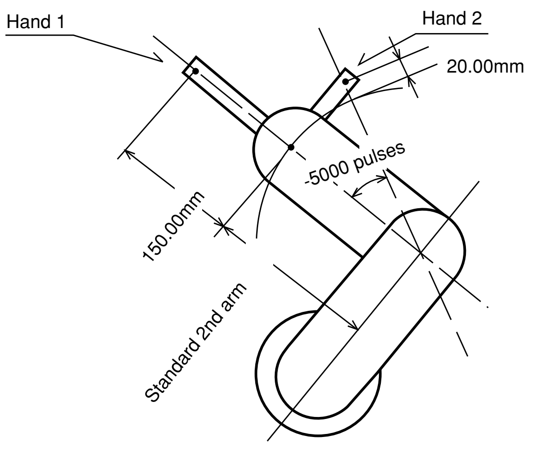
41.1 For SCARA Robots

1. When the <4th parameter> "R" is not specified

Hands installed on the second arm tip are selected (see below).

<1st parameter>	Number of offset pulses between the standard second arm position and the virtual second arm position of hand "n". "+" indicates the counterclockwise direction [pulse].
<2nd parameter>	Difference between the hand "n" virtual second arm length and the standard second arm length. [mm]
<3rd parameter>	Z-axis offset value for hand "n". [mm]

When the <4th parameter> "R" is not specified:



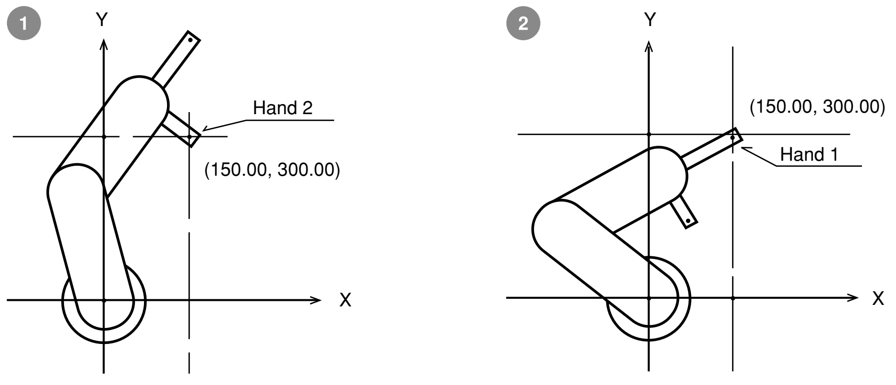
33803-R9-00

41 HAND

SAMPLE

```
HAND H1=       0    150.0      0.0
HAND H2=   -5000    20.00      0.0
P1=        150.00   300.00    0.00      0.00     0.00     0.00
CHANGE H2
MOVE P,P1 ························ Tip of hand 2 moves to P1. ❶
CHANGE H1
MOVE P,P1 ························ Tip of hand 1 moves to P1. ❷
HALT
```

SAMPLE:HAND



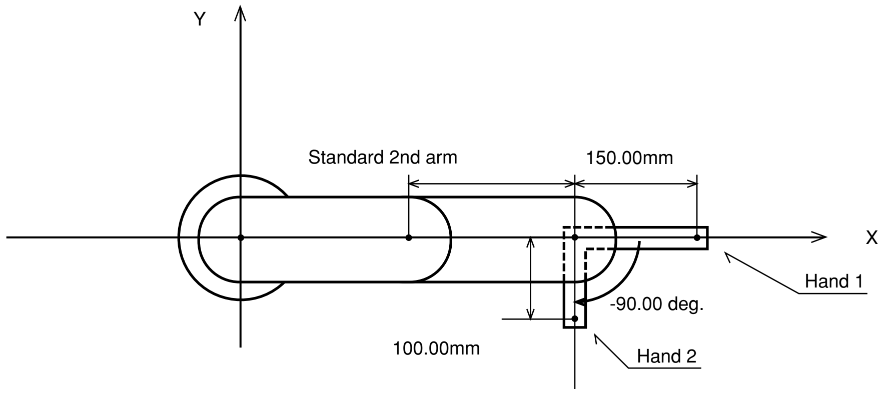
33802-R7-00

2. When the <4th parameter> "R" is specified

If the R-axis uses a servo motor, the hands that are offset from the R-axis rotating center are selected (see below).

<1st parameter>	When the current position of R-axis is 0.00, this parameter shows the angle of hand "n" from the X-axis plus direction in a Cartesian coordinate system. ("+"indicates the counterclockwise direction.) [degree]
<2nd parameter>	Length of hand "n". [mm] (>0)
<3rd parameter>	Z-axis offset amount for hand "n". [mm]

When the <4th parameter> "R" is specified

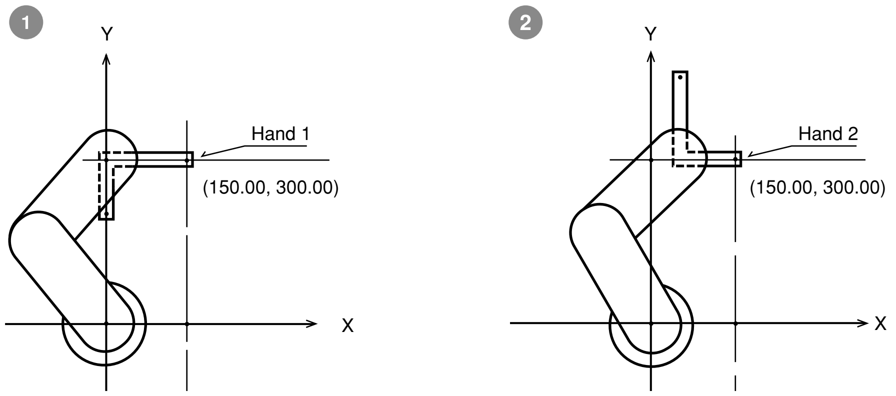


33804-R9-00

SAMPLE

```
HAND H1=    0.00   150.0     0.0        R
HAND H2= -90.00  100.00     0.0        R
P1=      150.00  300.00    0.00     0.00     0.00     0.00
CHANGE H1
MOVE P,P1 ························ Tip of hand 1 moves to P1. ❶
CHANGE H2
MOVE P,P1 ························ Tip of hand 2 moves to P1. ❷
HALT
```

SAMPLE:HAND



33804-R7-00

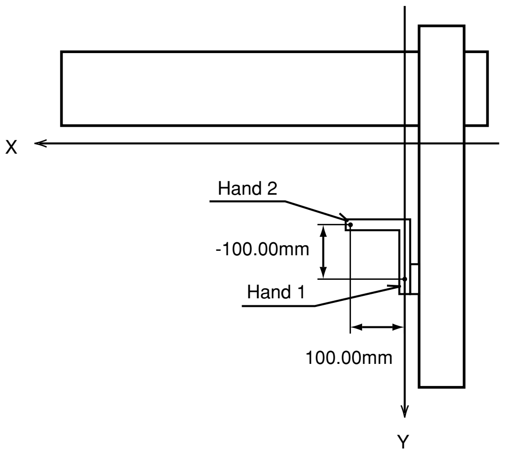
41.2 For Cartesian Robots

1. When the <4th parameter> "R" is not specified

Hands installed on the second arm tip are selected (see below).

<1st parameter>	Hand "n" X-axis offset amount [mm]
<2nd parameter>	Hand "n" Y-axis offset amount [mm]
<3rd parameter>	Hand "n" Z-axis offset amount [mm]

When the <4th parameter> "R" is not specified

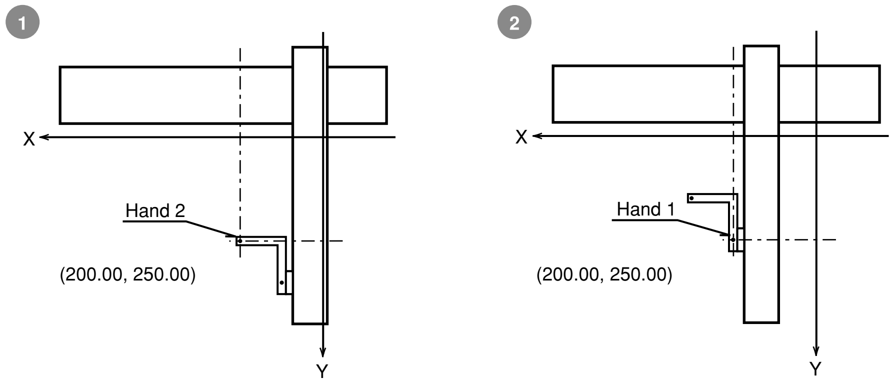


33805-R9-00

SAMPLE

```
HAND H1=    0.00     0.00     0.00
HAND H2=   100.0   -100.0   -100.0
P1=       200.00   250.00     0.00     0.00     0.00     0.00
CHANGE H2
MOVE P,P1 ························ Tip of hand 2 moves to P1. ❶
CHANGE H1
MOVE P,P1 ························ Tip of hand 1 moves to P1. ❷
HALT
```

SAMPLE:HAND



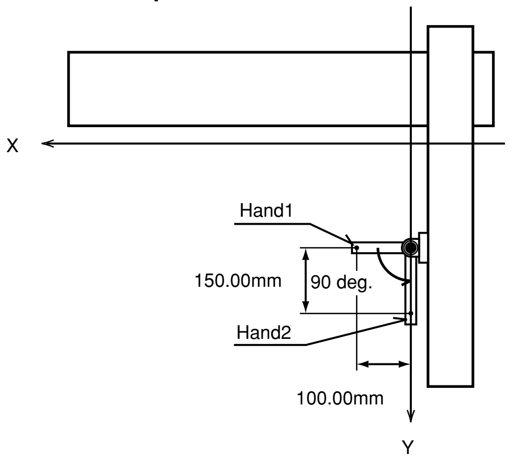
33806-R7-00

2. When the <4th parameter> "R" is specified

If the R-axis uses a servo motor, the hands that are offset from the R-axis rotating center are selected (see below).

<1st parameter>	When the current position of R-axis is 0.00, this parameter shows the angle of hand "n" from the X-axis plus direction in a Cartesian coordinate system. ("+"indicates the counterclockwise direction.) [degree]
<2nd parameter>	Length of hand "n". [mm] (>0)
<3rd parameter>	Z-axis offset amount for hand "n". [mm]

When the <4th parameter> "R" is specified

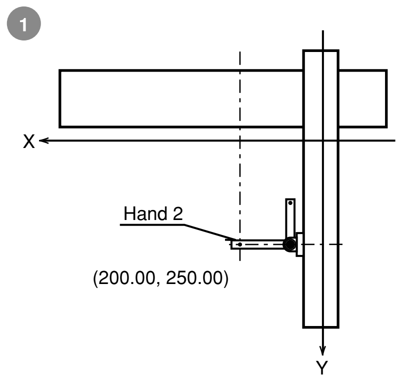


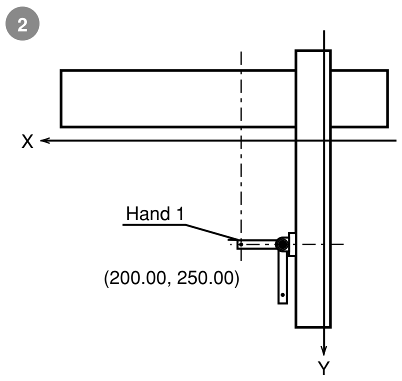
33806-R9-00

SAMPLE

```
HAND H1=    0.00  100.00    0.00        R
HAND H2=   90.00  150.00    0.00        R
P1=       200.00  250.00    0.00     0.00     0.00     0.00
CHANGE H2
MOVE P,P1 ························ Tip of hand 2 moves to P1. ❶
CHANGE H1
MOVE P,P1 ························ Tip of hand 1 moves to P1. ❷
HALT
```

SAMPLE:HAND





33808-R7-00

42 HOLD

Temporarily stops the program

Format

```
HOLD[│    <expression>    │]
     │<character string>│
```

Explanation Temporarily stops the program. When restarted, processing resumes from the next line after the HOLD statement. If an <expression> or <character string> is written in the statement, the contents of the <expression> or <character string> display on the programming box screen.

MEMO

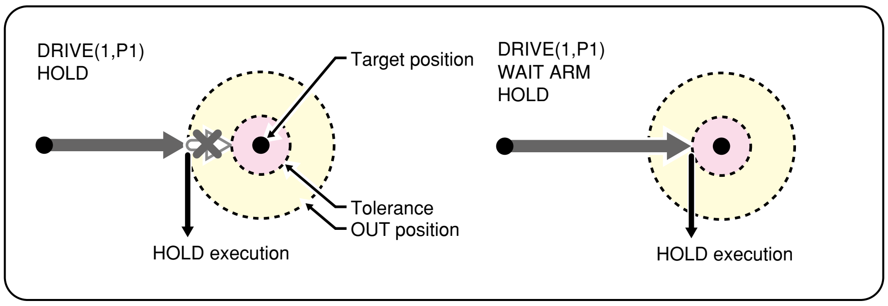
- HOLD is effective only in the task executed. The programs executed in other tasks continue execution.

SAMPLE

```
’ MAIN ROUTINE
*ST:
   MOVE P,P0,P1
   IF DI(20)=1 THEN
      HOLD "PROGRAM STOP"
   ENDIF
GOTO *ST
HALT
```

In PTP movement specified by movement commands such as MOVE and DRIVE, the next line's command is executed when the axis enters the effective OUT position range.
Therefore, if a HOLD command exists immediately after a PTP movement command, that HOLD command is executed before the axis arrives in the target position tolerance range.
Likewise, in interpolation movement during MOVE command, the next command is executed immediately after movement starts. Therefore, if a HOLD command exists immediately after the interpolation movement command during MOVE, that HOLD command is executed immediately after movement starts.
In either of the above cases, use the WAIT ARM command if desiring to execute the HOLD command after the axis arrives within the target position tolerance range.

HOLD command



33822-R7-00

43 HOLDALL

Temporality stops all programs.

Format

```
HOLD[|    <expression>    |]
     |<character string>|
```

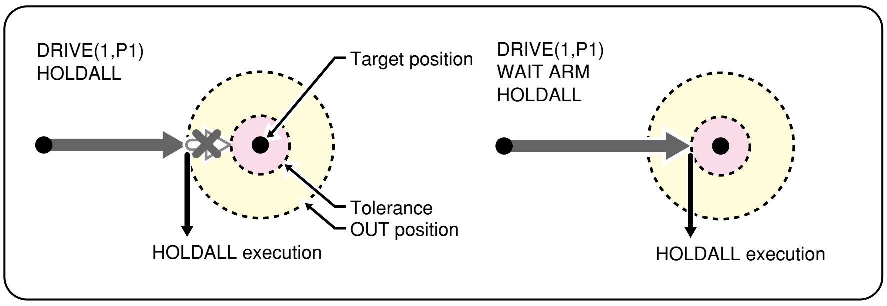
Explanation Temporality stops all programs. When restarted, the program that has executed HOLDALL is executed from the next line after the statement, and other programs are resumed from the line that has interrupted execution. If an <expression> or <character sting> is written in the statement, the contents of <expression> or <character string> displays on the programming box screen.

SAMPLE

```
' MAIN ROUTINE
*ST:
   MOVE P,P0,P1
   IF DI(20)=1 THEN
      HOLD "PROGRAM STOP"
   ENDIF
GOTO *ST
HALT
```

In PTP movement specified by movement commands such as MOVE and DRIVE, the next line's command is executed when the axis enters the effective OUT position range.
Therefore, if a HOLDALL command exists immediately after a PTP movement command, that HOLDALL command is executed before the axis arrives in the target position tolerance range.
Likewise, if the CONT option is specified in the interpolation movement during MOVE command, the next command is executed immediately after movement starts. Therefore, if a HOLDALL command exists immediately after the interpolation movement command with the CONT option specified during MOVE, that HOLDALL command is executed immediately after movement starts.
In either of the above cases, use the WAIT ARM command if desiring to execute the HOLDALL command after the axis arrives within the target position tolerance range.

HOLDALL command



33702-R9-00

44 IF

Evaluates a conditional expression value, and executes the command in accordance with the conditions

MEMO

- When the conditional expression used to designate the IF statement condition is a numeric expression, an expression value other than “0” indicates a TRUE status, and “0” indicates a FALSE status.

44.1 Simple IF statement

Format

```
IF <conditional expression> THEN| <label 1>           |[ELSE| <label 2>           |]
                                |<command statement 1>|     |<command statement 2>|
```

Explanation If the condition specified by the <conditional expression> is met, processing jumps either to the <label 1> which follows THEN, or to the next line after <command statement 1> is executed.

If the condition specified by the <conditional expression> is not met, the following processing occurs:

1. Processing either jumps to the <label 2> specified after the ELSE statement, or to the next line after <command statement 2> is executed.
2. If nothing is specified after the ELSE statement, no action is taken, and processing simply jumps to the next line.

SAMPLE

```
’ MAIN ROUTINE
*ST:
   MOVE P,P0,P1
   IF DI(20)=1 THEN *L1････ If DI (20) is "1", a jump to
                            *L1 occurs.
   DO(20)=1
   DELAY 100
*L1:
   IF DI(21)=1 THEN *ST ELSE *FIN
          ･･････････････････ If DI (21) is "1", a jump to
                            *ST occurs. If other than "1",
                            a jump to *FIN occurs.
*FIN:
HALT
```

44.2 Block IF statement

Format

```
IF <conditional expression 1> THEN
   <command block 1>
[ELSEIF <conditional expression 2> THEN
   <command block 2>]
[ELSE
   <command block n>]
ENDIF
```

Explanation If the condition specified by <conditional expression 1> is met, this statement executes the instructions specified in <command block 1>, then jumps to the next line after ENDIF.
When an ELSEIF statement is present and the condition specified by <conditional expression 2> is met, the instructions specified in <command block 2> are executed.
If all the conditions specified by the conditional expression are not met, <command block n> is executed.

SAMPLE

```
' MAIN ROUTINE
*ST:
   MOVE P,P0,P1
   IF DI(21,20)=1 THEN
      DO(20)= 1
      DELAY 100
      WAIT DI(20)=0
   ELSEIF DI(21,20)=2 THEN
      DELAY 100
   ELSE
      GOTO *FIN
   ENDIF
GOTO *ST
*FIN:
HALT
```

45 INPUT

Assigns a value to a variable specified from the programming box

Format

```
INPUT [<prompt statement>| ; |]  |<variable>      |[, |<variable>      |,...]
                         | , |   |<point variable>|   |<point variable>|
                         |   |   |<shift variable>|   |<shift variable>|
```

Explanation Assigns a value to the variable specified from the programming box.
The input definitions are as follows:

1. When two or more variables are specified by separating them with a comma (,), the specified input data items must also be separated with a comma (,).
2. At the <prompt statement>, enter a character string enclosed in quotation marks (" ") that will appear as a message requiring data input. When a semicolon (;) is entered following the prompt statement, a question mark (?) and a space will appear at the end of the message. When a comma (,) is entered, nothing will be displayed following the message.
3. When the <prompt statement> is omitted, only a question mark (?) and a space will be displayed.
4. The input data type must match the type of the corresponding variables. When data is input to a point variable or shift variable, insufficient elements are set to "0".
5. If only the ENTER key is pressed without making any entry, the program interprets this as a "0" or "null string" input. However, if specifying two or more variables, a comma (,) must be used to separate them.
6. If the specified variable is a character type and a significant space is to be entered before and after a comma (,), double quotation mark (") or character string, the character string must be enclosed in double quotation marks ("). Note that in this case, you must enter two double quotation marks in succession so that they will be identified as a double quotation mark input.
7. Pressing the ESC key skips this command.

MEMO

- If the variable and the value to be assigned are different types, the specified message displays, and a “waiting for input” status is established.
- When assigning alphanumeric characters to a character variable, it is not necessary to enclose the character string in double quotation marks (").

SAMPLE

```
INPUT  A
INPUT  "INPUT POINT NUMBER";A1
INPUT  "INPUT STRING",B$(0),B$(1)
INPUT  P100
HALT
```

46 INT

Truncates decimal fractions

Format

```
INT (<expression>)
```

Explanation This function acquires an integer value with decimal fractions truncated. The maximum integer value which does not exceed the <expression> value is acquired.

SAMPLE

```
A=INT(A(0))
B=INT(-1. 233)·················· "-2" is assigned to B.
```

47 JTOXY

Performs axis unit system conversions (pulse → mm)

Format

```
JTOXY [<robot number>] (<point expression>)
```

Values <robot number>.....................1 to 4

Explanation Converts the joint coordinate data (unit: pulse) specified by the <point expression> into Cartesian coordinate data (unit: mm, degree) of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
On YK500TW model robot, the first arm and the second arm rotation information are also set.

SAMPLE

```
P10=JTOXY(WHERE) ·············· Current position data is
                                converted to Cartesian
                                coordinate data.
```

Related commands	XYTOJ

48 LEFT$

Extracts character strings from the left end

Format

```
LEFT$ (<character string expression> , <expression>)
```

Values <expression>..........................0 to 25

Explanation This function extracts a character string with the digits specified by the <expression> from the left end of the character string specified by <character string expression>.
The <expression> value must be between 0 and 25, otherwise an error will occur.
If the <expression> value is 0, then LEFT$ will be a null string (empty character string).
If the <expression> value has more characters than the <character string expression>, LEFT$ will become the same as the <character string expression>.

SAMPLE

```
B$=LEFT$(A$,4) ················· 4 characters from the left end
                                 of A$ are assigned to B$.
```

Related commands	MID$, RIGHT$

49 LEFTY

Sets the SCARA robot hand system as a left-hand system

Format

```
LEFTY [<robot number>]
```

Values <robot number>.....................1 to 4

Explanation This statement specifies the robot specified by the <robot number> so that it moves as a left-handed system to a point specified in the Cartesian coordinates. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

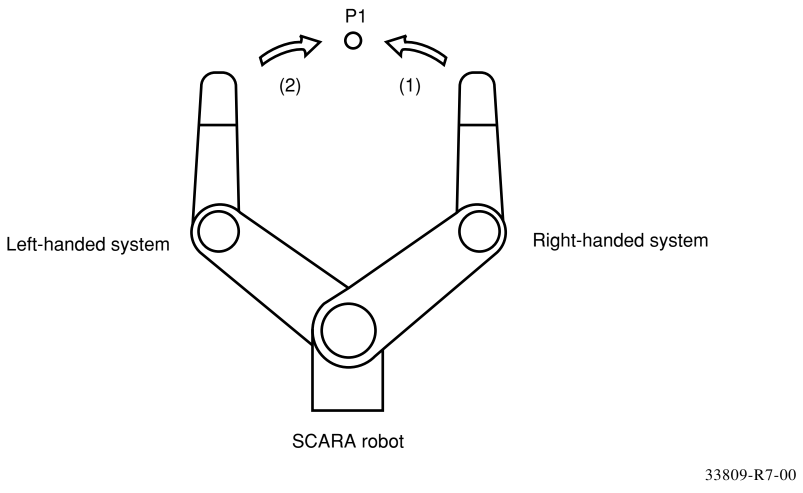
This statement only selects the hand system, and does not move the robot. If executed while the robot arm is moving, execution waits until movement is complete (positioned within tolerance range).

This command is only valid for SCARA robots.

SAMPLE

```
RIGHTY
MOVE P,P1
LEFTY
MOVE P,P1
RIGHTY
HALT
```

SAMPLE:LEFTY/RIGHTY



Related commands	RIGHTY

50 LEN

Acquires a character string length

Format

```
LEN(<character string expression>)
```

Explanation Returns the length (number of bytes) of the <character string expression>.

SAMPLE

```
B=LEN(A$)
```

51 LET

Assigns values to variables

Format

```
LET | <arithmetic assignment statement>       |
    | <character string assignment statement> |
    | <point assignment statement>            |
    | <shift assignment statement>            |
```

Explanation Directly executes the specified assignment statement. The right-side value is assigned to the left side. An assignment statement can also be directly written to the program without using a LET statement.

MEMO

- If the controller power is turned off during execution of a <point assignment statement> or <shift assignment statement>, a memory-related error such as the "9.2: Point check-sum error" or the "9.6: Shift check-sum error" may occur.

1. Arithmetic assignment statement

Format

```
[LET] | <arithmetic variable>       | = <expression>
      | <parallel output variable>  |
      | <internal output variable>  |
      | <arm lock output variable>  |
      | <timer output variable>     |
      | <serial output variable>    |
```

Values <expression>..........................Variable, function, numeric value

Explanation The <expression> value is assigned to the left-side variable.

SAMPLE

```
A!=B!+1
B%(1,2,3)=INT(10.88)
DO2()=&B00101101
MO(21,20)=2
LO(00)=1
TO(01)=0
SO12()=255
```

2. Character string assignment statement

Format

```
[LET] <character string variable> = <character string expression>
```

Explanation The <character string expression> value is assigned to the character string variable. Only the plus (+) arithmetic operator can be used in the <character string expression>. Other arithmetic operators and parentheses cannot be used.

SAMPLE

```
A$ ="YAMAHA"
B$ ="ROBOT"
D$ = A$ + "-" + B$

Execution result: YAMAHA-ROBOT
```

MEMO

- The "+" arithmetic operator is used to link character strings.

3. Point assignment statement

Format

```
[LET] <point variable> = <point expression>
```

Explanation Assigns <point expression> values to point variables.
Only 4 arithmetic operators (+, - *, /) can be used in the <point expression>.
Multiplication and division are performed only for constant or variable arithmetic operations.

• Addition/subtraction	Addition/subtraction is performed for each element of each axis.
• Multiplication	Multiplication by a constant or variable is performed for each element of each axis.
• Division	Division by a constant or variable is performed for each element of each axis.

Multiplication results vary according to the point data type.

• For pulse units	Assigned after being converted to an integer.
• For "mm" units	Assigned after being converted to a real number down to the 2nd decimal position.

SAMPLE
```
P1 =P10 ····················· Point 10 is assigned to P1.
P20=P20+P5 ·················· Each element of point 20 and
                              point 5 is summed and assigned
                              to P20.
P30=P30-P3 ·················· Each element of point 3 is
                              subtracted from point 30 and
                              assigned to P30.
P80=P70*4···················· Each element of point 70 is
                              multiplied by 4 and assigned
                              to P80.
P60=P5/3 ···················· Each element of point 5 is
                              divided by 3 and assigned to
                              P60.
```

MEMO

- Multiplication & division examples are shown below.
  - Permissible examples .... P15 * 5, P[E]/A, etc.
  - Prohibited examples .... P10 * P11, 3/P10, etc.

4. Shift assignment statement

Format
```
[LET] <shift variable> = <shift expression>
```

Explanation Assigns <shift expression> values to shift variables.
Only shift elements can be used in <shift expressions>, and only addition and subtraction arithmetic operators are permitted. Parentheses cannot be used.

- Addition/subtraction ....Addition/subtraction is performed for each element of each axis.

SAMPLE
```
S1=S0 ······················· "shift 0" is assigned to
                              "shift 1".
S2=S1+S0 ···················· Each element of "shift 1"
                              and "shift 0" is summed and
                              assigned to "shift 2".
```

MEMO

- Examples of <shift expression> addition/subtraction:
  - Permissible examples .... S1 + S2
  - Prohibited examples .... S1 + 3

52 LO

Arm lock output

Format

```
1. LOm ([b,･･･,b]) = <expression>
2. LO (mb,･･･,mb) = <expression>
```

Values

m: port number	0, 1
b: Bit definition	0 to 7
<expression>	Converted to an integer. The lower bits corresponding to the bits specified on the left side are valid.

Explanation This statement outputs the specified value to the LO port to either prohibit or allow axis movement.
LO(00) to LO(17) respectively correspond to axes 1 to 16. An arm lock ON status occurs at axes where bits are set, and axis movement is prohibited. Multiple bits must be specified in descending order from left to right (large → small).

MEMO

- This statement is valid at axes where movement is started.

SAMPLE

```
LO0()=&B00001010･･････････････ Prohibits movement at axes 2 and 4.
LO0(2,1)=&B10･････････････････ Prohibits movement at axis 3.
```

Related commands	RESET, SET

53 LOCx

Specifies/acquires point data for a specified axis or shift data for a specified element.

Format

```
1. LOCx (<point expression>) = <expression>
2. LOCx (<shift expression>) = <expression>
```

Values

Format 1: x	1 to 6 (axis setting), F (hand system flag setting), F1 (first arm rotation information), F2 (second arm rotation information).
Format 2: x	1 to 4 (element setting)
<expression>	For axis or element setting: coordinate value. For hand system flag setting: 1 (right-handed system) or 2 (left-handed system) 0 (no setting) For specifies the first arm rotation information and specifies the second arm rotation information (*1): 0, 1, -1

*1: For details regarding the first arm and the second arm rotation information, refer to Chapter 4 "3. Point data format".

Explanation

Direct format 1: Changes the value of the point data specified axis, the hand system flag, and the first arm and the second arm rotation information.

Format 2: Changes the value of a specified element from the shift data value.

MEMO

- Points where data is to be changed must be registered in advance. An error will occur if a value change is attempted at an unregistered point (where there are no coordinate values).

53 LOCx

Functions

Format

```
1. LOCx (<point expression>)
2. LOCx (<shift expression>)
```

Values

Format 1: x	1 to 6 (axis setting), F (hand system flag setting), F1 (first arm rotation information), F2 (second arm rotation information).
Format 2: x	1 to 4 (element setting)

Explanation Format 1: Acquires the value of the point data specified axis, the hand system flag, and the first arm and the second arm rotation information.
Format 2: Acquires a specified axis element from the shift data.

SAMPLE

```
LOC1(P10)=A(1) ·················· Axis 1 data of P10 is changed
                                  to the array A (1) value.
LOC2(S1)=B······················· Axis 2 data of S1 is changed
                                  to the B value.

A(1)=LOC1(P10) ·················· Axis 1 data of P10 is assigned
                                  to array A (1).
B(2)=LOC1(S1) ··················· The first element (X direction)
                                  of S1 is assigned to array B (2).
```

Related commands	Point variable, shift variable

54 LSHIFT

Left-shifts a bit

Format

```
LSHIFT (<expression 1> ,<expression 2>)
```

Explanation Shifts the <expression 1> bit value to the left by the amount of <expression 2>. Spaces left blank by the shift are filled with zeros (0).

SAMPLE

```
A=LSHIFT(&B10111011,2) ······ The 2-bit-left-shifted
                              &B10111011 value (&B11101100)
                              is assigned to A.
```

Related commands	RSHIFT

55 MCHREF

Acquires a machine reference

Format

```
MCHREF (<axis number>)
```

Values <axis number>.......................1 to 6

Explanation This function provides the return-to-origin or absolute-search machine reference (unit:%) for an axis specified by the <axis number>.
This function can only be used for axes whose return-to-origin method is set to the sensor or stroke end method.

SAMPLE

```
A=MCHREF(1) ······················ The  return-to-origin  machine
                                   reference of axis 1 of robot 1
                                   is assigned to variable A.
```

56 MID$

Acquires a character string from a specified position

Format

```
MID$ (<character string expression>,<expression 1>[,<expression 2>])
```

Values

<expression 1>.......................	1 to 255
<expression 2>.......................	0 to 255

Explanation This function extracts a character string of a desired length (number of characters) from the character string specified by <character string expression>. <expression 1> specifies the character where the extraction is to begin, and <expression 2> specifies the number of characters to be extracted.
An error will occur if the <expression 1> and <expression 2> values violate the permissible value ranges.
If <expression 2> is omitted, or if the number of characters to the right of the character of <expression 1> is less than the value of <expression 2>, then all characters to the right of the character specified by <expression 1> will be extracted.
If <expression 1> is longer than the character string, MID$ will be a null string (empty character string).

SAMPLE

```
B$=MID$(A$,2,4) ················ The 2nd to 4th characters (up
                                 to the 5th char.) of A$ are
                                 assigned to B$.
```

Related commands	LEFT$, RIGHT$

57 MO

Outputs a specified value to the MO port (internal output)

Format

```
1. MOm([b,･･･,b]) = <expression>
2. MO(mb,･･･,mb) = <expression>
```

REFERENCE

• For details regarding bit definitions, see Chapter 3 "10 Bit Settings".

Values

m: port number	2 to 7, 10 to 17, 20 to 27, 30 to 37
b: bit definition	0 to 7
<expression>	The integer-converted lower bits corresponding to the bit definition specified at the left side are valid.

Explanation Outputs a specified value to the MO port.
In order to maintain the sensor status and axis HOLD status at each axis, ports "30" to "37" cannot be used as output ports (these ports are for referencing only). (ports 32, 33, 36, and 37 are reserved by the system)
If multiple bits are specified, they are expressed from the left in descending order (large to small).
If the [b,…,b] data is omitted, all 8 bits are processed.

• Ports "30", "31", "34", and "35" outputs

Bit	7	6	5	4	3	2	1	0
Port 30	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1
Port 31	Axis 16	Axis 15	Axis 14	Axis 13	Axis 12	Axis 11	Axis 10	Axis 9
	Origin sensor status 0: ON; 1: OFF (Axis 1 is not connected)							
Port 34	Axis 8	Axis 7	Axis 6	Axis 5	Axis 4	Axis 3	Axis 2	Axis 1
Port 35	Axis 16	Axis 15	Axis 14	Axis 13	Axis 12	Axis 11	Axis 10	Axis 9
	HOLD status 0: No HOLD / 1: HOLD (Axis 1 is not connected)							

MEMO

• For details regarding MO ports "30" to "37", see Chapter 3 "9.5 Internal output variable".

SAMPLE

```
MO2()=&B10111000 ··············· MO (27, 25, 24, 23) are turned
                                 ON, and MO (26, 22, 21, 20)
                                 are turned OFF.
MO2(6,5,1)=&B010 ··············· MO (25) are turned ON, and MO
                                 (26, 21) are turned OFF.
MO3() = 15 ····················· MO (33, 32, 31, 30) are turned
                                 ON, and MO (37, 36, 35, 34)
                                 are turned OFF.
MO(37,35,27,20)=A··············· The contents of the 4 lower
                                 bits acquired when variable
                                 A is converted to an integer
                                 are output to MO (37, 35, 27,
                                 20), respectively.
```

Related commands	RESET, SET

58 MOTOR

Controls the motor power status.

Format

```
MOTOR | ON
      | OFF
      | PWR
```

Explanation This function controls the motor power on/off. The servo on/off of all robots can also be controlled at the same time.

- ON .... Turns on the motor power. All robot servos are also turned on at the same time.
- OFF.......... Turns off the motor power. All robot servos are also turned off at the same time to apply the dynamic brake. For the axis with the brake, the brake is applied to lock it.
- PWR .... Turns on only the motor power.

SAMPLE

```
MOTOR ON ·························· Turns on the motor power and
                                    all robot servos.
```

59 MOVE

Performs absolute movement of all robot axes

Format

```
MOVE [<robot number>] (<axis setting>,... ) | PTP | ,<point definition>,option ,option...
                                            | P   |
                                            | L   |
                                            | C   |
                                            | J   |
```

Explanation Executes absolute movement of the axis specified to a specified robot.
It is not enabled for axes of other robots or for auxiliary axes.
A robot numbers from 1 to 4 specifies a robot. If the robot number is omitted, robot 1 is specified. If the robot number is omitted, do not describe [ ], either.
Example: MOVE P, P0   Specifies robot 1.
Axis specifications from 1 to 6 specify axes (multiple axes specifiable). If the axis specification is omitted, all axes are specified.

- Movement type : PTP, linear interpolation, circular interpolation Merged PTP.
- Point data setting : Direct coordinate data input, point definition.
- Options : Speed setting, arch motion setting, STOPON condition setting, CONT setting, acceleration setting, deceleration setting, plane coordinate setting, port output setting (multiple ports outputs specifiable).
  The option can be omitted.

Options	PTP	Merged PTP	Linear interpolation	Arch interpolation	Remarks
Speed setting (SPEED)	○	○	○	○	Enabled only for specified MOVE statement
Speed setting (VEL)	×	×	○	○	Enabled only for specified MOVE statement
Arch motion	○	○	×	×	Enabled only for specified MOVE statement
STOPON condition setting	○	○	○	×	Enabled only by program execution
CONT setting	○	○	○	○	Enabled only for specified MOVE statement
Acceleration setting	○	○	○	×	Enabled only for specified MOVE statement
Deceleration setting	○	○	○	×	Enabled only for specified MOVE statement
Plane coordinate setting	×	×	×	○	Enabled only for specified MOVE statement
Port output setting	×	×	○	○	Enabled only for specified MOVE statement

59 MOVE

Movement type

● PTP (point-to-point) movement

Execution START condition: Movement of all specified axes is complete (within the tolerance range).
Execution END condition: All specified axes have entered the OUT position range.
When two or more axes are specified, they will reach their target positions simultaneously. The movement path of the axes is not guaranteed.

● Caution regarding commands which follow the MOVE P command:

If the next command following the MOVE P command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position tolerance range.
Example:

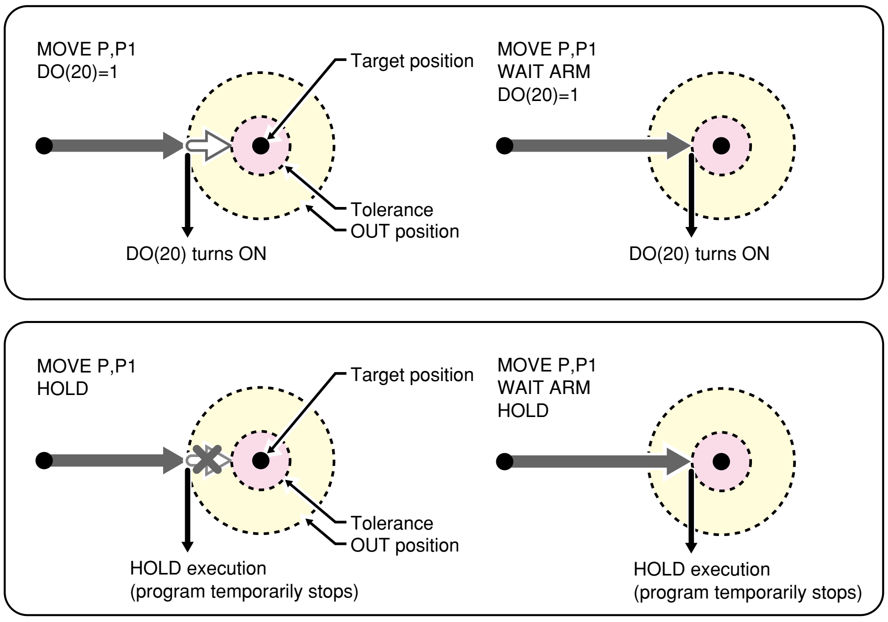
Signal output (DO, etc.)	Signal is output when axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when axis enters the OUT position range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	All programs in execution stop when axis enters the OUT position range, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops when axis enters the OUT position range. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop when axis enters the OUT position range. Therefore, the movement also stops.
WAIT	WAIT command is executed when axis enters the OUT position range.

The WAIT ARM statements are used to execute the next command after the axis enters the tolerance range.

MEMO

- The OUT position value is specified by parameter setting.
  This value can be changed from within the program by using the OUTPOS command.

MOVE command



33823-R7-00

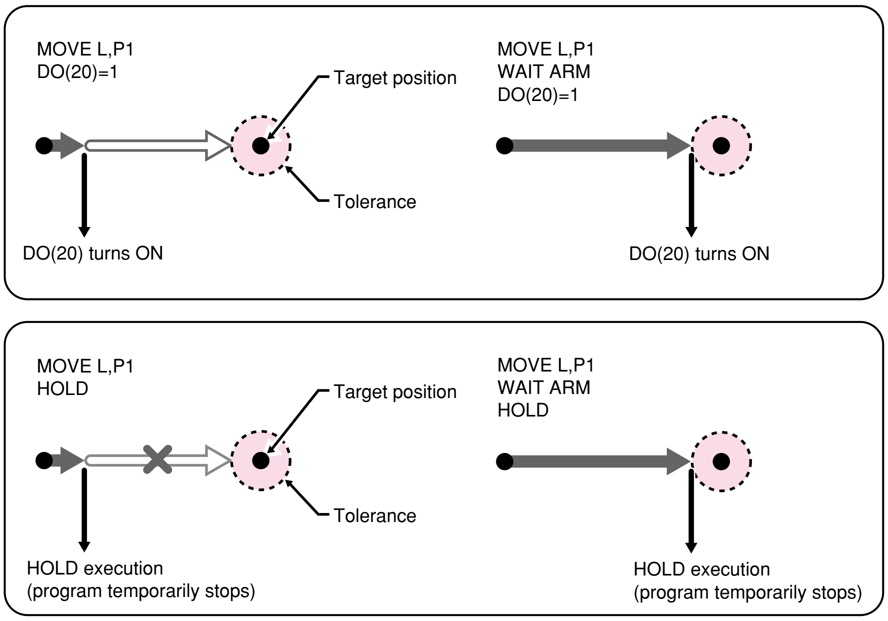
SAMPLE

```
MOVE P,P0·····················The main robot axis robot
                              1 moves from its current
                              position to the position
                              specified by P0. (the same
                              occurs for MOVE PTP, P0).
```

MEMO

- PTP movement is faster than interpolation movement, but when executing continuous movement to multiple points, a positioning stop occurs at each point.

MOVE command

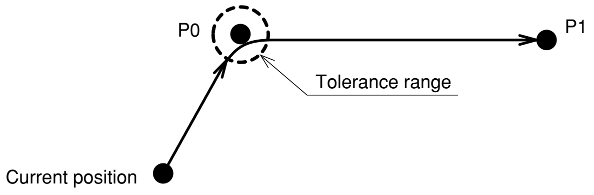


33824-R7-00

SAMPLE

```
MOVE L,P0,P1················The robot 1 moves from its
                            current position to the
                            position specified by P0, P1.
```

SAMPLE:MOVE L



33810-R7-00

● Circular interpolation movement

Execution START condition: Movement of all specified axes is complete (within the tolerance range).
Execution END condition: Movement of all specified axes has begun.
Execution of the immediately following command occurs immediately after axis movement begins.
When executing linear or circular interpolation in a continuous manner, the 2 movement paths are linked by connecting the deceleration and acceleration sections, enabling continuous movement without intermediate stops.
All movement axes arrive at the same time.
In circular interpolation, an arc is generated based on 3 points: the current position, an intermediate position, and the target position. **Therefore, circular interpolation must be specified by an even number of points.**

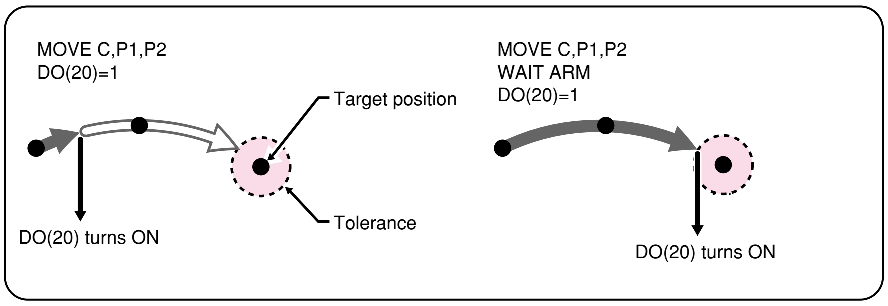
● Caution regarding commands which follow a MOVE C command:

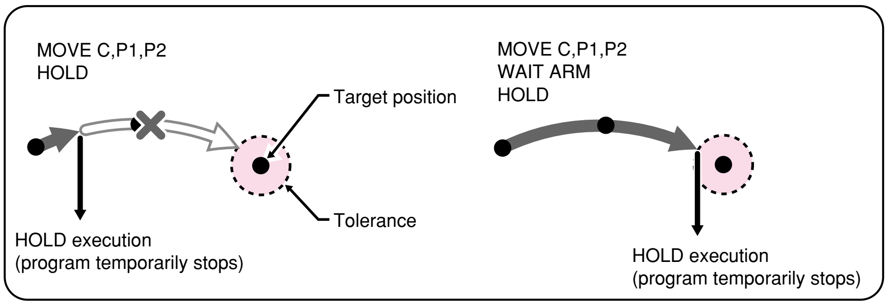
If the next command following the MOVE C command is an executable command such as a signal output command, that next command will start immediately after axis movement begins.
Example:

Signal output (DO, etc.)	Signal is output immediately after movement begins.
DELAY	DELAY command is executed and standby starts immediately after movement begins.
HALT	Program stops and is reset immediately after movement begins. Therefore, axis movement also stops.
HALTALL	All programs in execution stop when axis enters the OUT position range, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops immediately after movement begins. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop when axis enters the OUT position range. Therefore, the movement also stops.
WAIT	WAIT command is executed immediately after movement begins.

The WAIT ARM statement are used to execute the next command after the axis enters the tolerance range.

MOVE command



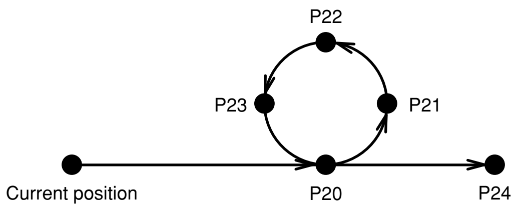


33825-R7-00

SAMPLE

```
MOVE L,P20 ··················· Linear movement occurs from
                               the current position to P20.
MOVE C,P21,P22,P23,P20 ·· Arc movement occurs through
                               points P21, P22, P23, P20.
MOVE L,P24 ··················· Linear movement occurs to P24.
```

SAMPLE:MOVE C



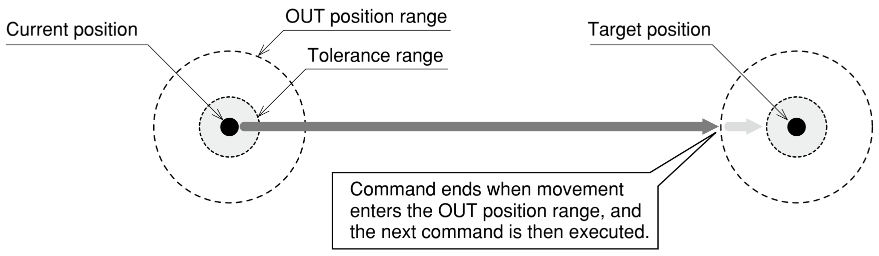
33811-R7-00

MEMO

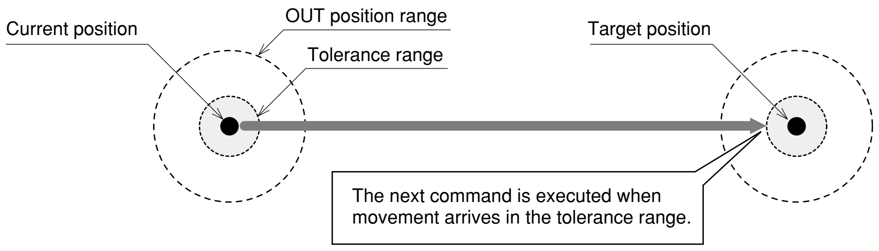
- In continuous interpolation operations, too, there are no stops at intermediate points. However, the maximum speed is slower than the PTP speed.
- Circular interpolation is possible within the following range: radius 1.00mm to 5,000.00mm.
- Circle distortion may occur, depending on the speed, acceleration, and the distance between points.
- On robots with an R-axis, the R-axis speed may become too fast and cause an error, depending on the R-axis movement distance.
- If a DELAY statement is executed after a MOVE L command, a DELAY timer is activated after the MOVE L command is executed. Therefore, **if a DELAY is desired after reaching the target point, use the WAIT ARM statement after the MOVE statement.**
  The same applies for other commands such as HALT, etc.
- On "Multi" type robots, the "5.37 : Specification mismatch" error will occur, and circular interpolation will be disabled.
- If the direction changes at an acute angle during interpolation movement, the acceleration/deceleration speed of the connection section may become too fast, causing an error. In this case, specify a slower acceleration/deceleration speed at the connection section, or use the WAIT ARM command to revise the operation pattern.

Movement command types and the corresponding movement

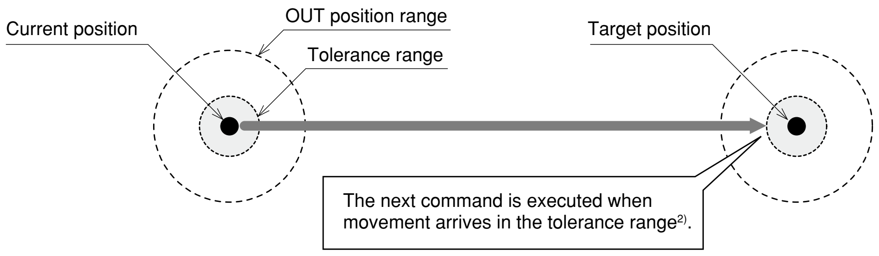
1. PTP movement


Current position
OUT position range
Tolerance range
Target position
Command ends when movement enters the OUT position range, and the next command is then executed.

2. WAIT ARM


Current position
OUT position range
Tolerance range
Target position
The next command is executed when movement arrives in the tolerance range.

3. STOPON conditional expression


Current position
OUT position range
Tolerance range
Target position
The next command is executed when movement arrives in the tolerance range2).

1) If a DELAY statement is executed after an interpolation operation, a DELAY timer is activated immediately after the movement starts. Therefore, if a DELAY is desired after reaching the target point, use the WAIT ARM statement after the MOVE command.
2) A deceleration and stop occurs at an intermediate point if the condition specified by the STOPON conditional expression is met (for details, see the "STOPON Condition Setting" item).

33703-R9-00

Point data setting types

● **Direct numeric value input** PTP Merged PTP Linear interpolation Circular interpolation

Format

```
p1 p2 p3 p4 p5 p6 [f][f1][f2]
```

Values

p1 to p6	Space-separated coordinate values for each axis.
f	Hand system flag (SCARA robot only)
f1	First arm rotation information (YK500TW model only).
f2	Second arm rotation information (YK500TW model only).

NOTE

- If both integers and real numbers are used together (mixed), all coordinate values will be handled in "mm/deg" units.

CAUTION

- When performing linear interpolation with a hand system flag specified, be sure that the same hand system is used at the current position and target position. If the same hand system is not used, an error will occur and robot movement will be disabled.
- When performing a linear interpolation, the current position's first arm and second arm rotation information must be the same as the movement destination's first arm and second arm rotation information. If the two are different, an error will occur and movement will be disabled.

Explanation Directly specifies coordinate values by a numeric value. If an integer is used, this is interpreted as "pulse" units, and if a real number (with decimal point) is used, this is interpreted as "mm/deg" units, with movement occurring accordingly. If both integers and real numbers are used together (mixed), all coordinate values will be handled in "mm/deg" units.

The types of movements in which this specification is possible are the PTP movement, the merged PTP movement, and the linear interpolation movement.

Hand system flags can be specified for SCARA robots when directly specifying the coordinate values in "mm" units.

To specify an extended hand system flag for SCARA robots, set either 1 or 2 at "f". If a number other than 1 or 2 is set, or if no number is designated, 0 will be set to indicate that there is no hand system flag.

1: Right-handed system is used to move to a specified position.

2: Left-handed system is used to move to a specified position.

Direct numeric value inputs can be used to set the first arm and the second arm rotation information (*1) only on YK500TW model robots where the coordinate system-of-units has been set as "mm".

To set extended the first arm and the second arm rotation information at the YK500TW model robot, a "-1", "0", or "1" value must be specified at f1 and f2. Any other value, or no the first arm and the second arm rotation information at all, will be processed as "0".

0: Indicates arm rotation information where movement to the "0" position has been specified.

1: Indicates arm rotation information where movement to the "1" position has been specified.

-1: Indicates arm rotation information where movement to the "-1" position has been specified.

*1: For details regarding the first arm and the second arm rotation information, refer to Chapter 4 "3. Point data format".

MEMO

- At SCARA robots with a hand system flag set in the movement destination's coordinate data, the specified hand system will have priority over the current arm type or LEFTY/RIGHTY setting.

SAMPLE

```
MOVE P,10000 10000 1000 1000 0 0
           ······················· PTP movement occurs from
                                   current position to the
                                   specified position.
```

CAUTION

• When moving the robot by linear or circular interpolation to a point where a hand system flag is specified, be sure that the same hand system is used at both the current and target positions. If the same hand system is not used, an error will occur and robot movement will be disabled.

CAUTION

• When performing a linear and circular interpolation, the current position's first arm and second arm rotation information must be the same as the movement destination's first arm and second arm rotation information. If the two are different, an error will occur and movement will be disabled.

● **Point definition** PTP Merged PTP linear interpolation Circular interpolation

Format

```
<point expression>[,<point expression>...]
```

Explanation Specifies a <point expression>. Two or more data items can be designated by separating them with a comma (,).
Circular interpolation must be specified by an even number of points.

MEMO

• At SCARA robots with a hand system flag set in the movement destination's coordinate data, the specified hand system will have priority over the current arm type or LEFTY/RIGHTY setting.

SAMPLE

```
MOVE   P,P1 ···················· Moves from the current position
                                 to the position specified by P1.
MOVE   P,P20,P0,P100······· Moves in sequence from the
                                 current position to positions
                                 specified by P20, P0, P100.
```

Option types

● Speed setting 1 PTP Merged PTP linear interpolation Circular interpolation

Format

```
1.  SPEED = <expression>
2.  S = <expression>
```

NOTE

• This option specifies only the maximum speed and does not guarantee movement at the specified speed.

Values <expression>..........................1 to 100 (units: %)

Explanation Specifies the program speed in an <expression>.
The actual speed will be as follows:

• [Robot max. speed (mm/sec)] × [automatic movement speed (%)] × [program movement speed (%)].

This option is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE  P,P10,S=10 ··········· Moves from the current
                             position to the position
                             specified by P10, at 10% of the
                             program movement speed.
```

● Speed setting 2 PTP Merged PTP linear interpolation Circular interpolation

Format

```
VEL = <expression>
```

NOTE

• This option specifies only the maximum composite speed and does not guarantee movement at the specified speed.

Values <expression>..........................For SCARA robot: 1 to 750
For XY robot: 1 to 1000 (units: mm/sec)

Explanation Specifies the maximum composite speed (in "mm/sec" units) of the XYZ axes in an <expression>. This option is specifiable when the robot is a SCARA robot or an XY robot and the type of movement is linear interpolation and circular interpolation movements.
This option is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE L,P10,VEL=100 ········ Moves from the current
                            position to the position
                            specified by P10 at the maximum
                            composite speed of 100 mm/sec.
                            of the XYZ attribute axis.
```

59 MOVE

● Arch motion setting PTP Merged PTP linear interpolation Circular interpolation

Format

```
x = <expression> [( <expression１>, <expression2>)]
```

Values	
x	Specifies an axis from A1 to A6.
<expression>	An integer value is processed in "pulse" units. A real number (with decimal point) is process in "mm/deg" units.
<expression 1>, <expression 2>	An integer value is processed in "pulse" units. A real number (with decimal point) is process in "mm/deg" units.

MEMO

- When there is a real value in any of the <expression>, <expression 1>, and <expression 2>, all expressions are handed as real value.
- The <expression 1> and <expression 2> are optional. These expressions specify the arch distance 1 and arch distance 2, respectively.

NOTE

- The axis arch distance parameters can be changed using ARCHP1/ARCHP2. The smaller the value, the shorter the movement execution time.

Explanation

1. The "x" specified axis begins moving toward the position specified by the <expression> (see "1" in the Fig. below).
2. When the axis specified by "x" moves the arch distance 1 or more, other axes move to their target positions ("2" shown in the figure below).
3. The axis specified by "x" moves to the target position so that the remaining movement distance becomes the arch distance 2 when the movement of other axes is completed ("3" shown in the figure below).
4. The command ends when all axis enter the OUT position range.

This option can be used only for PTP movement and merged PTP movement.
When the axis specified by "x" is the first arm or second arm of the SCARA robot or the axis 1 or axis 2 of the XY robot, the <expression> and target position value are limited to an integer (pulse units).

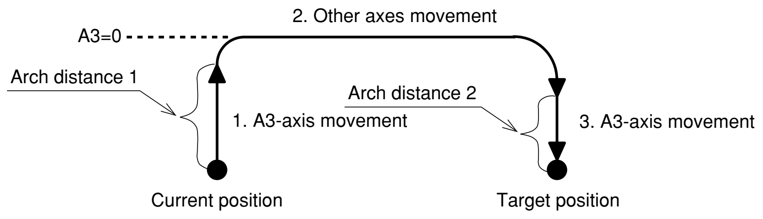
<expression> value:	SCARA and XY type robots.
Target position value:	SCARA type robot.

The values are indicated as the motor position rather than the coordinate values.

SAMPLE

```
MOVE P,P1,A3=0(150,100) ·········The A3-axis moves from the
                                 current position to the "0
                                 pulse" position.After that,
                                 other axes move to P1. Finally,
                                 the A3-axis moves to P1.
```

SAMPLE:MOVE A3



33704-R9-00

● STOPON condition setting PTP Merged PTP linear interpolation Circular interpolation

Format

```
STOPON <conditional expression>
```

Explanation Stops movement when the conditions specified by the conditional expression are met. Because this is a deceleration type stop, **there will be some movement (during deceleration) after the conditions are met.**
If the conditions are already met before movement begins, no movement occurs, and the command is terminated.
This option can only be used for PTP movement and linear interpolation movement.
This option is only possible by program execution.

SAMPLE

```
MOVE P,P100,STOPON DI(20)=1
            ··············Moves from the current position to
                          the position specified by P100. If
                          the "DI (20) = 1" condition is met
                          during movement, a deceleration and
                          stop occurs, and the next step is
                          then executed.
```

MEMO

- When the conditional expression used to designate the STOPON condition is a numeric expression, expression value other than "0" indicates a TRUE status, and "0" indicates a FALSE status.

NOTE

- The CONT setting can be used to reduce the PTP movement START positioning time.
- The path to the target point is not guaranteed.

● **CONT setting** PTP Merged PTP linear interpolation Circular interpolation

Format

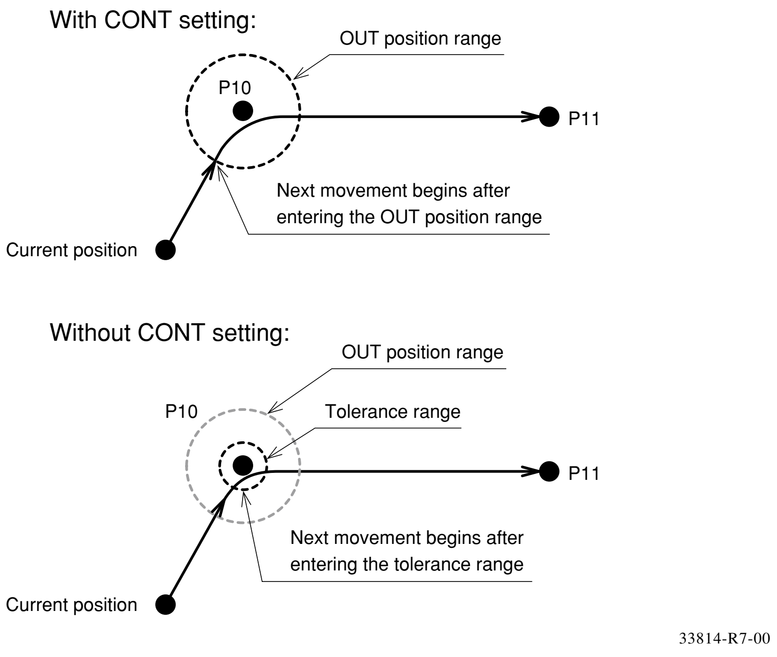
```
CONT
```

Explanation When movement is executed with CONT setting option, when all movable axes enter the OUT position range (with the command being terminated at that point), will begin movable axes to complete their movement into the tolerance range.
This option is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE  P,P10,P11,CONT
        ·············· Moves from the current position to
                       the position specified by P10, and
                       then moves to P11 without waiting
                       for the moving axes to arrive in the
                       tolerance range.
```

SAMPLE:MOVE CONT



● **Acceleration setting** PTP Merged PTP linear interpolation Circular interpolation

Format

```
ACC = <expression>
```

Values <expression>..........................1 to 100 (units: %)

Explanation Specifies the robot acceleration rate in an <expression>. The actual robot acceleration is determined by the acceleration coefficient parameter setting.
This option can be used only for linear interpolation movement and is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE L,P100,ACC=10 ········ Moves at an acceleration
                            rate of 10% from the current
                            position to the position
                            specified by P100.
```

● **Deceleration setting** PTP Merged PTP linear interpolation Circular interpolation

Format

```
DEC = <expression>
```

Values <expression>..........................1 to 100 (units: %)

Explanation Specifies the robot deceleration rate in an <expression>. The actual robot deceleration is determined by the acceleration coefficient parameter setting (the setting is specified as a percentage of the acceleration setting value (100%)).
This option can be used only for linear interpolation movement and is enabled only for the specified MOVE statement.

SAMPLE

```
  MOVE L,P100,DEC=20 ···· Moves at a deceleration
                          rate of 20% from the current
                          position to the position
                          specified by P100.
```

59 MOVE

● Coordinate plane setting

PTP　Merged PTP　linear interpolation　**Circular interpolation**

Format

```
| XY |
| YZ |
| ZX |
```

Values

XY	XY coordinate plane
YZ	YZ coordinate plane
ZX	ZX coordinate plane

Explanation When circular interpolation is executed by setting coordinates, this option executes circular interpolation so that the projection on the specified coordinate plane becomes a circle.
This option can be used for circular interpolation movement and is enabled only for the specified MOVE statement.

NOTE
- If no coordinate plane is specified, the robot moves along a 3-dimensional circle.
- When a 2-axis robot is used, the robot moves along a circle on the XY plane.

SAMPLE

```
P10 =  100.00 100.00 20.00 0.00 0.00 0.00
P11 =  150.00 100.00  0.00 0.00 0.00 0.00
P12 =  150.00 150.00 20.00 0.00 0.00 0.00
P13 =  100.00 150.00 40.00 0.00 0.00 0.00
MOVE P,P10
MOVE C,P11,P12
MOVE C,P13,P10 ·············· Moves continuously along a
                              3-dimensional circle generated
                              at P10, P11, P12, and P12,
                              P13, P10.
MOVE C,P11,P12,XY
MOVE C,P13,P10,XY ········· Moves continuously along a
                              circle on an XY plane generated
                              at P10, P11, P12, and P12,
                              P13, P10. Z-axis moves to the
                              position specified by P12 and P10
                              (the circle's target position).
```

● **Port output setting** PTP Merged PTP **linear interpolation** **Circular interpolation**

Format 1

```
|DO|m([b,···,b])=<expression 1>@<expression 2>
|MO|
|SO|
```

Format 2

```
|DO|(mb,···,mb)=<expression 1>@<expression 2>
|MO|
|SO|
```

Values	
m: port number	2 to 7, 10 to 17, 20 to 27
b: bit definition	0 to 7
<expression 1>	Value which is output to the specified port (only integers are valid).
<expression 2>	Position where the port output occurs. This position can be specified in "mm" units down to the 2nd decimal position.

CAUTION

• Output to ports "0" and "1" is not allowed at DO, MO, and SO.

REFERENCE

• For bit setting details, see Chapter 3 "10 Bit Settings".

Explanation

During linear interpolation or circular interpolation movement, this command option outputs the value of <expression 1> to the specified port when the robot reaches the <expression 2> distance (units: "mm") from the start position.

The <expression 2> numeric value represents a circle radius centered on the movement START point.

This command option can only be used with linear or circular interpolation movement, and it can be specified no more than 2 times per MOVE statement.

If multiple bits are specified, they are expressed from the left in descending order (large to small).

If the [b,…,b] data is omitted in format 1, all 8 bits are processed.

If no hardware port exists, nothing is output.

SAMPLE 1

```
MOVE P,P0
MOVE L,P1,DO2()=105@25.85
      ··············During linear interpolation movement to P1,
                    105 (&B01101001) is output to DO2() when the
                    robot reaches a distance of 25.85mm from P0.
```

SAMPLE 2

```
A!=10
B!=20
MOVE L,P2,MO(22)=1@A!,MO(22)=0@B!
      ··············After movement START toward P2, MO(22)
                    switches ON when the robot has moved a
                    distance of 10mm, and switches OFF when
                    the robot has moved a distance of 20mm.
```

Related commands	MOVEI, DRIVE, DRIVEI, WAIT ARM

60 MOVEI

Performs relative movement of all robot axes

Format

```
MOVEI│PTP│,<point definition> [, option [, option]...]
     │P  │
```

Explanation Executes relative position movement commands for the robot.
MOVEI is used for all specified axes.
It is not enabled for other groups, or for auxiliary axes.

• Movement type :	PTP
• Point data setting :	Direct coordinate data input, point definition.
• Options :	Speed setting

MEMO

- If the MOVEI statement is interrupted and then re-executed, the movement target position can be selected at the "MOVEI/DRIVEI start position" setting in the controller. For details, refer to the controller user's manual.

1) KEEP (default setting)	Continues the previous (before interruption) movement. The original target position remains unchanged.
2) RESET	Relative movement begins anew from the current position. The new target position is different from the original one (before interruption). (Backward compatibility)

Movement type

● PTP (point-to-point) movement

Execution START condition: Movement of all specified axes is complete (within the tolerance range).
Execution END condition: All specified axes have entered the OUT position range.
When two or more axes are specified, they will reach their target positions simultaneously. The movement path of the axes is not guaranteed.

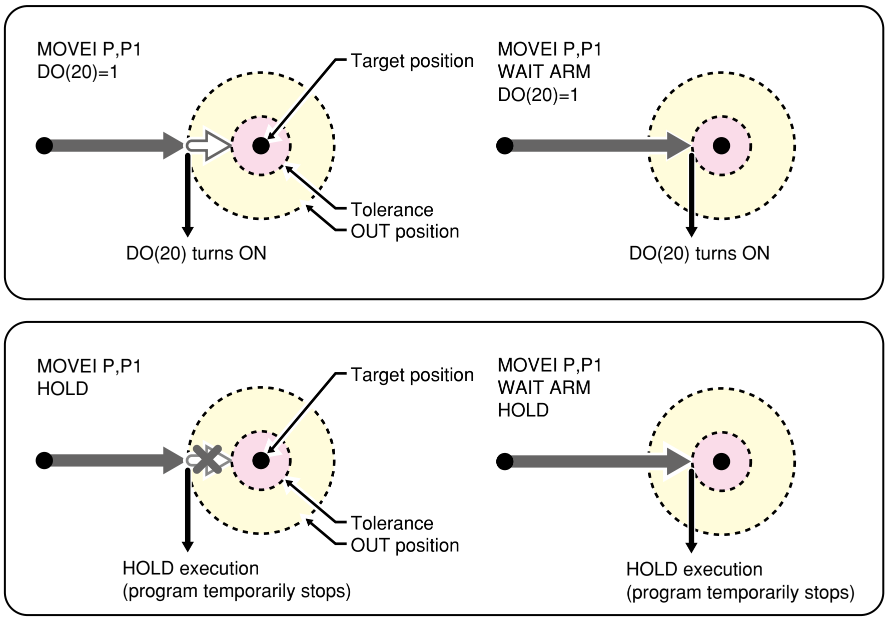
● Caution regarding commands which follow the MOVEI command:

If the next command following the MOVEI command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position tolerance range.
Example:

Signal output (DO, etc.)	Signal is output when axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when axis enters the OUT position range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	All programs in execution stop when axis enters the OUT position range, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops when axis enters the OUT position range. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop when axis enters the OUT position range. Therefore, the movement also stops.
WAIT	WAIT command is executed when axis enters the OUT position range.

The WAIT ARM statement are used to execute the next command after the axis enters the tolerance range.

MOVEI command



33826-R7-00

SAMPLE

```
MOVEI P,P0 ··················· From its current position,
                                the main robot axis moves (PTP
                                movement) the amount specified
                                by P0.
```

Point data setting types

● Direct numeric value input

PTP

Format

```
p1 p2 p3 p4 p5 p6 [f] [f1] [f2]
```

NOTE

- If both integers and real numbers are used together (mixed), all coordinate values will be handled in "mm/deg" units.

Values

p1 to p6	Space-separated coordinate values for each axis.
f	Hand system flag (SCARA robot only)
f1	First arm rotation information (YK500TW model only).
f2	Second arm rotation information (YK500TW model only).

Explanation Directly specifies coordinate values by a numeric value. If an integer is used, this is interpreted as "pulse" units, and if a real number is used, this is interpreted as "mm/deg" units, with movement occurring accordingly.
Hand system flags can be specified for SCARA robots when directly specifying the coordinate values in "mm" units.
To specify an extended hand system flag for SCARA robots, set either 1 or 2 at "f". If a number other than 1 or 2 is set, or if no number is designated, 0 will be set to indicate that there is no hand system flag.

1: Right-handed system is used to move to a specified position.
2: Left-handed system is used to move to a specified position.

Direct numeric value inputs can be used to set the first arm and the second arm rotation information (*1) only on YK500TW model robots where the coordinate system-of-units has been set as "mm".
To set extended the first arm and the second arm rotation information at the YK500TW model robot, a "-1", "0", or "1" value must be specified at f1 and f2. Any other value, or no the first arm and the second arm rotation information at all, will be processed as "0".

0: Indicates arm rotation information where movement to the "0" position has been specified.
1: Indicates arm rotation information where movement to the "1" position has been specified.
-1: Indicates arm rotation information where movement to the "-1" position has been specified.

*1: For details regarding the first arm and the second arm rotation information, refer to Chapter 4 "3. Point data format".

MEMO

- At SCARA robots with a hand system flag set in the movement destination's coordinate data, the specified hand system will have priority over the current arm type or LEFTY/RIGHTY setting.

SAMPLE

```
MOVEI P, 10000 10000 1000 1000 0 0
       ......... From its current position, the axis
                 moves (PTP movement) the specified
                 amount (pulse units).
```

● Point definition

PTP

Format

```
<point expression>[,<point expression>...]
```

Explanation Specifies a <point expression>. Two or more data items can be designated by separating them with a comma (,).

MEMO

- At SCARA robots with a hand system flag set in the movement destination's coordinate data, the specified hand system will have priority over the current arm type or LEFTY/RIGHTY setting.

SAMPLE

```
MOVEI P,P1 ··················· From its current position,
                                the axis moves (PTP movement)
                                the amount specified by P1.
```

60 MOVEI

Option types

● **Speed setting** PTP

Format

```
1.  SPEED = <expression>
2.  S = <expression>
```

Values <expression> ....1 to 100 (units: %)

NOTE
• This option specifies only the maximum speed and does not guarantee movement at the specified speed.

Explanation Specifies the program speed in an <expression>.
The actual speed will be as follows:
• [Robot max. speed (mm/sec)] × [automatic movement speed (%)] × [program movement speed (%)].
This option is enabled only for the specified MOVEI statement.

SAMPLE

```
  MOVEI P,P10,S=10 ······· From its current position, the
                           axis moves (PTP movement) the
                           amount specified by P1, at 10%
                           of the program movement speed.
```

Related commands	MOVE, DRIVE, DRIVEI, WAIT ARM

61 OFFLINE

Sets a specified communication port to the "offline" mode

Format

```
OFFLINE [│ETH│]
        [│CMU│]
```

Values <expression>..........................ETH, CMU, or no setting

Explanation Changes the communication mode parameter in order to switch the communication mode to OFFLINE.

ETH	Changes the Ethernet communication mode parameter to OFFLINE and clears the transmission and reception buffers.
CMU	Changes the RS-232C communication mode parameter to OFFLINE, resets the communication error, and clears the reception buffer.
No setting	Changes the Ethernet and RS-232C communication mode parameter to OFFLINE and clears the transmission and reception buffers.

SAMPLE

```
OFFLINE
SEND CMU TO A$
SEND CMU TO P10
ONLINE
HALT
```

62 ON ERROR GOTO

Jumps to a specified label when an error occurs

Format

```
1. ON ERROR GOTO <label>
2. ON ERROR GOTO 0
```

Values Error output information	ERR: Error code number
	ERL: Line number where error occurred

Explanation Even if an error occurs during execution of the robot language, this statement allows the program to jump to the error processing routine specified by the <label>, allowing the program to continue without being stopped (this is not possible for some serious errors.)
If "0" is specified instead of the <label>, the program stops when an error occurs, and an error message displays.
If ON ERROR GOTO "0" is executed at any place other than an error processing routine, the ON ERROR GOTO command is canceled (interruption canceled).
The error processing routine can process an error using the RESUME statement and the error output information (ERR, ERL).

MEMO

- If a serious error such as "17.4: Overload" occurs, the program execution stops.
- The most recently executed "ON ERROR GOTO <label>" statement is valid.
- If an error occurs during an error processing routine, the program will stop.
- "ON ERROR GOTO <label>" statements cannot be used within error processing routines.

SAMPLE

```
ON ERROR GOTO *ER1
FOR A = 0 TO 9
   P[A+10] = P[A]
NEXT A
*L99: HALT
' ERROR ROUTINE
*ER1:
IF ERR = &H0604 THEN *NEXT1········Checks to see if a
                                   "Point doesn't exist"
                                   error has occurred.
IF ERR = &H0606 THEN *NEXT2········Checks to see if a
                                   "Subscript out of range"
                                   error has occurred.
ON ERROR GOTO 0················Displays the error message and
                               stops the program.
*NEXT1:
   RESUME NEXT·················Jumps to the next line after
                               the error line and resumes
                               program execution.
*NEXT2:
   RESUME *L99·················Jumps to label *L99 and
                               resumes program execution.
```

Related commands	RESUME

63 ON to GOSUB

Executes the subroutine specified by the <expression> value

Format

```
ON<expression>GOSUB<label 1> [,<label 2>...]
            * GOSUB can also be expressed as "GO SUB".
```

Values <expression>..........................0 or positive integer

Explanation The <expression> value determines the program's jump destination.
An <expression> value of "1" specifies a jump to <label 1>, "2" specifies a jump to <label 2>, etc.
Likewise, (<expression> value "n" specifies a jump to <label n>.)
If the <expression> value is "0" or if the <expression> value exceeds the number of existing labels, no jump occurs, and the next command is executed.
After executing a jump destination subroutine, the next command after the ON to GOSUB statement is executed.

SAMPLE

```
' MAIN ROUTINE
*ST:
ON DI3() GOSUB *SUB1,*SUB2,*SUB3 ··· *SUB1 to *SUB3 are
                                     executed.
GOTO *ST ··························· Returns to *ST.
HALT
' SUB ROUTINE
*SUB1:
   MOVE P,P10,Z=0
   RETURN
*SUB2:
   DO(30) = 1
   RETURN
*SUB3:
   DO(30) = 0
   RETURN
```

Related commands	GOSUB, RETURN

64 ON to GOTO

Jumps to the label specified by the <expression> value

Format

```
ON<expression>GOTO<label 1> [,<label 2>...]
                  * GOTO can also be expressed as "GO TO".
```

Values <expression>..........................0 or positive integer

Explanation The <expression> value determines the program's jump destination.
An <expression> value of "1" specifies a jump to <label 1>, "2" specifies a jump to <label 2>, etc.
Likewise, (<expression> value "n" specifies a jump to <label n>.)
If the <expression> value is "0" or if the <expression> value exceeds the number of existing labels, no jump occurs, and the next command is executed.

SAMPLE

```
' MAIN ROUTINE
*ST:
ON DI3() GOTO *L1,*L2,*L3··········Jumps to *L1 to *L3 in
                                    accordance with the
                                    DI3() value.
GOTO *ST ··························Returns to *ST.
HALT
' SUB ROUTINE
*L1:
   MOVE P,P10,Z=0
   GOTO *ST
*L2:
   DO(30) = 1
   GOTO *ST
*L3:
   DO(30) = 0
   GOTO *ST
```

Related commands	GOTO

65 ONLINE

Sets the specified communication port to the "online" mode

Format

```
ONLINE   [│ETH │]
         [│CMU │]
```

Values <expression>..........................ETH, CMU, or no setting

Explanation Changes the communication mode parameter in order to switch the communication mode to ONLINE.

ETH	Changes the Ethernet communication mode parameter to ONLINE and clears the transmission and reception buffers.
CMU	Changes the RS-232C communication mode parameter to ONLINE, resets the communication error, and clears the reception buffer.
No setting...................	Changes the Ethernet and RS-232C communication mode parameter to ONLINE, resets the communication error, and clears the reception buffer.

SAMPLE

```
OFFLINE
SEND CMU TO A$
SEND CMU TO P10
ONLINE
HALT
```

66 ORD

Acquires a character code

Format

```
ORD (<character string expression>)
```

Explanation Acquires the character code of the first character in a <character string expression>.

SAMPLE

```
A=ORD("B")······················· 66 (=&H42) is assigned to A.
```

Related commands	CHR$

67 ORGORD

Specifies/acquires the robot's return-to-origin sequence

Format

```
ORGORD [<robot number>] <expression>
```

Values <robot number>.....................1 to 4
<expression>..........................n to nnnnnn (n : 0 to 6)

Explanation Sets the axis sequence parameter for return-to-origin and absolute search operation of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
The 1 to 6 axes are expressed as "1 to 6" values, respectively, and the <expression> value must be 1-digit to 6-digit integer.
The same axis cannot be specified twice.
After the specified axes are returned to their origin points in sequence, from left to right, the remaining axes return to their origin points simultaneously.
If the <expression> value is "0", all axes will be returned to their origin points simultaneously.

Functions

Format

```
ORGORD [<robot number>]
```

Values <robot number>.....................1 to 4

Explanation Acquires the axis sequence parameter for return-to-origin and absolute search operation of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=3
ORGORD A ························ A return-to-origin is executed
                                  first for axis 3.
ORIGIN   ························ After the return-to-origin of
                                  axis 3 of robot 1 is completed,
                                  a return-to-origin is executed
                                  for the remaining axis.
MOVE P,P0
A=ORGORD ························ The main group's return-to-
                                  origin sequence parameter is
                                  assigned to variable A.
HALT
```

Related commands	ORIGIN

68 ORIGIN

Performs a return-to-origin

Format

```
ORIGIN
```

Explanation This statement performs a return-to-origin of a robot, or an absolute search for a semi-absolute axis.
If the movement is stopped at an intermediate point, an "incomplete return-to-origin" status will occur.
When multiple robots are specified, the return-to-origin and absolute search are first performed for the robot 1 and then for the robots 2 to 4.

SAMPLE

```
ORIGIN   ·························· Performs return-to-origin.
```

Related commands	ORGORD, MCHREF

69 OUT

Turns ON the specified port output

Format

```
OUT | DOm([b,・・・,b]) | [, <time>]
    | DO(mb,・・・,mb)  |
    | MOm([b,・・・,b]) |
    | MO(mb,・・・,mb)  |
    | SOm([b,・・・,b]) |
    | SO(mb,・・・,mb)  |
    | LO0([b,・・・,b]) |
    | LO(0b,・・・,0b)  |
    | TO0([b,・・・,b]) |
    | TO(0b,・・・,0b)  |
```

Values

m: port number	2 to 7, 10 to 17, 20 to 27
b: bit definition	0 to 7
<expression>	1 to 3600000 (units: ms)

CAUTION

- Output to ports "0" and "1" are not allowed at DO, and SO.

REFERENCE

- For bit setting details, see Chapter 3 "10 Bit Settings".

Explanation This statement turns ON the specified port output and terminates the command. (The program proceeds to the next line.) Output to that port is then turned OFF after the time specified by the <expression> has elapsed. If the operation is stopped temporarily at an intermediate point and then restarted, that port's output is turned OFF when the remaining <expression> specified time has elapsed.

If this <expression> is omitted, the specified port's output remains ON.
Up to 16 OUT statements using <expressions> can be executed at the same time. Attempting to execute 17 or more OUT statements will activate error "6.26: Insufficient memory for OUT".

If multiple bits are specified, they are expressed from the left in descending order (large to small).
If no hardware port exists, nothing is output.

SAMPLE

```
OUT DO2(),200・・・・・・・・・・・・・・・・・・Turns DO(27 to 20) ON, then
                                  turns them OFF 200ms later.
OUT DO(37,35,27,20)・・・・・・・・・Turns DO(37, 35, 27, 20) ON.
```

Related commands	DO, MO, SO, TO, LO

70 OUTPOS

Specifies/acquires the OUT enable position parameter of the robot

Format

```
1. OUTPOS [<robot number>] <expression>
2. OUTPOS [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	1 to 6144000 (Unit: pulses)

Explanation Changes the parameter's OUT position of the robot axis specified by the <robot number> to the value indicated in the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

Format 1: The change is applied to all axes specified to specified robots.

Format 2: The change is applied only to the axis specified by <axis number>.

MEMO

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and command execution is stopped.

Functions

Format

```
OUTPOS [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the OUT position parameter value for the axis specified by the <expression> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

MEMO

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and command execution is stopped.

SAMPLE

```
' CYCLE WITH DECREASING OUTPOS
DIM SAV(3)
GOSUB *SAVE_OUTPOS
FOR A=1000  TO 10000  STEP 1000
   GOSUB *CHANGE_OUTPOS
   MOVE P,P0
   DO3(0)=1
   MOVE P,P1
   DO3(0)=0
NEXT A
GOSUB *RESTORE_OUTPOS
HALT
*CHANGE_OUTPOS:
   FOR B=1 TO 4
      OUTPOS(B)=A
   NEXT B
   RETURN
*SAVE_OUTPOS:
   FOR B=1 TO 4
      SAV(B-1)=OUTPOS(B)
   NEXT B
   RETURN
*RESTORE_OUTPOS:
   FOR B=1 TO 4
      OUTPOS(B)=SAV(B-1)
   NEXT B
   RETURN
```

71 PDEF

Defines the pallet used to execute pallet movement commands

Format

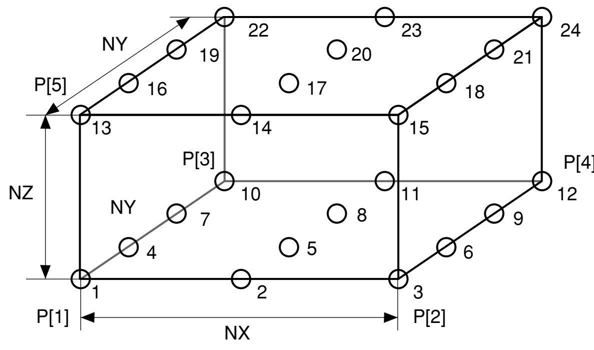
```
PDEF(<Pallet definition number> )=<expression 1>, <expression 2> [, <expression 3>]
```

Values

<Pallet definition number>.....	0 to 39
<point definition>	The point used for a pallet definition. Continuous 5 points starting with the specified point are used.
<expression 1>.......................	Number of points (NX) between P[1] and P[2].
<expression 2>.......................	Number of points (NY) between P[1] and P[3].
<expression 3>.......................	Number of points (NZ) between P[1] and P[5]. Total number of points: <expression 1> × <expression 2> × <expression 3> must be 32767 or less. Regarding the P[1] to P[5] definition, see the figure below.

Explanation Defines the pallets to permit execution of the pallet movement command.
Also changes the dividing conditions of previously defined pallet data.
After specifying the number of points per axis, the equally-spaced points for each axis are automatically calculated and defined in the sequence shown in the figure below.
If <expression 3> (Z-axis direction) is omitted, the height direction value becomes "1".
The total number of points defined for a single pallet must not exceed 32,767.

Automatic point calculation



33815-R7-00

SAMPLE

```
PDEF (1) =P3991,3,4,2·········· Pallet definition 1 is defined
                                as 3 x 4 x 2 by using P3991 to
                                P3995.
```

PMOVE

Executes a pallet movement command for the robot

Format

```
PMOVE [<robot number>] (<pallet definition number> ,
<pallet position number>)[,option[,option]...]
```

Values

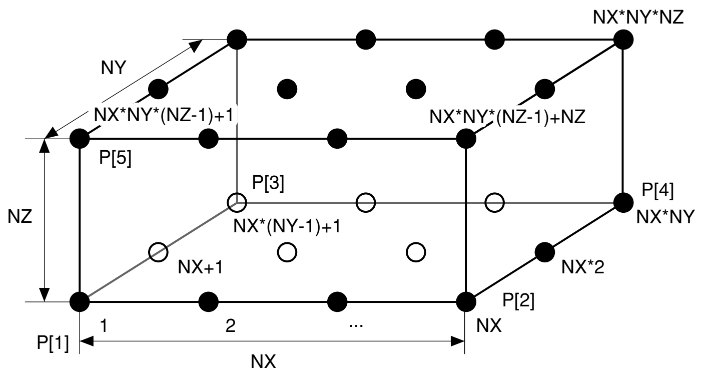
<robot number>.....................0 to 4
<pallet definition number>.....0 to 39
<pallet position number>.......1 to 32767

Explanation Executes a robot axis "pallet move" command. (The specified pallet numbers must be registered in advance.)
The PMOVE command applies to all main robot axes, and the PMOVE2 command applies to all sub robot axes. These commands do not apply to any other group axes, or to auxiliary axes.

• Movement type:	PTP
• Pallet definition number:	Numeric expression
• Pallet position number:	Numeric expression
• Options:	Speed setting, arch motion setting, STOPON condition setting

The position numbers for each pallet definition are shown below.

Position numbers for each pallet definition



33816-R7-00

MEMO

• Acquires the XYZ attribute axes move to the position determined by calculated values, the R attribute axis moves to the position specified by pallet point data P [1].

Options	PTP	Remarks
Speed setting (SPEED)	○	Enabled only for specified PMOVE statement
Arch motion	○	Enabled only for specified PMOVE statement
STOPON condition setting	○	Enabled only by program execution

SAMPLE

```
PMOVE(1,16) ····················· The main robot axis moves from
                                  its current position to the
                                  position specified by pallet
                                  position number 16 of pallet
                                  definition number 1.
```

72 PMOVE

Movement type

● **PTP (point-to-point) movement**

PTP movement begins after positioning of all movement axes is complete (within the tolerance range), and **the command terminates when the movement axes enter the OUT position range.** Although the movement axes reach their target positions simultaneously, their paths are not guaranteed.

● **Caution regarding commands which follow the PMOVE command:**

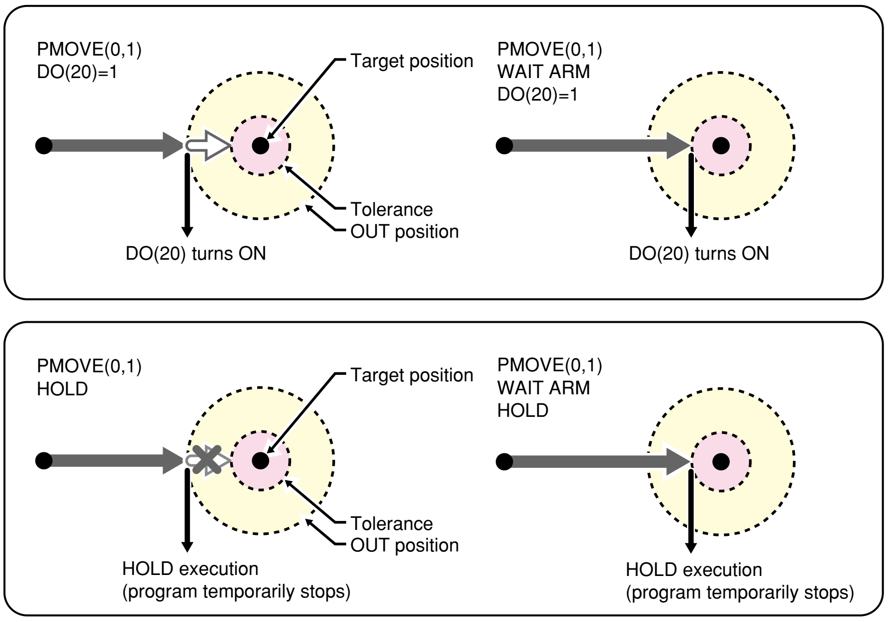
If the next command following the PMOVE command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position OUT position range.

Example:

Signal output (DO, etc.)	Signal is output when axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when axis enters the OUT position range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	All programs in execution stop when axis enters the OUT position range, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops when axis enters the OUT position range. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop when axis enters the OUT position range. Therefore, the movement also stops.
WAIT	WAIT command is executed when axis enters the OUT position range.

The WAIT ARM statement is used to execute the next command after the axis enters the tolerance range.

PMOVE command



33827-R7-00

Option types

● Speed setting PTP

Format

```
1.  SPEED = <expression>
2.  S = <expression>
```

Values <expression>..........................1 to 100 (units: %)

Explanation Specifies the program speed in an <expression>. The movement speed is the automatic movement speed multiplied by the program movement speed.
This option is enabled only for the specified PMOVE statement.

NOTE
- This option specifies only the maximum speed and does not guarantee movement at the specified speed.

SAMPLE

```
PMOVE(1,3),S=10············· Movement occurs at 10% of the
                             program speed, from the current
                             position to the position specified
                             by pallet position number 3 of
                             pallet definition number 1.
```

● Arch motion setting PTP

Format

```
x = <expression>[, x = <expression>...]
```

Values x	Specifies the Z,R,A,B axis.
<expression>..........................	An integer value is processed in "pulse" units. A real number (with decimal point) is process in "mm/deg" units.

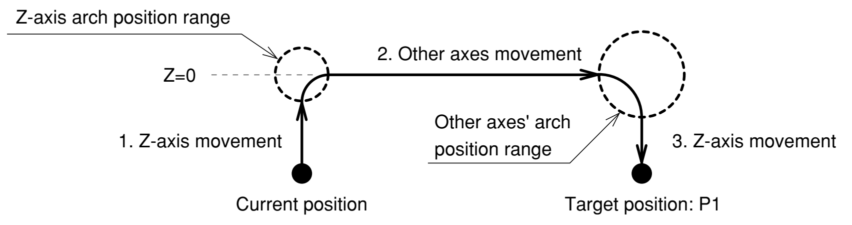
Explanation

1. The "x" specified axis begins moving toward the position specified by the <expression>.
2. When the "x" specified axis enters the arch position range, all other axes move toward the target position.
3. When all axes other than the "x" specified axis enter the arch position range, and the "x" specified axis enters the tolerance range of the position specified by the <expression>, the "x" specified axis then moves to the target position.
4. The command ends when all axis enter the OUT position range.

SAMPLE

```
PMOVE(1,A),Z=0 ········The Z-axis first moves from the current
                       position to the "0 pulse" position. Then the
                       other axes move to the position specified by
                       pallet position number A of pallet definition
                       number 1. Finally the Z-axis moves to the
                       position specified by pallet position number A.
```

SAMPLE:PMOVE Z



33817-R7-00

● **STOPON condition setting** PTP

Format

```
STOPON <conditional expression>
```

Explanation Stops movement when the conditions specified by the conditional expression are met. Because this is a deceleration type stop, there will be some movement (during deceleration) after the conditions are met.
If the conditions are already met before movement begins, no movement occurs, and the command is terminated.
This option is only possible by program execution.

SAMPLE

```
PMOVE(A,16),STOPON DI(20)=1
          ·············· Moves from the current position to
                         the position specified by pallet
                         position number 16 of pallet
                         definition number A, then decelerates
                         and stops when the condition "DI(20)
                         = 1" is met.
```

MEMO

- When the conditional expression used to designate the STOPON condition is a numeric expression, expression value other than “0” indicates a TRUE status, and “0” indicates a FALSE status.

Pn

Defines points within a program

Format

```
[LET] Pn = p1 p2 p3 p4 p5 p6[ f ][ f 1][ f 2]
```

Values

n	Point number: 0 to 29999.
p1 to p6	Point data: the range varies according to the format.
f	Hand system flag: 1 or 2 (for SCARA robots only).
f1	First arm rotation information : -1, 0, 1 (YK500TW model only).
f2	Second arm rotation information : -1, 0, 1 (YK500TW model only).

Explanation Defines the point data.

NOTE

• If both integers and real numbers are used together (mixed), all coordinate values will be handled in "mm/deg" units.

1. "n" indicates the point number.
2. Input data for "p1" to "p6" must be separated with a space (blank).
3. If all input data for "p1" to "p6" are integers (no decimal points), the movement units are viewed as "pulses". "p1" through "p6" then correspond to axis 1 through axis 6.
4. **If there is even 1 real number (with decimal point)** in the input data for "x" through "b", **the movement units are recognized as "mm".** In this case, "x" to "z" correspond to the x, y and z coordinates of a Cartesian coordinate system, while "r" to "b" correspond to axes 4 to 6.
5. The input data ranges are as follows:

For "pulse" units:	-6,144,000 to 6,144,000 range
For "mm" units:	-99,999.99 to 99,999.99 range

Hand system flags can be specified for SCARA robots when specifying point definition data in "mm" units.
To specify an extended hand system flag for SCARA robots, set either 1 or 2 at "f". If a number other than 1 or 2 is set, or if no number is designated, 0 will be set, indicating that there is no hand system flag.

1: Indicates a right-handed system point setting.
2: Indicates a left-handed system point setting.

The first arm and the second arm rotation information (*1) can be specified on YK500TW where point data is defined in "mm" units.
To set extended the first arm and the second arm rotation information at the YK500TW model robot, a "-1", "0", or "1" value must be specified at f1 and f2. Any other value, or no the first arm and the second arm rotation information at all, will be processed as "0".

0: Indicates arm rotation information where "0" has been specified.
1: Indicates arm rotation information where "1" has been specified.
-1: Indicates arm rotation information where "-1" has been specified.

*1: For details regarding the first arm and the second arm rotation information, refer to Chapter 4 "3. Point data format".

73 Pn

NOTE

- All input values are handled as constants.
- If controller power is turned off during execution of a point definition statement, a memory-related error such as "9.2: Point check-sum error" may occur.

SAMPLE

```
P1 =       0       0       0       0       0       0
P2 =  100.00  200.00   50.00    0.00    0.00    0.00
P3 =   10.00    0.00    0.00    0.00    0.00    0.00
P10=  P2
FOR A=10 TO 15
   P[A+1]=P[A]+P3
NEXT A
FOR A=10 TO 16
   MOVE P,P1,P[A]
NEXT A
HALT
```

Related commands	Point assignment statement (LET)

74 PPNT

Creates pallet point data

Format

```
PPNT(pallet definition number,pallet position number)
```

Explanation Creates the point data specified by the pallet definition number and the pallet position number.

SAMPLE

```
P10=PPNT(1,24) ·················· Creates, at P10, the point
                                  data specified by pallet
                                  position number 24 of pallet
                                  definition number 1.
```

Related commands	PDEF, PMOVE

75 PRINT

Displays the specified expression value at the programming box

Format

```
PRINT [<expression>] [| , |<expression>...] [| , |]
                      | ; |                   | ; |
```

Values <expression>..........................character string, numeric value, variable.

Explanation Displays a specified variable on the programming box screen.
Output definitions are as follows:

1. If numbers or character strings are specified in an <expression>, they display as they are. If variables or arrays are specified, the values assigned to the specified variables or arrays display.
2 If no <expression> is specified, only a line-feed occurs.
3. If the data length exceeds the screen width, a line-feed occurs, and the data wraps to the next line.
4. If a comma (,) is used as a display delimiter, a space (blank) is inserted between the displayed items.
5. If a semicolon (;) is used as a display delimiter, the displayed items appear in succession without being separated.
6. If the data ends with a delimiter, the next PRINT statement is executed without a line-feed. When not ended with a display delimiter, a line-feed occurs.

MEMO

- Data communication to the programming box screen occurs in order for the PRINT statement to be displayed there. Therefore, program execution may be delayed when several PRINT statements are executed consecutively.

SAMPLE

```
PRINT A Displays the value of variable A.
PRINT "A1 =";A1  ·············· Displays the value of variable
                                 A1 after "A1 =".
PRINT "B(0),B(1) = ";B(0);",";B(1)
PRINT P100 ······················ Displays the P100 value.
```

Related commands	INPUT

76 PSHFRC

Specifies/acquires a pushing thrust parameter.

Format

```
1. PSHFRC [<robot number>] <expression>
2. PSHFRC [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	-1000 to 1000 (unit: %)

Explanation Changes the pushing thrust parameter of the robot axis specified by the <robot number> to the value of <pushing thrust>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
If the F option is omitted in the PUSH statement, the pushing control is executed with the setting of the pushing thrust parameter.
Actual pushing thrust is as follows.
• Rated thrust x <pushing thrust>/100
In format 1, the change occurs at all axes.
In format 2, the change occurs at parameter of the axis specified by the <axis number>.

SAMPLE

```
PSHFRC (1) = 10 ················ Changes the pushing thrust
                                 parameter of axis 1 of robot 1
                                 to 10%
```

Functions

Format

```
PSHFRC [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the value of pushing thrust parameter of the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=PSHFRC (1) ···················· The pushing thrust parameter
                                  of axis 1 of robot 1 is
                                  assigned to variable A.
```

77 PSHJGSP

Specifies/acquires a pushing detection speed threshold parameter.

Format

```
1. PSHJGSP [<robot number>] <expression>
2. PSHJGSP [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	0: Invalid, 1 to 100 (units: %)

Explanation Changes the pushing detection speed threshold parameter of the robot axis specified by the <robot number> to the value indicated in the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

If the pushing detection speed threshold parameter is enabled, a pushing operation is detected only when the movement speed is below <expression> with the pushing thrust in the PUSH statement at the specified value.

The setting of <expression> can be specified as follows.

0: A pushing operation is detected if the pushing thrust reaches the specified value with the threshold setting invalid.

1 to 100: The movement speed in the PUSH statement is 100% to specify thresholds with a rate.

SAMPLE

```
PSHJGSP (1) = 50··············· Changes the pushing detection
                                 speed threshold parameter of
                                 axis 1 of robot 1 to 50%.
```

Functions

Format

```
PSHJGSP [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the value of the pushing detection speed threshold parameter of the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=PSHJGSP (1) ·················· The pushing detection speed
                                 threshold parameter of axis
                                 1 of robot 1 is assigned to
                                 variable A.
```

78 PSHMTD

Specifies/acquires a pushing type parameter.

Format

```
1. PSHMTD [<robot number>] <expression>
2. PSHMTD [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>.....................	1 to 4
<axis number>.......................	1 to 6
<expression>..........................	0: Totalizing method, 1: Resetting method

Explanation Changes the pushing type parameter of the robot axis specified by the <robot number> to the value of the <value>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
The pushing type in the PUSH statement can be specified as follows by the <value>.
0: The time for the pushing thrust to reach the specified value is totalized to execute the pushing control end detection.
1: The pushing control end detection is executed only when the pushing thrust continuously reaches the specified value. If the pushing thrust is lower than the specified value, the elapsed time is reset to 0.
In format 1, the change occurs at all axes.
In format 2, the change occurs at the parameter of the axis specified by the <axis number>.

SAMPLE

```
PSHMTD (1) = 1··················Changes the pushing type
                                parameter of axis 1 of robot 1
                                to the resetting type.
```

Functions

Format

```
PSHMTD [<robot number>] (<axis number>)
```

Values

<robot number>.....................	1 to 4
<axis number>.......................	1 to 6

Explanation Acquires the value of pushing type parameter of the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=PSHMTD (1)····················The pushing type parameter of
                                axis 1 of robot 1 is assigned
                                to variable A.
```

N O P Q R S T U V W X Y Z

79 PSHRSLT

Acquires the status when PUSH statement ends.

Format

```
PSHRSLT [<robot number>] <axis number>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the status at the completion of a "PUSH statement" that was executed for the axis specified by the <axis number> of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

0 ....The PUSH statement was ended by for a reason other than the arrival of the pushing time.

1 ....The PUSH statement was ended by the arrival of the pushing time

SAMPLE

```
PUSH(3,P1) ························ Moves the axis 3 of robot 1 is
                                    under the pushing control to
                                    the position specified with P1.
IF PSHRSLT(3) = 1 THEN ····· Ended by the arrival of the
                                    pushing time
GOTO *OK
ELSE      ·························· Ended for a reason other than
                                    the arrival of the pushing
                                    time
GOTO *NG
ENDIF
```

80 PSHSPD

Specifies/acquires the pushing movement speed parameter.

Format

```
1. PSHSPD [<robot number>] <expression>
2. PSHSPD [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<Pushing speed>	0: Invalid, 1 to 100 (units: %)

Explanation Changes the pushing movement speed parameter of the robot axis specified by the <robot number> to the value indicated in the <expression>. <robot number> can be omitted. If it is omitted, robot 1 is specified.
The motion speed in the PUSH statement is as follows.

- When there is no specification of the S or DS option in the PUSH statement:
  Maximum speed of a robot (mm/sec. or deg./sec.) x Pushing movement speed (%) x Automatic movement speed (%)

- When there is a specification of the S option in the PUSH statement:
  Maximum speed of a robot (mm/sec. or deg./sec.) x Pushing movement speed (%) x Automatic movement speed (%) x Program movement speed (%)

- When there is a specification of the DS option in the PUSH statement:
  Maximum speed of a robot (mm/sec. or deg./sec.) x Pushing movement speed (%) x Movement speed of an axis (%)

SAMPLE

```
PSHSPD (1) = 50 ················ Changes the pushing movement
                                 speed parameter of axis 1 of
                                 robot 1 to 50%.
```

Functions

Format

```
PSHSPD [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the value of the pushing movement speed parameter of the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=PSHSPD (1)···················· The pushing movement speed
                                 parameter of axis 1 of robot 1
                                 is assigned to variable A.
```

81 PSHTIME

Specifies/acquires the pushing time parameter.

Format

```
1. PSHTIME [<robot number>] <expression>
2. PSHTIME [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	1 to 32767 (unit: ms)

Explanation Changes the pushing time parameter of the robot axis specified by the <robot number> to the value indicated in the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
If the TIM option is omitted in the PUSH statement, the pushing control is executed with the setting of the pushing time parameter.
In format 1, the change occurs at all axes.
In format 2, the change occurs at the axis specified by the <axis number>.

SAMPLE

```
PSHTIME (1) = 1000··········· Changes the pushing time
                              parameter of axis 1 of robot 1
                              to 1000ms
```

Functions

Format

```
PSHTIME [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the value of pushing time parameter of the axis specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=PSHTIME (1) ················· The pushing time parameter of
                              axis 1 of robot 1 is assigned
                              to variable A.
```

82 PUSH

Executes a pushing operation for specified axes.

Format

```
PUSH[<robot number>](<axis number> , <expression>) [, option]
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	Motor position (mm, degree, pulse) or point expression

Explanation An absolute position movement of the robot axis specified by the <robot number> is executed while the pushing thrust in the direction of forwarding in the axis unit is controlled. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

- Movement type : Pushing PTP movement of specified axis
- Point data setting : Direct coordinate data input, point definition.
- Options : Pushing thrust setting, pushing time, pushing speed setting, STOPON setting

Movement type

● PTP (point-to-point) of specified axis

PTP movement begins after the operation of the axis specified by the <axis number> is completed (within the tolerance range), controlling the pushing thrust in the forwarding direction of the axis.

The conditions to start the pushing control are as follows.
- Immediately after the start of movement of an axis by the PUSH statement
- After the merge operation is completed (when the PUSH statement is specified in the line next to the movement command with CONT specified)

The conditions to terminate the command are as follows.
- The axis arrives within the tolerance range of the target position.
- The status where the pushing thrust of the axis reaches <pushing thrust value> elapses the time specified to <pushing time value>.

The end status for the PUSH statement can be confirmed with the PSHRSLT statement.

The conditions to cancel the pushing thrust are as follows.
- When a movement command is executed after the PUSH command including STOP is finished
- When a servo off occurs
- When the power source to the controller is interrupted and restarted

82 PUSH

If the next command following to the PUSH statement is an executable command such as a signal output command, the next command will start when the pushing conditions of an axis to be moved are satisfied, or when an axis arrives within the tolerance range of the target position.
Example:

Signal output (DO, etc.)	Signal is output when the pushing conditions are satisfied or within the tolerance range.
DELAY	DELAY command is executed and standby starts, when the pushing conditions are satisfied or within the tolerance range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	When the pushing conditions are satisfied or within the tolerance range, the programs in execution are all stopped, task 1 is reset, and other tasks are terminated. Therefore, axis movement also stops.
HOLD	Program temporarily stops when axis enters the OUT position range. Therefore, axis movement also stops.
HOLDALL	When the pushing conditions are satisfied or within the tolerance range, the programs in execution are all temporarily stopped. Therefore, axis movement also stops.
WAIT	WAIT command is executed, when the pushing conditions are satisfied or within the tolerance range.

SAMPLE

```
PUSH(1,P0) ··················· Axis 1 moves from its current position
                               to the position specified by P0.
```

Point data setting types

● Direct numeric value input

The motor position is specified directly in <expression>.
If the motor position's numeric value is an integer, this is interpreted as a "pulse" unit. If the motor position's numeric value is a real number, this is interpreted as a "mm/degrees" unit, and each axis will move from the 0-pulse position to a pulse-converted position.

SAMPLE

```
PUSH(1,10000) ················ Axis 1 of robot 1 moves from its current
                               position to the 100000 pulse position.
PUSH[2](2,90.00) ············· Axis 2 of robot 2 moves from its
                               current position to the 90° position
                               (when axis 2 is a rotating axis.)
```

● Point definition

Point data is specified in <expressions>. The axis data specified by the <axis number> is used. If the point expression is in "mm/degrees" units, movement for each axis occurs from the 0-pulse position to the pulse-converted position.

SAMPLE

```
PUSH(1,P1) ··················· Axis 1 of robot 1 moves from its current
                               position to the position specified by P1.
PUSH[2](2,P90) ··············· Axis 2 of robot 2 moves from its
                               current position to the position
                               specified by P90 (deg.) (when
                               axis 2 is a rotating axis.)
```

Option types

● Pushing thrust setting

Format

```
F=<expression>
```

Values <expression>..........................-1000 to 1000 (units: %)

Explanation The pushing thrust in the forwarding direction of an axis is specified as an <expression>.
The actual pushing thrust is determined as shown below.
• Rated thrust x <expression>/100
If <pushing thrust value> is omitted, <pushing thrust value> specified with the parameter is used.

SAMPLE

```
PUSH(1,10000),F=10 ········ Axis 1 of robot 1 moves from
                            its current position to the
                            100000 pulse position with the
                            pushing thrust at 10% of the
                            rated thrust.
```

● Pushing time setting

Format

```
TIM=<expression>
```

Values <expression>..........................1 to 32767 (units: ms)

Explanation The time to keep pushing with the specified pushing thrust is specified as an <expression>.
When the status where the pushing thrust reaches the specified value exceeds <expression>, the PUSH statement terminates.
If this option is omitted, the setting of the parameter is used.

SAMPLE

```
PUSH(1,10000),S=10 ········ Axis 1 moves from its current
                            position to the 100000 pulse
                            position with the speed at 10%
                            of the multiplication of the
                            pushing movement speed and
                            the automatic movement speed.
```

● Speed setting

Format

```
1. SPEED=<expression>
2. S=<expression>
```

Values <expression>..........................1 to 100 (units: %)

Explanation The program movement speed is specified in <expression>.
The actual speed is determined as shown below.

- Robot's max. speed (mm/sec, or deg/sec) x pushing movement speed (%) x automatic movement speed (%) x program movement speed (%)

This option is enabled only for the specified PUSH statement.

SAMPLE

```
PUSH(1,10000),S=10 ········ Axis 1 moves from its current
                            position to the 100000 pulse
                            position with the speed at 10%
                            of the multiplication of the
                            pushing movement speed and
                            the automatic movement speed.
```

Format

```
1. DSPEED=<expression>
2. DS=<expression>
```

Values <expression>..........................0.01 to 100.00 (units: %)

Explanation The axis movement speed is specified in <expression>.
The actual speed is determined as shown below.

- Robot's max. speed (mm/sec, or deg/sec) x pushing movement speed (%) x movement speed of an axis (%)

This option is enabled only for the specified PUSH statement.

- Movement always occurs at the DSPEED <expression> value (%) without being affected by the automatic movement speed value (%).

SAMPLE

```
PUSH(1,10000),DS=0.1 ····· Axis 1 moves from its current
                           position to the 100000 pulse
                           position with the speed at
                           0.1% of the pushing movement
                           speed.
```

● **STOPON conditions setting**

Format

```
STOPON <conditional expression>
```

Explanation Stops movement when the conditions specified by the conditional expression are met. **Because this is a deceleration type stop, there will be some movement (during deceleration) after the conditions are met.**
If the conditions are already met before movement begins, no movement occurs, and the command is terminated.
This option is enabled only by program execution.

SAMPLE

```
PUSH(1,10000),STOPON DI(20) = 1
        ·······················Axis 1 moves from its current
                               position toward the "10000
                               pulses" position and stops at
                               an intermediate point if the
                               "DI (20) = 1" condition is met.
                               The next step is then executed.
```

MEMO

- When the conditional expression used to designate the STOPON conditions is a numeric expression, an expression value other than “0” indicates a TRUE status, and “0” indicates a FALSE status.

Related commands	PSHFRC, PSHTIME, PSHMTD, PSHDSP, PSHSPD, PSHRSLT, CURTRQ, CURTQST

83 RADDEG

Performs a unit conversion (radians → degrees)

Format

```
RADDEG(<expression>)
```

Values <expression>..........................Angle (units: radians)

Explanation Converts the <expression> value to degrees.

SAMPLE

```
LOCR(P0)=RADDEG(ATN(B)) ··· Converts the variable B
                            arctangent value to degrees,
                            and assigns it to R-data of
                            P0.
```

Related commands	ATN, COS, DEGRAD, SIN, TAN

84 REM

Inserts a comment

Format

```
1. REM <character string>
2. ' <character string>
```

Explanation All characters which follow REM or an apostrophe (') are handled as a comment. This comment statement is used only to insert comments in the program, and it does not execute any command. The apostrophe (') can be entered at any point in the line.

SAMPLE

```
REM *** MAIN PROGRAM ***
   ' *** SUBROUTINE ***
HALT   ' HALT COMMAND
```

85 RESET

Turns OFF the bits of specified ports, or clears variables

Format 1

```
RESET | DOm([b,···,b])|
      | DO(mb,···,mb)
      | MOm([b,···,b])
      | MO(mb,···,mb)
      | TOn([b,···,b])
      | TO(n-b,···,nb)
      | LOn([b,···,b])
      | LO(nb,···,nb)
      | SOm([b,···,b])
      | SO(mb,···,mb) |
```

Format 2

```
RESET TCOUNTER
```

Values

m: port number	2 to 7, 10 to 17, 20 to 27
n: port number	0, 1
b: bit definition	0 to 7

CAUTION

• Output to ports "0" and "1" is not allowed at DO, and SO.

REFERENCE

• For details regarding bit definitions, see Chapter 3 "10 Bit Settings".

Explanation

Format 1: Turns the bits of specified ports OFF.

Format 2: Clears the 10ms counter variables (10ms counter variables are used to measure the time in 10ms units).

If multiple bits are specified, they are expressed from the left in descending order (large to small).

If the [b,…,b] data is omitted, all 8 bits are processed.

SAMPLE

```
RESET DO2()··················· Turns OFF DO(27 to 20).
RESET DO2(6,5,1) ··········· Turns OFF DO(26, 25, 21).
RESET (37,35,27,20)········ Turns OFF DO(37, 35, 27, 20).
RESET TCOUNTER··············· Clears the 10ms counter
                               variables.
```

Related commands	SET, DO, MO, SO, TO, LO

86 RESTART

Restarts another task during a temporary stop

Format

```
RESTART |Tn
        |"< "<program name>" >"
        |PGm
```

Values m: Program number ....0 to 99
n: Task number ....1 to 16

Explanation Restarts another task that has been temporarily stopped (SUSPEND status).
A task can be specified by the name or the number of a program in execution.

MEMO

- If a task (program) not temporarily stopped is specified and executed, an error occurs.

SAMPLE

```
START <SUB_PGM>,T2
   FLAG=1
*L0:
   IF FLAG=1 AND DI2(0)=1 THEN
      SUSPEND T2
      FLAG=2
   WAIT DI2(0)=0
   ENDIF
   IF FLAG=2 AND DI2(0)=1 THEN
      RESTART T2
      FLAG=1
      WAIT DI2(1)=0
   ENDIF
   MOVE P,P0
   MOVE P,P1
   GOTO *L0
   HALTALL
Program name:SUB_PGM
*SUBPGM:
' SUBTASK ROUTINE
*SUBTASK:
   DO2(0)=1
   DELAY 1000
   DO2(0)=0
   DELAY 1000
   GOTO *SUBPGM
   EXIT TASK
```

Related commands	CUT, EXIT TASK, START, SUSPEND

Reference For details, refer to the "Multi-Task" item.

87 RESUME

Resumes program execution after error recovery processing

Format

```
1. RESUME NEXT
2. RESUME <label>
```

REFERENCE

• For details, see "62 ON ERROR GOTO".

Explanation Resumes program execution after recovery from an error.
Depending on its location, a program can be resumed in the following 3 ways:

1. RESUME	The program resumes from the command which caused the error.
2. RESUME NEXT	The program resumes from the next command after the command which caused the error.
3. RESUME <label>	The program resumes from the command specified by the <label>.

MEMO

- The RESUME statement can also be executed in an error processing routine.
- "Error recovery processing is not possible for serious errors such as "17.4 : Overload", etc.

Related commands	ON ERROR GOTO

88 RETURN

Processing which was branched by GOSUB, is returned to the next line after GOSUB

Format

```
GOSUB <label>                    * GOSUB can also be expressed as "GO SUB".
   :
<label>:
   :
RETURN
```

Explanation Ends the subroutine and returns to the next line after the jump source GOSUB statement.

All subroutines (jump destinations) specified by a GOSUB statement must end with a RETURN statement. Using the GOTO statement, etc., to jump from a subroutine will cause an error such as the "5.12: Stack overflow", etc.

SAMPLE

```
*ST:
   MOVE P,P0
   GOSUB *CLOSEHAND
   MOVE P,P1
   GOSUB *OPENHAND
GOTO *ST
HALT
' SUB ROUTINE
*CLOSEHAND:
   DO(20) = 1
RETURN
*OPENHAND:
   DO(20) = 0
RETURN
```

Related commands	GOSUB

89 RIGHT$

Extracts a character string from the right end of another character string

Format

```
RIGHT$(<character string expression>,<expression>)
```

Values <expression>..........................0 to 75

Explanation This function extracts a character string with the digits specified by the <expression> from the right end of the character string specified by <character string expression>.
The <expression> value must be between 0 and 75, otherwise an error will occur.
If the <expression> value is 0, then RIGHT$ will be a null string (empty character string).
If the <expression> value has more characters than the <character string expression>, RIGHT$ will become the same as the <character string expression>.

SAMPLE

```
B$=RIGHT$(A$,4) ················ 4 characters from the right
                                 end of A$ are assigned to B$.
```

Related commands	LEFT$, MID$

90 RIGHTY

Sets the SCARA robot hand system to "Right"

Format

```
RIGHTY [<robot number>]
```

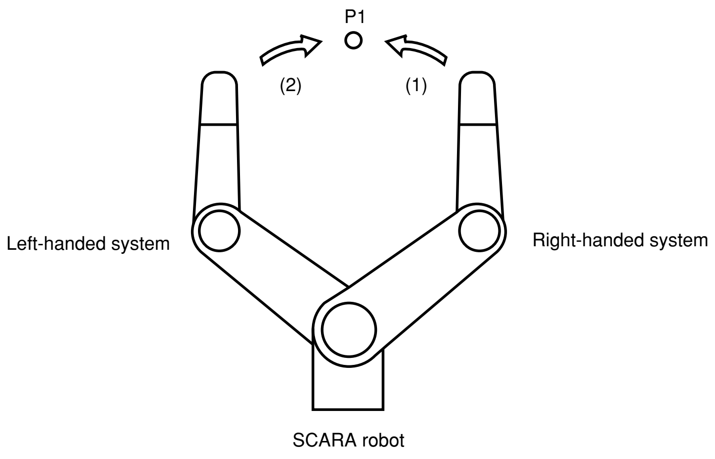
Values <robot number>....................1 to 4

Explanation This statement specifies the robot with <robot number> by the right-handed movement to a point specified in the Cartesian coordinate. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
This statement only selects the hand system, and does not move the robot. If executed while the robot arm is moving, execution waits until movement is complete (positioned within tolerance range).
This command is only valid for SCARA robots.

SAMPLE

```
RIGHTY   .......................... Specifies a Robot 1 “right-
                                    handed system” setting.(see
                                    Fig.1 below).
MOVE P,P1
LEFTY    .......................... Specifies a Robot 1 “left-
                                    handed system” setting.(see
                                    Fig.2 below).
MOVE P,P1
RIGHTY
HALT
```

SAMPLE:LEFTY/RIGHTY



33818-R7-00

Related commands LEFTY

91 RSHIFT

Shifts a bit value to the right

Format

```
RSHIFT(<expression 1>,<expression 2>)
```

Explanation Shifts the <expression 1> bit value to the right by the amount of <expression 2>. Spaces left blank by the shift are filled with zeros (0).

SAMPLE

```
A=RSHIFT(&B10111011,2)······The 2-bit-right-shifted
                            &B10111011 value (&B00101110)
                            is assigned to A.
```

Related commands	LSHIFT

92 SELECT CASE to END SELECT

Executes the specified command block in accordance with the <expression> value

Format

```
SELECT [CASE] <expression>
   CASE <expression list 1>
      [command block 1]
   [CASE <expression list 2>
      [command block 2]]
      :
   [CASE ELSE
      [command block n]]
END SELECT
```

Explanation These statements execute multiple command blocks in accordance with the <expression> value. The setting method is as follows.

1. The <expression list> following CASE statement comprises multiple numerical expressions and character expressions separated from each other by a comma (,).
2. If the <expression> value matches one of expressions contained in the <expression list>, the specified command block is executed. After executing the command block, the program jumps to the next command which follows the END SELECT statement.
3. If the <expression> value does not match any of the expressions contained in the <expression list>, the command block indicated after the CASE ELSE statement is executed. After executing the command block, the program jumps to the next command which follows the END SELECT statement.
4. If the <expression> value does not match any of the expressions contained in <expression list> and no CASE ELSE statement exists, the program jumps to the next command following the END SELECT statement.

SAMPLE

```
WHILE -1
SELECT CASE DI3()
   CASE 1,2,3
      CALL *EXEC(1,10)
   CASE 4,5,6,7,8,9,10
      CALL *EXEC(11,20)
   CASE ELSE
      CALL *EXEC(21,30)
END SELECT
WEND
HALT
```

93 SEND

Sends <read file> data to the <write file>

Format

```
SEND <read file> TO <write file>
```

NOTE

- Examples of erroneous writing to a read-only file:
  SEND CMU TO DIR
  SEND PNT TO SI()
- Examples of data format mismatches:
  SEND PGM TO PNT
  SEND SI() TO SFT

Explanation Sends <read file> data to the <write file>.
An entire DO, MO, TO, LO, SO, or SOW port (DO(), MO(), etc.), cannot be specified as a write file.
Moreover, some individual files (DOn(), MOn(), etc.) cannot be specified as a write file. For details, refer to Chapter 8 "Data file description".
Writing to read-only files (indicated by a "×" in the "WRITE" column of the table shown below) is not permitted.
Even if the READ and WRITE files are specified correctly, it may not be possible to execute them if there is a data format mismatch between the files.

Type	File Name	Definition Format		READ	WRITE
		All	Individual File		
User	All files	ALL	———	○	○
	Program	PGM	<bbbbbbbb>	○	○
	Point	PNT	Pn	○	○
	Point comment	PCM	PCn	○	○
	Parameter	PRM	/cccccc/	○	○
	Shift definition	SFT	Sn	○	○
	Hand definition	HND	Hn	○	○
	Pallet definition	PLT	PLn	○	○
Variable, Constant	Variable	VAR	ab...by	○	○
	Array variable	ARY	ab...by(x)	○	○
	Constant		“cc...c”	○	×
Status	Program directory	DIR	<<bbbbbbbb>>	○	×
	Parameter directory	DPM	———	○	×
	Machine reference	MRF	———	○	×
	Error log	LOG	———	○	×
	Remaining memory size	MEM	———	○	×
Device	DI port	DI()	DIn()	○	×
	DO port	DO()	DOn()	○	○
	MO port	MO()	MOn()	○	○
	TO port	TO()	TOn()	○	○
	LO port	LO()	LOn()	○	○
	SI port	SI()	SIn()	○	×
	SO port	SO()	SOn()	○	○
	SIW port	SIW()	SIWn()	○	×
	SOW port	SOW()	SOWn()	○	○
	RS-232C	CMU	———	○	○
	Ethernet	ETH	———	○	○
Other	File END code	EOF	———	○	×

N: Number a: Alphabetic character b: Alphanumeric character or underscore (_) ○ : Permitted
c: Alphanumeric character or special symbol x: Expression (array argument) y: Variable type × : Not permitted

MEMO

- The following cautions apply when a restart is performed after a stop occurred during execution of the SEND statement:
  1. When reading from RS-232C / Ethernet (SEND CMU TO XXX, SEND ETH TO XXX): When the SEND statement is stopped during data reading from the reception buffer, the data acquired up to that point is discarded.
  2. When writing to RS-232C / Ethernet (SEND XXX TO CMU, SEND XXX TO ETH): When the SEND statement is stopped during data writing to the transmission buffer, the data is written from the beginning.

SAMPLE

```
SEND PGM TO CMU················Outputs all user programs from
                               the RS-232C port.
SEND <PRG1> TO CMU·············Outputs the PRG1 program from
                               the RS-232C port.
SEND CMU TO PNT················Inputs a point data file from
                               the RS-232C port.
SEND "T1" TO CMU···············Outputs the "T1" character
                               string from the RS-232C port.
SEND CMU TO A$ ················Inputs character string data
                               to variable A$ from the RS-
                               232C port.
```

Reference For details, refer to Chapter 8 "Data file description".

94 SERVO

Controls the servo status

Format

```
SERVO [<robot number>]  | ON   | [(<axis number>)]
                        | OFF  |
                        | FREE |
                        | PWR  |
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation This statement controls the servo ON/OFF at the specified axes or all axes. When the <axis number> is specified, only the specified axis becomes an object. When not specified, all axes become an object.

- ON ......Turns the servo ON.
- OFF......Turns the servo OFF and applies the dynamic brake. Axes equipped with brakes are all locked by the brake.
- FREE.....Turns the servo OFF and releases the dynamic brake. The brakes are released at all axes with brakes.

CAUTION

- Keep out of the robot movement range while the motor power is turned OFF by the SERVO OFF statement. Always check that the Emergency Stop is ON when working within the robot movement area.

MEMO

- This statement is executed after the operation of all axes of the specified robot has been complete (after positioned within the tolerance).

SAMPLE

```
SERVO ON ·························· Turns servos ON at all axes
SERVO OFF ························· Axes equipped with brakes that
                                    turns the servos at all axes
                                    OFF are all locked by the
                                    brake.
SERVO FREE(3) ····················· The axis 3 (Z-axis) servo is turned
                                    OFF, and the brake is released.
```

95 SET

Turns the bit at the specified output port ON

Format

```
SET | DOm([b,···,b])|  [, <time>]
    | DO (mb,···,mb)|
    | MOm([b,···,b])|
    | MO (mb,···,mb)|
    | TOn([b,···,b])|
    | TO (nb,···,nb)|
    | LOn([b,···,b])|
    | LO (nb,···,nb)|
    | SOm([b,···,b])|
    | SO (mb,···,mb)|
```

Values

m: port number	2 to 7, 10 to 17, 20 to 27
n: port number	0, 1
b: bit definition	0 to 7
<time>	10 to 3600000 (units: ms)

CAUTION

- Output to ports "0" and "1" are not allowed at DO, and SO.

REFERENCE

- For bit setting details, see Chapter 3 "10 Bit Settings".

Explanation Turns ON the bits of specified ports.

The pulse output time (unit: ms) is specified by the <time> value. When the specified time elapses, the output is turned OFF, and command execution ends.

If multiple bits are specified, they are expressed from the left in descending order (large to small).

If the [b,…,b] data is omitted, all 8 bits are processed.

If no hardware port exists, nothing is output.

SAMPLE

```
SET DO2() ························ Turns ON DO(27 to 20).
SET DO2(6,5,1),200 ··········· DO(26,25,21) switches ON for
                               200ms.
SET DO(37,35,27,20) ········· Turns DO(37, 35, 27, 20) ON.
```

Related commands	RESET, DO, MO, SO, TO, LO

96 SHARED

Enables sub-procedure referencing without passing on the variable

Format

```
SHARED <variable>[()][,<variable>[()]... ]
```

NOTE

• The program level code is a program written outside the sub-procedure.

Explanation This statement allows variables declared with a program level code to be referenced with a sub-procedure without passing on the variables as dummy arguments.
The program level variable used by the sub-procedure is specified by the <variable> value.
A simple variable or an array variable followed by parentheses is specified. If an array is specified, that entire array is selected.

MEMO

• Normally, a <dummy argument> passes along the variable to a sub-procedure, but the SHARED statement allows referencing to occur without passing along the variable.
• The SHARED statement allows variables to be shared only between a program level code and sub-procedure which are within the same program level.

SAMPLE

```
DIM Y!(10)
X!=2. 5
Y!(10)=1. 2
CALL *DISTANCE
CALL *AREA
HALT
SUB *DISTANCE
   SHARED X!,Y!( ) ··········Variable referencing is
                             declared by SHARED.
   PRINT X!^2+Y!(10)^2······The variable is shared.
END SUB
SUB *AREA
   DIM Y!(10)
   PRINT X!*Y!(10)··········The variable is not shared.
END SUB
```

Related commands	SUB, END SUB

97 SHIFT

Sets the shift coordinates

Format

```
SHIFT<robot number>]    <shift variable>
                        OFF
```

Values <robot number>.....................1 to 4

Explanation Sets the shift coordinates in accordance with the shift data specified by the <shift variable> to the robot specified by the <robot number>.

MEMO

- This statement is executed after axis positioning is complete (within the tolerance range).

SAMPLE

```
SHIFT S1
MOVE P,P10
SHIFT S[A]
MOVE P,P20
HALT
```

Related commands	Shift definition statement, shift assignment statement

98 SIN

Acquires the sine value for a specified value

Format

```
SIN(<expression>)
```

Values <expression>	Angle (units: radians)

Explanation This function gives the sine value for the <expression> value.

SAMPLE

```
A(0)=SIN(B*2+C) ················ Assigns the expression B*2+C
                                 sine value to array A (0).
A(1)=SIN(DEGRAD(30)) ········ Assigns a 30.0° sine value to
                                 array A (1).
```

Related commands	ATN, COS, DEGRAD, RADDEG, TAN

99 Sn

Defines the shift coordinates in the program

Format

```
Sn = x y z r
```

Values

n	0 to 39
x, y, z, r	-99,999.99 to 99,999.99

NOTE

- All input values are handled as constants.
- If the controller power is turned off during execution of a shift coordinate definition statement, a memory-related error such as "9.6: Shift check-sum error" may occur.

Explanation Defines shift coordinate values in order to shift the coordinates for robot movement. Only "mm" units can be used for these coordinate values ("pulse" units cannot be used).

1. "n" indicates the shift number.
2. The "x" to "r" input data must be separated with spaces (blanks).
3. The "x" to "r" input data is recognized as "mm" unit data.
4. "x" to "z" correspond to the Cartesian coordinate system's x, y, z coordinate shift values, and "r" corresponds to the xy coordinates' rotational shift values.

SAMPLE

```
S0 =    0.00    0.00    0.00    0.00
S1 =  100.00  200.00   50.00   90.00
P3 =  100.00    0.00    0.00    0.00    0.00    0.00
SHIFT S0
MOVE P,P3
SHIFT S1
MOVE P,P3
HALT
```

Related commands	Shift assignment statement, SHIFT

100 SO

Outputs a specified value to the serial port

Format

```
1. [LET]SOm([b,···,b]) =<expression>
2. [LET]SO (mb,···,mb) =<expression>
```

Values

m: port number	2 to 7, 10 to 17, 20 to 27
b: bit definition	0 to 7

CAUTION

• Outputs to SO0() and SO1() are not possible.

REFERENCE

• For bit setting details, see Chapter 3 "10 Bit Settings".

Explanation Outputs a specified value to the SO port.
Only the <value> data's integer-converted lower bits corresponding to the bits defined at the left side can be output.
If multiple bits are specified, they are expressed from the left in descending order (large to small).
If the [b,…,b] data is omitted, all 8 bits are processed.
If no hardware port exists, nothing is output.

SAMPLE

```
SO2()=&B10111000 ·············· SO (27, 25, 24, 23) are turned
                                 ON, and SO (26, 22, 21, 20)
                                 are turned OFF.
SO2(6,5,1)=&B010 ············· SO (25) are turned ON, and SO
                                 (26, 21) are turned OFF.
SO3()=15 ······················· SO (33, 32, 31, 30) are turned
                                 ON, and SO (37, 36, 35, 34)
                                 are turned OFF.
SO(37,35,27,20)=A············· The lower 4 bits of integer-
                                 converted variable A are
                                 output to SO (37, 35, 27, 20).
```

Related commands	RESET, SET

101 SPEED

Changes the program movement speed

Format

```
SPEED [<robot number>] <expression>
```

Values

<robot number>	1 to 4
<expression>	1 to 100 (units: %)

Explanation Changes the program movement speed value of the robot specified by the <robot number> to the speed indicated by the <expression>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

This speed change applies to all the robot axes and auxiliary axes of the specified robot.

The operation speed is determined by multiplying the automatic movement speed (specified from the programming box and by the ASPEED command), by the program movement speed (specified by SPEED command, etc.).

Operation speed = automatic movement speed x program movement speed.

Example:

Automatic movement speed ... 80%

Program movement speed ... 50%

Movement speed = 40% (80% × 50%)

NOTE

- Automatic movement speed

  Specified by programming box operation or by the ASPEED command.

- Program movement speed

  Specified by SPEED commands or MOVE, DRIVE speed options.

Functions

Format

```
SPEED [<robot number>]
```

Values

<robot number>	1 to 4

Explanation Acquires the program movement speed value of a robot specified by the <robot number>.

The <robot number> can be omitted. When the <robot number> is omitted, the robot 1 is specified.

SAMPLE

```
ASPEED 100
SPEED 70
MOVE P,P0 ························ Moves from current position to
                                   P0 at a speed of 70% (=100 * 70).
SPEED 50
MOVE P, P1······················· Moves from current position to
                                   P1 at a speed of 50% (=100 * 50).
MOVE P,P2, S=10 ················· Moves from current position to
                                   P2 at a speed of 10% (=100 * 10).
HALT
```

Related commands	ASPEED

102 SQR

Acquires the square root of a specified value

Format

```
SQR(<expression>)
```

Values <expression>..........................0 or positive number.

Explanation Gives the square root of the <expression> value. An error occurs if the <expression> value is a negative number.

SAMPLE

```
A=SQR(X^2+Y^2)..................The square root of X^2+Y^2 is
                                assigned to variable A.
```

103 START

Starts a new task

Format

```
START  │ “< “<program name>” >”│[,Tn [, p]]
       │ PGm                   │
```

Values

m: Program number	0 to 99
n: Task number	1 to 16
p: Task priority ranking	1 to 64

Explanation Starts task “n” specified by the program with the “p” priority ranking.

If task number “n” is omitted, the task with the smallest number among the tasks yet to be started is automatically specified.

If a priority ranking is not specified, "32" is adopted as the priority ranking for this task.

The smaller the priority number, the higher the priority (high priority: 1 ↔ low priority: 64).

When a RUNNING status occurs at a task with higher priority, all tasks with lower priority also remain in a READY status.

SAMPLE

```
START <SUB_PGM>,T2,33
*ST:
   MOVE P,P0,P1
GOTO *ST
HALT
Program name:SUB_PGM
*SUBPGM:
‘SUBTASK ROUTINE
*SUBTASK:
   P100 = WHERE
   IF LOCZ(P100) > 10000 THEN
      DO(20) = 1
   ELSE
      DO(20) = 0
   ENDIF
GOTO *SUBPGM
EXIT TASK
```

Related commands	CUT, EXIT TASK, RESTART, SUSPEND, CHGPRI

104 STR$

Converts a numeric value to a character string

Format

```
STR$(<expression>)
```

Explanation Converts the value specified by the <expression> to a character string. The <expression> specifies an integer or real number value.

SAMPLE

```
B$=STR$(10.01)
```

Related commands	VAL

105 SUB to END SUB

Defines a sub-procedure

Format

```
SUB <label> [(<dummy argument> [, <dummy argument> ...])]
   <command block>
END SUB
```

Explanation Defines a sub-procedure.
The sub-procedure can be executed by a CALL statement. When the END SUB statement is executed, the program jumps to the next command after the CALL statement that was called. Definitions are as follows.

1. All variables declared within the sub-procedure are local variables, and these are valid only within the sub-procedure. Local variables are initialized each time the sub-procedure is called up.
2. Use a SHARED statement in order to use global variables (program level).
3. Use a <dummy argument> when variables are to be passed on. If two or more dummy arguments are used, separate them by a comma (,).
4. A valid <dummy argument> consists of a name of variable and an entire array (array name followed by parentheses). An error will occur if array elements (a <subscript> following the array name) are specified.

MEMO

- Sub-procedures cannot be defined within a sub-procedure.
- A label can be defined within a sub-procedure, but it cannot jump (by a GOTO or GOSUB statement) to a label outside the sub-procedure.
- Local variables cannot be used with PRINT and SEND statements.

SAMPLE 1

```
A=1
CALL *TEST
PRINT A
HALT
' SUB ROUTINE: TEST
SUB *TEST
   A=50  ........................ Handled as a different
                                  variable than the "A" shown
                                  above.
END SUB
```

MEMO

- In the above example, the program level variable "A" is unrelated to the variable "A" within the sub-procedure. Therefore, the value indicated in the 3rd line PRINT statement becomes "1".

SAMPLE 2

```
X% = 4
Y% = 5
CALL *COMPARE( REF X%, REF Y% )
PRINT X%,Y%
Z% = 7
W% = 2
CALL *COMPARE( REF Z%, REF W% )
PRINT Z%,W%
HALT
' SUB ROUTINE: COMPARE
SUB *COMPARE( A%, B% )
   IF A% < B% THEN
      TEMP% = A%
      A% = B%
      B% = TEMP%
   ENDIF
END SUB
```

MEMO

- In the above example, different variables are passed along as arguments to call the sub-procedure 2 times.

Related commands	CALL, EXIT SUB, SHARED

106 SUSPEND

Temporarily stops another task which is being executed

Format

```
SUSPEND | Tn                       |
        | “< “<program name>” >”   |
        | PGm                      |
```

Values

m: Program number	0 to 99
n: Task number	1 to 16

Explanation Temporarily stops (suspends) another task which is being executed. A task can be specified by the name or the number of a program in execution.
This statement can also be used for tasks with a higher priority ranking than this task itself.

MEMO

- If a task (program) not active is specified for the execution, an error occurs.

SAMPLE

```
START <SUB_PGM>,T2
SUSFLG=0
*L0:
   MOVE P,P0
   MOVE P,P1
   WAIT SUSFLG=1
   SUSPEND T2
   SUSFLG=0
GOTO *L0
HALT
Program name:SUB_PGM
*SUBPGM:
’ SUBTASK ROUTINE
*SUBTASK:
   WAIT SUSFLG=0
   DO2(0)=1
   DELAY 1000
   DO2(0)=0
   DELAY 1000
   SUSFLG=1
   GOTO *SUBPGM
   EXIT TASK
```

Related commands	CUT, EXIT TASK, RESTART, SUSPEND

107 SWI

Switches the program being executed

Format

```
SWI "<"<program name>">"
```

Explanation This statement switches from the current program to the specified program, starting from the first line.
Although the output variable status is not changed when the program is switched, the dynamic variables and array variables are cleared. The program name to be switched to must be enclosed in angular brackets (< >).

MEMO

- If the program specified as the switching target does not exist, a "3.3: Program doesn't exist" (code: &H0303) error occurs and operation stops.

SAMPLE

```
SWI <ABC> ························ Switches the execution program
                                   to "ABC".
```

108 TAN

Acquires the tangent value for a specified value

Format

```
TAN(<expression>)
```

Values <expression>..........................Angle (units: radians)

Explanation Gives a tangent value for the <expression> value. An error will occur if the <expression> value is a negative number.

SAMPLE

```
A(0)=B-TAN(C) ··················· The difference between the
                                  tangent values of variable B
                                  and variable C is assigned to
                                  array A (0).
A(1)=TAN(DEGRAD(20)) ········ The 20.0 ° tangent value is
                                  assigned to array A (1).
```

Related commands ATN, COS, DEGRAD, RADDEG, SIN

109 TCOUNTER

Timer & counter

Format

```
TCOUNTER
```

Explanation Outputs count-up values at 1ms intervals starting from the point when the TCOUNTER variable is reset.
After counting up to 65,535, the count is reset to 0.

SAMPLE

```
MOVE P,P0
WAIT ARM
RESET TCOUNTER
MOVE P,P1
WAIT ARM
A = TCOUNTER
PRINT TCOUNTER ················ Displays the P0 to P1 movement
                                time at the programming box
                                until movement enters the
                                tolerance range.
```

Related commands	RESET

110 TIME$

Acquires the current time

Format

```
TIME$
```

Explanation Acquires the current time in an hh:mm:ss format character string. "hh" is the hour, "mm" is the minutes, and "ss" is the seconds. The clock can be set in the SYSTEM mode's initial processing.

SAMPLE

```
A$=TIME$
PRINT TIME$
```

Related commands	DATE$, TIMER

111 TIMER

Acquires the current time

CAUTION

• The time indicated by the internal clock may differ somewhat from the actual time.

Format

```
TIMER
```

Functions Acquires the current time in seconds, counting from 12:00 midnight. This function is used to measure a program's run time, etc.
The clock can be set in the SYSTEM mode's initial processing.

SAMPLE

```
A%=TIMER
FOR B=1 TO 10
MOVE P,P0
MOVE P,P1
NEXT
A%=TIMER-A%
PRINT A%/60;":";A% MOD 60
HALT
```

Related commands	TIME$

112 TO

Outputs a specified value to the TO port

Format

```
1. [LET]TOm([b,···,b]) =<expression>
2. [LET]TO (mb,···,mb) =<expression>
```

Values	
m: port number	0, 1
b: bit definition	0 to 7

Explanation Outputs the specified value to the TO port. The output value is the expression's integer-converted lower bits corresponding to the bit definition specified at the left side.

If multiple bits are specified, they are expressed from the left in descending order (large to small).

If the [b,…,b] data is omitted, all 8 bits are processed.

The OFF/ON settings for bits which are being used in a SEQUENCE program have priority while the SEQUENCE program is running.

SAMPLE

```
TO0() = &B00000110
```

Related commands RESET, SET

113 TOLE

Specifies/acquires the tolerance parameter

Format

```
1. TOLE [<robot number>] <expression>
2. TOLE [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	Varies according to the motor which has been specified (unit: pulse)

Explanation Change the tolerance parameter of the robot axis specified by the <robot number> to the <expression> value (unit: pulse). The <robot number> can be omitted. If it is omitted, robot 1 is specified.
Format 1: The change is applied to all axes of the specified robot.
Format 2: The change is applied to only the axis specified by the <axis number> of the specified robot.

MEMO

- If an axis that is set to "no axis" in the system generation is specified, a "5.37: Specification mismatch" error occurs and the command execution is stopped.
- This statement is executed after positioning of the specified axes is complete (within the tolerance range).

Functions

Format

```
TOLE [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the tolerance parameter values for the axis-specified by the <axis number> among the robot axes specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
' CYCLE WITH DECREASING TOLERANCE
DIM TOLE(5)
FOR A=200 TO 80 STEP -20
   GOSUB *CHANGE_TOLE
   MOVE P,P0
   MOVE P,P1
NEXT A
C=TOLE(2) ························ The tolerance parameter of axis 2 of
                                   robot 1 is assigned to variable C.
HALT
*CHANGE_TOLE:
FOR B=1 TO 4
   TOLE(B)=A
NEXT B
RETURN
```

114 TORQUE

Specifies/acquires the maximum torque command value which can be set for a specified axis

Format

```
TORQUE [<robot number>] (<axis number>) = <expression>
```

Values

<robot number>	1 to 4
<axis number>	1 to 6
<expression>	1 to 100 (units: %)

Explanation Changes the maximum torque command value of the axis specified by the <axis number> of the robot specified by the <robot number> to the value of <expression>. The new value is enabled after the next movement command (MOVE or DRIVE statement, etc.) is executed. The torque parameter value does not change.

The conditions to cancel a torque limit are as follows.
- The TORQUE command for the same axis is executed.
- The controller power turned off and then on again.
- The axis polarity parameter is changed or a parameter is initialized.
- The server is turned off.

The conditions to temporarily change a torque limit are as follows.
- A return- to-origin is in execution.
- The PUSH statement is in execution (only the direction of a movement is changed).

The torque limit returns to the specified value when a movement command such as the next MOVE statement is executed.

CAUTION
- If the specified torque limit is too small, the axis may not move. In this case, press the emergency stop button before proceeding with the operation.
- If the specified value is less than the rated torque, an error may not occur even if the robot strikes an obstacle.

MEMO
- If an axis that is set to "no axis" in the system generation is specified, a "5.37 : Specification mismatch" error message displays and command execution is stopped.

Functions

Format

```
TORQUE [<robot number>] (<axis number>)
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Acquires the torque setting value for the axis specified by the <axis number> of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

114 TORQUE

SAMPLE

```
TORQUE (1) = 50 ················ Changes the max. torque of
                                 axis 1 of robot 1 to 50%.
DRIVE (1,P1)···················· Moves the axis 1 of robot 1
                                 from its current position to
                                 the point specified by P1.
                                 (Change the max. torque at
                                 the same time with the start
                                 of the movement.)
WAIT ARM ························ Waits for the completion of an
                                 operation of axis 1 of robot 1.
TORQUE (1) = 100··············· Returns the max. torque of
                                 axis 1 of robot 1 to the
                                 original value (100%).
MOVE P,P0 ······················· Moves the robot 1 from its
                                 current position to the point
                                 specified with P0.
                                 (Returns the max. torque of
                                 axis 1 to the original value
                                 (100%) at the same time with
                                 the start of a movement.)
```

Related commands	CURTRQ, PUSH

115 VAL

Converts character strings to numeric values

Format

```
VAL (<character string expression>)
```

Explanation Converts the numeric value of the character string specified in the <character string expression> into an actual numeric value.
The value may be expressed in integer format (binary, decimal, hexadecimal), or real number format (decimal point format, exponential format).
The VAL value becomes "0" if the first character of the character string is "+", "-", "&" or anything other than a numeric character.
If there are non-numeric characters or spaces elsewhere in the character string, all subsequent characters are ignored by this function.
However, for hexadecimal expressions, A to F are considered numeric characters.

SAMPLE

```
A=VAL("&B100001")
```

116 WAIT

Waits until the conditions of the DI/DO conditional expression are met

Format

```
WAIT <conditional expression> [,<expression>]
```

Values <expression>..........................1 to 3600000 (units: ms)

Explanation Establishes a "wait" status until the condition specified by the <conditional expression> is met. Specify the time-out period (unit: ms) in the <expression>.
If a time-out period has been specified, this command terminates if the time-out period elapses before the WAIT condition is met. The minimum wait time is 1ms but changes depending on the execution status of other tasks.

MEMO

- When the conditional expression is a numeric expression, an expression value other than “0” indicates a TRUE status, and “0” indicates a FALSE status.

SAMPLE

```
WAIT A=10 ························ A wait status continues until
                                   variable A becomes 10.
WAIT DI2( )=&B01010110 ······ Waits until DI(21),(22),(24),(26)
                                   are turned on, and
                                   DI(20),(23),(25),(27) is turned
                                   off.
WAIT DI2(4,3,2)=&B101 ······· Waits until DI(22) and DI(24)
                                   are turned on, and DI(23) is
                                   turned off.
WAIT DI(31)=1 OR DO(21)=1··· A wait status continues until
                                   either DI (31) or DO(21) turns ON.
WAIT DI(20)=1,1000 ··········· A wait status continues until
                                   DI(20) turns ON. If DI(20) fails
                                   to turn ON within 1 second, the
                                   command is terminated.
```

Related commands	DRIVE, DRIVEI, MOVE, MOVEI

117 WAIT ARM

Waits until the robot axis operation is completed

Format

```
WAIT ARM [<robot number>] [(<axis number>)]
```

Values

<robot number>	1 to 4
<axis number>	1 to 6

Explanation Establishes a "wait" status until axis movement of the robot specified by the <robot number> is completed (within the positioning tolerance range.)The <robot number> can be omitted. If it is omitted, robot 1 is specified.

If a specific axis of the specified robot has been specified by the <axis number>, this command will apply only to that axis. If there is no <axis number> setting, this command applies to all axes of the specified robot.

SAMPLE

```
WAIT ARM ························Waits for the movement
                                 completion of robot 1.
WAIT ARM[2](2)···················Waits for the movement
                                 completion of axis 2 of robot 2.
```

Related commands	DRIVE, DRIVEI, MOVE, MOVEI

118 WEIGHT

Specifies/acquires the tip weight parameter

Format

```
WEIGHT [<robot number>] <expression>
```

Values

<robot number>	1 to 4
<expression>	The range varies according to the robot which has been specified.

Explanation Changes the tip weight parameter of the robot specified by the <robot number> to the <expression> value. The <robot number> can be omitted. If it is omitted, robot 1 is specified. This change does not apply to auxiliary axes.

Functions

Format

```
WEIGHT [<robot number>]
```

Values

<robot number>	1 to 4

Explanation Acquires the tip weight parameter value of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
A=5
B=2
C=WEIGHT
WEIGHT A
MOVE P,P0
WEIGHT B
MOVE P,P1
WEIGHT C
D=WEIGHT ························ The tip weight parameter
                                   of robot 1 is assigned to
                                   variable D.
HALT
```

119 WEND

Ends the WHILE statement's command block

Format

```
WHILE <conditional expression>
   <command block>
WEND
```

Explanation Ends the command block which begins with the WHILE statement. A WEND statement must always be paired with a WHILE statement.
Jumping out of the WHILE to WEND loop is possible by using the GOTO statement, etc.

SAMPLE

```
A=0
WHILE DI3(0)=0
   A=A+1
   MOVE P,P0
   MOVE P,P1
   PRINT "COUNTER=";A
WEND
HALT
```

Related commands	WHILE

120 WHERE

Acquires the arm's current position (pulse coordinates)

Format

```
WHERE [<robot number>]
```

Values <robot number>.....................1 to 4

Explanation Acquires the arm's current position of the robot specified by the <robot number> in joint coordinates. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
P10=WHERE ························ The current position's pulse
                                   coordinate value is assigned
                                   to P10.
```

Related commands	WHRXY

121 WHILE to WEND

Repeats an operation for as long as a condition is met

Format

```
WHILE <conditional expression>
   <command block>
WEND
```

Explanation Executes the command block between the WHILE and WEND statements when the condition specified by the <conditional expression> is met, and then returns to the WHILE statement to repeat the same operation.
When the <conditional expression> condition is no longer met (becomes false), the program jumps to the next command after the WEND statement.
If the <conditional expression> condition is not met from the beginning (false), the command block between the WHILE and WEND statements is not executed, and a jump occurs to the next statement after the WEND statement.
Jumping out of the WHILE to WEND loop is possible by using the GOTO statement, etc.

MEMO

- When the conditional expression is a numeric expression, an expression value other than "0" indicates a TRUE status, and "0" indicates a FALSE status.

SAMPLE 1

```
A=0
WHILE DI3(0)=0
   A=A+1
   MOVE P,P0
   MOVE P,P1
   PRINT "COUNTER=";A
WEND
HALT
```

SAMPLE 2

```
A=0
WHILE -1 ························ Becomes an endless loop
                                 because the conditional
                                 expression is always TRUE
                                 (other than 0).
   A=A+1
   MOVE P,P0
   IF DI3(0)=1 THEN *END
   MOVE P,P1
   PRINT "COUNTER=";A
   IF DI3(0)=1 THEN *END
WEND
*END
HALT
```

122 WHRXY

Acquires the arm's current position in Cartesian coordinates

Format

```
WHRXY [<robot number>]
```

Values <robot number>.....................1 to 4

Explanation Acquires the arm's current position of the robot specified by the <robot number> in Cartesian coordinates. The <robot number> can be omitted. If it is omitted, robot 1 is specified.
On YK500TW model robots, the X-arm and Y-arm rotation information is also set.

SAMPLE

```
P10=WHRXY ························ The current position Cartesian
                                   coordinate value of robot 1 is
                                   assigned to P10.
```

Related commands	WHERE

123 XYTOJ

Converts the Cartesian coordinate data ("mm") to joint coordinate data ("pulse")

Format

```
XYTOJ [<robot number>] (<point expression>)
```

Values <robot number>.....................1 to 4

Explanation This function converts the Cartesian coordinate data (unit: mm, deg.) specified by the <point expression> to joint coordinate data (unit: pulse) of the robot specified by the <robot number>. The <robot number> can be omitted. If it is omitted, robot 1 is specified.

- When the command is executed, the data is converted based on the standard coordinates, shift coordinates and hand definition that were set.
- On SCARA robots, the converted result differs depending on whether right-handed or left-handed is specified.
- On the YK500TW model robot, the result varies, depending on the X-arm and Y-arm rotation information settings.
- To convert joint coordinate data to Cartesian coordinate data, use the JTOXY statement.

SAMPLE

```
P10=XYTOJ(P10) ·················· P10 is converted to joint
                                  coordinate data.
```

Chapter 8
Data file description

1 Overview

1.1 Data file types

This section explains data files used with a SEND statement and READ/WRITE online commands. There are 25 different types of data files.

1. Program file
2. Point file
3. Point comment file
4. Parameter file
5. Shift coordinate definition file
6. Hand definition file
7. Pallet definition file
8. All file
9. Program directory file
10. Parameter directory file
11. Variable file
12. Constant file
13. Array variable file
14. DI file
15. DO file
16. MO file
17. LO file
18. TO file
19. SI file
20. SO file
21. EOF file
22. Serial port communication file
23. SIW file
24. SOW file
25. Ethernet port communication file

1.2 Cautions

Observe the following cautions when handling data files.

- Only 1-byte characters can be used.
- All data is handled as character strings conforming to ASCII character codes.
- Only upper case alphabetic characters may be used in command statements (lower case characters are prohibited).
- Line lengths must not exceed 255 characters.
- A [cr/lf] data format designation indicates CR code (0Dh) + LF code (0Ah).
- The terms "reading" and "writing" used in this manual indicate the following data flow directions:
  Reading: controller → external communication device
  Writing: External communication device → controller

2 Program file

2.1 All programs

Format

```
PGM
```

Meaning

- Expresses all programs.
- When used as a readout file, all programs currently stored are read out.
- Write files are registered at the controller under the program name indicated at the NAME = <program name> line.
- If there is a specification of a program number in the case of a write file, the new program overwrites.
- If the program number is omitted in the case of a write file, the assignment is made to the smallest free number. If there are programs with the same name and with different program numbers, the oldest program is deleted.

DATA FORMAT

```
NAME = <program name> [cr/lf]
PGN=mmm
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
     :
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
     :
NAME = <program name> [cr/lf]
PGN=mmm
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
     :
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
[cr/lf]
```

Values

a	Character code
mmm	Program number: 1 to 100

- <Program names> are shown with 32 characters or less consisting of alphanumeric characters and underscore (_).
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND PGM TO CMU··············· Outputs all programs from
                                communication port.
SENDCMU TO PGM ················ Inputs all programs from
                                communication port.

Response:
NAME=TEST[cr/lf]
PGN=1
A=1[cr/lf]
RESET DO2()[cr/lf]
   :
HALT[cr/lf]
[cr/lf]
```

2.2 One program

Format

```
1.<program name>
2. PGmmm
```

Meaning

- Expresses a specified program.
- "mmm" is 1 to 100.
- Program name may be up to 32 characters consisting of alphanumeric characters and underscore "_" and must be enclosed by "<" and ">".
- If no program name is specified in format 1, the currently selected program is specified.
- An error occurs if the specified program name differs from the program name on the data.

DATA FORMAT

```
NAME=program name[cr/lf]
PGN=mmm
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
         :
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
[cr/lf]
```

Values

a	Character code
mmm	Program name:1 to 100

- <Program names> are shown with 32 characters or less consisting of alphanumeric characters and underscore (_).
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

MEMO

- At program writing operations, be sure to use the NAME statement to specify the program name. Program writing cannot occur if the program name is not specified.
- Writing into the currently selected program is not possible.
- When a sequence program is being executed, writing into the program name "SEQUENCE" is not possible.

SAMPLE

```
SEND <TEST1> TO CMU·········Outputs the program "TEST1"
                            from communication port.
SEND CMU TO <TEST1>·········Inputs the program "TEST1"
                            from communication port


Response:
NAME=TEST1[cr/lf]
A=1[cr/lf]
RESET DO2()[cr/lf]
    :
HALT[cr/lf]
[cr/lf]
```

3 Point file

3.1 All points

Format

```
PNT
```

Meaning
- Expresses all point data.
- When used as a readout file, all points currently stored are read out.
- When used as a write file, writing is performed with a point number.

DATA FORMAT (On robots other than YK500TW)

```
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
      :
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
[cr/lf]
```

DATA FORMAT(On robot YK500TW)

```
Pmmmm= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr [cr/lf]
Pmmmm= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr [cr/lf]
       :
Pmmmm= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr [cr/lf]
Pmmmm= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr [cr/lf]
[cr/lf]
```

NOTE
- Integer point data is recognized in pulse units, and real number point data is recognized in "mm" units.

Values

mmmm	Point No.: 0 to 29999
f	Coordinate sign: + / - / space
xxxxxx/../bbbbbb	Represent a numeric value of 8 digits or less. When a dot is included, this is treated as point data in "mm" units. Each piece of data is separated by one or more spaces.
t	Extended hand system flag setting for SCARA robots. 1: right hand system; 2: left hand system.
xr	Extended setting's first arm rotation information. 0: The "mm → pulse" converted angle data x (*1) range is –180.00° < x < = 180.00°. 1: The "mm → pulse" converted angle data x (*1) range is 180.00° < x < = 540.00°. -1: The "mm → pulse" converted angle data x (*1) range is -540.00° < x < = -180.00°.
yr	Extended setting's second arm rotation information. 0: The "mm → pulse" converted angle data x (*1) range is –180.00° < y < = 180.00°. 1: The "mm → pulse" converted angle data x (*1) range is 180.00° < y < = 540.00°. -1: The "mm → pulse" converted angle data x (*1) range is -540.00° < y < = -180.00°.

*1: The joint-coordinates-converted pulse data represents each arm's distance (converted to angular data) from its mechanical origin point.

- Hand system flags are valid only for SCARA robots, with the coordinate data specified in "mm" units.
- If a number other than "1" or "2" is specified for a hand system flag, or if no number is specified, this is interpreted as "0" setting (no hand system flag).

- The first arm and the second arm rotation information settings are available only on the YK500TW robot model where a "mm" units coordinate system has been set.
- The first arm and the second arm rotation information is processed as "0" if a numeral other than 0, 1, -1 has been specified, or if no numeral has been specified.
- A line containing only [cr/lf] is added at the end of the file to indicate the end of the file.

SAMPLE (On robots other than YK500TW)

```
SEND PNT TO CMU················Outputs all points from
                               communication port.
SEND CMU TO PNT················Inputs all points from
                               communication port.

Response:
P0 =       1       2       3       4       5       6   [cr/lf]
P1 =    1.00    2.00    3.00    4.00    5.00    6.00   [cr/lf]
P2 =    1.00    0.00    0.00    0.00    0.00    0.00   [cr/lf]
   :
P29999=-1.00    0.00    0.00    0.00    0.00    0.00   [cr/lf]
[cr/lf]
```

SAMPLE (On robot YK500TW)

```
SEND PNT TO CMU················Outputs all points from
                               communication port.
SEND CMU TO PNT················Inputs all points from
                               communication port.

Response:
P0 =       1        2        3       4       5       6   [cr/lf]
P1 =   426.20  -160.77  0.01   337.21  0.00  0.00  0  1  0  [cr/lf]
P2 =   -27.57  -377.84  0.36   193.22  0.00  0.00  0 -1  0  [cr/lf]
   :
P29999=-251.66 -419.51  0.00 -127.79  0.00  0.00  2 -1 -1  [cr/lf]
[cr/lf]
```

3.2 One point

Format

```
Pmmmm
```

Meaning
- Expresses a specified point.
- "mmmm" must be from 0 to 29999

DATA FORMAT (On robots other than YK500TW)

```
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
```

DATA FORMAT (On robot YK500TW)

```
Pmmmm= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr [cr/lf]
```

Values

mmmm	Point No.: 0 to 29999
f	Coordinate sign: + / - / space
xxxxxx/../bbbbbb	Represent a numeric value of 8 digits or less. When a dot is included, this is treated as point data in "mm" units. Each piece of data is separated by one or more spaces.
t	Extended hand system flag setting for SCARA robots. 1: right hand system; 2: left hand system.
xr	The first arm rotation information for the YK500TW robot. 0: The "mm → pulse" converted pulse data x (*1) range is –180.00° < x < = 180.00°. 1: The "mm → pulse" converted pulse data x (*1) range is 180.00° < x < = 540.00°. -1: The "mm → pulse" converted pulse data x (*1) range is –540.00° < x < = -180.00°.
yr	The second arm rotation information for the YK500TW robot. 0: The "mm → pulse" converted pulse data x (*1) range is –180.00° < y < = 180.00°. 1: The "mm → pulse" converted pulse data x (*1) range is 180.00° < y < = 540.00°. -1: The "mm → pulse" converted pulse data x (*1) range is –540.00° < y < = -180.00°.

*1: The joint-coordinates-converted pulse data represents each arm's distance (converted to angular data) from its mechanical origin point.

NOTE
- Integers indicate point data in "pulse" units, and real numbers in "mm" units.

- Hand system flags are valid only for SCARA robots, with the coordinate data specified in "mm" units.
- If a number other than "1" or "2" is specified for a hand system flag, or if no number is specified, this is interpreted as "0" setting (no hand system flag).
- The first arm and the second arm rotation information settings are available only on the YK500TW robot model where a "mm" units coordinate system has been set.
- The first arm and the second arm rotation information is processed as "0" if a numeral other than 0, 1, -1 has been specified, or if no numeral has been specified.

SAMPLE (On robots other than YK500TW)

```
SEND P100 TO CMU··············· Outputs the specified point
                                from communication port.
SEND CMU TO P100··············· Inputs the specified point from
                                communication port.

Response:
P100=   1    2    3    4    5    6[cr/lf]
```

SAMPLE (On robot YK500TW)

```
SEND P100 TO CMU··············· Outputs the specified point
                                from communication port.
SEND CMU TO P100··············· Inputs the specified point from
                                communication port.

Response:
P100=   1     2     3     4     5     6     0     1     0  [cr/lf]
```

4 Point comment file

4.1 All point comments

Format

```
PCM
```

Meaning
- Expresses all point comments.
- When used as a readout file, all point comments currently stored are read out.
- When used as a write file, writing is performed with a point comment number.

DATA FORMAT

```
PCmmmm= ssssssssssssssss[cr/lf]
PCmmmm= ssssssssssssssss[cr/lf]
      :
PCmmmm= ssssssssssssssss[cr/lf]
PCmmmm= ssssssssssssssss[cr/lf]
[cr/lf]
```

Values

mmmm	Point comment number: a number from 0 to 29999
ss...ss	Comment data: which can be up to 16 one-byte characters. If comment data exceeds 16 characters, then the 17th character onwards will be deleted.

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND PCM TO CMU··············· Outputs all point comments
                               from communication port.
SEND CMU TO PCM··············· Inputs all point comments from
                               communication port.

Response:
PC1＝ORIGIN POS[cr/lf]
PC3＝WAIT POS[cr/lf]
   :
PC3999＝ WORK100[cr/lf]
[cr/lf]
```

4.2 One point comment

Format

```
PCmmmmm
```

Meaning

- Expresses a specified point comment.
- "mmmm" represents a number from 0 to 29999.

DATA FORMAT

```
PCmmmm= ssssssssssssssss[cr/lf]
```

Values

mmmm	Point comment number: a number from 0 to 29999
ss...ss	Comment data: which can be up to 16 one-byte characters. If comment data exceeds 16 characters, then the 17th character onwards will be deleted.

SAMPLE

```
SEND PC1 TO CMU················ Outputs the specified point
                                comment from communication port.
SEND CMU TO PC1················ Inputs the specified point
                                comment from communication port.

Response:
PC1＝ORIGIN POS[cr/lf]
```

5 Parameter file

5.1 All parameters

Format

```
PRM
```

Meaning
- Expresses all parameters (including settings in "UTILITY" mode).
- When used as a readout file, all parameters currently stored are read out.
- When used as a write file, only the parameters specified by parameter labels are written.

DATA FORMAT

```
/parameter label/ ’ <comment> [cr/lf]
RC= xxxxxx [cr/lf]
/parameter label/   ’ <comment> [cr/lf]
R1= xxxxxx  R2= yyyyyy [cr/lf]
/parameter label/   ’ <comment> [cr/lf]
R1= xxxxxx, yyyyyy, zzzzzz, rrrrrr[cr/lf]
/parameter label/   ’ <comment> [cr/lf]
          :
[cr/lf]
```

Values

RC	Indicates the entire controller.
R?	Robot setting: ?: robot number
A	Represents an axis parameter. Data on each axis is separated by a comma.
<Comment>	Parameter name

- Parameter labels are shown with 8 alphabetic characters.
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

MEMO
- When writing parameter data, be sure that the servo is off.
- Parameters are already compatible with upper versions. However, parameters might not always be compatible with lower versions (upward compatibility).
- When you attempt to load a parameter file of new version into a controller of an earlier version, an error "10.14 : Undefined parameter found" may appear. If this happens, you may load the parameter by setting the "Skip undefined parameters" parameter to "VALID". For more details, refer to the "SYSTEM mode" – "Parameters" section in the robot controller user's manual. For more details, refer to the "SYSTEM mode" – "Other parameters" section in the robot controller user's manual.

SAMPLE

```
SEND PRM TO CMU················Outputs all parameters from
                               communication port.
SEND CMU TO PRM ···············Inputs all parameters from
                               communication port.

Response:
' V1.01/R0001
' V1.01/V1.01/V1.01/V1.01/V1.01/V1.01/V1.01/V1.01/
' -----/-----/-----/-----/-----/-----/-----/-----/
/RBTNUM/[cr/lf]
 R1=   3000  R2=  3010   [cr/lf]
/AXSNUM/[cr/lf]
 R1A=  5000,   5001,   5010,    5011 [cr/lf]
 R2A=     0,      0,      0,       0 [cr/lf]
/ATTRIB/
 R1A= 33792,  33792,  33792,   33792 [cr/lf]
 R1A=   256,    256,    256,     256 [cr/lf]
/WEIGHT/  '[kg][cr/lf]
 R1=      2  R2=    12   [cr/lf]
   :
[cr/lf]
```

5.2 One parameter

Format

```
/parameter label/
```

Meaning
- Parameter labels are shown with 8 alphabetic characters.
- When used as a readout file, only the parameter specified by a parameter label is read out.
- When used as a write file, only the parameter specified by a parameter label is written.

DATA FORMAT 1

```
/parameter label/  ' <comment>[cr/lf]
RC= xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT 2

```
/parameter label/  ' <comment>[cr/lf]
R?= xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT 3

```
/parameter label/  ' <comment>[cr/lf]
R?= xxxxxx,xxxxxx,xxxxxx,xxxxxx[cr/lf]
[cr/lf]
```

Values

RC	Indicates the entire controller.
R?	Robot setting: ? : robot number
A	Represents an axis parameter. Data on each axis is separated by a comma.
<Comment>	Parameter name

- Parameter labels are shown with 8 alphabetic characters.
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

MEMO
- When writing parameter data, be sure that the servo is off.
- Parameters are already compatible with upper versions. However, parameters might not always be compatible with lower versions (upward compatibility).
- When you attempt to load a parameter file of new version into a controller of an earlier version, an error "10.14 : Undefined parameter found" may appear.

SAMPLE

```
SEND  /ACCEL / TO CMU ······ Outputs the acceleration parameter
                             from communication port.
SEND CMU TO /ACCEL /········ Inputs the acceleration parameter
                             from communication port.

Response:
/ACCEL /  '[%]
R1A=    100,    100,    100,     100[cr/lf]
[cr/lf]
```

6 Shift coordinate definition file

6.1 All shift data

Format

```
SFT
```

Meaning
- Expresses all shift data.
- When used as a readout file, all shift data currently stored are read out.
- When used as a write file, writing is performed with a shift number.

DATA FORMAT

```
Sm  = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr [cr/lf]
SPm = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr [cr/lf]
SMm = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr [cr/lf]
      :
Sm  = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr [cr/lf]
SPm = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr [cr/lf]
SMm = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr [cr/lf]
[cr/lf]
```

Values

m	Shift No.: 0 to 9
f	Coordinate sign: + / - / space
xxxxxx/yyyyyy/../rrrrrr	Represent a numeric value of 8 digits or less, having 2 or less places below the decimal point.

- The SPm and SMm inputs are optional in writing files.
  SPm: shift coordinate range plus-side; SMm: shift coordinate range minus-side
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SFT TO CMU················Outputs all shift data from
                               communication port.
SEND CMU TO SFT················Inputs all shift data from
                               communication port.

Response:
S0 =    0.00    0.00    0.00    0.00 [cr/lf]
SP0=    0.00    0.00    0.00    0.00 [cr/lf]
SM0=    0.00    0.00    0.00    0.00 [cr/lf]
S1 =    1.00    1.00    1.00    1.00 [cr/lf]
      :
SM39=   9.00    9.00    9.00    9.00 [cr/lf]
[cr/lf]
```

6.2 One shift definition

Format

```
Sm
```

Meaning • Expresses a specified shift definition.

DATA FORMAT

```
Sm   = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr[cr/lf]
```

Values

m	Shift No.: 0 to 39
f	Coordinate sign: + / - / space
xxxxxx/yyyyyy/../rrrrrr	Represent a numeric value of 8 digits or less, having 2 or less places below the decimal point.

SAMPLE

```
SEND S0 TO CMU ················· Outputs the specified shift
                                 coordinate from communication port.
SEND CMU TO S0 ················· Inputs the specified shift
                                 coordinate from communication port.

Response:
S0 =0.00     0.00     0.00     0.00[cr/lf]
SP0=0.00     0.00     0.00     0.00[cr/lf]
SM0=0.00     0.00     0.00     0.00[cr/lf]
[cr/lf]
```

7 Hand definition file

7.1 All hand data

Format

```
HND
```

Meaning

- Expresses all hand data.
- When used as a readout file, all hand data currently stored are read out.
- When used as a write file, writing is performed with a hand number.

DATA FORMAT

```
Hm  = n,fxxxxxx, fyyyyyy, fzzzzzz [,{R}][cr/lf]
    :
Hm  = n,fxxxxxx, fyyyyyy, fzzzzzz [,{R}][cr/lf]
[cr/lf]
```

Values

m	Hand number: 0 to 31
n	Robot number: 1 to 4
f	Coordinate sign: + / - / space
xxxxxx/yyyyyy/zzzzzz	Represent a numeric value of 8 digits or less, having 2 or less places below the decimal point, or an integer of 7 digits or less. (This numeric format depends on the robot type setting and hand definition type.)
{R}	Indicates whether a hand is attached to the R-axis.

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND HND TO CMU················ Outputs all hand data from
                                communication port.
SEND CMU TO HND················ Inputs all hand data from
                                communication port.

Response:
H0 =  1,   0.00,   0.00,    0.00 [cr/lf]
H1 =  1,   1.00,   1.00,    1.00 [cr/lf]
H2 =  2,   2.00,   2.00,    2.00 [cr/lf]
H3 =  2,   3.00,   3.00,    3.00 [cr/lf]
H4 =  3,   4.00,   4.00,    4.00 [cr/lf]
H5 =  3,   5.00,   5.00,    5.00 [cr/lf]
H6 =  4,   6.00,   6.00,    6.00 [cr/lf]
H7 =  4,   7.00,   7.00,    7.00 [cr/lf]
[cr/lf]
```

7.2 One hand definition

Format

```
Hm
```

Meaning • Expresses a specified hand definition.

DATA FORMAT

```
Hm = n,fxxxxxx, fyyyyyy, fzzzzzz[,{R}][cr/lf]
```

Values

m	Hand number: 0 to 31
n	Robot number: 1 to 4
f	Coordinate sign: + / - / space
xxxxxx/yyyyyy/zzzzzz	Represent a numeric value of 8 digits or less, having 2 or less places below the decimal point, or an integer of 7 digits or less. (This numeric format depends on the robot type setting and hand definition type.)
{R}	Indicates whether a hand is attached to the R-axis.

SAMPLE

```
SEND H3 TO CMU ················· Outputs the specified hand definition
                                 data from communication port.
SEND CMU TO H3 ················· Inputs the specified hand definition
                                 data from communication port.

Response:
H3 =  2,   3.00,   3.00,    3.00[cr/lf]
```

8 Pallet definition file

8.1 All pallet definitions

Format

```
PLT
```

Meaning
- Expresses all pallet definitions.
- When used as a readout file, all pallet definitions currently stored are read out.
- When used as a write file, writing is performed with a pallet number.

DATA FORMAT (On robots other than YK500TW)

```
PLm [cr/lf]
PLN =  XY [cr/lf]
NX  =  nnn [cr/lf]
NY  =  nnn [cr/lf]
NZ  =  nnn [cr/lf]
PLP = ppppp [cr/lf]
P[1] = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
      :
P[5] = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
PLm [cr/lf]
      :
[cr/lf]
```

DATA FORMAT (On robot YK500TW)

```
PLm [cr/lf]
PLN =  XY [cr/lf]
NX  =  nnn [cr/lf]
NY  =  nnn [cr/lf]
NZ  =  nnn [cr/lf]
PLP = ppppp [cr/lf]
P[1]  = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr[cr/lf]
      :
P[5]  = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr[cr/lf]
PLm [cr/lf]
      :
[cr/lf]
```

Values

m	Pallet number: 0 to 39
nnn	Number of axis points: positive integer
ppppp	The point number used for a pallet definition. Continuous 5 points starting with the specified point are used.
f	Coordinate sign: + / - / space
xxxxxx/yyyyyy/../bbbbbb	Represent a numeric value of 8 digits or less. When a dot is included, this is treated as coordinate data in "mm" units. Each piece of data is separated by one or more spaces.
t	An extended hand system flag setting for SCARA robots. 1: Right-handed system, 2: Left-handed system

xr .... Extended setting for the first arm rotation information.
0: "mm" → pulse converted angle data x (*1) range: -180.00°< x <= 180.00°
1: "mm" → pulse converted angle data x (*1) range: 180.00° < x <= 540.00°
-1: "mm" → pulse converted angle data x (*1) range: -540.00° < x <= -180.00°

yr .... Extended setting for the second arm rotation information.
0: "mm" → pulse converted angle data x (*1) range: -180.00° < y <= 180.00°
1: "mm" → pulse converted angle data x (*1) range: 180.00° < y <= 540.00°
-1: "mm" → pulse converted angle data x (*1) range: -540.00° < y <= -180.00°

*1: The joint-coordinates-converted pulse data represents each arm's distance (converted to angular data) from its mechanical origin point.

- Hand system flags are enabled only when specifying the coordinate data in "mm" units for SCARA robots.
- Hand system flags and the first arm and the second arm rotation information are ignored during movement where pallet definitions are used.
- If a number other than 1 or 2 is set, or if no number is designated, then 0 will be set to indicate that there is no hand system flag.
- The first arm and the second arm rotation information settings are available only on the YK500TW robot model where a "mm" units coordinate system has been set.
- If a value other than "0", "1", "-1" is specified at the first arm and the second arm rotation information, or if no value is specified, this will be processed as "0".
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND PLT TO CMU··············· Outputs all pallet definitions
                               from communication port.
SEND CMU TO PLT··············· Inputs all pallet definitions
                               from communication port.
Response:
PL0[cr/lf]
PLN=XY[cr/lf]
NX =  3[cr/lf]
NY =  4[cr/lf]
NZ =  2[cr/lf]
PLP= 3996[cr/lf]
P[1]=   0.00    0.00    0.00    0.00    0.00    0.00  [cr/lf]
P[2]= 100.00    0.00    0.00    0.00    0.00    0.00  [cr/lf]
P[3]=   0.00  100.00    0.00    0.00    0.00    0.00  [cr/lf]
P[4]= 100.00  100.00    0.00    0.00    0.00    0.00  [cr/lf]
P[5]=   0.00    0.00   50.00    0.00    0.00    0.00  [cr/lf]
PL1[cr/lf]
PLN=  XY[cr/lf]
NX =  3[cr/lf]
NY =  4[cr/lf]
NZ =  2[cr/lf]
PLP= 3991[cr/lf]
P[1]=   0.00    0.00    0.00    0.00    0.00    0.00  [cr/lf]
P[2]= 100.00  100.00    0.00    0.00    0.00    0.00  [cr/lf]
P[3]=   0.00  200.00    0.00    0.00    0.00    0.00  [cr/lf]
P[4]= 100.00  200.00    0.00    0.00    0.00    0.00  [cr/lf]
P[5]=   0.00    0.00  100.00    0.00    0.00    0.00  [cr/lf]
[cr/lf]
```

8.2 One pallet definition

Format

```
PLm
```

Meaning
- Expresses a specified pallet definition.
- "m" must be from 0 to 39.

DATA FORMAT (On robots other than YK500TW)

```
PLm [cr/lf]
PLN   =  XY [cr/lf]
NX    =  nnn [cr/lf]
NY    =  nnn [cr/lf]
NZ    =  nnn [cr/lf]
PLP = ppppp [cr/lf]
P[1] = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
      :
P[5] = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
[cr/lf]
```

DATA FORMAT (On robot YK500TW)

```
PLm [cr/lf]
PLN   =  XY [cr/lf]
PLP = ppppp [cr/lf]
NX    =  nnn [cr/lf]
NY    =  nnn [cr/lf]
NZ    =  nnn [cr/lf]
P[1]  = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr[cr/lf]
      :
P[5]  = fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr[cr/lf]
[cr/lf]
```

NOTE
- Integers indicate point data in "pulse" units, and real numbers in "mm" units.

Values

m .... Pallet number: 0 to 39

nnn .... Number of points for each axis: positive integer

ppppp .... The point number used for a pallet definition. Continuous 5 points starting with the specified point are used.

f .... Coordinate sign: + / - / space

xxxxxx/yyyyyy/../bbbbbb.........
Represent a numeric value of 8 digits or less. When a dot is included, this is treated as point data in "mm" units. Each piece of data is separated by one or more spaces.

t .... An extended hand system flag setting for SCARA robots. 1: Right-handed system, 2: Left-handed system

xr .... Extended setting for the first arm rotation information.
0: "mm" → pulse converted angle data x (*1) range: -180.00° < x <= 180.00°
1: "mm" → pulse converted angle data x (*1) range: 180.00° < x <= 540.00°
-1: "mm" → pulse converted angle data x (*1) range: -540.00° < x <= -180.00°

yr .... Extended setting for the second arm rotation information.
0: "mm" → pulse converted angle data x (*1) range: -180.00° < y <= 180.00°
1: "mm" → pulse converted angle data x (*1) range: 180.00° < y <= 540.00°
-1: "mm" → pulse converted angle data x (*1) range: -540.00° < y <= -180.00°

*1: The joint-coordinates-converted pulse data represents each arm's distance (converted to angular data) from its mechanical origin point.

- Hand system flags are enabled only when specifying the coordinate data in "mm" units for SCARA robots.
- Hand system flags and the first arm and the second arm rotation information are ignored during movement where pallet definitions are used.
- If a number other than 1 or 2 is set, or if no number is designated, then 0 will be set to indicate that there is no hand system flag.
- The first arm and the second arm rotation information settings are available only on the YK500TW robot model where a "mm" units coordinate system has been set.
- If a value other than "0", "1", "-1" is specified at the first arm and the second arm rotation information, or if no value is specified, this will be processed as "0".
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND PL2 TO CMU················Outputs the specified pallet definition
                               from communication port as shown below.
SEND CMU TO PL2················Inputs the specified pallet definition
                               from communication port as shown below.
Response:
PL2[cr/lf]
PLN=XY[cr/lf]
NX=    3[cr/lf]
NY=    3[cr/lf]
NZ=    2[cr/lf]
PLP= 3986[cr/lf]
P[1]= 100.00  100.00   50.00   90.00    0.00    0.00  [cr/lf]
P[2]= 200.00  100.00   50.00   90.00    0.00    0.00  [cr/lf]
P[3]= 100.00  200.00   50.00   90.00    0.00    0.00  [cr/lf]
P[4]= 200.00  200.00   50.00   90.00    0.00    0.00  [cr/lf]
P[5]= 100.00   10.00  100.00   90.00    0.00    0.00  [cr/lf]
[cr/lf]
```

9 All file

9.1 All files

Format

```
ALL
```

Meaning Expresses the minimum number of data files required to operate the robot system.

NOTE
- For details of each file, refer to that file's explanation.

DATA FORMAT

```
[PGM] All program format
NAME=< program name >
aaaaa …aaaaaaaaaaaaaa[cr/lf]
:
aaaaa …aaaaaaaaaaaaaa[cr/lf]
[cr/lf]
[PNT] All point format
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
:
Pmmmm=fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t[cr/lf]
[cr/lf]
[PCM] All point comment format
PCmmmm= sssssssssssssss[cr/lf]
:
PCmmmm= sssssssssssssss[cr/lf]
[cr/lf]
[PRM] All parameter format
/parameter label/'<comment> [cr/lf]
RC= xxxxxx [cr/lf]
/parameter label/'<comment> [cr/lf]
R1= xxxxxx R2= yyyyyy [cr/lf]
:
[cr/lf]
[END] ALL files end
```

MEMO
- In writing files, [xxx] determines the data file's format, and this format is saved at the controller. Example: [HND]…All text data up the next [xxx] is saved at the controller as "all hand" format data.

SAMPLE

```
SEND ALL TO CMU················ Outputs all files of the entire
                                system from communication port.
SEND CMU TO ALL················ Inputs all files of the entire
                                system from communication port.
```

10 Program directory file

10.1 Entire program directory

Format

```
DIR
```

Meaning

- Expresses entire program directory.
- When used as a readout file, information on entire program directory is read out.
- Cannot be used as a write file.

DATA FORMAT

```
No.   Name         Line  Byte    RW/RO Date       Time[cr/lf]
nnnfgssssssss  llll  bbbbbb  xx    yy/mm/dd hh:mm[cr/lf]
     :
nnnfgssssssss  llll  bbbbbb  xx    yy/mm/dd hh:mm[cr/lf]
END[cr/lf]
```

Values

nnn	Program directory number: 3 digits
f	"o" at program compiling when a program object is created. "s" at sequence program compiling when a sequence object is created.
g	Shows an asterisk "*" for the currently selected program.
ssssssss	Program name: 32 digits
llll	Number of program lines: 4 digits
bbbbbb	Byte size of program: 6 digits
xx	File attribute: 2 digits RW: Readable/writable RO: Not writable (read only)
yy/mm/dd	Date when the program was updated: 8 digits (including the "/" marks)
hh:mm	Time when the program was updated: 5 digits

SAMPLE

```
SEND DIR TO CMU················ Outputs information on all program
                                directory from communication port.
Response:
No.   Name         Line  Byte   RW/RO Date        Time[cr/lf]
1o*   12345678     5     21     RW    01/06/20    10:35[cr/lf]
2     PGM1         5     66     RW    01/06/20    10:35[cr/lf]
3     PGM2         5     66     RW    01/06/20    10:35[cr/lf]
4     PGM3         5     66     RW    01/06/20    10:35[cr/lf]
5     PGM4         5     66     RW    01/06/20    10:35[cr/lf]
6     PGM5         5     66     RW    01/06/20    10:35[cr/lf]
7     PGM6         5     66     RW    01/06/20    10:35[cr/lf]
8s    SEQUENCE     1     15     RW    01/06/20    10:35[cr/lf]
END[cr/lf]
```

10.2 One program

Format

```
"<<" <program name>" >>"
```

Meaning
- Expresses information on one program.
- The program name is enclosed in <<>> double brackets.

DATA FORMAT

```
No.   Name        Line  Byte    RW/RO Date       Time[cr/lf]
nnnfgssssssss   llll  bbbbbb  xx    yy/mm/dd hh:mm[cr/lf]
```

Values

nnn	Program directory number: 3 digits
f	"o" at program compiling when a program object is created. "s" at sequence program compiling when a sequence object is created.
g	Shows an asterisk "*" for the currently selected program.
ssssssss	Program name: 32 digits
llll	Number of program lines: 4 digits
bbbbbb	Byte size of program: 6 digits
xx	File attribute: 2 digits RW: Readable/writable RO: Not writable (read only)
yy/mm/dd	Date when the program was updated: 8 digits (including the "/" marks)
hh:mm	Time when the program was updated: 5 digits

MEMO
- Indicates the compiled execution program (program objects compiled for a robot program, or sequence objects compiled for a sequence program).

SAMPLE

```
SEND <<TEST>> TO CMU········ Outputs information on the specified
                             program from communication port.

Response:
No.   Name        Line  Byte   RW/RO Date        Time[cr/lf]
3o*   PGM2        5     66     RW    01/06/20   10:35[cr/lf]
```

11 Parameter directory file

11.1 Entire parameter directory

Format

```
DPM
```

Meaning
- Expresses entire parameter directory.
- When used as a readout file, information on entire parameter directory is read out.
- Cannot be used as a write file.

DATA FORMAT

```
/parameter label/ ' <comment>[cr/lf]
/parameter label/ ' <comment>[cr/lf]
      :
/parameter label/ ' <comment>[cr/lf]
 [cr/lf]
```

Values <comment> .... Parameter name

- Parameter labels are shown with 6 alphabetic characters.
- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND DPM TO CMU··············· Outputs information on all parameter
                               directory from communication port.
Response:
/RBTNUM/ ' Robot number (V8.01/R1001)[cr/lf]
/AXES  / ' Number of axes[cr/lf]
/AXSNUM/ ' Axis number(V1.01/V1.01/V1.01/V1.01/-----/-----/-----/-----/)[cr/lf]
/ATTRIB/ ' Axis attribute[cr/lf]
/WEIGHT/ ' Tip weight[kg][cr/lf]
/ORIGIN/ ' Origin sequence[cr/lf]
/RORIEN/ ' R axis orientation[cr/lf]
   :
/CURPNO/ ' Output port number[cr/lf]
/CURPT1/ ' Number of compare point 1[cr/lf]
/CURPT2/ ' Number of compare point 2[cr/lf]
[cr/lf]
```

12 Variable file

12.1 All variables

Format

```
VAR
```

Meaning

- Expresses all global variables.
- When used as a readout file, all global variables currently stored are read out.
- When used as a write file, a specified global variable is written.

DATA FORMAT

```
<variable name>t = xxxxxx [cr/lf]
<variable name>t = xxxxxx [cr/lf]
      :
<variable name>t = xxxxxx [cr/lf]
 [cr/lf]
```

Values

<Variable name>	Global variable defined in the program. Variable name is shown with 32 characters or less consisting of alphanumeric characters and underscore ("_").
t	Type of variable / !: real type, %: integer type, $: character string type
xxxxxx	Differs depending on the type of variable:

Integer type:	integer of 8 digits or less
Real type:	real number of 7 digits or less including decimal fractions
Character type:	character string of 255 characters or less

▪ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

12 Variable file

SAMPLE 1

```
SEND VAR TO CMU················ Outputs all global variables
                                 from communication port.

Response:
SGI0=0[cr/lf]
SGI1=1111[cr/lf]
SGI2=2222[cr/lf]
SGI3=3333[cr/lf]
SGI4=4444[cr/lf]
SGI5=5555[cr/lf]
SGI6=6666[cr/lf]
SGI7=7777[cr/lf]
SGR0=0[cr/lf]
SGR1=1.1111E3[cr/lf]
SGR2=2.2222E3[cr/lf]
SGR3=3.3333E3[cr/lf]
SGR4=4.4444E3[cr/lf]
SGR5=5.5555E3[cr/lf]
SGR6=6.6666E3[cr/lf]
SGR7=7.7777E3[cr/lf]
B1%=111[cr/lf]
B2%=222[cr/lf]
C1$="CNS_1"[cr/lf]
C2$="CNS_2"[cr/lf]
[cr/lf]
```

SAMPLE 2

```
SEND CMU TO VAR················ Inputs all global variables
                                 from communication port.
```

12.2 One variable

Format

```
<variable name>t
```

Meaning • Expressed one variable.

DATA FORMAT

```
xxxxxx [cr/lf]
```

NOTE
- SGIx indicates an integer type static variable.
- SGRx indicates a real type static variable.

Values

<Variable name>	Global variable defined in the program. Variable name is shown with 32 characters or less consisting of alphanumeric characters and underscore ("_").	
t	Type of variable / !: real type, %: integer type, $: character string type	
xxxxxx	Differs depending on the type of variable:	
	Integer type:	integer of 8 digits or less
	Real type:	real number of 7 digits or less including decimal fractions
	Character type:	character string of 255 characters or less

MEMO
- Dynamic global variables are registered during program execution. Variables cannot be referred to unless they are registered.

SAMPLE 1

```
SEND SGI6 TO CMU[cr/lf] ···Outputs the specified variable
                           SGI6 from communication port.

Response:
6666[cr/lf]
```

SAMPLE 2

```
SEND CMU TO SGI6[cr/lf]  ··Inputs the specified variable
                           SGI6 from communication port.

Response:
6666 [cr/lf]·······Data input to the controller.
OK [cr/lf]·········Result output from the controller.
```

13 Constant file

13.1 One character string

Format

```
"<character string>"
```

Meaning
- Expresses a specified character string.
- When used as a readout file, the specified character string is read out.
- Cannot be used as a write file.

DATA FORMAT

```
sssss...ssssss[cr/lf]
```

Values sssss...ssssss............................Character string: 255 characters or less

- Output of a double quotation (") is shown with two successive double quotations.

SAMPLE

```
SEND ""YAMAHA ROBOT"" TO CMU
         ························· Outputs the specified character
                                   string from communication port.
Response:
"YAMAHA ROBOT"[cr/lf]
```

14 Array variable file

14.1 All array variables

Format

```
ARY
```

Meaning
- Expresses all array variables.
- When used as a readout file, all array variables are read out.
- When used as a write file, writing is performed with a specified array variable.

DATA FORMAT

```
<variable name>t(l{,m{,n}}) = xxxxxx [cr/lf]
<variable name>t(l{,m{,n}}) = xxxxxx [cr/lf]
      :
<variable name>t(l{,m{,n}}) = xxxxxx [cr/lf]
[cr/lf]
```

Values

<Variable name>	Global variable defined in the program. Variable name is shown with 32 characters or less consisting of alphanumeric characters and underscore ("_").
t	Type of variable / !: real type, %: integer type, $: character string type
l, m, n	Indicate array arguments.
xxxxxx	Differs depending on the type of array variable. Integer type: integer of 8 digits or less Real type: real number of 7 digits or less including decimal fractions Character type: character string of 255 characters or less

▪ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE 1

```
SEND ARY TO CMU··········· Outputs all global array variables from communication port.
Response:
A!(0)=0[cr/lf]
A!(1)=1.E2[cr/lf]
A!(2)=2.E2[cr/lf]
B%(0,0)=0[cr/lf]
B%(0,1)=1111[cr/lf]
B%(1,0)=2222[cr/lf]
B%(1,1)=3333[cr/lf]
C$(0,0,0)="ARY1"[cr/lf]
C$(0,0,1)="ARY2"[cr/lf]
C$(0,1,0)="ARY3"[cr/lf]
C$(0,1,1)="ARY4"[cr/lf]
C$(1,0,0)="ARY5"[cr/lf]
C$(1,0,1)="ARY6"[cr/lf]
C$(1,1,0)="ARY7"[cr/lf]
C$(1,1,1)="ARY8"[cr/lf]
[cr/lf]
```

SAMPLE 2

```
SEND CMU TO ARY··········· Inputs all global array variables from communication port.
```

14.2 One array variable

Format

```
<variable name>t(l {,m {,n }} )
```

Meaning • Expresses one array variable.

DATA FORMAT

```
xxxxxx [cr/lf]
```

Values

<Variable name>	Global variable defined in the program. Variable name is shown with 32 characters or less consisting of alphanumeric characters and underscore ("_").
t	Type of variable / !: real type, %: integer type, $: character string type
l, m, n	Indicate array arguments.
xxxxxx	Differs depending on the type of array variable. Integer type: integer of 8 digits or less Real type: real number of 7 digits or less including decimal fractions Character type: character string of 255 characters or less

MEMO

- Array variables defined by the DIM statement are registered during compiling. Array variables cannot be referred to unless they are registered.

SAMPLE 1

```
SEND C1$(2) TO CMU[cr/lf] ···· Outputs the specified array variable
                               C1$(2) from communication port.
Response:
YAMAHA ROBOT[cr/lf]
```

SAMPLE 2

```
SEND CMU TO C1$(2)[cr/lf] ···· Inputs the specified array variable
                               C1$(2) from communication port.
Response:
OK[cr/lf]
```

15 DI file

15.1 All DI information

Format

```
DI()
```

Meaning

- Expresses all DI (parallel input variable) information.
- When used as a readout file, all DI information is read out.
- Cannot be used as a write file.

DATA FORMAT

```
DI0()=&Bnnnnnnnn [cr/lf]
DI1()=&Bnnnnnnnn [cr/lf]
      :
DI27()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n .... "0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND DI() TO CM················ Outputs all DI information
                                 from communication port.

Response:
DI0()=&B10001001[cr/lf]
DI1()=&B00000010[cr/lf]
DI2()=&B00000000[cr/lf]
      :
DI7()=&B00000000[cr/lf]
DI10()=&B00000000[cr/lf]
DI11()=&B00000000[cr/lf]
DI12()=&B00000000[cr/lf]
      :
DI17()=&B00000000[cr/lf]
DI20()=&B00000000[cr/lf]
      :
DI26()=&B00000000[cr/lf]
DI27()=&B00000000[cr/lf]
[cr/lf]
```

15.2 One DI port

Format

```
DIm()
```

Meaning

- Expresses the status of one DI port.
- When used as a readout file, the specified DI port status is read out.
- Cannot be used as a write file.

DATA FORMAT

```
DIm()=&Bnnnnnnnn[cr/lf]
```

Values

m	0 to 7, 10 to 17, 20 to 27
n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

SAMPLE

```
SEND DI5() TO CMU············· Outputs the DI5 port status
                               from communication port.

Response:
DI5()=&B00000000[cr/lf]
```

16 DO file

16.1 All DO information

Format

```
DO()
```

Meaning

- Expresses all DO (parallel output variable) information.
- When used as a readout file, all DO information is read out.
- Cannot be used as a write file.

DATA FORMAT

```
DO0()=&Bnnnnnnnn [cr/lf]
DO1()=&Bnnnnnnnn [cr/lf]
       :
DO27()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND DO() TO CMU ·············· Outputs all DO information
                                from communication port.

Response:
DO0()=&B10001001[cr/lf]
DO1()=&B00000010[cr/lf]
DO2()=&B00000000[cr/lf]
      :
DO7()=&B00000000[cr/lf]
DO10()=&B00000000[cr/lf]
DO11()=&B00000000[cr/lf]
DO12()=&B00000000[cr/lf]
      :
DO17()=&B00000000[cr/lf]
DO20()=&B00000000[cr/lf]
      :
DO26()=&B00000000[cr/lf]
DO27()=&B00000000[cr/lf]
[cr/lf]
```

16.2 One DO port

Format

```
DOm()
```

Meaning

- Expresses the status of one DO port.
- When used as a readout file, the specified DO port status is read out.
- When used as a write file, the value is written to the specified DO port. However, writing to DO0() and DO1() is prohibited.

- Readout file

DATA FORMAT

```
DOm()=&Bnnnnnnnn[cr/lf]
```

- Write file

DATA FORMAT

```
&Bnnnnnnnn[cr/lf] or k[cr/lf]
```

Values

m	Port number: 0 to 7, 10 to 17, 20 to 27
n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).
k	Integer from 0 to 255

MEMO

- Writing to DO0() and DO1() is prohibited. Only referencing is permitted.

SAMPLE 1

```
SEND DO5() TO CMU············ Outputs the DO5 port status
                              from communication port.

Response:
DO5()=&B00000000[cr/lf]
```

SAMPLE 2

```
SEND CMU TO DO5()············ Inputs the DO5 port status
                              from communication port.
&B00000111

Response:
OK[cr/lf]
```

17 MO file

17.1 All MO information

Format

```
MO()
```

Meaning

- Expresses all MO (internal output variable) information.
- When used as a readout file, all MO information is read out.
- Cannot be used as a write file.

DATA FORMAT

```
MO0()=&Bnnnnnnnn [cr/lf]
MO1()=&Bnnnnnnnn [cr/lf]
      :
MO33()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n ...."0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND MO() TO CMU ·············· Outputs all MO information
                                from communication port.

Response:
MO0()=&B10001001[cr/lf]
MO1()=&B00000010[cr/lf]
MO2()=&B00000000[cr/lf]
:
MO7()=&B00000000[cr/lf]
MO10()=&B00000000[cr/lf]
MO11()=&B00000000[cr/lf]
MO12()=&B00000000[cr/lf]
:
MO17()=&B00000000[cr/lf]
MO20()=&B00000000[cr/lf]
:
MO32()=&B00000000[cr/lf]
MO33()=&B00000000[cr/lf]
[cr/lf]
```

17.2 One MO port

Format

```
MOm()
```

Meaning
- Expresses the status of one MO port.
- When used as a readout file, the specified MO port status is read out.
- When used as a write file, the value is written to the specified MO port. However, writing to MO30() to MO37() is prohibited.

- Readout file

DATA FORMAT

```
MOm()=&Bnnnnnnnn[cr/lf]
```

- Write file

DATA FORMAT

```
&Bnnnnnnnn[cr/lf] or k[cr/lf]
```

Values

m	Port number: 0 to 7, 10 to 17, 20 to 27, 30 to 37
n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).
k	Integer from 0 to 255

MEMO
- Writing to MO30() to MO37() is prohibited. Only reference is permitted.

SAMPLE 1

```
SEND MO5() TO CMU············ Outputs the MO5 port status
                              from communication port.

Response:
MO5()=&B00000000[cr/lf]
```

SAMPLE 2

```
SEND CMU TO MO5() ············ Inputs the MO5 port status
                               from communication port.
&B00000111

Response:
OK[cr/lf]
```

18 LO file

18.1 All LO information

Format

```
LO()
```

Meaning

- Expresses all LO (internal output variable) information.
- When used as a readout file, all LO information is read out.
- Cannot be used as a write file.

DATA FOMAT

```
LO0()=&Bnnnnnnnn [cr/lf]
LO1()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n ...."0" or "1" (total of 8 digits). Corresponds to 07, 06, …, 00, reading from the left.

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND LO() TO CMU ·············· Outputs all LO status from
                                communication port.

Response:
LO0()=&B10001001[cr/lf]
LO1()=&B00100100[cr/lf]
[cr/lf]
```

18.2 One LO port

Format

```
LOm()
```

Meaning
- Expresses the status of one LO port.
- When used as a readout file, the specified LO port status is read out.
- When used as a write file, the value is written to the specified LO port.

- Readout file

DATA FORMAT

```
LOm()=&Bnnnnnnnn[cr/lf]
```

- Write file

DATA FORMAT

```
&Bnnnnnnnn[cr/lf] or k[cr/lf]
```

Values

m	Port number: 0, 1
n	"0" or "1" (total of 8 digits). Corresponds to 07, 06, ..., 00, reading from the left.
k	Integer from 0 to 255

SAMPLE 1

```
SEND LO0() TO CMU············ Outputs the LO0 port status
                               from communication port.

Response:
LO0()=&B00000000[cr/lf]
```

SAMPLE 2

```
SEND CMU TO LO0()············ Inputs the LO0 port status
                               from communication port.
&B00000111

Response:
OK[cr/lf]
```

19 TO file

19.1 All TO information

Format

```
TO()
```

Meaning

- Expresses all TO (timer output variable) information.
- When used as a readout file, all TO information is read out.
- Cannot be used as a write file.

DATA FORMAT

```
TO0()=&Bnnnnnnnn [cr/lf]
TO1()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n	"0" or "1" (total of 8 digits). Corresponds to 07, 06, …, 00, reading from the left ("m" is the port No.).

▪ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND TO() TO CMU ·············· Outputs all TO status from
                                 communication port.

Response:
TO0()=&B10001001[cr/lf]
TO1()=&B10001001[cr/lf]
[cr/lf]
```

19.2 One TO port

Format

```
TOm()
```

Meaning
- Expresses the status of one TO port.
- When used as a readout file, the specified TO port status is read out.
- When used as a write file, the value is written to the specified TO port.

- Readout file

DATA FORMAT

```
TOm()=&Bnnnnnnnn[cr/lf]
```

- Write file

DATA FORMAT

```
&Bnnnnnnnn[cr/lf] or k[cr/lf]
```

Values

m	Port number: 0, 1
n	"0" or "1" (total of 8 digits). Corresponds to 07, 06, …, 00, reading from the left ("m" is the port No.).
k	Integer from 0 to 255

SAMPLE 1

```
SEND TO0() TO CMU············· Outputs the TO0 port status
                               from communication port.

Response:
TO0()=&B00000000[cr/lf]
```

SAMPLE 2

```
SEND CMU TO TO0()············· Inputs the TO0 port status
                               from communication port.
&B00000111

Response:
OK[cr/lf]
```

20 SI file

20.1 All SI information

Format

```
SI()
```

Meaning

- Expresses all SI (serial input variable) information.
- When used as a readout file, all SI information is read out.
- Cannot be used as a write file.

DATA FORMAT

```
SI0()=&Bnnnnnnnn [cr/lf]
SI1()=&Bnnnnnnnn [cr/lf]
     :
SI27()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

▪ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SI() TO CMU ·············· Outputs all SI status from
                                 communication port.

Response:
SI0()=&B10001001[cr/lf]
SI1()=&B00000010[cr/lf]
SI2()=&B00000000[cr/lf]
:
SI7()=&B00000000[cr/lf]
SI10()=&B00000000[cr/lf]
SI11()=&B00000000[cr/lf]
SI12()=&B00000000[cr/lf]
:
SI17()=&B00000000[cr/lf]
SI20()=&B00000000[cr/lf]
:
SI26()=&B00000000[cr/lf]
SI27()=&B00000000[cr/lf]
[cr/lf]
```

20.2 One SI port

Format

```
SIm()
```

Meaning
- Expresses the status of one SI port.
- When used as a readout file, the specified SI port status is read out.
- Cannot be used as a write file.

DATA FORMAT

```
SIm()=&Bnnnnnnnn[cr/lf]
```

Values

m	Port number: 0 to 7, 10 to 17, 20 to 27
n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

SAMPLE

```
SEND SI5() TO CMU············ Outputs the SI5 port status
                              from communication port.

Response:
SI5()=&B00000000[cr/lf]
```

21 SO file

21.1 All SO information

Format

```
SO()
```

Meaning

- Expresses all SO (serial output variable) information.
- When used as a readout file, all SO information is read out.
- Cannot be used as a write file.

DATA FORMAT

```
SO0()=&Bnnnnnnnn [cr/lf]
SO1()=&Bnnnnnnnn [cr/lf]
      :
SO27()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values

n ...."0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SO() TO CMU ·············· Outputs all SO status from
                                communication port.

Response:
SO0()=&B10001001[cr/lf]
SO1()=&B00000010[cr/lf]
SO2()=&B00000000[cr/lf]
:
SO7()=&B00000000[cr/lf]
SO10()=&B00000000[cr/lf]
SO11()=&B00000000[cr/lf]
SO12()=&B00000000[cr/lf]
:
SO17()=&B00000000[cr/lf]
SO20()=&B00000000[cr/lf]
:
SO26()=&B00000000[cr/lf]
SO27()=&B00000000[cr/lf]
[cr/lf]
```

21.2 One SO port

Format

```
SOm()
```

Meaning
- Expresses the status of one SO port.
- When used as a readout file, the specified SO port status is read out.
- When used as a write file, the value is written to the specified SO port. However, writing to SO0() and SO1() is prohibited.

- Readout file

DATA FORMAT

```
SOm()=&Bnnnnnnnn[cr/lf]
```

- Write file

DATA FORMAT

```
&Bnnnnnnnn[cr/lf] or k[cr/lf]
```

Values

m	Port number: 0 to 7, 10 to 17, 20 to 27
n	"0" or "1" (total of 8 digits). Corresponds to m7, m6, …, m0, reading from the left ("m" is the port No.).
k	Integer from 0 to 255

MEMO
- Writing to SO0() and SO1() is prohibited. Only reference is permitted.

SAMPLE 1

```
SEND SO5() TO CMU············ Outputs the SO5 port status
                              from communication port.

Response:
SO5()=&B00000000[cr/lf]
```

SAMPLE 2

```
SEND CMU TO SO5() ············ Inputs the SO5 port status
                               from communication port.
&B00000111

Response:
OK[cr/lf]
```

22 EOF file

22.1 EOF data

Format

```
EOF
```

Meaning

- This file is a special file consisting only of a ^Z (=1Ah) code. When transmitting data to an external device through the communication port, the EOF data can be used to add a ^Z code at the end of file.
- When used as a readout file, ^Z (=1Ah) is read out.
- Cannot be used as a write file.

DATA FORMAT

```
^Z (=1Ah)
```

SAMPLE

```
SEND PGM TO CMU
SEND EOF TO CMU················Outputs EOF data from
                               communication port.

NAME=TEST1[cr/lf]
A=1[cr/lf]
 :
HALT[cr/lf]
[cr/lf]
^Z
```

MEMO

- A "^Z" code may be required at the end of the transmitted file, depending on the specifications of the receiving device and application.

23 Serial port communication file

23.1 Serial port communication file

Format

```
CMU
```

Meaning
- Expresses the serial communication port.
- Depends on the various data formats.

SAMPLE

```
SEND PNT TO CMU················ Outputs all point data from
                                communication port.
SEND CMU TO PNT  ·············· Inputs all point data from
                                communication port.
```

24 SIW file

24.1 All SIW

Format

```
SIW()
```

Meaning

- Expresses all SIW (serial word input) data.
- Reads out all SIW information in hexadecimal digit when used as a readout file.
- Cannot be used as a write file.

DATA FORMAT

```
SIW( 0)=&Hnnnn [cr/lf]
SIW( 1)=&Hnnnn [cr/lf]
       :
SIW(15)=&Hnnnn [cr/lf]
[ cr/lf]
```

Values

n ....0 to 9, A to F: 4 digits (hexadecimal)

▪ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SIW() TO CMU············Outputs all SIW data from
                             communication port.

Response:
SIW(0)=&H1001[cr/lf]
SIW(1)=&H0010[cr/lf]
SIW(2)=&H0000[cr/lf]
 :
SIW(15)=&H0000[cr/lf]
[cr/lf]
```

24.2 One SIW data

Format

```
SIW(m)
```

Meaning

- Expresses one SIW status.
- Reads out all SIW information in hexadecimal digit when used as a readout file.
- Cannot be used as a write file.

DATA FORMAT

```
SIW(m)=&Hnnnn [cr/lf]
```

Values

m	0 to 15
n	0 to 9, A to F: 4 digits (hexadecimal)

SAMPLE

```
SEND SIW(5) TO CMU ··········· Outputs SIW(5) from
                               communication port.

Response:
SIW(5)=&H1001[cr/lf]
```

25 SOW file

25.1 All SOW

Format

```
SOW()
```

Meaning
- Expresses all SOW (serial word output) data.
- Reads out all SOW information in hexadecimal digit when used as a readout file.
- Cannot be used as a write file.

DATA FORMAT

```
SOW( 0)=&Hnnnn [cr/lf]
SOW( 1)=&Hnnnn [cr/lf]
       :
SOW(15)=&Hnnnn [cr/lf]
[ cr/lf]
```

Values n ....0 to 9, A to F: 4 digits (hexadecimal)

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SOW() TO CMU············Outputs all SOW data from
                             communication port.

Response:
SOW(0)=&H1001[cr/lf]
SOW(1)=&H0010[cr/lf]
SOW(2)=&H0000[cr/lf]
  :
SOW(15)=&H0000[cr/lf]
[cr/lf]
```

25.2 One SOW data

Format

```
SOW(m)
```

Meaning

- Expresses one SOW status.
- When used as a readout file, the specified SOW port status is read out.
- When used as a write file, the value is written to the specified SOW. However, writing to SOW0() and SOW1() is prohibited.

- Readout file

DATA FORMAT

```
SOW(m)=&Hnnnn [cr/lf]
```

- Write file

DATA FORMAT

```
&Hnnnn
```

Values

m	2 to 15
n	0 to 9, A to F: 4 digits (hexadecimal)

SAMPLE 1

```
SEND SOW(5) TO CMU ··········· Outputs SOW(5) from
                               communication port.

Response:
SOW(5)=&H1001[cr/lf]
```

SAMPLE 2

```
SEND CMU TO SOW(5) ··········· Input to SOW(5) from
                               communication port.
&H1001

Response:
OK[cr/lf]
```

26 Ethernet port communication file

26.1 Ethernet port communication file

Format

```
ETH
```

Meaning
- Expresses the Ethernet port.
- Depends on the various data formats.

SAMPLE

```
SEND PNT TO ETH················Outputs all point data from
                               the Ethernet port.
SEND ETH TO PNT················Inputs all point data from the
                               Ethernet port.
```

Chapter 9

User program examples

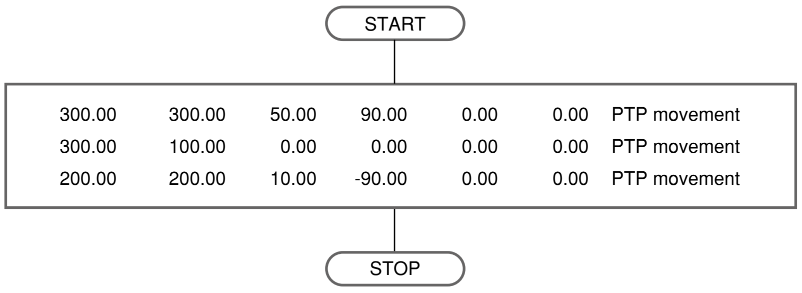
1 Basic operation

1.1 Directly writing point data in program

■ **Overview**

The robot arm can be moved by PTP (point-to-point) motion by directly specifying point data in the program.

Processing flow


START
300.00 300.00 50.00 90.00 0.00 0.00 PTP movement
300.00 100.00 0.00 0.00 0.00 0.00 PTP movement
200.00 200.00 10.00 -90.00 0.00 0.00 PTP movement
STOP

33C01-R7-00

SAMPLE

```
MOVE P,  300.00  300.00   50.00    90.00    0.00    0.00
MOVE P,  300.00  100.00    0.00     0.00    0.00    0.00
MOVE P,  200.00  200.00   10.00   -90.00    0.00    0.00
HALT
```

1.2 Using point numbers

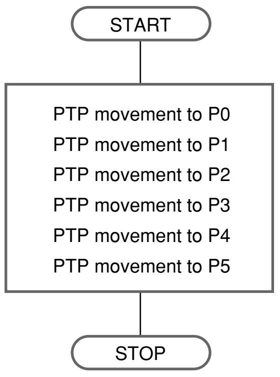
■ **Overview**

Coordinate data can be specified by using point numbers in a program. Coordinate data should be entered beforehand in "MANUAL>POINT" mode, for example as shown below.

```
POINT DATA

P0=      0.00     0.00     0.00     0.00     0.00     0.00
P1=    100.00     0.00   150.00    30.00     0.00     0.00
P2=      0.00   100.00    50.00     0.00     0.00     0.00
P3=    300.00   300.00     0.00     0.00     0.00     0.00
P4=    300.00   100.00   100.00    90.00     0.00     0.00
P5=    200.00   200.00     0.00     0.00     0.00     0.00
```

Processing flow



33C02-R7-00

```
SAMPLE 1

MOVE P,P0
MOVE P,P1
MOVE P,P2
MOVE P,P3
MOVE P,P4
MOVE P,P5
HALT
```

```
SAMPLE 2

FOR J=0 TO 5
   MOVE P,P[J]
NEXT J
HALT
```

Although the same operation is executed by both SAMPLE 1 and SAMPLE 2, the program can be shortened by using point Nos. and the FOR statement.

1.3 Using shift coordinates

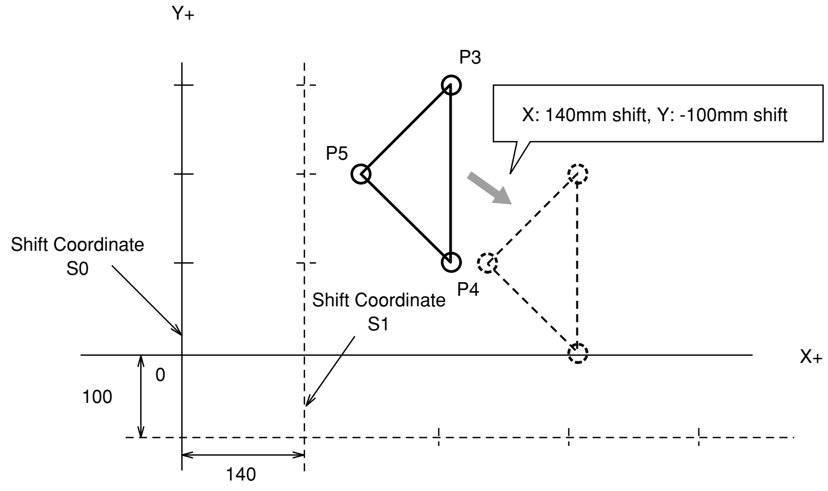
■ **Overview**

In the example shown below, after PTP movement from P3 to P5, the coordinate system is shifted +140mm along the X-axis and -100mm along the Y-axis, and the robot then moves from P3 to P5 again. The shift coordinate data is set in S1 and P3, P4, P5 are set as described in the previous section ("1.2 Using point numbers").

SHIFT DATA

```
S0=     0.00    0.00    0.00    0.00
S1=   140.00 -100.00    0.00    0.00
```

Shift Coordinate



33C03-R7-00

SAMPLE

```
SHIFT S0 ......................... Shift 0.
FOR J=3 TO 5...................... Repeated movement from P3 to P5.
   MOVE P, P[J]
NEXT J
SHIFT S1 ......................... Changed to "shift 1".
   FOR K=3 TO 5 .................. Repeated movement occurs in
                                   the same manner from P3 to P5.
   MOVE P,P[K]
NEXT K
```

1.4 Palletizing

1.4.1 Calculating point coordinates

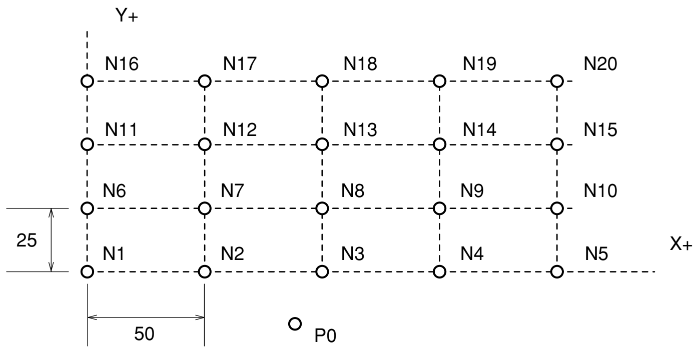
▪ **Overview**

Repetitive movement between a fixed work supply position P0 and each of the equally spaced points on a pallet can be performed with the following program.

In the drawing below, points N1 to N20 are on Cartesian coordinates, consisting of 5 points positioned at a 50mm pitch in the X-axis direction and 4 points at a 25mm pitch in the Y-axis direction. The robot arm moves from point to point in the order of P0-N1-P0-N2...N5-P0-N6-P0... while repeatedly moving back and forth between point P0 and each pallet.

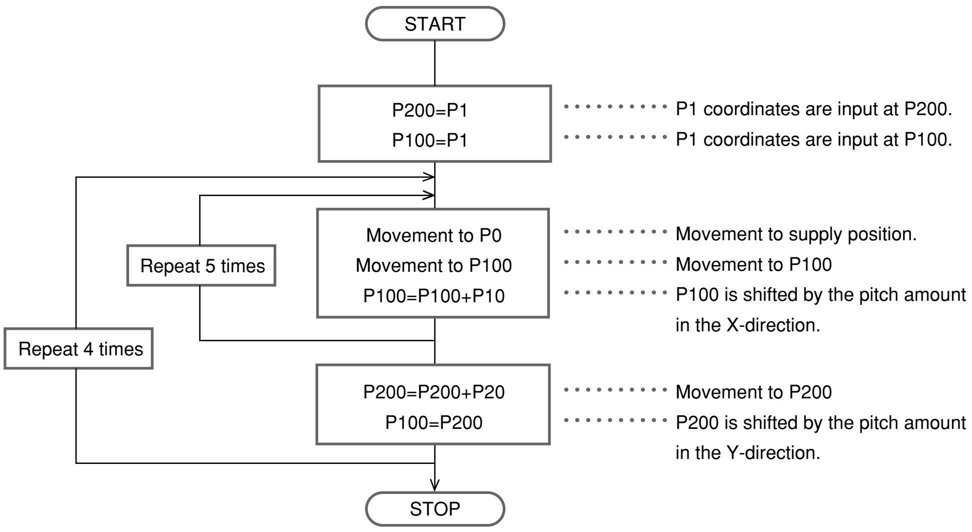
```
POINT DATA
Work supply position:
   P0=       0.0     0.0     0.0       0.0     0.0     0.0
X-axis pitch:
   P10=     50.0     0.0     0.0       0.0     0.0     0.0
Y-axis pitch:
   P20=      0.0    25.0     0.0       0.0     0.0     0.0
N1 position:
   P1 =    100.0    50.0     0.0       0.0     0.0     0.0
```

Calculating point coordinates



33C04-R7-00

Processing flow


START
P200=P1
P100=P1
P1 coordinates are input at P200.
P1 coordinates are input at P100.
Repeat 5 times
Repeat 4 times
Movement to P0
Movement to P100
P100=P100+P10
Movement to supply position.
Movement to P100
P100 is shifted by the pitch amount in the X-direction.
P200=P200+P20
P100=P200
Movement to P200
P200 is shifted by the pitch amount in the Y-direction.
STOP

31C05-R7-00

SAMPLE

```
P100=P1
P200=P1
FOR J=1 TO 4
   FOR K=1 TO 5
      MOVE P,P0
      MOVE P,P100
      P100=P100+P10
   NEXT K
   P200=P200+P20
   P100=P200
NEXT J
```

1.4.2 Utilizing pallet movement

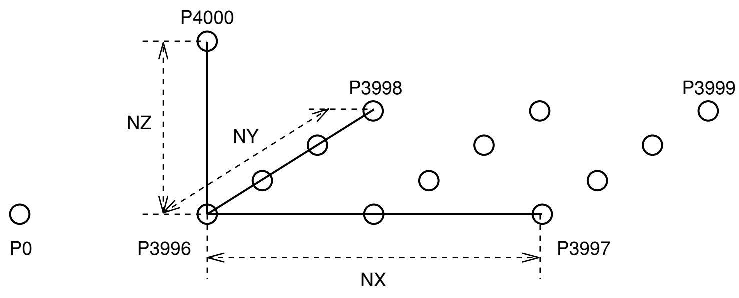
■ **Overview**

Repetitive movement between a fixed work supply position P0 and each of the equally spaced points on a pallet can be performed with the following program. In the drawing below, points N1 to N24 are on Cartesian coordinates, consisting of 3 points positioned at a 50mm pitch in the X-axis direction, 4 points at a 50mm pitch in the Y-axis direction, and 2 points at 100mm pitch in the Z-axis direction. The robot arm moves from point to point in the order of P0-N1-P0-N2...-N5-P0-N6... while repeatedly moving back and forth between point P0 and each pallet.

POINT DATA

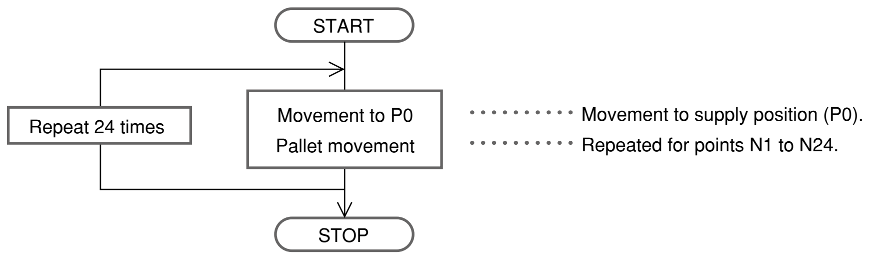
```
Work supply position:
   P0=      0.00     0.00   100.00      0.00     0.00     0.00
Pallet definition:
   PL0
NX=   3
NY=   4
NZ=   2
PLP=  3996  (P3996 to P4000 are used)
P3996=    100.00    50.00   100.00      0.00     0.00     0.00
P3997=    200.00    50.00   100.00      0.00     0.00     0.00
P3998=    100.00   200.00   100.00      0.00     0.00     0.00
P3999=    200.00   200.00   100.00      0.00     0.00     0.00
P4000=    100.00    50.00   200.00      0.00     0.00     0.00
```

Utilizing pallet movement



33C06-R7-00

Processing flow


START
Repeat 24 times
Movement to P0
Pallet movement
Movement to supply position (P0).
Repeated for points N1 to N24.
STOP

33C07-R7-00

SAMPLE

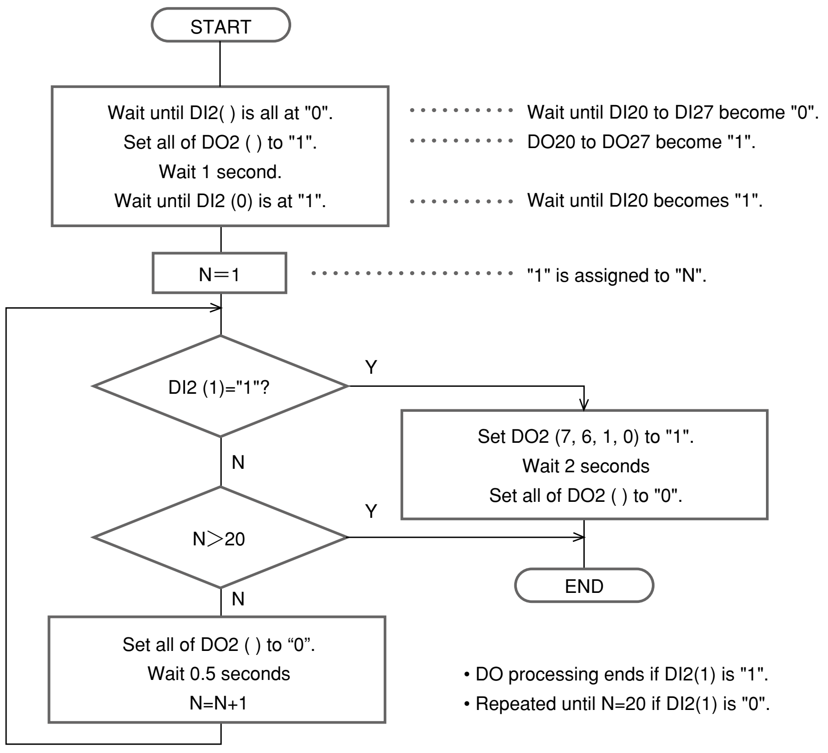
```
FOR I=1 TO 24 ·················· Repeated for I = 1 to 24.
   MOVE P,P0,Z=0.00 ·········· Movement to supply position.
   PMOVE (0,I),Z=0.00 ······· Movement to pallet point.
NEXT I
MOVE P,P0,Z=0.00
```

1.5 DI/DO (digital input and output) operation

■ **Overview**

The following example shows general-purpose signal input and output operations through the STD.

Processing flow


START
Wait until DI2( ) is all at "0".
Set all of DO2 ( ) to "1".
Wait 1 second.
Wait until DI2 (0) is at "1".
Wait until DI20 to DI27 become "0".
DO20 to DO27 become "1".
Wait until DI20 becomes "1".
N=1
"1" is assigned to "N".
DI2 (1)="1"?
Y
N
Set DO2 (7, 6, 1, 0) to "1".
Wait 2 seconds
Set all of DO2 ( ) to "0".
N>20
Y
N
END
Set all of DO2 ( ) to "0".
Wait 0.5 seconds
N=N+1
• DO processing ends if DI2(1) is "1".
• Repeated until N=20 if DI2(1) is "0".

33C08-R7-00

SAMPLE

```
WAIT DI2( )=0 ··················· Wait until DI20 to DI27 become "0".
DO2( )=&B11111111·············· DO20 to DO27 become "1".
DELAY 1000
WAIT DI2(0)=1 ··················· Wait until DI21 becomes "1".
N=1
*LOOP1:
IF DI2(1)=1 THEN *PROGEND··· Jumps to *PROGEND if DI21 = 1.
IF N>20 THEN *ALLEND ········ Ended in N > 20 (jumps to *ALLEND).
DO2( )=0 ························ DO20 to DO27 become "0".
DELAY 500
N=N+1
GOTO *LOOP1 ····················· Loop is repeated.
' END ROUTINE
*PROGEND: ························ End processing.
DO2(7,6,1,0)=&B1111··········· Set DO27, 26, 21, 20 to "1".
DELAY 2000······················· Wait 2 seconds
DO2( )=0 ························ Set DO20 to "0".
*ALLEND:
HALT
```

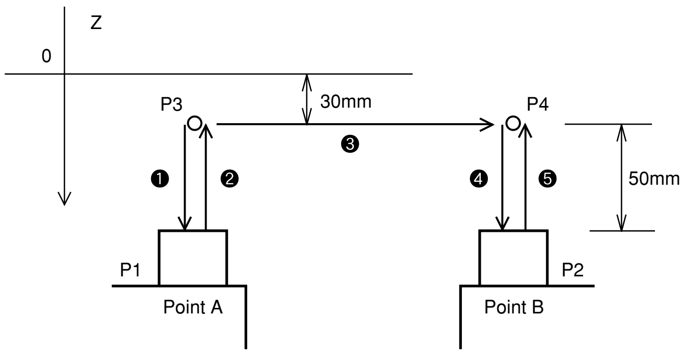
2 Application

2.1 Pick and place between 2 points

■ Overview

The following is an example for picking up a part at point A and placing it at point B.

Pick and place between 2 points



33C09-R7-00

■ Precondition

1. Set the robot movement path.
   - Movement path: P3→P1→P3→P4→P2→P4
   - Locate P3 and P4 respectively at a position 50mm above P1 and P2 and set the P1 and P2 positions by teaching.

2. I/O signal

DO (20)	Chuck (gripper) open/close =	0: open, 1: close

   - A 0.1 second wait time is set during chuck open and close.

SAMPLE: When calculating to find P3 and P4

```
   P3=P1 ························ P1 coordinates are assigned to P3.
   P4=P2 ························ P2 coordinates are assigned to P4.
   LOC3(P3)=LOC3(P3)-50.0····· P3 is shifted 50mm in Z UP direction.
   LOC3(P4)=LOC3(P4)-50.0····· P4 is shifted 50mm in Z UP direction.
   MOVE P,P3
   GOSUB *OPEN
   MOVE P,P1
   GOSUB *CLOSE
   MOVE P,P3
   MOVE P,P4
   MOVE P,P2
   GOSUB *OPEN
   MOVE P,P4
   HALT
*OPEN:   ························ Chuck OPEN routine.
   DO2(0)=0
   DELAY 100
   RETURN
*CLOSE:  ························ Chuck CLOSE routine.
   DO2(0)=1
   DELAY 100
   RETURN
```

SAMPLE: When using arch motion

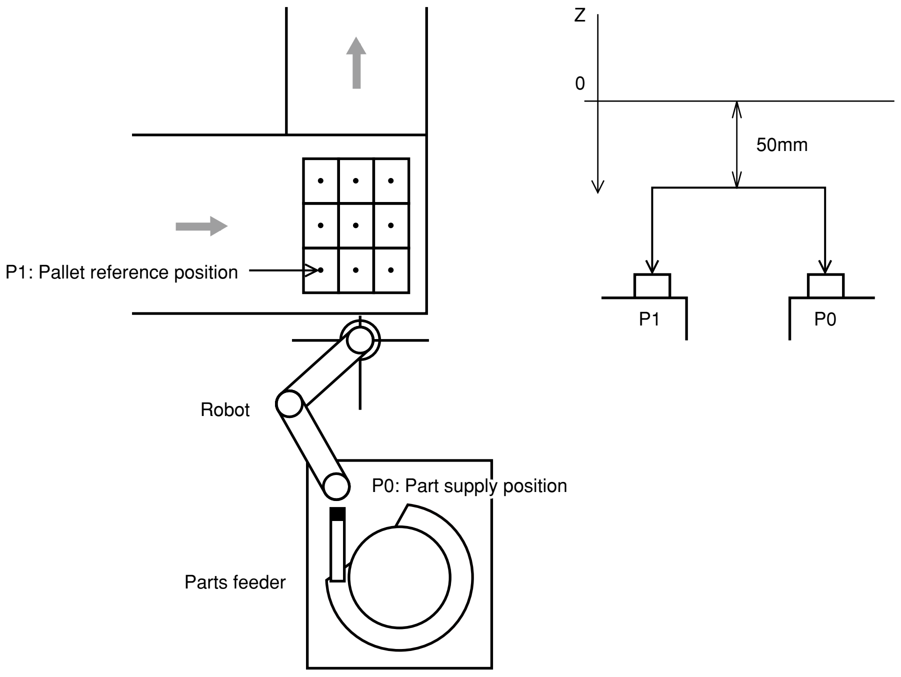
```
   P4=P2 ························· P2 coordinates are assigned to P4.
   LOC3(P4)=LOC3(P4)-50.0··· The  axis  3  data  of  P4  is
                             shifted by 50mm.
   GOSUB *OPEN
   MOVE P,P1,A3=30.0········ Arch motion at A3 = 30mm.
   GOSUB *CLOSE
   MOVE P,P2,A3=30.0········ Arch motion at A3 = 30mm.
   GOSUB *OPEN
   MOVE P,P4
   HALT
*OPEN:   ························· Chuck OPEN routine.
   DO2(0)=0
   DELAY 100
   RETURN
*CLOSE:  ························· Chuck CLOSE routine.
   DO2(0)=1
   DELAY 100
   RETURN
```

2.2 Palletizing

■ **Overview**

The following is an example for picking up parts supplied from the parts feeder and placing them on a pallet on the conveyor. The pallet is ejected when full.

Palletizing



33C10-R7-00

■ **Precondition**

1. I/O signal

DI (30)	Component detection sensor	1: Parts are supplied
DI (31)	Pallet sensor	1: Pallet is loaded

DO (30)	Robot hand open/close	0: Open / 1: Close
DO (31)	Pallet eject	1: Eject

Robot hand open/close time is 0.1 second and pallet eject time is 0.5 seconds.

2. The points below should be input beforehand as point data.

P0	Part supply position
P1	Pallet reference position
P10	X direction pitch
P11	Y direction pitch

3. Vertical movement is performed to a position Z=50mm above the pallet and parts feeder.

SAMPLE 1: When point is calculated

```
WHILE -1 ························ All repeated (-1 is always TRUE).
FOR A=0 TO 2
FOR B=0 TO 2
   WAIT DI(31)=1 ·············· Wait until a pallet "present"
                                status occurs.
   WAIT DI(30)=1 ·············· Wait until the supplied component
                                "present" status occurs.
   DO(30)=0 ··················· Robot hand OPENS.
   DELAY 100
   MOVE P,P0,A3=50.0 ·········· Movement to supply position.
   DO(30)=1 ··················· Robot hand CLOSES.
   DELAY 100
   P100=P1+P10*B+P11*A ········ Next point is calculated.
   MOVE P,P100,A3=50.0 ········ Movement to calculated point.
   DO(30)=0 ··················· Robot hand OPENS.
   DELAY 100
NEXT
NEXT
DRIVE (3,0) ···················· Only Z-axis moves to 0.
DO(31)=1 ······················· Pallet is ejected.
DELAY 500
DO(31)=0
WEND     ························ Loop is repeated.
HALT
```

SAMPLE 2: When using the palletizing function

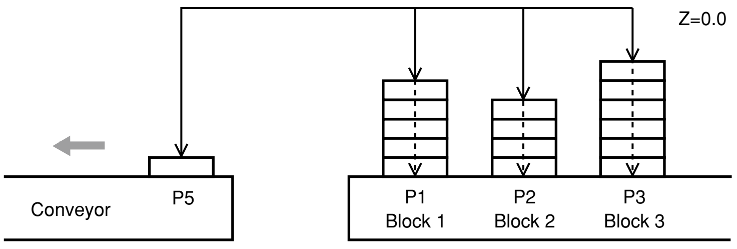
```
* Precondition: Must be defined at pallet "0".
WHILE -1 ························ All repeated.
FOR A=1 TO 9
   WAIT DI(31)=1 ·············· Wait until a pallet "present"
                                status occurs.
   WAIT DI(30)=1 ·············· Wait until the supplied component
                                "present" status occurs.
   DO(30)=0 ··················· Robot hand OPENS.
   DELAY 100
   MOVE P,P0,A3=50.0 ·········· Movement to supply position.
   DO(30)=1 ··················· Robot hand CLOSES.
   DELAY 100
   PMOVE(0,A),A3=50.0 ········· Movement to pallet point.
   DO(30)=0 ··················· Robot hand OPENS.
   DELAY 100
NEXT
DRIVE(3,0) ····················· Only Z-axis moves to 0.
DO(31)=1 Pallet is ejected.
DELAY 500
DO(31)=0
WEND     ························ Loop is repeated.
HALT
```

2.3 Pick and place of stacked parts

■ **Overview**

The following is an example for picking up parts stacked in a maximum of 6 layers and 3 blocks and placing them on the conveyor.
The number of parts per block may differ from others.
Parts are detected with a sensor installed on the robot hand.

Pick and place of stacked parts



33C11-R7-00

■ **Precondition**

1. I/O signal

DI (30)	Component detection sensor	1: Parts are supplied
DI (31)	Robot hand open/close	0: Open / 1: Close

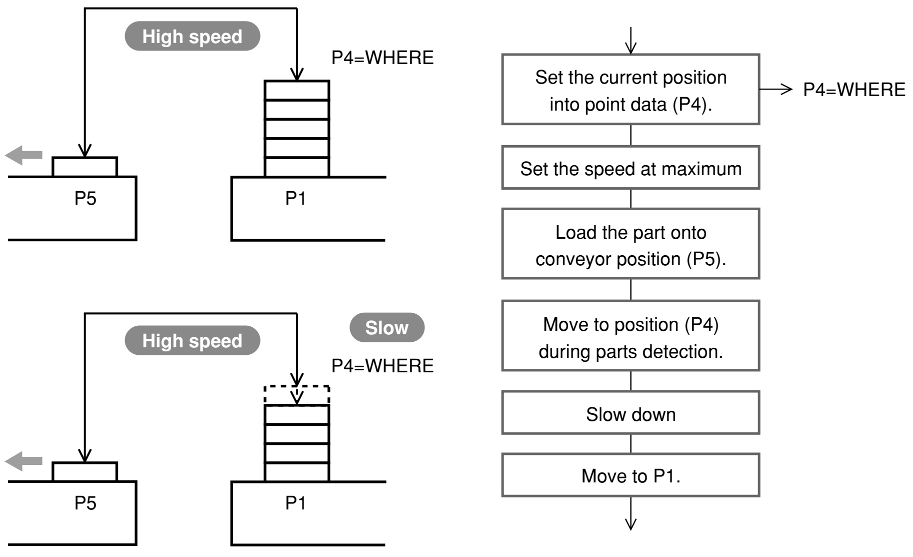
• Robot hand open/close time is 0.1 seconds.

2. The points below should be input beforehand as point data.

P1	Bottom of block 1
P2	Bottom of block 2
P3	Bottom of block 3
P5	Position on conveyor

3. Movement proceeds at maximum speeds but slows down when in proximity to the part.

Processing flow


High speed
P4=WHERE
P5
P1
High speed
Slow
P4=WHERE
P5
P1
Set the current position into point data (P4).
P4=WHERE
Set the speed at maximum
Load the part onto conveyor position (P5).
Move to position (P4) during parts detection.
Slow down
Move to P1.

33C12-R7-00

4. Use a STOPON condition in the MOVE statement for sensor detection during movement.

SAMPLE

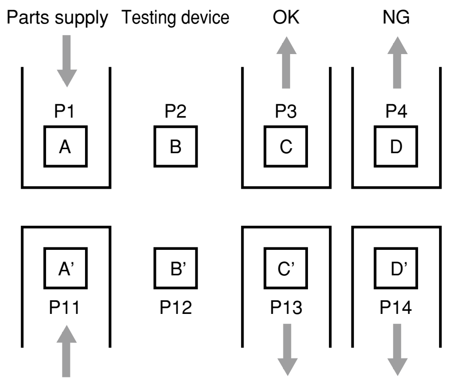
```
FOR A=1 TO 3
SPEED 100
GOSUB *OPEN
P6=P[A]
LOC3(P6)=0.00
MOVE P,P6,A3=0.0
WHILE -1
   SPEED 20
   MOVE P,P[A],STOPON DI3(0)=1
   IF DI3(0)=0 THEN *L1
   ' SENSOR ON
   P4=JTOXY(WHERE)
   GOSUB *CLOSE
   SPEED 100
   MOVE P,P5,A3=0.0
   GOSUB *OPEN
   MOVE P,P4,A3=0.0
WEND
*L1: ' SENSOR OFF
NEXT A
SPEED 100
DRIVE (3,0)
HALT
*OPEN:
DO3(0)=0
DELAY 100
RETURN
*CLOSE:
DO3(0)=1
DELAY 100
RETURN
```

2.4 Parts inspection (Multi-tasking example)

■ **Overview**

One robot is used to inspect two different parts and sort them according to the OK/NG results.
The robot picks up the part at point A and moves it to the testing device at point B. The testing device checks the part and sends it to point C if OK or to point D if NG.
The part at point A′ is picked up and moved to the testing device at point B' in the same way. The testing device checks the part and sends it to point C′ if OK or to point D′ if NG.
It is assumed that 10 to 15 seconds are required for the testing device to issue the OK/NG results.

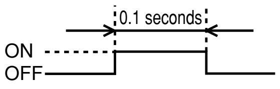
Parts inspection (Multi-tasking example)



33C13-R7-00

NOTE

- *1: As the start signal, supply a 0.1 second pulse signal to the testing device.

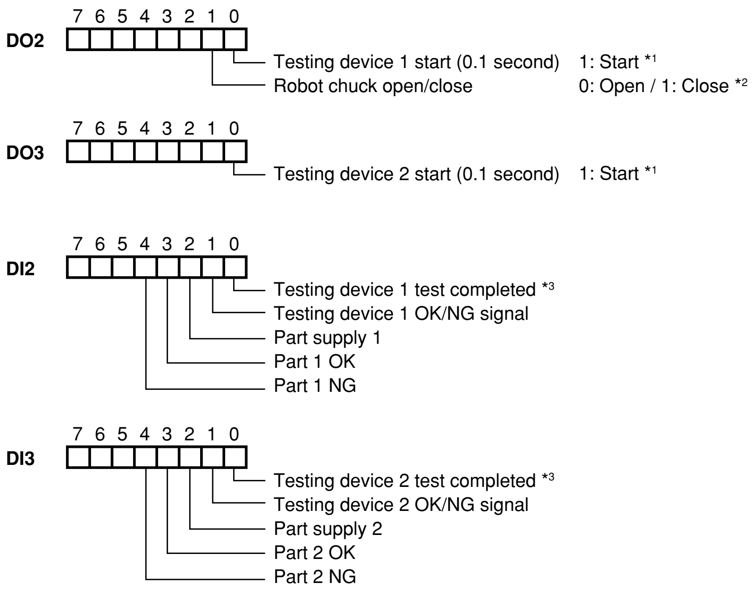


- *2: Chuck open and close time is 0.1 seconds.
- *3: Each time a test is finished, the test completion signal and OK/NG signal are sent from the testing device. After testing, the test completion signal turns ON (=1), and the OK/ NG signal turns ON (=1) when the result is OK and turns OFF (=0) when NG.

■ **Precondition**

1. I/O signal

I/O signal


DO2
7 6 5 4 3 2 1 0
Testing device 1 start (0.1 second) 1: Start *1
Robot chuck open/close 0: Open / 1: Close *2
DO3
7 6 5 4 3 2 1 0
Testing device 2 start (0.1 second) 1: Start *1
DI2
7 6 5 4 3 2 1 0
Testing device 1 test completed *3
Testing device 1 OK/NG signal
Part supply 1
Part 1 OK
Part 1 NG
DI3
7 6 5 4 3 2 1 0
Testing device 2 test completed *3
Testing device 2 OK/NG signal
Part supply 2
Part 2 OK
Part 2 NG

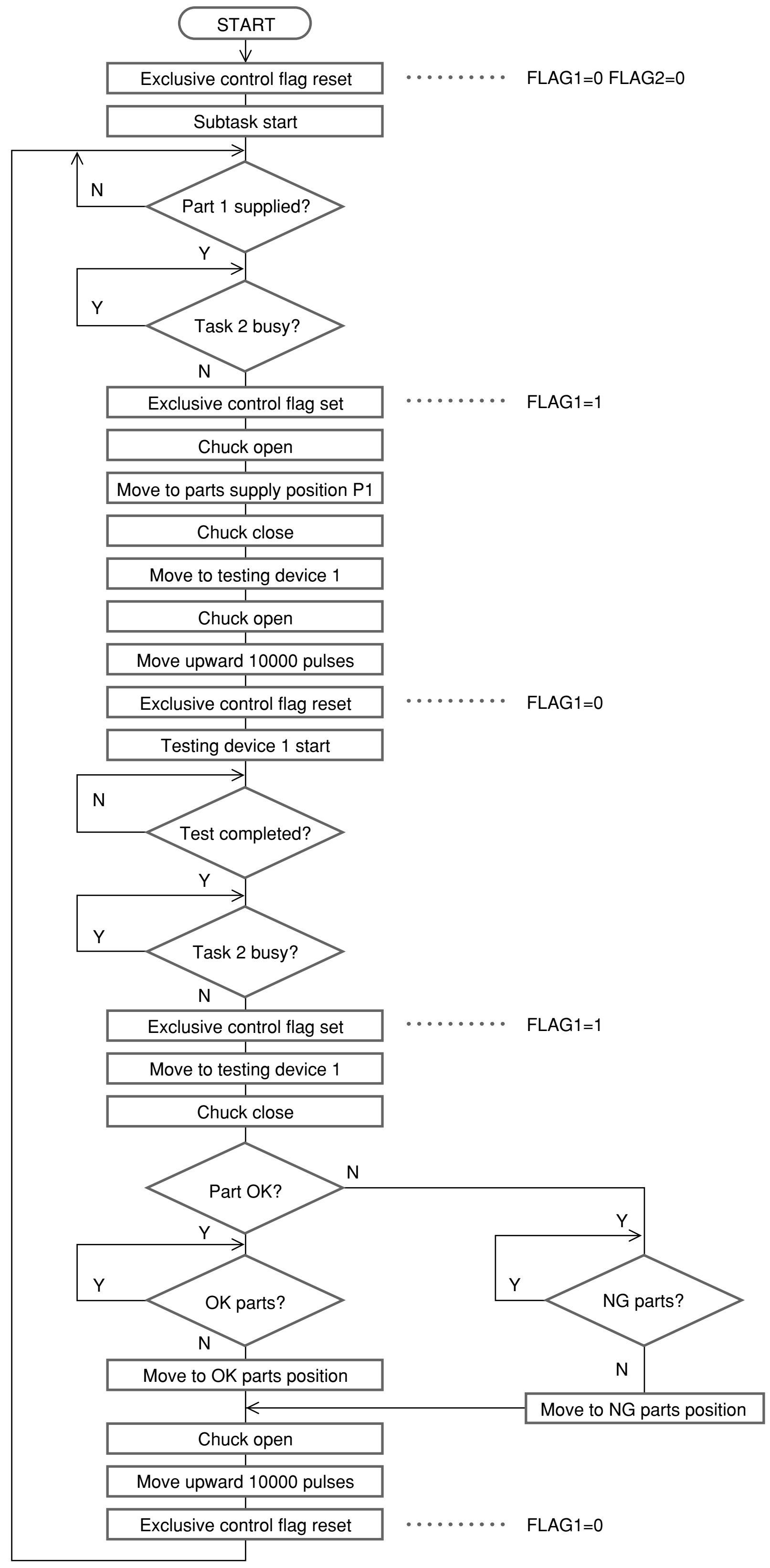
33C14-R7-00

2. The main task (task 1) is used to test part 1 and the subtask (task 2) is used to test part 2.
3. An exclusive control flag is used to allow other tasks to run while waiting for the test completion signal from the testing device.

FLAG1	0: Executing Task 1	(Task 2 execution enabled)
	1: Task 1 standby	(Task 2 execution disabled)
FLAG2	0: Executing Task 2	(Task 1 execution enabled)
	1: Task 2 standby	(Task 1 execution disabled)

4. Flow chart

Processing flow



33C15-R7-00

Task 2 (subtask) runs in the same flow.

■ **Program example**

SAMPLE

```
<Main task>
FLAG1=0                                                          <Subtask>
FLAG2=0
UPPOS=0.0
START <SUB_PGM>,T2   Subtask Start
*L1:
WAIT DI2(2)=1        Part supply standby                         *S1:
WAIT FLAG2=0         Other tasks waiting for standby status.    WAIT DI3(2)=1
FLAG1=1              Exclusive control flag set                  WAIT FLAG1=0
GOSUB *OPEN          Chuck open                                  FLAG2=1
MOVE P,P1,Z=UPPOS    Move to part supply position                GOSUB *OPEN
GOSUB *CLOSE         Chuck close                                 MOVE P,P11,Z=UPPOS
MOVE P,P2,Z=UPPOS    Move to testing device                      GOSUB *CLOSE
GOSUB *OPEN          Chuck open                                  MOVE P,P12,Z=UPPOS
DRIVEI (3,-10000)    Move Z-axis upward Z 10,000 pulses          GOSUB *OPEN
FLAG1=0              Exclusive control flag reset                DRIVEI (3,-10000)
DO2(0)=1             Testing device start                        FLAG2=0
DELAY 100                                                        DO3(0)=1
DO2(0)=0                                                         DELAY 100
WAIT DI2(0)=1        Test completion standby                     DO3(0)=0
WAIT FLAG2=0         Task completion standby                     WAIT DI3(0)=1
FLAG1=1              Exclusive control flag set                  WAIT FLAG1=0
MOVE P,P2,Z=UPPOS    Move to testing device                      FLAG2=1
GOSUB *CLOSE         Chuck close                                 MOVE P,P12,Z=UPPOS
IF DI2(1)=1 THEN     Test                                        GOSUB *CLOSE
'GOOD                                                            IF DI3(1)=1 THEN
WAIT DI4(2)=0        Part movement standby                       'GOOD
MOVE P,P3,Z=UPPOS    Move to OK parts position                   WAIT DI3(3)=0
ELSE                                                             MOVE P,P13,Z=UPPOS
'NG                                                              ELSE
WAIT DI2(4)=0        Part movement standby                       'NG
MOVE P,P4,Z=UPPOS    Move to NG parts position                   WAIT DI3(4)=0
ENDIF                                                            MOVE P,P14,Z=UPPOS
GOSUB *OPEN          Chuck open                                  ENDIF
DRIVEI (3,-10000)    Move Z-axis upward Z 10,000 pulses          GOSUB *OPEN
FLAG1=0              Exclusive control flag reset                DRIVEI (3,-10000)
GOTO *L1                                                         FLAG2=0
*OPEN:                                                           GOTO *S1
DO2(1)=0
DELAY 100
RETURN
*CLOSE:
DO2(1)=1
DELAY 100
RETURN
```

2.5 Connection to an external device through RS-232C (example 1)

■ **Overview**

Point data can be written in a program by using an external device connected to the RCX340 series controller via the RS-232C port.

■ **Precondition**

1. Input to the external device from the controller
   SDATA/X/Y [cr/lf]

2. Output to the controller from the external device

NOTE
• (cr/lf) indicates CR code (=0Dh) + LF code (=0Ah).

POINT DATA

```
P10=    156.42   243.91     0.00     0.00     0.00     0.00    [cr/lf]
```

SAMPLE

```
' INIT
   VCMD$="SDATA/X/Y"
   P0=  0.00    0.00    0.00    0.00    0.00    0.00
' MAIN ROUTINE
   MOVE P, P0
*ST:
   SEND VCMD$ TO CMU
   SEND CMU TO P10
   MOVE P, P10
GOTO *ST
```

MEMO

• "SEND xxx TO CMU" outputs the contents specified by "xxx" through the RS-232C.
• "SEND CMU TO xxx" sends data into the files specified by "xxx" through the RS-232C.

2.6 Connection to an external device through RS-232C (example 2)

■ **Overview**

Point data can be created from the desired character strings and written in a program by using an external device connected to the RCX340 controller via the RS-232C port.

■ **Precondition**

1. Input to the external device from the controller
   SDATA/X/Y [cr/lf]

2. Output to the controller from the external device
   X=156.42, Y=243.91 [cr/lf]

NOTE
- (cr/lf) indicates CR code (=0Dh) + LF code (=0Ah).

MEMO
- "SEND xxx TO CMU" outputs the contents specified by "xxx" through the RS-232C.
- "SEND CMU TO xxx" sends data into the files specified by "xxx" through the RS-232C.
- The LEN ( ) function obtains the length of the character string.
- The MID$ ( ) function obtains the specified character string from among the character strings.
- The VAL ( ) function obtains the value from the character string.

SAMPLE

```
' INT
   VCMD$="SDATA/X/Y"
   VIN$=""
   VX$=""
   VY$=""
   P0=      0.00    0.00    0.00     0.00    0.00    0.00
   P11=   100.00  100.00    0.00     0.00    0.00    0.00
' MAIN ROUTINE
   MOVE P, P0
*ST:
   SEND VCMD$ TO CMU
   SEND CMU TO VIN$
   I=1
   VMAX=LEN(VIN$)
*LOOP:
   IF I>VMAX THEN GOTO *E_LOOP
   C$=MID$(VIN$,I ,1)
   IF C$="X" THEN
      I=I+2
      J=I
*X_LOOP:
      C$=MID$(VIN$, J, 1)
      IF C$="," THEN
*X1_LP:
         L=J-I
         VX$=MID$(VIN$, I, L)
         I=J+1
         GOTO *LOOP
      ENDIF
      J=J+1
      IF J>VMAX  THEN  GOTO  *X1_LP
      GOTO  *X_LOOP
   ENDIF
   IF C$="Y"  THEN
      I=I+2
      J=I
*Y_LOOP:
      C$=MID$(VIN$, J, 1)
      IF C$=","THEN
*Y1_LP:
         L=J-I
         VY$=MID$(VIN$, I, L)
         I=J+1
         GOTO  *LOOP
      ENDIF
      J=J+1
      IF  J>VMAX  THEN  GOTO  *Y1_LP
      GOTO *Y_LOOP
   END IF
   I=I+1
   GOTO *LOOP
*E_LOOP:
   WX=VAL(VX$)
   WY=VAL(VY$)
   LOC1(P11)=WX
   LOC2(P11)=WY
   MOVE P, P11
GOTO  *ST
```

Chapter 10
Online commands

1 Online Command List

Online commands can be used to operate the controller via an RS-232C interface or via an Ethernet. This chapter explains the online commands which can be used. For details regarding the RS-232C and Ethernet connection methods, refer to the "RCX340 Controller User's Manual".

About termination codes

During data transmission, the controller adds the following codes to the end of a line of transmission data.

- RS-232C
  - CR (0Dh) and LF (0Ah) are added to the end of the line when the "Termination code" parameter of communication parameters is set to "CRLF".
  - CR (0Dh) is added to the end of the line when the "Termination code" parameter of communication parameters is set to "CR".
- Ethernet
  - CR (0Dh) and LF (0Ah) are added to the end of the line.

When data is received, then the data up to CR (0Dh) is treated as one line regardless of the "Termination code" parameter setting, so LF (0Ah) is ignored.
The termination code is expressed as [cr/lf] in the detailed description of each online command stated in "2 Operation and setting commands" onwards in this chapter.

1.1 Online command list: Function specific

Key operation

Operation type		Command	Option
Change mode	AUTO mode	AUTO	
	MANUAL mode	MANUAL	
Program	Reset program	RESET	
	Execute program	RUN	
	Execute one line	STEP	
	Skip one line	SKIP	
	Execute to next line	NEXT	
	Stop program	STOP	
Set break point		BREAK	m, n (m: break point No., n: line)
Change manual speed		MSPEED	k (k : 1-100)
Move to absolute reset position		ABSADJ	k, 0 or k, 1 (k : 1-6)
Return-to-origin		ORGRTN	k (k : 1-6)
Manual movement (inching)		INCH	k+ or k- (k : X, Y, Z, R, A, B)
Manual movement (jog)		JOG	k+ or k- (k : X, Y, Z, R, A, B)
Point data teaching		TEACH	m (m : point No.)

Utility

Operation type	Command	Option
Copy program 1 to program 2 Copy points "m - n" to point "k" Copy point comments "m - n" to point comment "k"	COPY	<program 1> TO <program 2> Pm-Pn TO Pk PCm-PCn TO PCk
Delete program Delete points "m - n" Delete point comments "m - n" Delete pallet "m"	ERA	<program> Pm-Pn PCm-PCn PLm
Rename "program 1" to "program 2"	REN	<program 1> TO <program 2>
Change program attribute	ATTR	<program> TO s (s : RW/RO)
Initialize data Program Point Shift Hand Pallet Point comment All data except parameters Parameter All data (MEM+PRM)	INIT	PGM PNT SFT HND PLT PCM MEM PRM ALL
Initialize data Communication parameter	INIT	CMU
Initialize data Error log	INIT	LOG
Setting Setting Access level Sequence execution flag	ACCESS SEQUENCE	k k
Check or set date Check or set time	DATE TIME	

Conditions:
1. Always executable.
2. Not executable during inputs from the programming box.
3. Not executable during inputs from the programming box, and while the program is running.
4. Not executable during inputs from the programming box, while the program is running, and when specific restrictions apply.

Data handling

Operation type		Command	Option	Condition
Acquiring status	Access level	?	ACCESS	1
	Arm status		ARM	
	Break point status		BREAK	
	Mode indication		MOD	
	Return-to-origin status		ORIGIN	
	Servo status		SERVO	
	Sequence execution flag status		SEQUENCE	
	Version information		VER	
	Current robot position (pulse coordinate)		WHERE	
	Current robot position (XY coordinate)		WHRXY	
	Task number		TASKS	
	Task operation status		TSKMON	
	Selected shift status		SHIFT	
	Selected hand status		HAND	
	Remaining memory capacity		MEM	
	Emergency stop status		EMG	
	Numerical data		Numerical expression	
	Character string data		Character string expression	
	Point data		Point expression	
	Shift data		Shift expression	
Data readout		READ		2
Data write		WRITE		2

Robot language independent execution

Operation type	Command	Option	Condition
Robot language executable independently			4

Control code

Operation type	Command	Option	Condition
Execution language interruption	^C(=03H)		1

Conditions:
1. Always executable.
2. Not executable during inputs from the programming box.
3. Not executable during inputs from the programming box, and while the program is running.
4. Not executable during inputs from the programming box, while the program is running, and when specific restrictions apply.

1.2 Online command list: In alphabetic order

Command	Option	Meaning	Condition
?	ACCESS	Acquire access level	1
	ARM	Acquire arm status	1
	BREAK	Acquire break point status	1
	CONFIG	Acquire controller configuration	1
	EMG	Acquire emergency stop status	1
	EXELVL	Acquire execution level	1
	HAND	Acquire selected hand status	1
	MEM	Acquire remaining memory capacity	1
	MOD	Acquire mode indication	1
	MSG [m, n]	Acquire error message	1
	OPSLOT	Acquire option slot status	1
	ORIGIN	Acquire return-to-origin status	1
	SELFCHK	Acquire error status by self-diagnosis	1
	SEQUENCE	Acquire sequence execution flag status	1
	SERVO	Acquire servo status	1
	SHIFT	Acquire selected shift status	1
	MSPEED	Acquire MANUAL speed status	1
	TASKS	Acquire task number	1
	TSKMON	Acquire task operation status	1
	VER	Acquire version	1
	WHERE	Acquire current robot position (pulse coordinate)	1
	WHRXY	Acquire current robot position (XY coordinate)	1
	WHRXYEX	Acquire current robot position (XY coordinate)	1
	Shift expression	Acquire shift data	1
	Point expression	Acquire point data	1
	Numeric expression	Acquire numeric data	1
	Character string expression	Acquire character string data	1
^C (=03H)		Execution language interruption	1
ABSADJ	k, 0 or k, 1 (k : 1-6)	Move to absolute reset position	3
ACCESS	k	Set access level	3
ATTR	<program> TO s (s : RW/RO)	Change program attribute	3
AUTO		Change mode: AUTO mode	3
BREAK	m, n (m: break point No., n: line)	Set break point	4
COPY	<program1> to <program2>	Copy program 1 to program 2	3
	PCm-PCn TO PCk	Copy point comments "m - n" to point comments "k"	3
	Pm-Pn TO Pk	Copy points "m - n" to points "k"	3
DATE		Check or set the date	2
EMGRST		Reset internal emergency stop flag	1
ERA	<program>	Delete program	3
	PCm-PCn	Delete point comments "m" to "n"	3
	PLm	Delete pallet "m"	3
	Pm-Pn	Delete points "m" to "n"	3
EXELVL	k	Execution level	3
INCH	k+ or k- (k : 1 to 6)	Manual movement (inching)	3
INIT	ALL	Initialize all data (MEM+PRM)	3
	CMU	Initialize communication parameter	3
	HND	Initialize hand data	3
	LOG	Initialize error history	3
	MEM	Initialize all memory data except parameters	3
	PCM	Initialize point comment data	3
	PGM	Initialize program data	3
	PLT	Initialize pallet data	3
	PNT	Initialize point data	3

Command	Option	Meaning	Condition
	PRM	Initialize parameter data	3
	SFT	Initialize shift data	3
JOG	k+ or k- (k : 1 to 6)	Manual movement (jog)	3
MANUAL		Change mode: MANUAL mode	3
MRKSET	k (k : 1-6)	Absolute reset on each axis	3
MSGCLR		Setting Clear line message	1
MSPEED	k (k : 1-100)	Change manual speed	3
NEXT		Execute program to next line	4
ORGRTN	k (k : 1-6)	Return-to-origin	3
READ		Read data	2
REN	<program 1> TO <program 2>	Change program name from "1" to "2"	3
RESET		Reset program	4
RUN		Execute program	4
SEQUENCE	k	Set sequence execution flag	3
SKIP		Program: Skip one line	4
STEP		Program: Execute one line	4
STOP		Stop program	2
TEACH	m (m: point number)	Point data teaching	3
TIME		Check or set time	2
WRITE		Write data	2
-		Robot language executable independently	4

Conditions:
1. Always executable.
2. Not executable during inputs from the programming box.
3. Not executable during inputs from the programming box, and while the program is running.
4. Not executable during inputs from the programming box, while the program is running, and when specific restrictions apply.

2 Operation and setting commands

2.1 Program operations

1. Register task

Command format

```
@LOAD    “<” <program name> “>”  [,Tn [, p]] [cr/lf]
         PGm
```

Response format

```
OK[cr/lf]
```

Values

m	Program number: 0 to 99
n	Task No: 1 to 16
P	Task priority ranking: 1 to 64

Meaning Registers the specified program into "task n" with "priority p". The registered program enters the STOP status. When "task number n" is omitted, the task with the smallest number of those that have not been started is specified automatically. When "task priority p" is omitted, "32" is specified.
The smaller value, the higher priority. The larger value, the lower priority (high 1 to low 64).
When the task with a high task priority is in the RUNNING status, the task with a low task priority still remains in the READY status.

SAMPLE

```
Command:   @LOAD <PG_MAIN>, T1 [cr/lf]···Registers the
                                          program to task 1.
Response:  OK [cr/lf]
```

2. Reset program

Command format

```
1.@RESET  [cr/lf]
2.@RESET  Tn                     [cr/lf]
          “<” <program name> “>”
          PGm
```

Response format

```
OK[cr/lf]
```

Values

n	Task No: 1 to 16
m	Program number: 0 to 99

Meaning Executes the program reset.
Command format 1 resets all programs. When restarting the program, the main program or the program that has been executed last in task 1 is executed from its beginning.
Command format 2 resets only the specified program. When restarting the program that has been reset, this program is executed from its beginning.

SAMPLE

```
Command:   @RESET [cr/lf]········ Resets all programs.
Response:  OK [cr/lf]
Command:   @RESET T3 [cr/lf]···· Resets only the program
                                 that is executed by T3.
Response:  OK [cr/lf]
```

3. Program execution

Command format

```
1.@RUN  [cr/lf]
2.@RUN │Tn                        │ [cr/lf]
       │"<" <program name> ">"    │
       │PGm                       │
```

Response format

```
OK[cr/lf]
```

Values n .... Task No: 1 to 16
m .... Program number: 0 to 99

Meaning Executes or stops the current program.
Command format 1 executes all programs in the STOP status.
Command format 2 executes only the specified program in the STOP status.

SAMPLE

```
Command:   @RUN [cr/lf]··········· Executes all programs in
                                   the STOP status.
Response:  OK [cr/lf]
Command:   @RUN T3 [cr/lf] ······· Executes only the program
                                   in the STOP status that is
                                   registered in T3.
Response:  OK [cr/lf]
```

4. Stop program

Command format

```
1.@STOP [cr/lf]
2.@STOP |Tn                          |[cr/lf]
        |"<" <program name> ">"      |
        |PGm                         |
```

Response format

```
OK[cr/lf]
```

NOTE

• Programs can be executed only in AUTO mode.

Values n ....Task No: 1 to 16
m ....Program number: 0 to 99

Meaning Stops the program.
Command format 1 stops all programs.
Command format 2 stops only the specified program.

SAMPLE

```
Command:   @STOP [cr/lf]·········· Stops all programs.
Response:  OK [cr/lf]
Command:   @STOP T3 [cr/lf]······ Stops only the program
                                  that is executed by T3.
Response:  OK [cr/lf]
```

5. Execute one program line

Command format

```
@STEP  |Tn                          |[cr/lf]
       |"<" <program name> ">"      |
       |PGm                         |
```

Command format

```
OK[cr/lf]
```

Values n ....Task No: 1 to 16
m ....Program number: 0 to 99

Meaning Executes one line of the specified program. When executing one line of the GOSUB statement or CALL statement, the program operation enters the subroutine or sub-procedure.

SAMPLE

```
Command:   @STEP T3 [cr/lf]······ Executes one line of the
                                  program that is executed
                                  by T3.
Response:  OK [cr/lf]
```

6. Skip one program line

Command format

```
@SKIP  | Tn                          | [cr/lf]
       | “<” <program name> “>”      |
       | PGm                         |
```

Response format

```
OK[cr/lf]
```

Values

n ....Task No: 1 to 16
m ....Program number: 0 to 99

Meaning Skips one line of the specified program. When skipping one line of the GOSUB statement or CALL statement, all subroutines or sub-procedures are skipped.

SAMPLE

```
Command:   @SKIP T3 [cr/lf]······Skips one line of the
                                 program that is executed
                                 by T3.
Response:  OK [cr/lf]
```

7. Execute program to next line

Command format

```
@NEXT  | Tn                          | [cr/lf]
       | “<” <program name> “>”      |
       | PGm                         |
```

Response format

```
OK[cr/lf]
```

Values

n ....Task No: 1 to 16
m ....Program number: 0 to 99

Meaning Executes the specified program to the next line. When executing the GOSUB statement or CALL statement, the subroutine or sub-procedure is executed one line.

SAMPLE

```
Command:   @NEXT T3 [cr/lf]······Skips one line of the
                                 program that is executed
                                 by T3.
Response:  OK [cr/lf]
```

8. Execute program to line before specified line

Command format

```
@RUNTO | Tn                       | , k [cr/lf]
       | “<” <program name> “>”   |
       | PGm                      |
```

Command format

```
OK[cr/lf]
```

Values

n	Task No: 1 to 16
m	Program number: 0 to 99
K	Specified line number: 1 to 9999

Meaning Executes the specified program to the line before the specified line.

SAMPLE

```
Command:   @RUNTO T3, 15 [cr/lf]··· Executes the program that
                                    is executed by T3 to the
                                    15th line.
Response:  OK [cr/lf]
```

9. Skip program to line before specified line

Command format

```
@SKIPTO | Tn                       | , k [cr/lf]
        | “<” <program name> “>”   |
        | PGm                      |
```

Command format

```
OK[cr/lf]
```

Values

n	Task No: 1 to 16
m	Program number: 0 to 99
K	Line number to set a break point: 1 to 9999

Meaning Skips the specified program to the line before the specified line.

SAMPLE

```
Command:   @SKIPTO T3, 15 [cr/lf]·· Skips the program that is
                                    executed by T3 to the 14th
                                    line.
Response:  OK [cr/lf]
```

10. Set break point

Command format

```
1.@BREAK│"<" <program name> ">"│(n[,n,n,...]), k [cr/lf]
        │ PGm                  │
2.@BREAK│"<" <program name> ">"│0 [cr/lf]
        │ PGm                  │
3.@BREAK 0 [cr/lf]
```

Command format

```
OK[cr/lf]
```

Values

m	Program number: 0 to 99
n	Specified line number: 1 to 9999
K	Set/Cancel: 0: Set, 1: Cancel

Meaning Sets a break point to pause the program during program execution.
Command format 1 sets or cancels a break point in the specified line of the specified program. Multiple lines can also be specified.
Command format 2 cancels all break points set in the specified program.
Command format 3 cancels all break points.

SAMPLE

```
 Command:@BREAK PG3 (1, 3), 1 [cr/lf]··· Sets a break point in
                                         the first and third
                                         lines of PG3.
Response:  OK [cr/lf]
```

2.2 MANUAL mode operation

1. Changing the MANUAL mode speed

Command format

```
@MSPEED[<robot number>] k[cr/lf]
```

Response format

```
OK[cr/lf]
```

Values

<robot number>	1 to 4
k	Manual movement speed: 1 to 100

Meaning Changes the manual mode movement speed of the robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
Command:   @MSPEED 50[cr/lf]
Response:  OK[cr/lf]
```

2. Point data teaching

Command format

```
@TEACH[<robot number>] mmmmm[cr/lf]
@TCHXY[<robot number>] mmmmm[cr/lf]
```

Response format

```
OK[cr/lf]
```

Values

<robot number>	1 to 4
mmmmm	Point number for registering point data: 0 to 29999

Meaning Registers the current robot position as point data for the specified point number. If point data is already registered in the specified point number, then that point data will be overwritten.
The unit of the point data may vary depending on the command.

TEACH	"pulse" units
TCHXY	"mm" units

SAMPLE

```
Command:   @TEACH 100[cr/lf]
Response:  OK[cr/lf]
```

2.3 Clearing output message buffer

Command format

```
@MSGCLR [cr/lf]
```

Response format

```
OK[cr/lf]
```

Values Clears the output message buffer of the controller. After the messages have been output by the PRINT statement, etc., the messages remaining in the buffer are cleared.

SAMPLE

```
Command:   @MSGCLR [cr/lf]
Response:  OK[cr/lf]
```

2.4 Setting input data

Command format

```
@INPUT | SET d |  ( [cr/lf]
       | CAN   |
       | CLR   |
```

Response format

```
OK[cr/lf]
```

Values d: Input data..........................Value that is matched to the type of the variable specified by the INPUT statement.
(Character string is enclosed by " ".)

Meaning Sets the input data to the data request by the INPUT statement.

SET	Sets the data.
CAN	Cancels the data request.
CLR	Clears the data that is not received by the controller.

SAMPLE

```
Command:   @INPUT "DATA" [cr/lf]
Response:  OK [cr/lf]
```

3 Reference commands

3.1 Acquiring return-to-origin status

Command format 1

```
@?ORIGIN[cr/lf]
```

Response format 1

```
x [cr/lf]
OK [cr/lf] ]
```

Command format 2

```
@?ORIGIN <robot number> [cr/lf]
```

Response format 2

```
x y{,y{,{...}}} [cr/lf]
OK [cr/lf]
```

Values

<robot number>.....................	1 to 4
x: Robot return-to-origin status..	0: Incomplete, 1: Complete
y: Axis return-to-origin status..	Shows the status of the axis 1, axis 2, ..., axis 6 from the left. 0: Incomplete, 1: Complete (Omitted when the axis is not connected.)

Meaning Acquires return-to-origin status.
Command format 1 acquires the return-to-origin status of all robots while command format 2 acquires the status of the specified robot.

SAMPLE

```
Command:   @?ORIGIN 2 [cr/lf]
Response:  0 1,1,0,1 ··············· Axis 3 of the robot 2 is
                                       in the return-to-origin
                                       incomplete status.
           OK [cr/lf]
```

3.2 Acquiring the servo status

Command format

```
@?SERVO [<robot number>] [cr/lf]
```

Response format

```
x y{,y{,{...}}} [cr/lf]
OK [cr/lf]
```

Values

<robot number>....................	1 to 4
x: Robot servo status	0: Servo off status, 1: Servo on status
y: Axis servo status	Shows the status of the axis 1, axis 2, ..., axis 6 from the left. 0: Mechanical brake on + dynamic brake on status 1: Servo on status 2: Mechanical brake off + dynamic brake off status (Omitted when the axis is not connected.)

Meaning Acquires the servo status. <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
Command:   @?SERVO[3] [cr/lf]
Response:  0 0,1,0,0 ·············· Only the axis 2 of the
                                    robot 3 is in the servo on
                                    status.
           OK [cr/lf]
```

3.3 Acquire motor power status

Command format

```
@?MOTOR [cr/lf]
```

Response format

```
x [cr/lf]
OK [cr/lf]
```

Values

x: Motor power status............	0: Motor power off status 1: Motor power on status 2: Motor power on + all robot servo on status

Meaning Acquires the motor power status.

SAMPLE

```
Command:   @?MOTOR [cr/lf]
Response:  2
           OK [cr/lf]
```

3.4 Acquiring the access level

Command format

```
@?ACCESS[cr/lf]
```

Response format

```
LEVELk[cr/lf]
```

Values k ....Access level: 0 to 1

Meaning Acquires the access level.

REFERENCE

- For a detailed description of the access level, refer to the Controller user's manual.

SAMPLE

```
Command:   @?ACCESS[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.5 Acquiring the break point status

Command format

```
@?BREAK[cr/lf]
```

Response format

```
k1,k2,k3,k4[cr/lf]
```

Values kn ....Line number on which break point "n" is set: 1 to 9999

Meaning Acquires the break point status.
When kn is 0, this means no break point is set.
When a break point is set in the COMMON program, the line number shows +10000.

SAMPLE

```
Command:   @?BREAK[cr/lf]
Response:  12,35,0,0[cr/lf]
```

3.6 Acquiring the mode status

Command format

```
@?MODE[cr/lf]
```

Response format

```
k[cr/lf]
OK[cr/lf]
```

Values k .... Mode status

Meaning Acquires the controller mode status.

SAMPLE

```
Command:   @?MODE[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.7 Acquiring the sequence program execution status

Command format

```
@?SEQUENCE[cr/lf]
```

Response format

```
1.  1,s[cr/lf]
2.  3,s[cr/lf]
3.  0[cr/lf]
```

Values

s	The sequence program's execution status is indicated as 1 or 0.
1	Program execution is in progress.
0	Program execution is stopped.

Meaning Acquires the sequence program execution status.
Response output means as follows:

1	Enabled
3	Enabled and output is cleared at emergency stop
0	Disabled

SAMPLE

```
Command:   @? SEQUENCE[cr/lf]
Response:  0[cr/lf]
```

3.8 Acquiring the version information

Command format

```
@?VER[cr/lf]
```

Response format

```
cv,cr-mv-dv1,dr1/dv2,dr2[cr/lf]
```

Values

cv	Host version number
cr	Host revision number (Rxxxx)
mv	PLO version number (Vx.xx)
dv? (?: 1, 2)	Driver version number (Vx.xx)
dr? (?: 1, 2)	Driver revision number (Rxxxx)

Meaning Acquires the version information.

SAMPLE

```
Command:   @?VER[cr/lf]
Response:  V8.02,R1021-V5.10-V1.01,R0001/V1.01,R0001[cr/lf]
```

3.9 Acquiring the current positions

1. Acquiring the current positions on pulse unit coordinates

Command format

```
@?WHERE[cr/lf]
```

Response format

```
xxxxxx yyyyyy zzzzzz rrrrrr aaaaaa bbbbbb[cr/lf]
```

Values

xxxxxx	Current position of axis 1 in "pulse" units
yyyyyy	Current position of axis 2 in "pulse" units
	:
bbbbbb	Current position of axis 6 in "pulse" units

Meaning Acquires the current positions.
WHERE: Acquires the current position of the specified robot.

SAMPLE

```
Command:   @?WHERE[cr/lf]
Response:  1000    2000     3000    -40000      0      0[cr/lf]
```

2. Acquiring the current positions on XY coordinates

Command format

```
@?WHRXY[cr/lf]
```

Response format

```
xxxxxx yyyyyy zzzzzz rrrrrr aaaaaa bbbbbb[cr/lf]
```

Values

xxxxxx	Current position of axis 1 in "mm" or "deg" units
yyyyyy	Current position of axis 2 in "mm" or "deg" units
	:
bbbbbb	Current position of axis 6 in "mm" or "deg" units

Meaning Acquires the current positions.
WHRXY: Acquires the current positions of the specified robot..

SAMPLE

```
Command:   @?WHRXY[cr/lf]
Response:  100.00   200.00   300.00   -40.00   0.00   0.00[cr/lf]
```

3.10 Acquiring the tasks in RUN or SUSPEND status

Command format

```
@?TASKS[cr/lf]
```

Response format

```
n{,n{,{...}}}[cr/lf]
```

Values n: Task number ....1 to 16 (Task currently run or suspended)

Meaning Acquires the tasks in RUN or SUSPEND status.

SAMPLE

```
Command:   @?TASKS[cr/lf]
Response:  1,3,4,6[cr/lf]
```

3.11 Acquiring the tasks operation status

Command format

```
@?TSKMON[cr/lf]
```

Response format

```
nfp,{nfp},{nfp},{nfp},{nfp},{nfp},{nfp},{nfp}[cr/lf]
```

Values

n : Line number being executed in each task	1 to 9999
f : Status of each task	R: RUN U: SUSPEND S: STOP
p : Priority level of each task	17 to 47

Meaning Acquires the status of each task in order from Task 1 to Task 8.

SAMPLE

```
Command:   @?TSKMON[cr/lf]
Response:  11R32,,43U32,,,,129R31,[cr/lf]
```

3.12 Acquiring the shift status

Command format

```
@?SHIFT[<robot number>][cr/lf]
```

Response format

```
m[cr/lf]
```

Values

<robot number>	1 to 4
m	Shift number selected for the specified robot: 0 to 39

Meaning Acquires the shift status of the robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
Command:   @?SHIFT[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.13 Acquiring the hand status

Command format

```
@?HAND[<robot number>][cr/lf]
```

Response format

```
m[cr/lf]
```

Values

<robot number>	1 to 4
m	Hand number selected for the specified robot: 0 to 31

Meaning Acquires the hand status of the robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified.

SAMPLE

```
Command:   @?HAND[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.14 Acquiring the remaining memory capacity

Command format

```
@?MEM[cr/lf]
```

Response format

```
k/m[cr/lf]
```

Values

k	Remaining source area (unit: bytes)
m	Remaining global identifier area (unit: bytes)

Meaning Acquires the remaining memory capacity.

SAMPLE

```
Command:   @?MEM[cr/lf]
Response:  102543/1342[cr/lf]
           OK[cr/lf]
```

3.15 Acquiring the emergency stop status

Command format

```
@?EMG[cr/lf]
```

Response format

```
k[cr/lf]
```

Values k ....Emergency stop status / 0: normal operation, 1: emergency stop

Meaning Acquires the emergency stop status by checking the internal emergency stop flag.

SAMPLE

```
Command:   @?EMG[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.16 Acquiring various values

1. Acquiring the value of a numerical expression

Command format

```
@? "numerical expression" [cr/lf]
```

Response format

```
"numerical value" [cr/lf]
```

Meaning Acquires the value of the specified numerical expression.
The numerical expression's value format is "decimal" or "real number".

SAMPLE 1

```
Command:   @?SQR(100*5)[cr/lf]
Response:  2.23606E01[c/lf]
           OK[cr/lf]
```

SAMPLE 2

```
Command:   @?LOC1(WHERE)[cr/lf]
Response:  102054[cr/lf]
           OK[cr/lf]
```

2. Acquiring the value of a character string expression

Command format

```
@? "character string expression" [cr/lf]
```

Response format

```
"character string " [cr/lf]
```

Meaning Acquires the value (character string) of the specified character string expression.

SAMPLE

```
If A$="ABC" and B$="DEF"
Command:   @?A$+B$+"123"[cr/lf]
Response:  ABCDEF123[cr/lf]
           OK[cr/lf]
```

3. Acquiring the value of a point expression

Command format

```
@? "point data expression" [cr/lf]
```

Response format

```
"point data" [cr/lf]
```

Meaning Acquires the value (point data) of the specified point expression.

SAMPLE

```
Command:   @?P1+WHRXY[cr/lf]
Response:  10.41  -1.60  52.15  3.00  0.00  0.00  0[cr/lf]
           OK[cr/lf]
```

4. Acquiring the value of a shift expression

Command format

```
@? "shift expression" [cr/lf]
```

Response format

```
"shift data" [cr/lf]
```

Meaning Acquires the value (shift data) of the specified shift expression.

SAMPLE

```
Command:   @?s1[cr/lf]
Response:  25.00  12.60  10.00  0.00[cr/lf]
           OK[cr/lf]
```

4 Operation commands

4.1 Absolute reset

Command format

```
@ABSADJ [<robot number>]  k,f[cr/lf]
@MRKSET [<robot number>]  k[cr/lf]
```

Response format

```
At the start of movement：RUN[cr/lf]
At the completion of movement：END[cr/lf]
```

Values

<robot number>	1 to 4
k	Designated axis: 1 to 6
f	Movement direction / 0: + direction, 1: - direction

Meaning Performs the absolute reset operation of a robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified.

ABSADJ	Moves the specified robot axis to an absolute reset position.
MRKSET	Performs absolute reset on the specified robot axis.

SAMPLE

```
Command:   @ABSADJ 1,0[cr/lf]
Response:  RUN[cr/lf] ············ Start of movement
           END[cr/lf] ············ Completion of movement
```

MEMO
• ABSADJ and MRKSET can be used at mark format axes.

4.2 Return-to-origin operation

Command format

```
@ORGRTN[<robot number>]  k[cr/lf]
```

Response format

```
At the start of movement：RUN[cr/lf]
At the completion of movement：END[cr/lf]
```

Values

<robot number>	1 to 4
k	Specified axis: 1 to 6

Meaning Performs the return-to-origin operation on the specified axis of the robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified. For the axis with the semi-absolute specifications, when the return-to-origin is executed, the absolute search operation is performed.

SAMPLE

```
Command:   @ORGRTN 1[cr/lf]
Response:  RUN[cr/lf] ············ Start of movement
           END[cr/lf] ············ Completion of movement
```

4.3 Manual movement: inching

Command format

```
@INCH [<robot number>] km [cr/lf]
@INCHXY [<robot number>] km [cr/lf]
@INCHT  [<robot number>] km [cr/lf]
```

Response format

```
At the start of movement：RUN[cr/lf]
At the completion of movement：END[cr/lf]
```

Values

<robot number>	1 to 4
k	Specified axis: 1 to 6
m	Movement direction / +, -

Meaning Manually moves (inching motion) the specified axis of the robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified. The robot performs the same motion as when moved manually in inching motion with the programming box jog keys (moves a fixed distance each time a jog key is pressed).
The unit of the movement amount and operation type by command are shown below.

INCH	"pulse" units. Only the specified axis moves.
INCHXY	"mm" units. According to the robot configuration, the arm tip of the robot moves in the direction of the Cartesian coordinate system.
INCHT	"mm" units. According to the robot configuration, the hand attached to the arm tip of the robot moves.

SAMPLE

```
Command:   @INCH 1+[cr/lf]
Response:  RUN [cr/lf] ············· Start of movement
           END [cr/lf] ············· Completion of movement
```

4.4 Manual movement: jog

Command format

```
@JOG [<robot number>] km [cr/lf]
@JOGXY [<robot number>] km [cr/lf]
@JOGT [<robot number>] km [cr/lf]
```

Response format

```
At the start of movement：RUN[cr/lf]
At the completion of movement：END[cr/lf]
```

Values

<robot number>	1 to 4
k	Designated axis: 1 to 6
m	Movement direction / +, -

Meaning Manually moves (jog motion) the specified axis of the robot specified by the <robot number>. <robot number> can be omitted. If it is omitted, robot 1 is specified. The robot performs the same motion as when holding down the programming box jog keys in manual mode.

To continue the operation, it is necessary for the JOG command to input the execution continue process (^V(=16H)) by the online command at intervals of 200ms. If not input, the error stop occurs.

Additionally, after the movement has started, the robot stops when any of the statues shown below arises.

- When software limit was reached.
- When interlock signal was turned off.
- When STOP key on the programming box was pressed.
- When an online command (^C (=03H)) to interrupt execution was input.

The unit of the movement amount and operation type by command are shown below.

JOG	"pulse" units. Only the specified axis moves.
JOGXY	"mm" units. According to the robot configuration, the arm tip of the robot moves in the direction of the Cartesian coordinate system.
JOGT	"mm" units. According to the robot configuration, the hand attached to the arm tip of the robot moves.

SAMPLE

```
Command:   @JOG 1+[cr/lf]
Response:  RUN[cr/lf] ············· Start of movement
           END[cr/lf] ············· Completion of movement
```

5 Data file operation commands

5.1 Copy operations

1. Copying a program

Command format

```
@COPY    <program name 1> TO <program name 2>  [cr/lf]
         PGn
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

<program name 1>	Program name in copy source (32 characters or less consisting of alphanumeric characters and underscore)
<program name 2>	Program name in copy destination (32 characters or less consisting of alphanumeric characters and underscore)
n: Program number	0 to 99

Meaning Copies the program specified by the <program name 1> or <program number> to <program name 2>.

SAMPLE

```
Command:   @COPY <TEST1> TO <TEST2> [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

2. Copying point data

Command format

```
@COPY  Pmmmmm-Pnnnnn  TO  Pkkkkk[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

mmmmm	Top point number in copy source: 0 to 29999
nnnnn	Last point number in copy source: 0 to 29999
kkkkk	Top point number in copy destination: 0 to 29999

Meaning Copies the point data between Pmmmmm and Pnnnnn to Pkkkkk.

SAMPLE

```
Command:   @COPY P101-P200 TO P1101[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

3. Copying point comments

Command format

```
@COPY  PCmmmmm-PCnnnnn  TO  PCkkkkk[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

mmmmm	Top point comment number in copy source: 0 to 29999
nnnnn	Last point comment number in copy source: 0 to 29999
kkkkk	Top point comment number in copy destination: 0 to 29999

Meaning Copies the point comments between PCmmmmm and PCnnnnn to PCkkkkk.

SAMPLE

```
Command:   @COPY PC101-PC200 TO PC1101[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

5.2 Erase

1. Erasing a program

Command format

```
@ERA | <program name> | [cr/lf]
     | PGn            |
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

<program name>	Program name to be erased (32 characters or less consisting of alphanumeric characters and underscore)
n: Program number	0 to 99

Meaning Erases the designated program.

SAMPLE

```
Command:   @ERA <TEST1> [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

2. Erasing point data

Command format

```
@ERA  Pmmmmm-Pnnnnn[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values	
mmmmm	Top point number to be erased: 0 to 29999
nnnnn	Last point number to be erased: 0 to 29999

Meaning Erases the point data between Pmmmmm and Pnnnnn.

SAMPLE

```
Command:   @ERA P101-P200[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

3. Erasing point comments

Command format

```
@ERA  PCmmmmm-PCnnnnn[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values	
mmmmm	Top point comment number to be erased: 0 to 29999
nnnnn	Last point comment number to be erased: 0 to 29999

Meaning Erases the point comments between PCmmmmm and PCnnnnn.

SAMPLE

```
Command:   @ERA PC101-PC200[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

4. Erasing point name

Command format

```
@ERA PNmmmmm-PNnnnnn [cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

mmmmm	Top point name number to be erased: 0 to 29999
nnnnn	Last point name number to be erased: 0 to 29999

Meaning Erases the point names between PNmmmmm and PNnnnnn.

SAMPLE

```
Command:   @ERA PC101-PC200[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

5. Erasing pallet data

Command format

```
@ERA  PLm[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

m	Pallet number to be erased: 0 to 39

Meaning Erases the PLm pallet data.

SAMPLE

```
Command:   @ERA PL1[cr/lf]
Response:  OK[cr/lf]
```

6. Erasing hand

Command format

```
@ERA Hm [cr/lf]
```

Response format

```
At process start : RUN [cr/lf]
At process end : END [cr/lf]
```

Values m ....Hand number to be erased: 0 to 31

Meaning Erases the hand definition data of "Hm".

SAMPLE

```
Command:   @ERA H2 [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

7. Erasing shift

Command format

```
@ERA Sm [cr/lf]
```

Response format

```
At process start : RUN [cr/lf]
At process end : END [cr/lf]
```

Values m ....Shift number to be erased: 0 to 39

Meaning Erases the shift data of "Sm".

SAMPLE

```
Command:   @ERA S1 [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

8. Erasing area check output setting

Command format

```
@ERA ACm [cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values m ....Area check output setting number to be erased: 0 to 31

Meaning Erases the area check output setting of "ACm".

SAMPLE

```
Command:   @ERA AC3 [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

9. Erasing general-purpose Ethernet port

Command format

```
@ERA GPm [cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values m ....General-purpose Ethernet port number to be erased: 0 to 15

Meaning Erases the general-purpose Ethernet port of "GPm".

SAMPLE

```
Command:   @ERA GP5 [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

5.3 Rename program name

Command format

```
@REN <program name 1> TO <program name 2> [cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

<program name 1>	Program name before renaming (32 characters or less consisting of alphanumeric characters and underscore)
<program name 2>	Program name after renaming (32 characters or less consisting of alphanumeric characters and underscore)
n: Program number	0 to 99

Meaning Changes the name of the specified program.

SAMPLE

```
Command:   @REN ＜TEST1＞ TO ＜TEST2＞[cr/lf]
Response:  RUN [cr/lf] ······· Process start
           END [cr/lf] ······· Process end
```

5.4 Changing the program attribute

Command format

```
@ATTR  | <program name> | TO s [cr/lf]
       | PGn            |
```

Response format

```
OK[cr/lf]
```

Values

<program name>	Program name to change the attribute (32 characters or less consisting of alphanumeric characters and underscore)
s: Attribute	RW: Readable and writable, RO: Read only, H: Hide
n: Program number	0 to 99

Meaning Changes the attribute of the program specified by the <program name> or <program number>.

SAMPLE

```
Command:   @ATTR <TEST1> TO RO[cr/lf]
Response:  OK[cr/lf]
```

5.5 Initialization process

1. Initializing the memory area

Command format

```
@INIT <memory area>[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values

<memory area> ....Memory area to be initialized.
One of the following memory areas is specified.

PGM	Initializes the program area.
PNT	Initializes the point data area.
SFT	Initializes the shift data area.
HND	Initializes the hand data area.
PLT	Initializes the pallet data area.
PCM	Initializes the point comment area.
PNM	Initializes the point name area.
ACO	Initializes the area check output setting area.
GEP	Initializes the general-purpose Ethernet port setting area.
MEM	Initializes the above areas (PGM ... all data up to PCM).
PRM	Initializes the parameter area.
ALL	Initializes all areas (MEM+PRM).

Meaning Initializes the memory area.

SAMPLE

```
Command:   @INIT PGM[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

2. Initializing the communication port

Command format

```
@INIT <communication port> [cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Values <communication port> ....Communication port to be initialized

Specify any of the ports shown below for the communication port.

CMU ....Initializes the RS-232C port.

ETH..Initializes the Ethernet port.

Meaning Initializes the communication port.

For information about the communication port initial settings, refer to the Controller user's manual.

SAMPLE

```
Command:   @INIT CMU [cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

3. Initializing the alarm log

Command format

```
@ INIT LOG[cr/lf]
```

Response format

```
At process start：RUN [cr/lf]
At process end：END [cr/lf]
```

Meaning Initializes the error log.

SAMPLE

```
Command:   @INIT LOG[cr/lf]
Response:  OK[cr/lf]
```

5.6 Data readout processing

Command format

```
@READ <readout file> [cr/lf]
```

Response format

```
At process start：BEGIN [cr/lf]
   (Data output: The contents may vary depending on
the readout file.)
At process end：END [cr/lf]
```

NOTE

- For more information about files, refer to the earlier Chapter 8 "Data file description".

Values <readout file> .... Designate a readout file name.

Meaning Reads out the data from the designated file.

Online commands that are input through the RS-232C port have the same meaning as the following command.

- SEND <readout file> TO CMU

Commands via Ethernet have the same meaning as the following command.

- SEND <readout file> TO ETH

Type	Readout file name	Definition format	
		All	Separate file
User memory	All files	ALL	———
	Program	PGM	<bb…b>>
	Point data	PNT	Pn
	Point comment	PCM	PCn
	Point name	PNM	PNn
	Parameter	PRM	/cccccccc/
	Shift definition	SFT	Sn
	Hand definition	HND	Hn
	Pallet definition	PLT	PLn
	General-purpose Ethernet port	GEP	GPn
	Input/output name	ION	iNMn(n)
	Area check output	ACO	ACn
Variable, constant	Variable	VAR	ab...by
	Array variable	ARY	ab...by(x)
	Constant	———	"cc...c"
Status	Program directory	DIR	<<bb…b>>
	Parameter directory	DPM	———
	Machine reference	MRF	———
	Machine reference (mark axis)	ARP	———
	Controller configuration	CFG	———
	Option board	OPT	———
	Self-diagnosis	SCK	———
	Alarm log	LOG	———
Device	DI port	DI()	DIn()
	DO port	DO()	DOn()
	MO port	MO()	MOn()
	TO port	TO()	TOn()
	LO port	LO()	LOn()
	SI port	SI()	SIn()
	SO port	SO()	SOn()
	SIW port	SIW()	SIWn()
	SOW port	SOW()	SOWn()
Others	File end code	EOF	———

a: Alphabetic character b: Alphanumeric character or underscore (_) c: Alphanumeric character or symbol
i: I/O type n: Number x: Expression (Array argument) y: variable type

SAMPLE

```
Command: @READ PGM [cr/lf]
         @READ P100 [cr/lf]
         @READ DINM2(0) [cr/lf]
```

5.7 Data write processing

Command format

```
@WRITE <write file> [cr/lf]
```

Response format

```
Input request display   ▶ READY[cr/lf]
After input is completed ▶ OK [cr/lf]
```

NOTE

• For more information about files, refer to the earlier Chapter 8 "Data file description".

Values <write file> .... Designate a write file name.

Meaning Writes the data in the designated file.
Online commands that are input through the RS-232C port have the same meaning as the following command.
• SEND CMU TO <write file>
Commands via Ethernet have the same meaning as the following command.
• SEND ETH TO <write file>

MEMO

• At the DO, MO, TO, LO, SO, SOW ports, an entire port (DO(), MO(), etc.) cannot be designated as a WRITE file.
• Some separate files (DOn(), MOn(), etc.) cannot be designated as a WRITE file. For details, see Chapter 8 "Data file description".

Type	Write file name	Definition format	
		All	Separate file
User memory	All files	ALL	———
	Program	PGM	<bb…b>>
	Point data	PNT	Pn
	Point comment	PCM	PCn
	Point name	PNM	PNn
	Parameter	PRM	/cccccccc/
	Shift definition	SFT	Sn
	Hand definition	HND	Hn
	Pallet definition	PLT	PLn
	General-purpose Ethernet port	GEP	GPn
	Input/output name	ION	iNMn(n)
	Area check output	ACO	ACn
Variable, constant	Variable	VAR	ab...by
	Array variable	ARY	ab...by(x)
Device	DO port	———	DOn()
	MO port	———	MOn()
	TO port	———	TOn()
	LO port	———	LOn()
	SO port	———	SOn()
	SOW port	———	SOWn()

a: Alphabetic character b: Alphanumeric character or underscore (_) c: Alphanumeric character or symbol
i: I/O type n: Number x: Expression (Array argument) y: variable type

SAMPLE

```
Command: @WRITE PRM [cr/lf]
         @WRITE P100 [cr/lf]
         @WRITE DINM2(0) [cr/lf] f]
```

6 Utility commands

6.1 Setting the sequence program execution flag

Command format

```
@SEQUENCE  k[cr/lf]
```

Response format

```
OK[cr/lf]
```

Values

k	Execution flag / 0: disable, 1: enable, 3: enable (DO reset)

Meaning Sets the sequence program execution flag.

SAMPLE

```
Command:   @ SEQUENCE 1[cr/lf]
Response:  OK[cr/lf]
```

6.2 Setting the date

Command format

```
@DATE  yy/mm/dd[cr/lf]
```

Response format

```
OK[cr/lf]
```

NOTE

- To change only the year or month, the slash (/) following it can be omitted.

  Example:

  To set the year to 2007, enter 07(cr/lf).

  To set the month to June, enter /06(cr/lf).

Values

yy/mm/dd	Date to be set. (year, month, day)
yy	Lower 2 digits of the year (00 to 99)
mm	Month (01 to 12)
dd	Day (01 to 31)

Meaning Sets a date in the controller

MEMO

- The currently set values are used for the omitted items.
- If only [cr/lf] is transmitted, then the date remains unchanged.
- If an improbable date is entered, then "5.2: Data error" occurs.

SAMPLE 1

```
To change only the day,
 //15[cr/lf]····················· Day is set to 15th.
```

SAMPLE 2

```
Command:   @DATE  14/01/14[cr/lf]
Response:  OK[cr/lf]
```

6.3 Setting the time

Command format

```
@TIME hh:mm:ss[cr/lf]
```

Response format

```
OK[cr/lf]
```

Values

hh:mm:ss	Current time
hh	hour (00 to 23)
mm	minute (00 to 59)
ss	second (00 to 59)

Meaning Sets the time of the controller.

MEMO
- The currently set values are used for the omitted items.
- If only [cr/lf] is transmitted, then the time remains unchanged.
- If an improbable time is entered, then "5.2: Data error" occurs.

SAMPLE 1

```
To change only the minute,
 :20:[cr/lf]····················· Minute is set to 20 minutes.
```

SAMPLE 2

```
Command:   @TIME 10:21:35[cr/lf]
Response:  OK[cr/lf]
```

7 Executing the robot language independently

Command format

```
@"robot language"[cr/lf]
```

Response format 1

```
OK[cr/lf] or NG=gg.eee [cr/lf]
```

Response format 2

```
At process start: RUN [cr/lf] or NG=gg.eee [cr/lf]
At process end: END [cr/lf] or NG=gg.eee [cr/lf]
```

Values

OK, END	Command ended correctly.
NG	An error occurred.
RUN	Command starts correctly.
gg: Alarm group number	0 to 99
ccc: Alarm classification number	0 to 999

Meaning Robot language commands can be executed.

- Independently executable commands can only be executed.
- Command format depends on each command to be executed.

SAMPLE 1

```
Command:   @SET DO(20) [cr/lf]
Response:  OK[cr/lf]
```

SAMPLE 2

```
Command:   @MOVE P,P100,S=20[cr/lf]
Response:  RUN [cr/lf] ············· Process start
           END [cr/lf] ············· Process end
```

8 Control codes

Command format

```
^C (=03H)
```

Response format

```
NG=1.8
```

Meaning Interrupts execution of the current command.

SAMPLE

```
Command:   @MOVE P,P100,S=20[cr/lf]
           ^C
Response:  NG=1.8[cr/lf]
```

Chapter 11
Appendix

1 Reserved word list

The words shown below are reserved for robot language and cannot be used as identifiers (variables, etc.).

A	DEC	HND	MOVET
ABS	DECEL	HOLD	MRF
ABSADJ	DEF	HOLDALL	MRKSET
ABSRPOS	DEGRAD	**I**	MSG
ACC	DELAY	IDIST	MSGCLR
ACCEL	DI	IF	MSPEED
ACCESS	DIM	IMP	**N**
ACO	DIR	INCH	NAME
ALL	DIST	INCHT	NEXT
ALM	DO	INCHXY	NOT
ALMRST	DPM	INIT	**O**
AND	DRIVE	INPUT	OFF
ARCHP1	DRIVEI	INT	OFFLINE
ARCHP2	DRV	ION	ON
• ARM	DS	**J**	ONLINE
ARMCND	DSPEED	JL	OPEN
ARMSEL	**E**	JOG	OPT
ARMTYP	ELSE	JOGT	OR
ARP	ELSEIF	JOGXY	ORD
ARY	EMG	JTOXY	ORGORD
ASPEED	END	**L**	ORGRTN
ATN	ENDIF	LEFT	ORIGIN
ATN2	EOF	LEFTY	OUT
ATTR	EQV	LEN	OUTPOS
AXWGHT	ERA	LET	**P**
B	ERL	LINEMODE	P
BIN	ERR	LOAD	PATH
BREAK	ERROR	LOC1	PC
C	ETH	LOC2	PCM
CALL	ETHSTS	LOC3	PDEF
CASE	EXIT	LOC4	PGM
CFG	EXITTASK	LOC5	PGMTSK
CHANGE	**F**	LOC6	PGN
CHGPRI	FN	LOCF	PLS
CHR	FOR	LOG	PLT
CLOSE	FREE	LSHIFT	PMOVE
CMU	**G**	**M**	PNM
CNT	GEP	MAINPG	PNT
CONT	GEPSTS	MCHREF	PPNT
COPY	GO	MEM	PRINT
COS	GOSUB	MID	PRM
CURTQST	GOTO	MNS	PSHFRC
CURTRQ	**H**	MOD	PSHJGSP
CUT	HALT	MODE	PSHMTD
D	HALTALL	MOTOR	PSHRSLT
DATE	HAND	MOVE	PSHSPD
DBP	HEX	MOVEI	PSHTIME

PUSH	SET	SWI	VEL
PWR	SETGEP	SYNCHK	VER
R	SETPW	**T**	**W**
RADDEG	SFT	XYTOJ	WAIT
RBT	SGI	**Y**	WEIGHT
READ	SGR	YZ	WEND
REF	SHARED	**Z**	WHERE
REM	SHIFT	TAG	WHILE
REN	SI	TAN	WHRXY
RESET	SID	TASKS	WHRXYEX
RESTART	SIN	TCHXY	WRITE
RESUME	SIW	TCOUNTER	**X**
RETURN	SKIP	TEACH	XOR
RIGHT	SKIPTO	THEN	XY
RIGHTY	SO	TIM	XYTOJ
RSHIFT	SOD	TIME	**Y**
RUN	SOW	TIMER	YZ
RUNTO	SPEED	TO	**Z**
S	SQR	TOLE	ZX
S	START	TORQUE	
SCK	STEP	TSKECD	
SELECT	STOP	TSKMON	
SEND	STOPON	TSKPGM	
SEQCMPL	STR	**V**	
SEQUENCE	SUB	VAL	
SERVO	SUSPEND	VAR	

Because the following names are used as system variable names, they cannot be used at the beginning of other variable names (n: numeric value).

Acn	FN	PCn	Sn
DIn	GPn	Pn	Son
DINMn	Hn	PNn	SONMn
DOn	LOn	SIn	TOn
DONMn	MOn	SINMn	

Variable name usage examples

- Although keywords which are reserved as robot language words cannot be used as they are, **they can be used as variable names if alphanumeric characters are added to them.**

Example: "ABS" cannot be used, but "ABS1" or "ABSX" can be used.

- Keywords reserved as system variables cannot be used at the beginning of other variable names, even if alphanumeric characters are added to them.

Example: "FN" cannot be used. "FNA" and "FN123" also cannot be used.

2 Robot Language Lists: Command list in alphabetic order

No.	Command	Function	Online	Type
A				
1	ABS	Acquires the absolute value of a specified value.	-	Command Statements
2	ABSRPOS	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)	-	Command Statements/ Functions
3	ACCEL	Specifies/acquires the acceleration coefficient parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP1	Specifies/acquires the arch position 1 parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP2	Specifies/acquires the arch position 2 parameter of a specified robot.	○	Command Statements/ Functions
5	ARMCND	Acquires the current arm status of a specified robot.	-	Functions
6	ARMSEL	Acquires the current “hand system” setting of a specified robot.	-	Functions
7	ARMTYP	Acquires the “hand system” setting of a specified robot.	-	Functions
8	ASPEED	Specifies/acquires the AUTO movement speed of a specified robot.	○	Command Statements/ Functions
9	ATN	Acquires the arctangent of the specified value.	-	Functions
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.	-	Functions
10	AXWGHT	Specifies/acquires the axis tip weight parameter of a specified robot.	○	Command Statements/ Functions
C				
11	CALL	Calls a sub-procedure.	×	Command Statements
12	CHANGE	Switches the hand of a specified robot.	○	Command Statements
13	CHGPRI	Changes the priority ranking of a specified task.	○	Command Statements
14	CHR$	Acquires a character with the specified character code.	-	Functions
15	COS	Acquires the cosine value of a specified value.	-	Functions
16	CURTQST	Acquires the current torque against the rated torque of a specified axis.	-	Functions
17	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.	-	Functions
18	CUT	Terminates a task currently being executed or temporarily stopped.	○	Command Statements
D				
19	DATE$	Acquires the date as a "yy/mm/dd" format character string.	-	Functions
20	DECEL	Specifies/acquires the deceleration rate parameter of a specified robot.	○	Command Statements/ Functions
21	DEF FN	Defines the functions that can be used by the user.	×	Command Statements
22	DEGRAD	Converts a specified value to radians (↔RADDEG).	-	Functions
23	DELAY	Waits for the specified period (units: ms).	×	Command Statements
24	DI	Acquires the input from the parallel port.	-	Functions
25	DIM	Declares the array variable name and the number of elements.	×	Command Statements
26	DIST	Acquires the distance between 2 specified points.	-	Functions
27	DO	Outputs a specified value to the DO port.	○	Command Statements
28	DRIVE	Moves a specified axis of a specified robot to an absolute position.	○	Command Statements
28	DRIVE	(With T-option) Executes an absolute movement command for a specified axis.	○	Command Statements
29	DRIVEI	Moves a specified axis of a specified robot to a relative position.	○	Command Statements

No.	Command	Function	Online	Type
E				
30	END SELECT	Terminates the SELECT CASE statement.	×	Command Statements
31	END SUB	Terminates the sub-procedure definition.	×	Command Statements
32	ERL	Gives the line No. where an error occurred.	-	Functions
32	ERR	Gives the error code number of an error which has occurred.	-	Functions
33	EXIT FOR	Terminates the FOR to NEXT statement loop.	×	Command Statements
34	EXIT SUB	Terminates the sub-procedure defined in SUB to END.	×	Command Statements
35	EXIT TASK	Terminates its own task which is in progress.	×	Command Statements
F				
36	FOR to NEXT	Controls repetitive operations. Executes the FOR to NEXT statement repeatedly until a specified value is exceeded.	×	Command Statements
G				
37	GOSUB to RETURN	Jumps to a subroutine with the label specified by a GOSUB statement, and executes that subroutine.	×	Command Statements
38	GOTO	Unconditionally jumps to the line specified by a label.	×	Command Statements
H				
39	HALT	Stops the program and performs a reset.	×	Command Statements
40	HALTALL	Stops and resets all programs.	×	Command Statements
41	HAND	Defines the hand of a specified robot.	○	Command Statements
42	HOLD	Temporarily stops the program.	×	Command Statements
43	HOLDALL	Temporarily stops all programs.	×	Command Statements
I				
44	IF	Allows control flow to branch according to conditions.	×	Command Statements
45	INPUT	Assigns a value to a variable specified from the programming box.	○	Command Statements
46	INT	Acquires an integer for a specified value by truncating all decimal fractions.	-	Functions
J				
47	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)	-	Functions
L				
48	LEFT$	Extracts a character string comprising a specified number of digits from the left end of a specified character string.	-	Functions
49	LEFTY	Sets the hand system of a specified robot to "Left."	○	Command Statements
50	LEN	Acquires the length (number of bytes) of a specified character string.	-	Functions
51	LET	Executes a specified assignment statement.	○	Command Statements
52	LO	Outputs a specified value to the LO port to enable/disable axis movement.	○	Command Statements
53	LOCx	Specifies/acquires point data for a specified axis or shift data for a specified element.	-	Command Statements/ Functions
54	LSHIFT	Shifts a value to the left by the specified number of bits. (↔RSHIFT)	-	Functions
M				
55	MCHREF	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.	-	Functions
56	MID$	Extracts a character string of a desired length from a specified character string.	-	Functions
57	MO	Outputs a specified value to the MO port.	○	Command Statements
58	MOTOR	Controls the motor power status.	○	Command Statements
59	MOVE	Performs absolute movement of all axes of a specified robot.	○	Command Statements

No.	Command	Function	Online	Type
60	MOVEI	Performs relative movement of all axes of a specified robot.	○	Command Statements
O				
61	OFFLINE	Sets a specified communication port to the "offline" mode.	○	Command Statements
62	ON ERROR GOTO	If an error occurs during program execution, this command allows the program to jump to the error processing routine specified by the label without stopping the program, or it stops the program and displays the error message.	×	Command Statements
63	ON to GOSUB	Jumps to a subroutine with labels specified by a GOSUB statement in accordance with the conditions, and executes that subroutine.	×	Command Statements
64	ON to GOTO	Jumps to label-specified lines in accordance with the conditions.	×	Command Statements
65	ONLINE	Sets the specified communication port to the "online" mode.	○	Command Statements
66	ORD	Acquires the character code of the first character in a specified character string.	-	Functions
67	ORGORD	Specifies/acquires the axis sequence parameter for performing return-to-origin and an absolute search operation in a specified robot.	○	Command Statements/ Functions
68	ORIGIN	Performs a return-to-origin.	○	Command Statements
69	OUT	Turns ON the bits of the specified output ports and the command statement ends.	×	Command Statements
70	OUTPOS	Specifies/acquires the OUT enable position parameter of a specified robot.	○	Command Statements/ Functions
P				
71	PDEF	Defines the pallet used to execute pallet movement commands.	○	Command Statements
72	PMOVE	Executes the pallet movement command of a specified robot.	○	Command Statements
73	Pn	Defines points within a program.	○	Command Statements
74	PPNT	Creates point data specified by a pallet definition number and pallet position number.	-	Functions
75	PRINT	Displays a character string at the programming box screen.	○	Command Statements
76	PSHFRC	Specifies/acquires the pushing thrust parameter.	○	Command Statements/ Functions
77	PSHJGSP	Specifies/acquires the pushing check speed threshold parameter.	○	Command Statements/ Functions
78	PSHMTD	Specifies/acquires the pushing method parameter.	○	Command Statements/ Functions
79	PSHRSLT	Acquires the status at the end of the PUSH statement.	-	Functions
80	PSHSPD	Specifies/acquires the pushing movement speed parameter.	○	Functions
81	PSHTIME	Specifies/acquires the pushing time parameter.	○	Functions
82	PUSH	Executes a pushing operation in the axis unit.	○	Command Statements
R				
83	RADDEG	Converts a specified value to degrees. (↔DEGRAD)	-	Functions
84	REM	Expresses a comment statement.	×	Command Statements
85	RESET	Turns the bit of a specified output port OFF.	○	Command Statements
86	RESTART	Restarts another task during a temporary stop.	○	Command Statements
87	RESUME	Resumes program execution after error recovery processing.	×	Command Statements
88	RETURN	Returns the processing branching with GOSUB to the next line of GOSUB.	×	Command Statements
89	RIGHT$	Extracts a character string comprising a specified number of digits from the right end of a specified character string.	-	Functions
90	RIGHTY	Sets the hand system of a specified robot to “Right.”	○	Command Statements
91	RSHIFT	Shifts a value to the right by the specified number of bits. (↔LSHIFT)	-	Functions

No.	Command	Function	Online	Type
S				
92	SELECT CASE to END SELECT	Allows control flow to branch according to conditions.	×	Command Statements
93	SEND	Sends a file.	○	Command Statements
94	SERVO	Controls the servo ON/OFF of a specified axis or all axes of a specified robot.	○	Command Statements
95	SET	Turns the bit at the specified output port ON.	△	Command Statements
96	SHARED	Enables reference with a sub-procedure without transferring a variable.	×	Command Statements
97	SHIFT	Sets the shift coordinate for a specified robot by using the shift data specified by a shift variable.	○	Command Statements
98	SIN	Acquires the sine value for a specified value.	-	Functions
99	Sn	Defines the shift coordinates within the program.	○	Command Statements
100	SO	Outputs a specified value to the SO port.	○	Command Statements
101	SPEED	Changes the program movement speed of a specified robot.	○	Command Statements
102	SQR	Acquires the square root of a specified value.	-	Functions
103	START	Specifies the task number and priority ranking of a specified program, and starts that program.	○	Command Statements
104	STR$	Converts a specified value to a character string (↔VAL)	-	Functions
105	SUB to END SUB	Defines a sub-procedure.	×	Command Statements
106	SUSPEND	Temporarily stops another task which is being executed.	×	Command Statements
107	SWI	Switches the program being executed, then begins execution from the first line.	×	Command Statements
T				
108	TAN	Acquires the tangent value for a specified value.	-	Functions
109	TCOUNTER	Outputs count-up values at 10ms intervals starting from the point when the TCOUNTER variable is reset.	-	Functions
110	TIME$	Acquires the current time as an "hh:mm:ss" format character string.	-	Functions
111	TIMER	Acquires the current time in seconds, counting from 12:00 midnight.	-	Functions
112	TO	Outputs a specified value to the TO port.	○	Command Statements
113	TOLE	Specifies/acquires the tolerance parameter of a specified robot.	○	Command Statements/ Functions
114	TORQUE	Specifies/acquires the maximum torque command value which can be set for a specified axis of a specified robot.	○	Command Statements/ Functions
V				
115	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)	-	Functions
W				
116	WAIT	Waits until the conditions of the DI/DO conditional expression are met (with time-out).	×	Command Statements
117	WAIT ARM	Waits until the axis operation of a specified robot is completed.	×	Command Statements
118	WEIGHT	Specifies/acquires the tip weight parameter of a specified robot.	○	Command Statements/ Functions
119	WEND	Terminates the command block of the WHILE statement.	×	Command Statements
120	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).	-	Functions
121	WHILE to WEND	Controls repeated operations.	×	Command Statements
122	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).	-	Functions
X				
123	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).	-	Functions

3 Robot Language Lists: Function Specific

Program commands

General commands

No.	Command	Function	Online	Type
25	DIM	Declares the array variable name and the number of elements.	×	Command Statements
51	LET	Executes a specified assignment statement.	○	Command Statements
84	REM	Expresses a comment statement.	×	Command Statements

Arithmetic commands

No.	Command	Function	Online	Type
1	ABS	Acquires the absolute value of a specified value.	-	Command Statements
9	ATN	Acquires the arctangent of the specified value.	-	Functions
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.	-	Functions
15	COS	Acquires the cosine value of a specified value.	-	Functions
22	DEGRAD	Converts a specified value to radians (↔RADDEG).	-	Functions
26	DIST	Acquires the distance between 2 specified points.	-	Functions
46	INT	Acquires an integer for a specified value by truncating all decimal fractions.	-	Functions
54	LSHIFT	Shifts a value to the left by the specified number of bits. (↔RSHIFT)	-	Functions
83	RADDEG	Converts a specified value to degrees. (↔DEGRAD)	-	Functions
91	RSHIFT	Shifts a value to the right by the specified number of bits. (↔LSHIFT)	-	Functions
98	SIN	Acquires the sine value for a specified value.	-	Functions
102	SQR	Acquires the square root of a specified value.	-	Functions
108	TAN	Acquires the tangent value for a specified value.	-	Functions

Date / time

No.	Command	Function	Online	Type
19	DATE $	Acquires the date as a "yy/mm/dd" format character string.	-	Functions
109	TCOUNTER	Outputs count-up values at 10ms intervals starting from the point when the TCOUNTER variable is reset.	-	Functions
110	TIME $	Acquires the current time as an "hh:mm:ss" format character string.	-	Functions
111	TIMER	Acquires the current time in seconds, counting from 12:00 midnight.	-	Functions

Character string operation

No.	Command	Function	Online	Type
14	CHR $	Acquires a character with the specified character code.	-	Functions
48	LEFT $	Extracts a character string comprising a specified number of digits from the left end of a specified character string.	-	Functions
50	LEN	Acquires the length (number of bytes) of a specified character string.	-	Functions
56	MID $	Extracts a character string of a desired length from a specified character string.	-	Functions
66	ORD	Acquires the character code of the first character in a specified character string.	-	Functions
89	RIGHT $	Extracts a character string comprising a specified number of digits from the right end of a specified character string.	-	Functions
104	STR $	Converts a specified value to a character string (↔VAL)	-	Functions
115	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)	-	Functions

Point, coordinates, shift coordinates

No.	Command	Function	Online	Type
12	CHANGE	Switches the hand of a specified robot.	○	Command Statements
41	HAND	Defines the hand of a specified robot.	○	Command Statements
47	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)	-	Functions
49	LEFTY	Sets the hand system of a specified robot to "Left."	○	Command Statements
53	LOCx	Specifies/acquires point data for a specified axis or shift data for a specified element.	-	Command Statements/ Functions
73	Pn	Defines points within a program.	○	Command Statements
74	PPNT	Creates point data specified by a pallet definition number and pallet position number.	-	Functions
90	RIGHTY	Sets the hand system of a specified robot to "Right."	○	Command Statements
99	Sn	Defines the shift coordinates in the program.	○	Command Statements
97	SHIFT	Sets the shift coordinate for a specified robot by using the shift data specified by a shift variable.	○	Command Statements
123	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).	-	Functions

Branching commands

No.	Command	Function	Online	Type
33	EXIT FOR	Terminates the FOR to NEXT statement loop.	×	Command Statements
36	FOR to NEXT	Controls repetitive operations. Executes the FOR to NEXT statement repeatedly until a specified value is exceeded.	×	Command Statements
37	GOSUB to RETURN	Jumps to a subroutine with the label specified by a GOSUB statement, and executes that subroutine.	×	Command Statements
38	GOTO	Unconditionally jumps to the line specified by a label.	×	Command Statements
44	IF	Allows control flow to branch according to conditions.	×	Command Statements
63	ON to GOSUB	Jumps to a subroutine with labels specified by a GOSUB statement in accordance with the conditions, and executes that subroutine.	×	Command Statements
64	ON to GOTO	Jumps to label-specified lines in accordance with the conditions.	×	Command Statements
92	SELECT CASE to END SELECT	Allows control flow to branch according to conditions.	×	Command Statements
121	WHILE to WEND	Controls repeated operations.	×	Command Statements

Error control

No.	Command	Function	Online	Type
62	ON ERROR GOTO	If an error occurs during program execution, this command allows the program to jump to the error processing routine specified by the label without stopping the program, or it stops the program and displays the error message.	×	Command Statements
87	RESUME	Resumes program execution after error recovery processing.	×	Command Statements
32	ERL	Gives the line No. where an error occurred.	-	Functions
32	ERR	Gives the error code number of an error which has occurred.	-	Functions

Program & task control

Program control

No.	Command	Function	Online	Type
11	CALL	Calls a sub-procedure.	×	Command Statements
39	HALT	Stops the program and performs a reset.	×	Command Statements
40	HALTALL	Stops all programs, resets task 1, and terminates all other tasks.	×	Command Statements
42	HOLD	Temporarily stops the program.	×	Command Statements
43	HOLDALL	Temporarily stops all programs.	×	Command Statements
107	SWI	Switches the program being executed, performs compiling, then begins execution from the first line.	×	Command Statements

Task control

No.	Command	Function	Online	Type
13	CHGPRI	Changes the priority ranking of a specified task.	-	Command Statements
18	CUT	Terminates a task currently being executed or temporarily stopped.	○	Command Statements
35	EXIT TASK	Terminates its own task which is in progress.	×	Command Statements
82	PUSH	Executes a pushing operation in the axis unit.	○	Command Statements
86	RESTART	Restarts another task during a temporary stop.	○	Command Statements
103	START	Specifies the task number and priority ranking of a specified task, and starts that task.	○	Command Statements
106	SUSPEND	Temporarily stops another task which is being executed.	×	Command Statements

Robot control

Robot operations

No.	Command	Function	Online	Type
12	CHANGE	Switches the hand of a specified robot.	○	Command Statements
28	DRIVE	Moves a specified axis of a specified robot to an absolute position.	○	Command Statements
29	DRIVEI	Moves a specified axis of a specified robot to a relative position.	○	Command Statements
41	HAND	Defines the hand of a specified robot.	○	Command Statements
49	LEFTY	Sets the hand system of a specified robot to “Left.”	○	Command Statements
58	MOTOR	Controls the motor power status.	○	Command Statements
59	MOVE	Performs absolute movement of all axes of a specified robot.	○	Command Statements
60	MOVEI	Performs relative movement of all axes of a specified robot.	○	Command Statements
68	ORIGIN	Performs a return-to-origin.	○	Command Statements
72	PMOVE	Executes the pallet movement command of a specified robot.	○	Command Statements
90	RIGHTY	Sets the hand system of a specified robot to “Right.”	○	Command Statements
94	SERVO	Controls the servo ON/OFF of a specified axis or all axes of a specified robot.	○	Command Statements

Status acquisition

No.	Command	Function	Online	Type
2	ABSRPOS	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)	-	Command Statements/ Functions
5	ARMCND	Acquires the current arm status of a specified robot.	-	Functions
6	ARMSEL	Acquires the current “hand system” setting of a specified robot.	-	Functions
7	ARMTYP	Acquires the “hand system” setting of a specified robot.	-	Functions
16	CURTQST	Acquires the current torque against the rated torque of a specified axis.	-	Functions
55	MCHREF	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.	-	Functions
79	PSHRSLT	Acquires the status at the end of the PUSH statement.	-	Functions
80	PSHSPD	Specifies/acquires the pushing movement speed parameter.	○	Command Statements/ Functions
81	PSHTIME	Specifies/acquires the pushing time parameter.	○	Command Statements/ Functions
117	WAIT ARM	Waits until the axis operation of a specified robot is completed.	×	Command Statements
120	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).	-	Functions
122	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).	-	Functions

Status change

No.	Command	Function	Online	Type
3	ACCEL	Specifies/acquires the acceleration coefficient parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP1	Specifies/acquires the arch position 1 parameter of a specified robot.	○	Command Statements/ Functions
4	ARCHP2	Specifies/acquires the arch position 2 parameter of a specified robot.	○	Command Statements/ Functions

No.	Command	Function	Online	Type
8	ASPEED	Specifies/acquires the AUTO movement speed of a specified robot.	○	Command Statements/ Functions
10	AXWGHT	Specifies/acquires the axis tip weight parameter of a specified robot.	○	Command Statements/ Functions
20	DECEL	Specifies/acquires the deceleration rate parameter of a specified robot.	○	Command Statements/ Functions
67	ORGORD	Specifies/acquires the axis sequence parameter for performing return-to-origin and an absolute search operation in a specified robot.	○	Command Statements/ Functions
70	OUTPOS	Specifies/acquires the OUT enable position parameter of a specified robot.	○	Command Statements/ Functions
71	PDEF	Defines the pallet used to execute pallet movement commands.	○	Command Statements
76	PSHFRC	Specifies/acquires the pushing thrust parameter.	○	Command Statements/ Functions
77	PSHJGSP	Specifies/acquires the pushing check speed threshold parameter.	○	Command Statements/ Functions
78	PSHMTD	Specifies/acquires the pushing method parameter.	○	Command Statements/ Functions
101	SPEED	Changes the program movement speed of a specified robot.	○	Command Statements
113	TOLE	Specifies/acquires the tolerance parameter of a specified robot.	○	Command Statements/ Functions
118	WEIGHT	Specifies/acquires the tip weight parameter of a specified robot.	○	Command Statements/ Functions

Input/output & communication control

Input/output control

No.	Command	Function	Online	Type
23	DELAY	Waits for the specified period (units: ms).	×	Command Statements
27	DO	Outputs a specified value to the DO port.	○	Command Statements
52	LO	Outputs a specified value to the LO port to enable/disable axis movement.	○	Command Statements
57	MO	Outputs a specified value to the MO port.	○	Command Statements
69	OUT	Turns ON the bits of the specified output ports and the command statement ends.	×	Command Statements
85	RESET	Turns the bit of a specified output port OFF.	○	Command Statements
95	SET	Turns the bit at the specified output port ON.	△	Command Statements
100	SO	Outputs a specified value to the SO port.	○	Command Statements
112	TO	Outputs a specified value to the TO port.	○	Command Statements
116	WAIT	Waits until the conditions of the DI/DO conditional expression are met (with time-out).	×	Command Statements

Communication control

No.	Command	Function	Online	Type
65	ONLINE	Sets the specified communication port to the "online" mode.	○	Command Statements
61	OFFLINE	Sets a specified communication port to the "offline" mode.	○	Command Statements
93	SEND	Sends a file.	○	Command Statements

4 Functions: in alphabetic order

No.	Function	Type	Function
A			
1	ABS	Arithmetic function	Acquires the absolute value of a specified value.
2	ABSRPOS	Arithmetic function	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)
3	ACCEL	Arithmetic function	Acquires the acceleration coefficient parameter of a specified robot.
4	ARCHP1	Arithmetic function	Acquires the arch position 1 parameter of a specified robot.
4	ARCHP2	Arithmetic function	Acquires the arch position 2 parameter of a specified robot.
5	ARMCND	Arithmetic function	Acquires the current arm status of a specified robot.
6	ARMSEL	Arithmetic function	Acquires the current “hand system” setting of a specified robot.
7	ARMTYP	Arithmetic function	Acquires the “hand system” setting of a specified robot.
8	ASPEED	Arithmetic function	Sets the automatic movement speed.
9	ATN	Arithmetic function	Acquires the arctangent of the specified value.
9	ATN2	Arithmetic function	Acquires the arctangent of the specified X-Y coordinates.
10	AXWGHT	Arithmetic function	Acquires the axis tip weight parameter of a specified robot.
C			
14	CHR$	Character string function	Acquires a character with the specified character code.
15	COS	Arithmetic function	Acquires the cosine value of a specified value.
16	CURTQST	Arithmetic function	Acquires the current torque against the rated torque of a specified axis.
17	CURTRQ	Arithmetic function	Acquires the current torque value of the specified axis of a specified robot.
D			
19	DATE$	Character string function	Acquires the date as a "yy/mm/dd" format character string.
20	DECEL	Arithmetic function	Acquires the deceleration rate parameter of a specified robot.
22	DEGRAD	Arithmetic function	Converts a specified value to radians (↔RADDEG).
26	DIST	Arithmetic function	Acquires the distance between 2 specified points.
E			
32	ERL	Arithmetic function	Gives the line No. where an error occurred.
32	ERR	Arithmetic function	Gives the error code number of an error which has occurred.
I			
46	INT	Arithmetic function	Acquires an integer for a specified value by truncating all decimal fractions.
J			
47	JTOXY	Point function	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)
L			
48	LEFT$	Character string function	Extracts a character string comprising a specified number of digits from the left end of a specified character string.
50	LEN	Arithmetic function	Acquires the length (number of bytes) of a specified character string.
53	LOCx	Point function	Specifies/acquires point data for a specified axis or shift data for a specified element.
54	LSHIFT	Arithmetic function	Shifts a value to the left by the specified number of bits. (↔RSHIFT)

No.	Function	Type	Function
M			
55	MCHREF	Arithmetic function	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.
56	MID$	Character string function	Extracts a character string of a desired length from a specified character string.
O			
66	ORD	Arithmetic function	Acquires the character code of the first character in a specified character string.
67	ORGORD	Arithmetic function	Acquires the axis sequence parameter for performing return-to-origin and an absolute search operation of a specified robot.
70	OUTPOS	Arithmetic function	Acquires the OUT enable position parameter of a specified robot.
P			
74	PPNT	Point function	Creates point data specified by a pallet definition number and pallet position number.
76	PSHFRC	Arithmetic function	Specifies/acquires a pushing thrust parameter.
77	PSHJGSP	Arithmetic function	Specifies/acquires a pushing detection speed threshold parameter.
78	PSHMTD	Arithmetic function	Specifies/acquires a pushing type parameter.
79	PSHRSLT	Arithmetic function	Acquires the status when PUSH statement ends.
80	PSHSPD	Arithmetic function	Specifies/acquires the pushing movement speed parameter.
81	PSHTIME	Arithmetic function	Acquires the status at the end of the PUSH statement.
R			
83	RADDEG	Arithmetic function	Converts a specified value to degrees. (↔DEGRAD)
89	RIGHT$	Character string function	Extracts a character string comprising a specified number of digits from the right end of a specified character string.
91	RSHIFT	Arithmetic function	Shifts a value to the right by the specified number of bits. (↔LSHIFT)
S			
98	SIN	Arithmetic function	Acquires the sine value for a specified value.
102	SQR	Arithmetic function	Acquires the square root of a specified value.
104	STR$	Character string function	Converts a specified value to a character string (↔VAL)
T			
108	TAN	Arithmetic function	Acquires the tangent value for a specified value.
109	TCOUNTER	Arithmetic function	Outputs count-up values at 10ms intervals starting from the point when the TCOUNTER variable is reset.
110	TIME$	Character string function	Acquires the current time as an "hh:mm:ss" format character string.
111	TIMER	Arithmetic function	Acquires the current time in seconds, counting from 12:00 midnight.
113	TOLE	Arithmetic function	Acquires the tolerance parameter of a specified robot.
114	TORQUE	Arithmetic function	Acquires the maximum torque command value which can be set for a specified axis of a specified robot.
V			
115	VAL	Arithmetic function	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)
W			
118	WEIGHT	Arithmetic function	Acquires the tip weight parameter of a specified robot.
120	WHERE	Point function	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).

No.	Function	Type	Function
122	WHRXY	Point function	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).
X			
123	XYTOJ	Point function	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).

5 Functions: operation-specific

Point related functions

No.	Function name	Function
47	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)
53	LOCx	Acquires point data for a specified axis or shift data for a specified element.
74	PPNT	Creates point data specified by a pallet definition number and pallet position number.
120	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).
122	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).
123	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).

Parameter related functions

No.	Function name	Function
2	ABSRPOS	Acquires the machine reference of the specified axis of a specified robot. (Valid only for axes where the return-to-origin method is set as "mark method".)
3	ACCEL	Acquires the acceleration coefficient parameter of a specified robot.
4	ARCHP1	Acquires the arch position 1 parameter of a specified robot.
4	ARCHP2	Acquires the arch position 2 parameter of a specified robot.
5	ARMCND	Acquires the current arm status of a specified robot.
6	ARMSEL	Acquires the current “hand system” setting of a specified robot.
7	ARMTYP	Acquires the “hand system” setting of a specified robot.
10	AXWGHT	Acquires the axis tip weight parameter of a specified robot.
16	CURTQST	Acquires the current torque against the rated torque of a specified axis.
17	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.
20	DECEL	Acquires the deceleration rate parameter of a specified robot.
50	LEN	Acquires the length (number of bytes) of a specified character string.
55	MCHREF	Acquires the return-to-origin or absolute-search machine reference for a specified robot axis.
66	ORD	Acquires the character code of the first character in a specified character string.
67	ORGORD	Acquires the axis sequence parameter for performing return-to-origin and an absolute search operation of a specified robot.
70	OUTPOS	Acquires the OUT enable position parameter of a specified robot.
79	PSHRSLT	Acquires the status at the end of the PUSH statement.
113	TOLE	Acquires the tolerance parameter of a specified robot.
114	TORQUE	Acquires the maximum torque command value which can be set for a specified axis of a specified robot.
118	WEIGHT	Acquires the tip weight parameter of a specified robot.

Numeric calculation related functions

No.	Function name	Function
1	ABS	Acquires the absolute value of a specified value.
9	ATN	Acquires the arctangent of the specified value.
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.
15	COS	Acquires the cosine value of a specified value.
22	DEGRAD	Converts a specified value to radians (↔RADDEG).
26	DIST	Acquires the distance between 2 specified points.
46	INT	Acquires an integer for a specified value by truncating all decimal fractions.
54	LSHIFT	Shifts a value to the left by the specified number of bits. (↔RSHIFT)
83	RADDEG	Converts a specified value to degrees. (↔DEGRAD)
91	RSHIFT	Shifts a value to the right by the specified number of bits. (↔LSHIFT)
98	SIN	Acquires the sine value for a specified value.
102	SQR	Acquires the square root of a specified value.
108	TAN	Acquires the tangent value for a specified value.
115	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)

Character string calculation related functions

No.	Function name	Function
14	CHR $	Acquires a character with the specified character code.
19	DATE $	Acquires the date as a "yy/mm/dd" format character string.
48	LEFT $	Extracts a character string comprising a specified number of digits from the left end of a specified character string.
56	MID $	Extracts a character string of a desired length from a specified character string.
89	RIGHT $	Extracts a character string comprising a specified number of digits from the right end of a specified character string.
104	STR $	Converts a specified value to a character string (↔VAL)

Parameter related functions

No.	Function name	Function
32	ERL	Gives the line No. where an error occurred.
32	ERR	Gives the error code number of an error which has occurred.
109	TCOUNTER	Outputs count-up values at 1ms intervals starting from the point when the TCOUNTER variable is reset.
110	TIME $	Acquires the current time as an "hh:mm:ss" format character string.
111	TIMER	Acquires the current time in seconds, counting from 12:00 midnight.

Index

Index

Revision record

Manual version	Issue date	Description
Ver. 1.00	Apr. 2014	First edition
Ver. 1.10	Aug. 2014	Clerical errors were corrected, etc.

Programming Manual

RCX340

Aug. 2014
Ver. 1.10
This manual is based on Ver. 1.10 of Japanese manual.

YAMAHA MOTOR CO., LTD. IM Operations

IM Operations

882 Soude, Nakaku, Hamamatsu, Shizuoka, 435-0054, Japan
Tel. 81-53-460-6103 Fax. 81-53-460-6811

Robot manuals can be downloaded from our company website.
Please use the following for more detailed information.
http://global.yamaha-motor.com/business/robot/

YAMAHA

YAMAHA MOTOR CO., LTD.