Cat. No. I232E-EN-02

SCARA Robots
YRCX Series

YRCX Robot Controller

PROGRAMMING MANUAL

OMRON

Introduction

Our sincere thanks for your purchase of this OMRON YRCX robot controller.

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

Applicable controllers: YRCX

Safety precautions

Be sure to read before using

Before using the OMRON 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 OMRON 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.
Keep this manual carefully so that the operator can refer to it when needed. Also make sure that this manual reaches the end user.

CONTENTS

CONTENTS

Chapter 9 PATH Statements

Chapter 10 Data file description

Chapter 1

Writing Programs

1 The OMRON Robot Language

The OMRON 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 OMRON robot language, including actual examples on how its commands are used.

2 Characters

The characters and symbols used in the OMRON 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 "125 SUB to END SUB" in Chapter 8.

NOTE
- For details regarding user defined functions, refer to "22 DEF FN" in Chapter 8.

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. A line-feed on the program's final line creates a blank line, so be careful not to do so.

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.

1

2

3

4

5

6

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 YRCX 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, refer to "4.6 Sequence program specifications" in Chapter 7.
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, refer to "4.6 Sequence program specifications" in Chapter 7.

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, refer to Chapter 13 "1. Reserved word list", regarding system variables, refer to Chapter 3 "9 System Variables".

6 LABEL Statement

Defines a *label* on a program line.

Format

```
*label:
```

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

1. A *label* must begin with an alphabetic or numeric character.
2. Alphanumeric and underscore (_) can be used as the remaining *label* characters. Special symbols 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 (') 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:

- The shaded section can be omitted.
- *The italic items* should be written in the specific format.
- 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 written 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 -2,147,483,648 to 2,147,483,647
		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 $+1.0\times10^{38}$
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 -2,147,483,648 to 2,147,483,647 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 $+1.0\times10^{38}$ 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 &H80000000 to &H7FFFFFFF.

1
2
3
4
5
6

3 Character constants

Character type constants are character string data enclosed in double 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

```
"OMRON ROBOT"
"EXAMPLE OF""A""" .................. EXAMPLE OF "A"
PRINT "COMPLETED"
"OMRON ROBOT"
```

Chapter 3
Variables

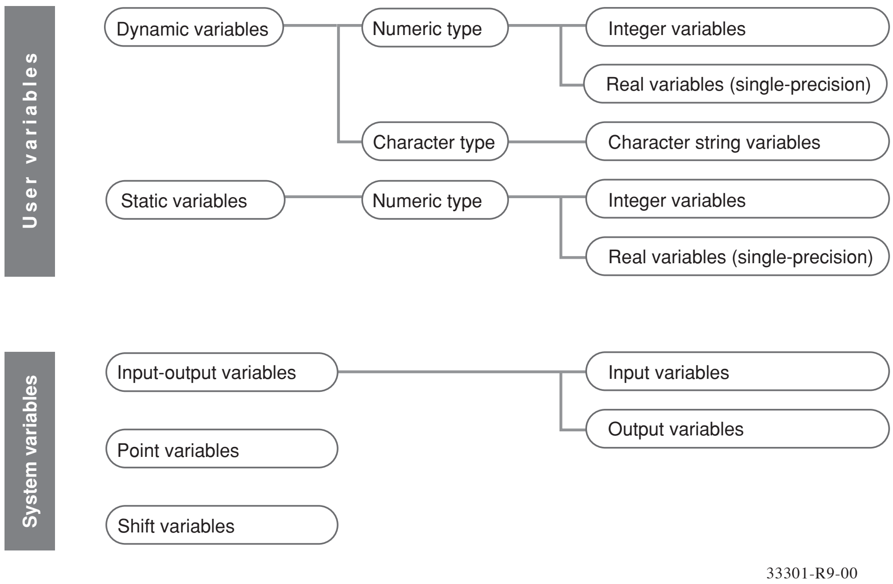
1 Outline

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



REFERENCE For details regarding array variables, refer to "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 -2,147,483,648 to 2,147,483,647 (Signed hexadecimal constants: &H80000000 to &H7FFFFFFF)
		Real variables (single-precision) -1.0×10^{38} to $+1.0\times10^{38}$
	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 -2,147,483,648 to 2,147,483,647
		Real variables (single-precision) -1.0×10^{38} to $+1.0\times10^{38}$
Array variables	Numeric type	Integer array variables -2,147,483,648 to 2,147,483,647
		Real array variables (single-precision) -1.0×10^{38} to $+1.0\times10^{38}$
	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 variables	External signal / status inputs	DI, SI, SIW, SID
	Output variables	External signal / status outputs	DO, SO, SOW, SOD
Point variables		Handles point data	Pnnnn
Shift variables		Specifies the shift coordinate No. as a numeric constant or expression	Sn

REFERENCE For details, refer to "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) and user-defined function. 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         ........................Use is not permitted
                                    (Reserved word)
FNAME       ........................Use is not permitted
                                    (FN: user-defined function)
S91         ........................Use is not permitted
                                    (Sn: pre-defined variable)
```

REFERENCE For details regarding reserved words, refer to Chapter 13 "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 online command.

REFERENCE For details regarding clearing of static variables, refer to "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 variables	STR1$
%	Integer variables	CONT0%, ACT%(1)
!	Real 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 -2,147,483,648 to 2,147,483,647 (in signed hexadecimal, this range is expressed as &H80000000 to &H7FFFFFFF).

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

Real variables

NOTE

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

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$ = "OMRON"
          R2$(2) = R1$ + "MOTOR" ··········· "OMRON 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, Array variables must be declared by DIM statement 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, refer to "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 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 8 "54 LET (Assignment Statement)"

1 2 3 4 5 6

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.
- 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 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

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)]
```

1
2
3
4
5
6

9.2 Shift 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

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 in Format 1, bits 0 to 7 are all selected.

```
Examples: A%=DI1()
           →Input status of ports DI(17) to DI(10)
            is assigned to variable 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 (high to low).
- A "0" is input if an input port does not actually exist.

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 in Format 1, 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 (high to low).
- If an output port does not actually exist, the data is not output externally.

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 in Format 1, bits 0 to 7 are all selected.

Functions Internal output variables which are used only in the controller, can set the status and refer. 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 Port 31	Axis 8 Axis 16	Axis 7 Axis 15	Axis 6 Axis 14	Axis 5 Axis 13	Axis 4 Axis 12	Axis 3 Axis 11	Axis 2 Axis 10	Axis 1 Axis 9
	Origin sensor status 0: OFF / 1: ON (Axis 1 is not connected)							
Port 34 Port 35	Axis 8 Axis 16	Axis 7 Axis 15	Axis 6 Axis 14	Axis 5 Axis 13	Axis 4 Axis 12	Axis 3 Axis 11	Axis 2 Axis 10	Axis 1 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" (HOLD).
- 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 (high to low).

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 in Format 1, bits 0 to 7 are all selected.

Functions The contents of this variable can be set the status 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.)
LO1()=&B0010
  →LO(11) is set to 1(ON),then movement of axis 10 is prohibited.
LO1(2,0)=3
  →LO(12) and LO(10) are set to 1(ON),
   then movements of axes 11 and 9 are prohibited.
```

MEMO

- When specifying multiple bits, specify them from left to right in descending order (high to low).
- 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 "12.401 Arm locked" 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 in Format 1, 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 "4.2 Input/output variables" in Chapter 7.

```
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 (high to low).

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 in Format 1, 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 (high to law).
- A "0" is input if a serial port does not actually exist.

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 in Format 1, 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 turn 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) turn 1(ON), then A%=5.)

          SO3()=B%
            →Changes the output status of SO(37) to
             SO(30) to one indicated by B%.
            (If B% is 123, 123 is expressed B01111011 as a binary number,
            that means SO(37) and SO(32) turn 0(OFF), the other bits turn 1(ON).)

          SO4(5,4,0)=&B101
            →DO(45) and DO(40) turn 1(ON), DO(44) turns 0(OFF).
```

MEMO

• When specifying multiple bits, specify them from left to right in descending order (high to law).
• If a serial port does not actually exist, the data is not output externally.

9.10 Serial word input

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

Format

```
SIW(m)
```

Values m : port number .... 2 to 15
The acquisition range is 0 (&H0000) to 65,535 (&HFFFF).

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

MEMO

- The information is handled as unsigned word data.
- "0" is input if a 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 number 2, 4, 6, 8, 10, 12, 14
The acquisition range is -2,147,483,648(&H80000000) to 2,147,483,647(&H7FFFFFFF).

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

MEMO

- The information is handled as signed double word data.
- "0" is input if a serial port does not actually exist.
- The lower port number data is placed at the lower address.
  For example, if SIW(2) =&H2345, SIW(3) =&H0001, then SID(2) =&H00012345.

1
2
3
4
5
6

9.12 Serial word output

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

Format

```
SOW(m)
```

Values m : port number .... 2 to 15
The output range is 0 (&H0000) to 65,535 (&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 of 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 port does not actually exist, the data 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 number 2, 4, 6, 8, 10, 12, 14
The output range is -2,147,483,648(&H80000000) to 2,147,483,647(&H7FFFFFFF).

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

MEMO

- The information is handled as signed double word data.
- If a serial port does not actually exist, the data is not output externally.
- The lower port number data is placed at the lower address.
  For example, if SOW(2) =&H2345, SOW(3) =&H0001, then SOD(2) =&H00012345.

1
2
3
4
5
6

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, 2-digit consisting of the port number and the bit number must be specified in parentheses and must be separated by commas. Up to 8 bits can be written.

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

11.1 Valid range of dynamic (array) variables

Dynamic (array) 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.

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 8 "111 SHARED").

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)

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 array is cleared in the same manner.

- 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
- When the HALTALL statement was executed in the program (HALT statement does not clear dynamic variables).

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

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, I, XOR (Logical OR, exclusive OR)
12	EQV (Logical equivalence)
13	IMP (Logical implication)

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

Results:   OMRON ROBOT LANGUAGE
           OMRON 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.

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 8 digits or less with a plus or minus sign, and can be specified from –99999999 to 99999999. 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

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 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

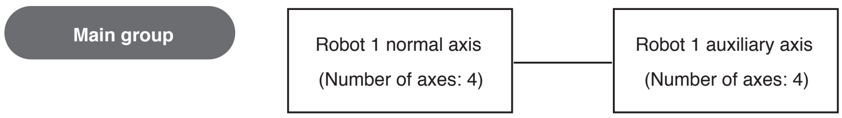
YRCX can be used to control multiple robots (up to 4).
The multi-task function also enables multiple robots to move asynchronously.
To use this function, settings for multiple robots or settings for auxiliary axes must be made in the system prior to shipment.
The following settings are possible to the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

- Robot 1 (4 axes)
- Robot 1 (4 axes) + robot 2 (4 axes) (when using the YC-LINK/E option)
- Robot 1 (4 axes) + robot 2 (4 axes) + robot 3 (4 axes) + robot 4 (4 axes)
  (when a master controller is YRCX
  and the YC-LINK/E option is used.)

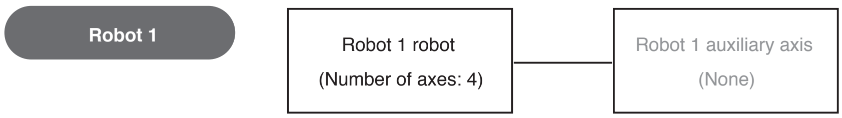
Each robot consists of normal axes and auxiliary axes.
When using one robot without auxiliary axes, the setting is made only to normal axes.

Axes configuration

1. For robot 1



2. For 1 robot with no auxiliary axes used



33501-R9-00

1
2
3
4
5
6

2 Command list with a robot setting

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 MOVET PMOVE WAIT ARM	DRIVEI MOVEI PATH SERVO
Coordinate control	CHANGE LEFTY RIGHTY	HAND PATH SHIFT
Status change	ACCEL ARCHP2 ARMTYP AXWGHT MSPEED OUTPOS TOLE	ARCHP1 ARMSEL ASPEED DECEL ORGORD SPEED WEIGHT WEIGHTG
Point operation	JTOXY XYTOJ	WHERE WHRXY
Parameter reference	ACCEL ARCHP2 AXWGHT ORGORD TOLE	ARCHP1 ARMTYP DECEL OUTPOS WEIGHT WEIGHTG
Status reference	ABSRPOS ARMSEL CURTQST MCHREF WHRXY	ARMCND ARMTYP CURTRQ WHERE
Torque control	TORQUE TRQTIME	TRQSTS CURTRQ

- An axis specified as an auxiliary axis cannot be moved with the MOVE, MOVEI, MOVET 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-task 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 1 and 64.
- Smaller values have higher priority, and larger values have lower priority
  (High priority: 1 ⇔ 64: low priority).

2 Task definition method

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 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 numbers are then assigned, and the program starts.

SAMPLE

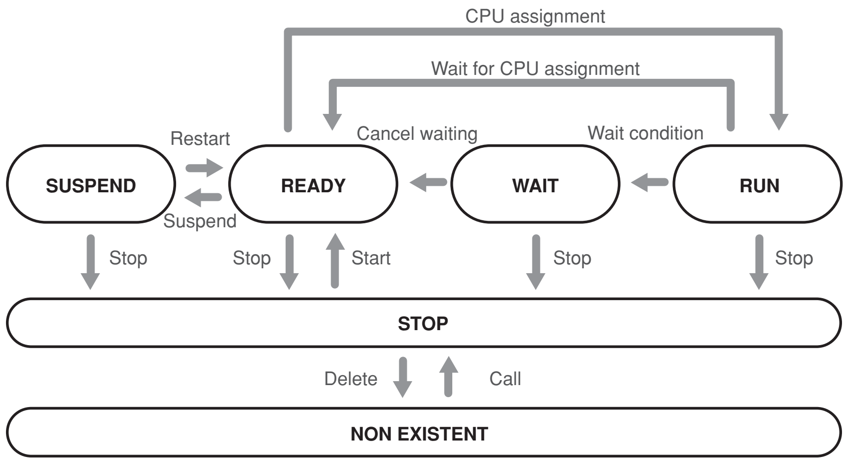
```
'MAIN TASK(TASK1)
START <SUB_PGM>,T2 ·················<SUB_PGM> is started as Task 2
*ST1:
MOVE P,P1,P0
    IF DO(20)= 1 THEN
        HALTALL
    ENDIF
GOTO *ST
HALTALL

Program name:SUB_PGM
'SUB TASK(TASK2)
*IOTASK:    ························Task 2 begins here
    IF DI(21)=1 THEN
        DO(30)=1
    ELSE
        DO(30)=0
    ENDIF
GOTO *IOTASK ······················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 START command is executed, a specified program is registered in the task and placed in RUN status. If the task number (1 to 16) is not specified by the START command, the task with the smallest number among the tasks yet to be started is automatically specified. For details regarding the START command, refer to "123 START" in Chapter 8.

MEMO

- When the LOAD command is executed, a specified program is registered in the task and placed in a STOP status. For details of the LOAD command, refer to "1. Register task" of "2.1 Program operations" in Chapter 12.
- If another program is already registered in the task number specified by the START command or the LOAD command, the "6.215: Task running" error will occur.
- When programs are registered in all task numbers and the START command or the LOAD command is executed without specifying the task number, the "6.263: Too many Tasks" error will occur.
- When the HALTALL command is executed, all tasks termitate and the task enters the NON EXISTENT (no task registration) status. When the main program is specified, the HALTALL command registers the main program in the task 1 and stops at the beginning line. When the main program is not specified, the HALTALL command registers the program that has been executed last (current program) in the task 1 and stops at the beginning line.
  For details regarding the main program, refer to "Setting the main program" of YRCX operator's manual.

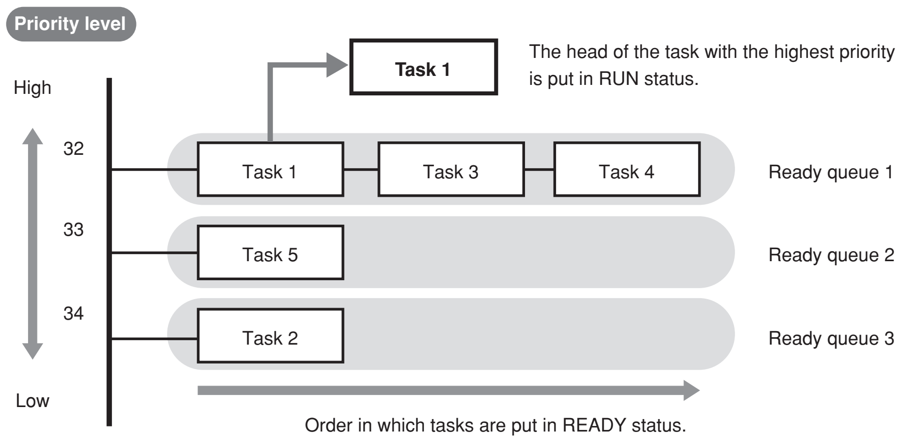
3.2 Task scheduling

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



33602-R7-00

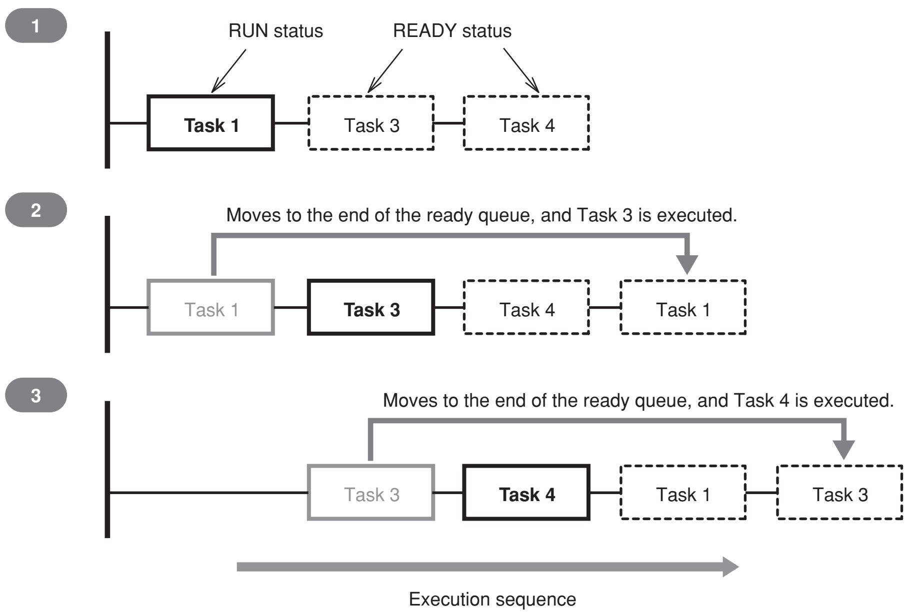
A RUN status task will be moved to the end of the ready queue if placed in a READY status by any of the following causes:

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



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 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 WAIT status is executed, the following transition happens.

- Task for which a command causing 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 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 "waiting for an event" status. In this case, WAIT status is established at the task which executed the MOVE command, and that task is moved to the end of the ready queue. 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 DRIVEI SERVO	MOVEI PMOVE WAIT ARM	MOVET PATH	DRIVE MOTOR
	Parameter command	ACCEL DECEL WEIGHT	ARCHP1 OUTPOS WEIGHTG	ARCHP2 TOLE	AXWGHT ORGORD
	Robot status change command	CHANGE MSPEED	SHIFT SPEED	LEFTY	ASPEED
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

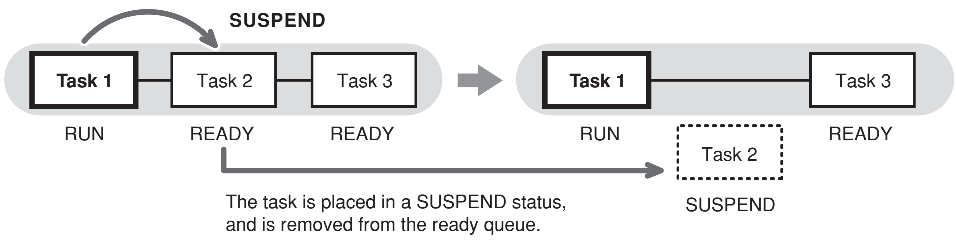
- The tasks are not put in WAIT status if the event has been established before the above commands are executed.

3.4 Suspending tasks (SUSPEND)

The SUSPEND command temporarily stops tasks other than task 1 and places them in SUSPEND status.
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)



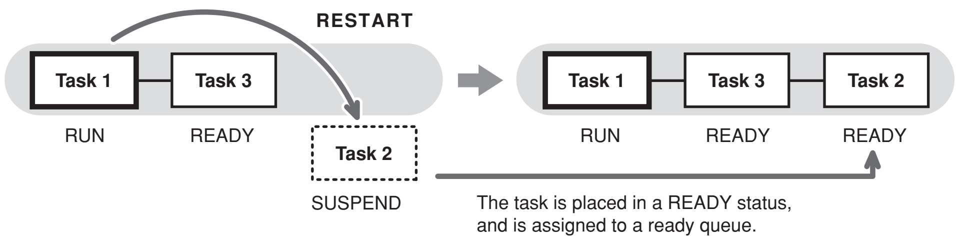
33604-R7-00

3.5 Restarting tasks (RESTART)

Tasks in the SUSPEND status can be restarted with the RESTART command.
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)



33605-R7-00

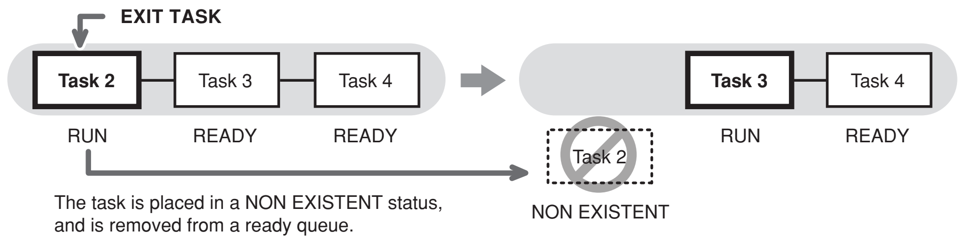
3.6 Deleting tasks

Task self-delete (EXIT TASK)

Tasks can delete themselves and set to the NON EXISTENT (no task registration) status by using the EXIT TASK command.
When the EXIT TASK command is executed, the status transition takes place as follows.

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

Task self-delete (EXIT TASK)



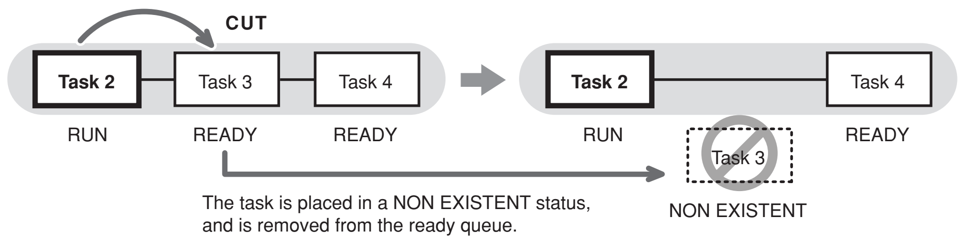
33606-R7-00

Other-task delete (CUT)

Tasks can also delete the other tasks and put in the NON EXISTENT (no task registration) status by using the CUT command.
When the CUT command is executed, the status transition takes place as follows.

- Task that executed the CUT command → RUN
- Specified task → NON EXISTENT

Other-task delete (CUT)



33607-R7-00

MEMO

- If a SUSPEND command is executed for a WAIT-status task, the commands being executed by that task are ended.

3.7 Stopping tasks

All tasks stop if any of the following cases occurs.

1. **HALTALL command is executed. (stop & reset)**
   All programs are reset and task is put in the NON EXISTENT status. When the main program is specified, the HALTALL command registers the main program in the task 1 and all tasks are put in the STOP status at the beginning line. When the main program is not specified, the HALTALL command registers the program that has been executed last (current program) in the task 1 and all tasks are put in the STOP status at the beginning line.

2. **HOLDALL 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 the SUSPEND status after being placed in the READY status).

4. **When the emergency stop button on the programming box is pressed or the emergency stop signal is cut off.**
   All tasks are put in the 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 the 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 <SUB_TSK2>,T2
START <SUB_TSK3>,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
HALTALL

Program name:SUB_TSK2
'TASK2      ..................... Task 2 begins here.
*ST2:
    IF DI(20) = 1
        MO(20) = 1
        DELAY 100
    ELSE
        MO(20) = 0
    ENDIF
GOTO *ST2
EXIT TASK  Ends here.

Program name:SUB_TSK3
'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

All global variables, static variables, input/output variables, point data, shift coordinate definition data, hand definition data, and pallet definition data 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. Take great care when sharing the same variables and data.

6 Cautionary Items

A freeze 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 <SUB_PGM>,T2,31 ········· <SUB_PGM> is started as task 2
                                        at the priority level of 31.
    ENDIF
GOTO *MAIN_TASK
HALTALL

Program name:SUB_PGM
'==============
'       TASK2
'==============
*TASK2:
    DRIVE(1,P1)
    WAIT ARM(1)
    DRIVE(1,P2)
    WAIT ARM(1)
    FLAG1 = 0
EXIT TASK
```

Chapter 7

Sequence function

1 Sequence function

NOTE

- While the "DI10: sequence control input" is ON, a sequence program runs according to its own cycle, regardless of robot program starts and stops.
- The "DO12: Sequence program running" dedicated signal output occurs while a sequence program is being executed.

Besides normal robot programs, the YRCX controller can execute high-speed processing programs (sequence programs) in response to the robot input/output (DI, DO, MO, LO, TO, SI, SO) signals.

- This function allows to monitor the input/output signals of sensors, push button switches, solenoid valves, etc. and move them. The sequence program starts running simultaneously the controller is turned on.
- The sequence program can be written in the same robot language used for robot programs. (The ladder logic are not necessary).
- Naming the program "SEQUENCE" makes the controller recognize as sequence program.
- For details regarding conditions to execute a sequence program, refer to "3 Executing a sequence program" in this Chapter.
- General-purpose outputs are not reset by the program reset while the sequence function is running.

MEMO

In the manner shown below, the reset of general-purpose output can be set while the sequence program compile.

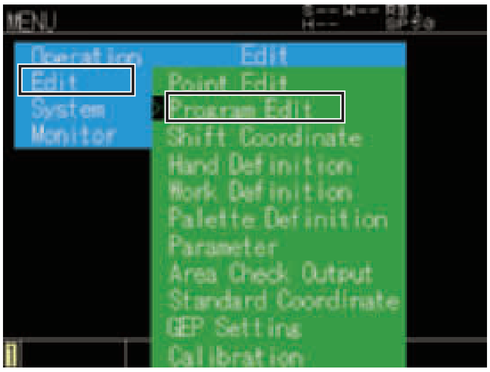
- Set a sequence flag value of the controller parameter at "3".
- Select "Output Reset Enable" on the sequence execution flag dialogue in the support software "SCARA-YRCX Studio".

2 Creating a sequence program

2.1 Programming method

Step 1 *Select (Program Edit) from (Edit) menus on the "MENU" screen of the programming box.*

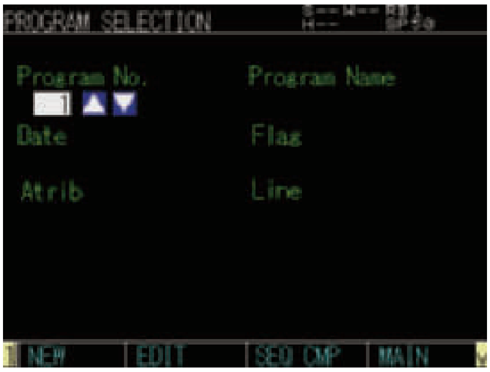
▶ Step 1 Program edit



34701-R9-00

Step 2 *Press the F1 key (NEW) on the "PROGRAM SELECTION" screen.*

▶ Step 2 Program selection



34702-R9-00

Step 3 *Enter "SEQUENCE" on the program name entry screen, and press the (OK) button.*

34703-R9-00

▶ **Step 3** **Creating new program**

Step 4 *Use the cursor keys (▲/▼) to select "SEQUENCE" on the "PROGRAM SELECTION" screen, and then press the F2 key (EDIT).*

34704-R9-00

▶ **Step 4** **Program selection**

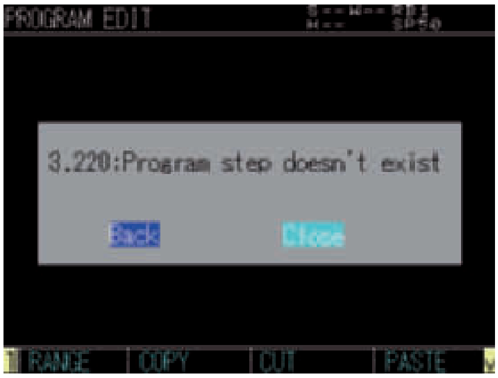
NOTE

- When creating a new program, the alarm occurs since no program is written. This alarm does not occur when the robot language exists already in a program.

Step 5 *"3.220: Program step doesn't exist" message appears when creating a new program, and press (Close).*

34705-R9-00

▶ Step 5 "Program step doesn't exist" message



Step 6 *Input a program on "PROGRAM SELECTION" screen.*

Although usable commands are resricted, editing method is same as the standard robot program.

Commands which can be input are explained at "4 Programming a sequence program" in this Chapter.

34706-R9-00

▶ Step 6 Program edit

2.2 Compiling

Compile and create an executable sequence program.

Step 1 *Press the F3 key (SEQ CMP) on the "PROGRAM SELECTION" screen.*

34707-R9-00

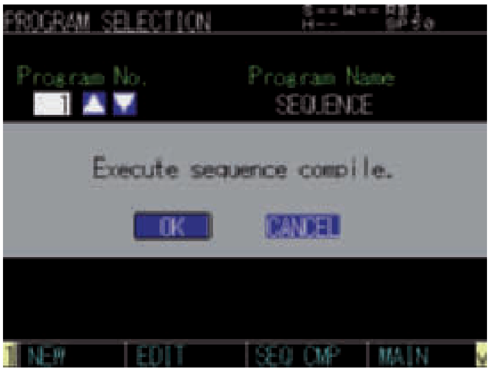
▶ Step 1 Program selection

Step 2 *The confirmation message will appear whether you execute sequence compile.*

Press (OK) to compile the program.

34708-R9-00

▶ Step 2 Sequence compiling

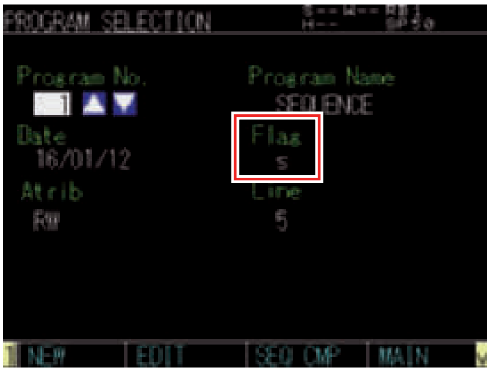


MEMO

- If there is a syntax error in the program, an error message appears. When the compiling ends without any error, the display returns to the "PROGRAM SELECTION" screen and the letter "s" appears in "Flag". This means that the sequence program has been compiled successfully and is ready for use.

34709-R9-00

▶ MEMO Success of compiling



MEMO

- The sequence execution program is erased and the Flag's letter "s" disappears in the following cases. In these cases the sequence function cannot be used.

  1. When the sequence program was erased.
  2. When the sequence program was edited.
  3. When the program data was initialized.
  4. When the "9.729: Sequence object destroyed." alarm occured.

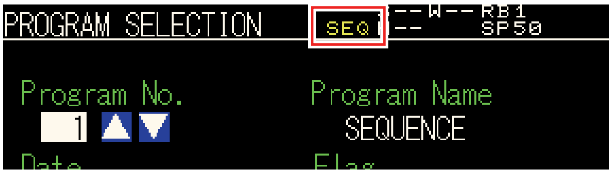
3 Executing a sequence program

All the following conditions must be satisfied to execute a sequence program.

1. The sequence program has been compiled.
2. The sequence program execution flag is enabled.
   (For details regarding the sequence program execution flag, refer to the YRCX operator's manual.)
3. The external sequence control input (DI10) contact is ON.

Sequence program execution in progress

Indicated during execution



34710-R9-00

3.1 Sequence program STEP execution

The sequence program may be executed line by line while checking one command line at a time. This step execution can be executed in the same way as normal programs.
For details, refer to the YRCX operator's manual.
When the step is executed, satisfying the conditions described in the previous section is not required.

4 Programming a sequence program

When programming a sequence program, you may use only assignment statements comprised of input/output variables and logical operators.

4.1 Assignment statements

Format

```
output variable = expression
```

Values *expression* .... Any one of the following can be used.
- Parallel input/output variables
- Internal output variables
- Arm lock output variables
- Timer output variables
- Serial input/output variable
- The logic operation expression shown above

4.2 Input/output variables

Each variable must be specified in a 1-bit format

```
·Correct examples              DO(35)
                               MO(24)
                               DI(16)
·Incorrect examples    DO(37, 24)
                               DI3(4)
                               MO3()
```

4.2.1 Input variables

● **Parallel input variables**

Format

```
DI(mb)   m: Port number ·············· 0 to 7, 10 to 17, 20 to 27
         b: bit definition ············ 0 to 7
```

These variables show the status of the parallel input signal.

● **Serial input variables**

Format

```
SI(mb)   m: Port number ·············· 0 to 7, 10 to 17, 20 to 27
         b: bit definition ············ 0 to 7
```

Indicates a serial input signal status. Only referencing can occur. No controls are possible.

7
8
9
10
11
12
13

4.2.2 Output variables

● Parallel output variables

Format

```
DO(mb)    m: Port number ·············· 0 to 7, 10 to 17, 20 to 27
          b: bit definition ············ 0 to 7
```

A parallel output is specified, or the output status is referenced. Ports 0 and 1 are for referencing only, and no outputs can occur there.

● Internal output variables

Format

```
MO(mb)    m: Port number ·············· 0 to 7, 10 to 17, 20 to 27, 30 to 37
          b: bit definition ············ 0 to 7
```

These variables are used within the controller. Ports 30 to 37 are for referencing only and ON/OFF can not be controlled.

● Arm lock output variables

Format

```
LO(mb)    m: port number ·············· 0, 1
      b: bit definition················ 0 to 7
```

These variables are used to prohibit the arm (axis) movement. Movement is prohibited when ON. LO(00) to LO(07) corresponds to arm 1 to arm 8, LO(10) to LO(17) corresponds to arm 9 to arm 16, respectively.

● Timer output variables

Format

```
TO(mb)    m: port number ·············· 0, 1
          b: bit definition ············ 0 to 7
```

There are a total of 16 timer output variables: TO(00) to TO(17). The timer of each variable is defined by the timer definition statement TIM00 to 17.

● Serial output variables

Format

```
SO(mb)    m: Port number ·············· 0 to 7, 10 to 17, 20 to 27
          b: bit definition ············ 0 to 7
```

Control or reference serial output signal status. Port 0 is for referencing only, and no controls are possible.

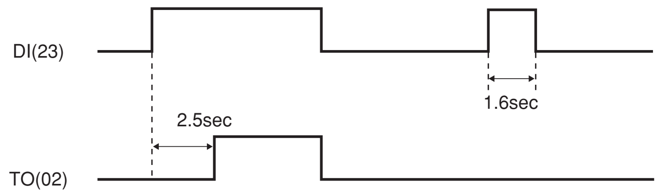
Timer usage example

SAMPLE

```
TIM02 = 2500 ···················· Timer 02 is set to 2.5 seconds.
TO(02) = DI(23) ················· Timer starts when DI(23) switches ON.
```

- When DI(23) is ON, after 2.5 seconds, TO(02) is set ON.
- When DI(23) is OFF, TO(02) is also OFF.
- When DI(23) isn't ON after 2.5 second or more, TO(02) does not change to ON.

Timer usage example: Timing chart



33701-R7-00

4.3 Timer definition statement

Format

```
TIMmb=time     m: Port number·······  0, 1
               b: bit definition·····  0 to 7
```

Values *time* .... 100 to 999,900ms (0.1 to 999.9 second)

Meaning The timer definition statement sets the timer value of the timer output variable. This definition statement may be anywhere in the program.
When the timer definition statement is omitted, the timer setting value of the variable is 0.
TIM00 to 17 correspond to the timer output variables TO(00) to (17).
However, since the units are set every 100ms, values less than 99ms are truncated.

4.4 Logical operators

Operators	Functions	Meaning
NOT, ~	Logical NOT	Reverses the bits.
AND, &	Logical AND	Becomes "1" when both bits are "1".
OR, I	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".

4.5 Priority of logic operations

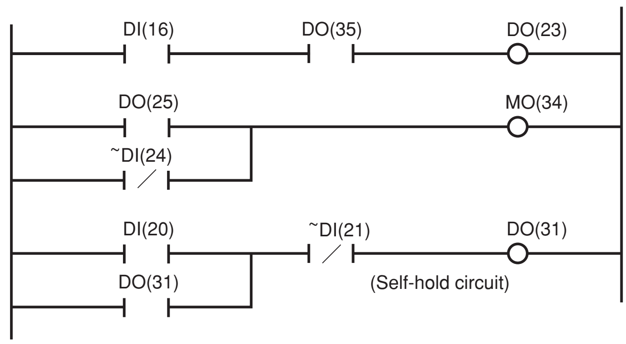
Priority Ranking	Operation Content
1	Expressions in parentheses
2	NOT, ~ (Logical NOT)
3	AND, & (Logical AND)
4	OR, I (Logical OR)
5	XOR (Exclusive OR)
6	EQV (Logical equivalence operator)
7	IMP (Logical implication operator)

● Example with a ladder statement substitution

SAMPLE

```
DO(23)=DI(16)&DO(35)
MO(34)=DO(25) | ~DI(24)
DO(31)=(DI(20) | DO(31))&~DI(21)
```

Ladder diagram



33702-R7-00

MEMO

- "NOT" cannot be used prior to the first parenthesis " (" or on the left of an expression. For example, the following commands cannot be used.
  - •DO(21)=~(DI(30) | DI(32))
  - •~DO(30)=DI(22)&DI(27)
- Numeric values cannot be assigned on the right of an expression.
  - •MO(35)=1
  - •DO(26)=0
- There is no need to define a "HALT" or "HOLD" statement at the end of the program.
- The variables used in sequence programs are shared with robot programs, so be careful not to make improper changes when using the same variables between them.

4.6 Sequence program specifications

Item	Specification
Commands	Logical NOT, AND, OR, XOR, EQV, IMP
I/O	Same as robot language
Program capacity	16,384 bytes (A maximum of 2,048 variables can be specified.)
Scan time	1 to 4ms depending on the number of steps (This changes automatically.)

Chapter 8

Robot Language Lists

How to read the robot language table

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

No.	Name	Description	Online	Type
26	DIM	Declares array variable	–	Command

(1) No.

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

Example of "No." column

No.

26 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 (,) to separate.

MEMO
- Array subscripts can be "0 to a specified value", with their total number being the *constant* + 1.
- A "9.300: 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) Description

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.	Name	Description	Online	Type
A				
1	ABS	Acquires the absolute value of a specified value.	✓	Function
2	ABSRPOS	Acquires the machine reference value for specified robot axes. (Valid only for axes whose return-to-origin method is set as "mark".)	✓	Function
3	ACCEL	Specifies/acquires the acceleration coefficient parameter of a specified robot.	✓	Command / Function
4	ARCHP1	Specifies/acquires the arch position 1 parameter of a specified robot.	✓	Command / Function
4	ARCHP2	Specifies/acquires the arch position 2 parameter of a specified robot.	✓	Command / Function
5	ARMCND	Acquires the current arm status of a specified robot.	✓	Function
6	ARMSEL	Specifies/acquires the current "hand system" setting of a specified robot.	✓	Command / Function
7	ARMTYP	Specifies/acquires the "hand system" setting of a specified robot.	✓	Command / Function
8	ASPEED	Specifies/acquires the AUTO movement speed of a specified robot.	✓	Command / Function
9	ATN	Acquires the arctangent of the specified value.	✓	Function
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.	✓	Function
10	AXWGHT	Specifies/acquires the axis tip weight parameter of a specified robot.	✓	Command / Function
C				
11	CALL	Calls a sub-procedure.	–	Command
12	CHANGE	Switches the hand of a specified robot.	✓	Command
13	CHGPRI	Changes the priority ranking of a specified task.	✓	Command
14	CHR$	Acquires a character with the specified character code.	✓	Function
15	CLOSE	Close the specified General Ethernet Port.	✓	Command
16	COS	Acquires the cosine value of a specified value.	✓	Function
17	CURTQST	Acquires the current torque value ratio of a specified axis to the rated torque.	✓	Function
18	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.	✓	Function
19	CUT	Terminates another task currently being executed or temporarily stopped.	✓	Command
D				
20	DATE$	Acquires the date as a "yy/mm/dd" format character string.	✓	Function
21	DECEL	Specifies/acquires the deceleration rate parameter of a specified robot.	✓	Command / Function
22	DEF FN	Defines the functions that can be used by the user.	–	Command
23	DEGRAD	Converts a specified value to radians (↔RADDEG).	✓	Function
24	DELAY	Waits for the specified period (units: ms).	–	Command
25	DI	Acquires the specified DI status.	✓	Function
26	DIM	Declares the array variable name and the number of elements.	–	Command
27	DIST	Acquires the distance between 2 specified points.	✓	Function
28	DO	Outputs a specified value to the DO port or acquires the DO status.	✓	Command / Function
29	DRIVE	Moves a specified axis of a specified robot to an absolute position.	✓	Command

No.	Name	Description	Online	Type
30	DRIVEI	Moves a specified axis of a specified robot to a relative position.	✓	Command
E				
31	END SELECT	Terminates the SELECT CASE statement.	–	Command
32	END SUB	Terminates the sub-procedure definition.	–	Command
33	ERR / ERL	Acquires the error code number of an error which has occurred / the line number where an error occurred.	✓	Function
34	ETHSTS	Acquires the Ethernet port status.	✓	Function
35	EXIT FOR	Terminates the FOR to NEXT statement loop.	–	Command
36	EXIT SUB	Terminates the sub-procedure defined by the SUB to END statement.	–	Command
37	EXIT TASK	Terminates its own task which is in progress.	–	Command
F				
38	FOR to NEXT	Executes the FOR to NEXT statement repeatedly until a specified value is exceeded.	–	Command
G				
39	GEPSTS	Acquires the General Ethernet Port status.	✓	Function
40	GOSUB to RETURN	Jumps to a subroutine with the label specified by GOSUB statement, and executes that subroutine.	–	Command
41	GOTO	Unconditionally jumps to the line specified by a label.	–	Command
H				
42	HALT	Stops the program and performs a reset.	–	Command
43	HALTALL	Stops and resets all programs.	–	Command
44	HAND	Defines the hand of a specified robot.	✓	Command
45	HOLD	Temporarily stops the program.	–	Command
46	HOLDALL	Temporarily stops all programs.	–	Command
I				
47	IF	Allows control flow to branch according to conditions.	–	Command
48	INPUT	Assigns a value to a variable specified from the programming box.	✓	Command
49	INT	Acquires an integer for a specified value by truncating all decimal fractions.	✓	Function
J				
50	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)	✓	Function
L				
51	LEFT$	Extracts a character string comprising a specified number of digits from the left end of a specified character string.	✓	Function
52	LEFTY	Sets the hand system of a specified robot to the left-handed system.	✓	Command
53	LEN	Acquires the length (byte count) of a specified character string.	✓	Function
54	LET	Executes a specified assignment statement.	✓	Command
55	LO	Outputs a specified value to the LO port to enable/disable axis movement or acquires the LO status.	✓	Command / Function
56	LOCx	Specifies/acquires point data for a specified axis or shift data for a specified element.	✓	Command / Function
57	LSHIFT	Shifts a value to the left by the specified bit count. (↔RSHIFT)	✓	Function

No.	Name	Description	Online	Type
M				
58	MCHREF	Acquires the return-to-origin or absolute-search machine reference value for specified robot axes. (Valid only for axes whose return-to-origin method is set as "sensor" or "stroke-end".)	✓	Function
59	MID$	Extracts a character string of a desired length from a specified character string.	✓	Function
60	MO	Outputs a specified value to the MO port or acquires the MO status.	✓	Command / Function
61	MOTOR	Controls the motor power status.	✓	Command
62	MOVE	Performs absolute movement of all axes of a specified robot.	✓	Command
63	MOVEI	Performs relative movement of all axes of a specified robot.	✓	Command
64	MOVET	Performs relative movement of all axes of a specified robot when the tool coordinate is selected.	✓	Command
65	MTRDUTY	Acquires the motor load factor of the specified axis.	✓	Function
O				
66	OFFLINE	Sets a specified communication port to the "offline" mode.	✓	Command
67	ON ERROR GOTO	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
68	ON to GOSUB	Jumps to a subroutine with labels specified by a GOSUB statement in accordance with the conditions, and executes that subroutine.	–	Command
69	ON to GOTO	Jumps to label-specified lines in accordance with the conditions.	–	Command
70	ONLINE	Sets the specified communication port to the "online" mode.	✓	Command
71	OPEN	Opens the specified General Ethernet Port.	✓	Command
72	ORD	Acquires the character code of the first character in a specified character string.	✓	Function
73	ORGORD	Specifies/acquires the axis sequence parameter for performing return-to-origin and an absolute search operation in a specified robot.	✓	Command / Function
74	ORIGIN	Performs return-to-origin.	✓	Command
75	OUT	Turns ON the bits of the specified output ports and terminates the command statement.	–	Command
76	OUTPOS	Specifies/acquires the "OUT position" parameter of a specified robot.	✓	Command / Function
P				
77	PATH	Specifies the PATH motion path.	–	Command
78	PATH END	Ends the path setting for PATH motion.	–	Command
79	PATH SET	Starts the path setting for PATH motion.	–	Command
80	PATH START	Starts the PATH motion.	–	Command
81	PDEF	Defines the pallet used to execute pallet movement commands.	✓	Command
82	PGMTSK	Acquires the task number in which a specified program is registered.	✓	Function
83	PGN	Acquires the program number from a specified program name.	✓	Function
84	PMOVE	Executes the pallet movement command of a specified robot.	✓	Command
85	Pn	Defines points within a program.	✓	Command
86	PPNT	Creates point data specified by a pallet definition number and pallet position number.	✓	Function
87	PRINT	Displays a character string at the programming box screen.	–	Command

No.	Name	Description	Online	Type
88	PSHFRC	Specifies/acquires the "Push force" parameter.	✓	Command / Function
89	PSHJGSP	Specifies/acquires the push judge speed threshold parameter.	✓	Command / Function
90	PSHMTD	Specifies/acquires the push method parameter.	✓	Command / Function
91	PSHRSLT	Acquires the status at the end of the PUSH statement.	✓	Function
92	PSHSPD	Specifies/acquires the push speed parameter.	✓	Command / Function
93	PSHTIME	Specifies/acquires the push time parameter.	✓	Command / Function
94	PUSH	Executes a pushing operation in the axis unit.	✓	Command
R				
95	RADDEG	Converts a specified value to degrees. (⇔DEGRAD)	✓	Function
96	REM	Expresses a comment statement.	–	Command
97	RESET	Turns the bit of a specified output port OFF.	✓	Command
98	RESTART	Restarts another task during a temporary stop.	✓	Command
99	RESUME	Resumes program execution after error recovery processing.	–	Command
100	RETURN	Returns the processing branching with GOSUB to the next line of GOSUB.	–	Command
101	RIGHT$	Extracts a character string comprising a specified number of digits from the right end of a specified character string.	✓	Function
102	RIGHTY	Sets the hand system of a specified robot to the right-handed system.	✓	Command
103	RSHIFT	Shifts a value to the right by the specified bit count. (⇔LSHIFT)	✓	Function
S				
104	SELECT CASE to END SELECT	Allows control flow to branch according to conditions.	–	Command
105	SEND	Sends a file.	✓	Command
106	SERVO	Controls the servo ON/OFF of a specified axis or all axes of a specified robot.	✓	Command
107	SET	Turns the bit at the specified output port ON.	–	Command
108	SETGEP	Sets the General Ethernet Port.	✓	Command
109	SGI	Assigns the value to a specified integer type static variable / acquires the value of a specified integer type static variable.	✓	Command / Function
110	SGR	Assigns the value to a specified real type static variable / acquires the value of a specified real type static variable.	✓	Command / Function
111	SHARED	Enables reference with a sub-procedure without transferring a variable.	–	Command
112	SHIFT	Sets the shift coordinate for a specified robot by using the shift data specified by a shift variable.	✓	Command
113	SI	Acquires a specified SI status.	✓	Function
114	SID	Acquires a specified serial input's double-word information status.	✓	Function
115	SIN	Acquires the sine value for a specified value.	✓	Function
116	SIW	Acquires a specified serial input's word information status.	✓	Function
117	Sn	Defines the shift coordinates within the program.	✓	Command
118	SO	Outputs a specified value to the SO port or acquires the SO status.	✓	Command / Function
119	SOD	Outputs a specified serial output's double-word information or acquires the output status.	✓	Command / Function
120	SOW	Outputs a specified serial output's word information or acquires the output status.	✓	Command / Function

No.	Name	Description	Online	Type
121	SPEED	Changes the program movement speed of a specified robot.	✓	Command
122	SQR	Acquires the square root of a specified value.	✓	Function
123	START	Specifies the task number and priority ranking of a specified program, and starts that program.	✓	Command
124	STR$	Converts a specified value to a character string (↔VAL).	✓	Function
125	SUB to END SUB	Defines a sub-procedure.	–	Command
126	SUSPEND	Temporarily stops another task which is being executed.	–	Command
127	SWI	Switches the program being executed, then begins execution from the first line.	–	Command
T				
128	TAN	Acquires the tangent value for a specified value.	✓	Function
129	TCOUNTER	Outputs count-up values at 1ms intervals starting from the point when the TCOUNTER variable is reset.	✓	Function
130	TIME$	Acquires the current time as an "hh:mm:ss" format character string.	✓	Function
131	TIMER	Acquires the current time in seconds, counting from midnight.	✓	Function
132	TO	Outputs a specified value to the TO port or acquires the TO status.	✓	Command / Function
133	TOLE	Specifies/acquires the tolerance parameter of a specified robot.	✓	Command / Function
134	TORQUE	Specifies/acquires the maximum torque command value which can be set for a specified axis of a specified robot.	✓	Command / Function
135	TSKPGM	Acquires the program number which is registered in a specified task.	✓	Function
V				
136	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)	✓	Function
W				
137	WAIT	Waits until the conditions of the DI/DO conditional expression are met (with time-out).	–	Command
138	WAIT ARM	Waits until the axis operation of a specified robot is completed.	–	Command
139	WEIGHT	Specifies/acquires the tip weight (kg) parameter of a specified robot.	✓	Command / Function
140	WEIGHTG	Specifies/acquires the tip weight (g) parameter of a specified robot.	✓	Command / Function
141	WEND	Terminates the command block of the WHILE statement.	–	Command
142	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).	✓	Function
143	WHILE to WEND	Controls repeated operations.	–	Command
144	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).	✓	Function
X				
145	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).	✓	Function

Operation-specific

Program commands

General commands

No.	Command	Description	Online	Type
26	DIM	Declares the array variable name and the number of elements.	–	Command
54	LET	Executes a specified assignment statement.	✓	Command
96	REM	Expresses a comment statement.	–	Command

Arithmetic commands

No.	Command	Description	Online	Type
1	ABS	Acquires the absolute value of a specified value.	✓	Function
9	ATN	Acquires the arctangent of the specified value.	✓	Function
9	ATN2	Acquires the arctangent of the specified X-Y coordinates.	✓	Function
16	COS	Acquires the cosine value of a specified value.	✓	Function
23	DEGRAD	Converts a specified value to radians (↔RADDEG).	✓	Function
27	DIST	Acquires the distance between 2 specified points.	✓	Function
49	INT	Acquires an integer for a specified value by truncating all decimal fractions.	✓	Function
57	LSHIFT	Shifts a value to the left by the specified bit count. (↔RSHIFT)	✓	Function
95	RADDEG	Converts a specified value to degrees. (↔DEGRAD)	✓	Function
103	RSHIFT	Shifts a value to the right by the specified bit count. (↔LSHIFT)	✓	Function
115	SIN	Acquires the sine value for a specified value.	✓	Function
122	SQR	Acquires the square root of a specified value.	✓	Function
128	TAN	Acquires the tangent value for a specified value.	✓	Function

Date / time

No.	Command	Description	Online	Type
20	DATE $	Acquires the date as a "yy/mm/dd" format character string.	✓	Function
129	TCOUNTER	Outputs count-up values at 1ms intervals starting from the point when the TCOUNTER variable is reset.	✓	Function
130	TIME $	Acquires the current time as an "hh:mm:ss" format character string.	✓	Function
131	TIMER	Acquires the current time in seconds, counting from midnight.	✓	Function

Character string operation

No.	Command	Description	Online	Type
14	CHR $	Acquires a character with the specified character code.	✓	Function
51	LEFT $	Extracts a character string comprising a specified number of digits from the left end of a specified character string.	✓	Function
53	LEN	Acquires the length (byte count) of a specified character string.	✓	Function
59	MID $	Extracts a character string of a desired length from a specified character string.	✓	Function

No.	Command	Description	Online	Type
72	ORD	Acquires the character code of the first character in a specified character string.	✓	Function
101	RIGHT $	Extracts a character string comprising a specified number of digits from the right end of a specified character string.	✓	Function
124	STR $	Converts a specified value to a character string (↔VAL).	✓	Function
136	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)	✓	Function

Point, coordinates, shift coordinates

No.	Command	Description	Online	Type
12	CHANGE	Switches the hand of a specified robot.	✓	Command
44	HAND	Defines the hand of a specified robot.	✓	Command
50	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)	✓	Function
52	LEFTY	Sets the hand system of a specified robot to the left-handed system.	✓	Command
56	LOCx	Specifies/acquires point data for a specified axis or shift data for a specified element.	✓	Command / Function
77	PATH	Sets the movement path.	–	Command
85	Pn	Defines points within a program.	✓	Command
86	PPNT	Creates point data specified by a pallet definition number and pallet position number.	✓	Function
102	RIGHTY	Sets the hand system of a specified robot to the right-handed system.	✓	Command
117	Sn	Defines the shift coordinates within the program.	✓	Command
112	SHIFT	Sets the shift coordinate for a specified robot by using the shift data specified by a shift variable.	✓	Command
144	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).	✓	Function

Branching commands

No.	Command	Description	Online	Type
35	EXIT FOR	Terminates the FOR to NEXT statement loop.	–	Command
38	FOR to NEXT	Executes the FOR to NEXT statement repeatedly until a specified value is exceeded.	–	Command
40	GOSUB to RETURN	Jumps to a subroutine with the label specified by GOSUB statement, and executes that subroutine.	–	Command
41	GOTO	Unconditionally jumps to the line specified by a label.	–	Command
47	IF	Allows control flow to branch according to conditions.	–	Command
68	ON to GOSUB	Jumps to a subroutine with labels specified by a GOSUB statement in accordance with the conditions, and executes that subroutine.	–	Command
69	ON to GOTO	Jumps to label-specified lines in accordance with the conditions.	–	Command
104	SELECT CASE to END SELECT	Allows control flow to branch according to conditions.	–	Command
142	WHILE to WEND	Controls repeated operations.	–	Command

Error control

No.	Command	Description	Online	Type
33	ERR / ERL	Acquires the error code number of an error which has occurred / the line number where an error occurred.	✓	Function
67	ON ERROR GOTO	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
99	RESUME	Resumes program execution after error recovery processing.	–	Command

Program & task control

Program control

No.	Command	Description	Online	Type
11	CALL	Calls a sub-procedure.	–	Command
42	HALT	Stops the program and performs a reset.	–	Command
43	HALTALL	Stops and resets all programs.	–	Command
45	HOLD	Temporarily stops the program.	–	Command
46	HOLDALL	Temporarily stops all programs.	–	Command
82	PGMTSK	Acquires the task number in which a specified program is registered.	✓	Function
83	PGN	Acquires the program number from a specified program name.	✓	Function
109	SGI	Assigns/acquires the value to a specified integer type static variable.	✓	Command / Function
110	SGR	Assigns/acquires the value to a specified real type static variable.	✓	Command / Function
127	SWI	Switches the program being executed, then begins execution from the first line.	–	Command
135	TSKPGM	Acquires the program number which is registered in a specified task.	✓	Function

Task control

No.	Command	Description	Online	Type
13	CHGPRI	Changes the priority ranking of a specified task.	✓	Command
19	CUT	Terminates another task currently being executed or temporarily stopped.	✓	Command
37	EXIT TASK	Terminates its own task which is in progress.	–	Command
98	RESTART	Restarts another task during a temporary stop.	✓	Command
123	START	Specifies the task number and priority ranking of a specified program, and starts that program.	✓	Command
126	SUSPEND	Temporarily stops another task which is being executed.	–	Command

Robot control

Robot operations

No.	Command	Description	Online	Type
29	DRIVE	Moves a specified axis of a specified robot to an absolute position.	✓	Command
30	DRIVEI	Moves a specified axis of a specified robot to a relative position.	✓	Command
61	MOTOR	Controls the motor power status.	✓	Command
62	MOVE	Performs absolute movement of all axes of a specified robot.	✓	Command
63	MOVEI	Performs relative movement of all axes of a specified robot.	✓	Command
64	MOVET	Performs relative movement of all axes of a specified robot when the tool coordinate is selected.	✓	Command
74	ORIGIN	Performs return-to-origin.	✓	Command
84	PMOVE	Executes the pallet movement command of a specified robot.	✓	Command
94	PUSH	Executes a pushing operation in the axis unit.	✓	Command
106	SERVO	Controls the servo ON/OFF of a specified axis or all axes of a specified robot.	✓	Command

Status acquisition

No.	Command	Description	Online	Type
2	ABSRPOS	Acquires the machine reference value for specified robot axes. (Valid only for axes whose return-to-origin method is set as "mark".)	✓	Function
5	ARMCND	Acquires the current arm status of a specified robot.	✓	Function
6	ARMSEL	Specifies/acquires the current "hand system" setting of a specified robot.	✓	Command / Function
7	ARMTYP	Specifies/acquires the "hand system" setting of a specified robot.	✓	Command / Function
17	CURTQST	Acquires the current torque value ratio of a specified axis to the rated torque.	✓	Function
58	MCHREF	Acquires the return-to-origin or absolute-search machine reference value for specified robot axes. (Valid only for axes whose return-to-origin method is set as "sensor" or "stroke-end".)	✓	Function
65	MTRDUTY	Acquires the motor load factor of the specified axis.	✓	Function
91	PSHRSLT	Acquires the status at the end of the PUSH statement.	✓	Function
92	PSHSPD	Specifies/acquires the push speed parameter.	✓	Command / Function
93	PSHTIME	Specifies/acquires the push time parameter.	✓	Command / Function
138	WAIT ARM	Waits until the axis operation of a specified robot is completed.	–	Command
141	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).	✓	Function
143	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).	✓	Function

Status change

No.	Command	Description	Online	Type
3	ACCEL	Specifies/acquires the acceleration coefficient parameter of a specified robot.	✓	Command / Function
4	ARCHP1	Specifies/acquires the arch position 1 parameter of a specified robot.	✓	Command / Function

No.	Command	Description	Online	Type
4	ARCHP2	Specifies/acquires the arch position 2 parameter of a specified robot.	✓	Command / Function
8	ASPEED	Specifies/acquires the AUTO movement speed of a specified robot.	✓	Command / Function
10	AXWGHT	Specifies/acquires the axis tip weight parameter of a specified robot.	✓	Command / Function
12	CHANGE	Switches the hand of a specified robot.	✓	Command
21	DECEL	Specifies/acquires the deceleration rate parameter of a specified robot.	✓	Command / Function
44	HAND	Defines the hand of a specified robot.	✓	Command
52	LEFTY	Sets the hand system of a specified robot to the left-handed system.	✓	Command
73	ORGORD	Specifies/acquires the axis sequence parameter for performing return-to-origin and an absolute search operation in a specified robot.	✓	Command / Function
76	OUTPOS	Specifies/acquires the "OUT position" parameter of a specified robot.	✓	Command / Function
81	PDEF	Defines the pallet used to execute pallet movement commands.	✓	Command
88	PSHFRC	Specifies/acquires the "Push force" parameter.	✓	Command / Function
89	PSHJGSP	Specifies/acquires the push judge speed threshold parameter.	✓	Command / Function
90	PSHMTD	Specifies/acquires the push method parameter.	✓	Command / Function
102	RIGHTY	Sets the hand system of a specified robot to the right-handed system.	✓	Command
108	SETGEP	Sets the General Ethernet Port.	✓	Command
121	SPEED	Changes the program movement speed of a specified robot.	✓	Command
133	TOLE	Specifies/acquires the tolerance parameter of a specified robot.	✓	Command / Function
139	WEIGHT	Specifies/acquires the tip weight (kg) parameter of a specified robot.	✓	Command / Function
140	WEIGHTG	Specifies/acquires the tip weight (g) parameter of a specified robot.	✓	Command / Function

PATH control

No.	Command	Description	Online	Type
77	PATH	Specifies the PATH motion path.	–	Command
78	PATH END	Ends the path setting for PATH motion.	–	Command
79	PATH SET	Starts the path setting for PATH motion.	–	Command
80	PATH START	Starts the PATH motion.	–	Command

Torque control

No.	Command	Description	Online	Type
17	CURTQST	Acquires the current torque value ratio of a specified axis to the rated torque.	✓	Function
18	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.	✓	Function
94	PUSH	Executes a pushing operation in the axis unit.	✓	Command
134	TORQUE	Specifies/acquires the maximum torque command value which can be set for a specified axis of a specified robot.	✓	Command / Function

Input/output & communication control

Input/output control

No.	Command	Description	Online	Type
24	DELAY	Waits for the specified period (units: ms).	–	Command
28	DO	Outputs a specified value to the DO port or acquires the DO status.	✓	Command / Function
55	LO	Outputs a specified value to the LO port to enable/disable axis movement or acquires the LO status.	✓	Command / Function
60	MO	Outputs a specified value to the MO port or acquires the MO status.	✓	Command / Function
75	OUT	Turns ON the bits of the specified output ports and terminates the command statement.	–	Command
97	RESET	Turns the bit of a specified output port OFF.	✓	Command
107	SET	Turns the bit at the specified output port ON.	–	Command
113	SI	Acquires a specified SI status.	✓	Function
114	SID	Acquires a specified serial input's double-word information status.	✓	Function
116	SIW	Acquires a specified serial input's word information status.	✓	Function
108	SO	Outputs a specified value to the SO port or acquires the SO status.	✓	Command / Function
119	SOD	Outputs a specified serial output's double-word information or acquires the output status.	✓	Command / Function
120	SOW	Outputs a specified serial output's word information or acquires the output status.	✓	Command / Function
132	TO	Outputs a specified value to the TO port or acquires the TO status.	✓	Command / Function
137	WAIT	Waits until the conditions of the DI/DO conditional expression are met (with time-out).	–	Command

Communication control

No.	Command	Description	Online	Type
15	CLOSE	Close the specified General Ethernet Port.	✓	Command
34	ETHSTS	Acquires the Ethernet port status.	✓	Function
39	GEPSTS	Acquires the General Ethernet Port status.	✓	Function
66	OFFLINE	Sets a specified communication port to the "offline" mode.	✓	Command
70	ONLINE	Sets the specified communication port to the "online" mode.	✓	Command
71	OPEN	Opens the specified General Ethernet Port.	✓	Command
105	SEND	Sends a file.	✓	Command

Functions: in alphabetic order

No.	Function	Type	Description
A			
1	ABS	Arithmetic function	Acquires the absolute value of a specified value.
2	ABSRPOS	Arithmetic function	Acquires the machine reference value for specified robot axes. (Valid only for axes whose return-to-origin method is set as "mark".)
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	Acquires the AUTO movement speed of a specified robot.
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.
16	COS	Arithmetic function	Acquires the cosine value of a specified value.
17	CURTQST	Arithmetic function	Acquires the current torque value ratio of a specified axis to the rated torque.
18	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.
21	DECEL	Arithmetic function	Acquires the deceleration rate parameter of a specified robot.
23	DEGRAD	Arithmetic function	Converts a specified value to radians (↔RADDEG).
27	DIST	Arithmetic function	Acquires the distance between 2 specified points.
E			
33	ERR / ERL	Arithmetic function	Acquires the error code number of an error which has occurred / the line number where an error occurred.
34	ETHSTS	Arithmetic function	Acquires the Ethernet port status.
G			
39	GEPSTS	Arithmetic function	Acquires the General Ethernet Port status.
I			
49	INT	Arithmetic function	Acquires an integer for a specified value by truncating all decimal fractions.
J			
50	JTOXY	Point function	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)
L			
51	LEFT$	Character string function	Extracts a character string comprising a specified number of digits from the left end of a specified character string.
53	LEN	Arithmetic function	Acquires the length (byte count) of a specified character string.

8

9

10

11

12

13

No.	Function	Type	Description
56	LOCx	Point function	Acquires point data for a specified axis or shift data for a specified element.
57	LSHIFT	Arithmetic function	Shifts a value to the left by the specified bit count. (↔RSHIFT)
M			
58	MCHREF	Arithmetic function	Acquires the return-to-origin or absolute-search machine reference for specified robot axes. (Valid only for axes whose return-to-origin method is set as "sensor" or "stroke-end".)
59	MID$	Character string function	Extracts a character string of a desired length from a specified character string.
65	MTRDUTY	Character string function	Acquires the motor load factor of the specified axis.
O			
72	ORD	Arithmetic function	Acquires the character code of the first character in a specified character string.
73	ORGORD	Arithmetic function	Acquires the axis sequence parameter for performing return-to-origin and an absolute search operation of a specified robot.
76	OUTPOS	Arithmetic function	Acquires the "OUT position" parameter of a specified robot.
P			
82	PGMTSK	Arithmetic function	Acquires the task number in which a specified program is registered.
83	PGN	Arithmetic function	Acquires the program number from a specified program name.
86	PPNT	Point function	Creates point data specified by a pallet definition number and pallet position number.
88	PSHFRC	Arithmetic function	Acquires the "Push force" parameter.
89	PSHJGSP	Arithmetic function	Acquires the push judge speed threshold parameter.
90	PSHMTD	Arithmetic function	Acquires the push method parameter.
91	PSHRSLT	Arithmetic function	Acquires the status at the end of the PUSH statement.
92	PSHSPD	Arithmetic function	Acquires the push speed parameter.
93	PSHTIME	Arithmetic function	Acquires the push time parameter.
R			
95	RADDEG	Arithmetic function	Converts a specified value to degrees. (↔DEGRAD)
101	RIGHT$	Character string function	Extracts a character string comprising a specified number of digits from the right end of a specified character string.
103	RSHIFT	Arithmetic function	Shifts a value to the right by the specified bit count. (↔LSHIFT)
S			
109	SGI	Arithmetic function	Acquires the value of a specified integer type static variable.
110	SGR	Arithmetic function	Acquires the value of a specified real type static variable.
113	SI	Arithmetic function	Acquires a specified SI status.
114	SID	Arithmetic function	Acquires a specified serial input's double-word information status.
115	SIN	Arithmetic function	Acquires the sine value for a specified value.
116	SIW	Arithmetic function	Acquires a specified serial input's word information status.
122	SQR	Arithmetic function	Acquires the square root of a specified value.
124	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 1ms intervals starting from the point when the TCOUNTER variable is reset.

No.	Function	Type	Description
130	TIME$	Character string function	Acquires the current time as an "hh:mm:ss" format character string.
131	TIMER	Arithmetic function	Acquires the current time in seconds, counting from midnight.
133	TOLE	Arithmetic function	Acquires the tolerance parameter of a specified robot.
134	TORQUE	Arithmetic function	Acquires the maximum torque command value which can be set for a specified axis of a specified robot.
135	TSKPGM	Arithmetic function	Acquires the program number which is registered in a specified task.
V			
136	VAL	Arithmetic function	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)
W			
139	WEIGHT	Arithmetic function	Acquires the tip weight (kg) parameter of a specified robot.
140	WEIGHTG	Arithmetic function	Acquires the tip weight (g) parameter of a specified robot.
141	WHERE	Point function	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).
143	WHRXY	Point function	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).
X			
144	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	Description
50	JTOXY	Converts joint coordinate data to Cartesian coordinate data of a specified robot. (↔XYTOJ)
56	LOCx	Acquires point data for a specified axis or shift data for a specified element.
86	PPNT	Creates point data specified by a pallet definition number and pallet position number.
141	WHERE	Reads out the current position of the arm of a specified robot in joint coordinates (pulse).
143	WHRXY	Reads out the current position of the arm of a specified robot as Cartesian coordinates (mm, degrees).
144	XYTOJ	Converts the point variable Cartesian coordinate data to the joint coordinate data of a specified robot. (↔JTOXY).

Parameter related functions

No.	Function name	Description
2	ABSRPOS	Acquires the machine reference value for specified robot axes. (Valid only for axes whose return-to-origin method is set as "mark".)
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.
17	CURTQST	Acquires the current torque value ratio of a specified axis to the rated torque.
18	CURTRQ	Acquires the current torque value of the specified axis of a specified robot.
21	DECEL	Acquires the deceleration rate parameter of a specified robot.
53	LEN	Acquires the length (byte count) of a specified character string.
58	MCHREF	Acquires the return-to-origin or absolute-search machine reference for specified robot axes. (Valid only for axes whose return-to-origin method is set as "sensor" or "stroke-end".)
65	MTRDUTY	Acquires the motor load factor of the specified axis.
72	ORD	Acquires the character code of the first character in a specified character string.
73	ORGORD	Acquires the axis sequence parameter for performing return-to-origin and an absolute search operation of a specified robot.
76	OUTPOS	Acquires the "OUT position" parameter of a specified robot.
88	PSHFRC	Acquires the "Push force" parameter.
89	PSHJGSP	Acquires the push judge speed threshold parameter.
90	PSHMTD	Acquires the push method parameter.
91	PSHRSLT	Acquires the status at the end of the PUSH statement.
92	PSHSPD	Acquires the push speed parameter.
93	PSHTIME	Acquires the push time parameter.
133	TOLE	Acquires the tolerance parameter of a specified robot.
134	TORQUE	Acquires the maximum torque command value which can be set for a specified axis of a specified robot.
139	WEIGHT	Acquires the tip weight (kg) parameter of a specified robot.
140	WEIGHTG	Acquires the tip weight (g) parameter of a specified robot.

Program related functions

No.	Function name	Description
82	PGMTSK	Acquires the task number in which a specified program is registered.
83	PGN	Acquires the program number from a specified program name.
135	TSKPGM	Acquires the program number which is registered in a specified task.

Numeric calculation related functions

No.	Function name	Description
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.
16	COS	Acquires the cosine value of a specified value.
23	DEGRAD	Converts a specified value to radians (↔RADDEG).
27	DIST	Acquires the distance between 2 specified points.
49	INT	Acquires an integer for a specified value by truncating all decimal fractions.
57	LSHIFT	Shifts a value to the left by the specified bit count. (↔RSHIFT)
95	RADDEG	Converts a specified value to degrees. (↔DEGRAD)
103	RSHIFT	Shifts a value to the right by the specified bit count. (↔LSHIFT)
115	SIN	Acquires the sine value for a specified value.
122	SQR	Acquires the square root of a specified value.
128	TAN	Acquires the tangent value for a specified value.
136	VAL	Converts the numeric value of a specified character string to an actual numeric value. (↔STR$)

Character string calculation related functions

No.	Function name	Description
14	CHR $	Acquires a character with the specified character code.
20	DATE $	Acquires the date as a "yy/mm/dd" format character string.
51	LEFT $	Extracts a character string comprising a specified number of digits from the left end of a specified character string.
59	MID $	Extracts a character string of a desired length from a specified character string.
101	RIGHT $	Extracts a character string comprising a specified number of digits from the right end of a specified character string.
124	STR $	Converts a specified value to a character string (↔VAL).
130	TIME $	Acquires the current time as an "hh:mm:ss" format character string.

Other functions

No.	Function name	Description
33	ERR / ERL	Acquires the error code number of an error which has occurred / the line number where an error occurred.
34	ETHSTS	Acquires the Ethernet port status.
39	GEPSTS	Acquires the General Ethernet Port status.
109	SGI	Acquires the value of a specified integer type static variable.
110	SGR	Acquires the value of a specified real type static variable.
129	TCOUNTER	Outputs count-up values at 1ms intervals starting from the point when the TCOUNTER variable is reset.
131	TIMER	Acquires the current time in seconds, counting from 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 the machine reference value (axes: mark method)

Format

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

Values *Robot number*........................1 to 4 (If not input, robot 1 is specified.)
Axis number...........................1 to 6

Explanation Acquires the machine reference value of axes specified by an <*axis number*>.
This function is valid only for axes whose return-to-origin method is set as "Mark", not for "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 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................1 to 100 (units: %)

Explanation Changes the acceleration coefficient parameter of the robot axis specified by the <*robot number*> to the value specified by the <*expression*>.
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*>.

Functions

Format

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

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation The acceleration coefficient parameter value is acquired for the axis specified by the <*axis number*> among the robot axes specified by the <*robot number*>.

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 arch position 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 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression..............................0 to 99999999 (Unit: pulses)

Explanation ARCHP1 corresponds to the arch position 1 parameter; ARCHP2 corresponds to the arch position 2 parameter, respectively. Changes the parameter's arch position to the value indicated in the <*expression*>.
Format 1 changes all axes specified by <*robot number*>.
Format 2 changes the only axis specified by <*axis number*> to the value indicated in the <*expression*>.

Functions

Format

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

Format

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

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation ARCHP1 corresponds to the arch position 1 parameter; ARCHP2 corresponds to the arch position 2 parameter, respectively.
Acquires the arch position parameter value of the axis specified at <*axis number*>.

4 ARCHP1 / ARCHP2

SAMPLE

```
ARCHP1 3 =10 ···················· The arch position 1 parameter value of
                                  the 3rd axis of robot 1 changes to 10
                                  pulses.
ARCHP2 3 =20 ···················· The arch position 2 parameter value of
                                  the 3rd axis of robot 1 changes to 20
                                  pulses.

    .
    .
    .
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).
NEXT
```

5 ARMCND

Acquires the current arm status

Format

```
ARMCND  [robot number]
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)

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

SAMPLE

```
A=ARMCND    ..................... The current arm status of robot 1 is
                                   assigned to variable A.
IF A=1 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 (If not input, robot 1 is specified.)
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*>.

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 (If not input, robot 1 is specified.)

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 arm type is "1" for a right-handed system, and "2" for a left-handed system.

SAMPLE

```
A=ARMSEL     ..................... The current hand system selection of
                                   robot 1 is assigned to the variable A.
IF A=1 THEN  ..................... The hand system selection is
                                   a right-handed system.
 MOVE P,P100,Z=0
ELSE         ..................... The hand system selection 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 (If not input, robot 1 is specified.)
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*>.

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 (If not input, robot 1 is specified.)

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 arm type is "1" for a right-handed system, and "2" for a left-handed system.

SAMPLE

```
A=ARMTYP   The arm type value of robot 1 is assigned to the variable A.
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
```

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

8 ASPEED

Sets/acquires the AUTO movement speed of a specified robot

Format

```
ASPEED [robot number] expression
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)
expression..............................1 to 100 (units: %)

Explanation Changes the automatic movement speed of the robot specified by the <*robot number*> to the value indicated in the <*expression*>.
This speed change applies to all 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%)

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.

Functions

Format

```
ASPEED [robot number]
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Acquires the automatic movement speed value of the robot specified by the <*robot number*>.

SAMPLE

```
SPEED 70
ASPEED 100
MOVE P,P0      ..................... Movement from the current position to P0
                                     occurs at 70% speed (=100 * 70) of the
                                     robot 1.
ASPEED 50
MOVE P,P1      ..................... Movement from the current position to P1
                                     occurs at 35% speed (=50 * 70) of the robot 1.
MOVE P,P2,S=10 ..................... Movement from the current position to P2
                                     occurs at 5% speed (=50 * 10) of the robot 1.
HALT
```

Related commands	SPEED

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

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 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................Varies according to the specified robot.

Explanation Changes the axis tip weight parameter for the specified axis to the <*expression*> value.
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 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the axis tip weight parameter value for the specified axis.
This statement is valid in systems with "MULTI" axes and auxiliary axes.

SAMPLE

```
A=5
B=0
C=AXWGHT(1) ..................... Axis tip weight value of the axis 1 of
                                  the robot 1 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 of the axis
                                  1 of the robot 1 is set again.
HALT
```

Related commands	WEIGHT, WEIGHTG

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, refer to Chapter 3 "8 Value Pass-Along & Reference Pass-Along".

MEMO

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.

- 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 or GOSUB statement are used, or depending on the use status of identifiers.
- Always use the END SUB or EXT SUB 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 "5.212: 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, EXIT SUB, SHARED

12 CHANGE

Switches the hand

Format

```
CHANGE  [robot number]   Hn
                         OFF
```

Values *robot number* ....1 to 4 (If not input, robot 1 is specified.)
n: hand number ....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.
Before hand switching can occur, the hands must be defined at the HAND statement, the programming box, or the SCARA-YRCX Studio.
For details, refer to section "44 HAND". If the hand data with another robot setting is specified, "6.258: Illegal robot no" error occurs.

SAMPLE

```
HAND H1=          0      150.000      0.000
HAND H2=      -5000      20.0000      0.000
P1=150.000 300.000 0.000 0.000 0.000 0.000

CHANGE H2   ..................... Changes the hand of the robot 1 to hand 2.
MOVE P,P1   ..................... Moves the hand 2 tip of the robot 1 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

8

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

Format

CHGPRI	Tn *<program name>* PGm	,p

Values
m: Program number ....0 to 100
n: Task number ....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
HALTALL
Program name:SUB_PGM
'SUBTASK ROUTINE
*SUBTASK:
    IF LOC3(WHERE) > 10000 THEN
        DO(20) = 1
        GOTO *SUBTASK
    ENDIF
    DO(20) = 0
GOTO *SUBTASK
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 CLOSE

Closes the specified General Ethernet Port

Format

```
CLOSE GPm
```

Values m: General Ethernet Port number ....0 to 7

Explanation Closes the communication port of the specified General Ethernet Port.

SAMPLE

```
OPEN GP1    ...................... Opens the General Ethernet Port 1.
SEND "123" TO GP1 ................ Sends the character strings "123" from
                                   the General Ethernet Port 1.
SEND GP1 TO A$ ................... Receives the data from the General
                                   Ethernet Port 1 and Saves the received
                                   data in the variable A$.
CLOSE GP1   ...................... Closes the General Ethernet Port 1.
```

Related commands	OPEN, SEND, SETGEP, GEPSTS

16 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 C 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

17 CURTQST

Acquires the ratio of the current torque (current) value of axis against the rated torque (current) value

Format

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

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the percentage (-1000 to 1000 [%]) of the current torque value against the rated torque value of axis.
Plus/minus signs indicate the direction.

SAMPLE

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

18 CURTRQ

Acquires the ratio of the current torque (current) value of axis against the maximum torque (current) value

Format

```
CURTRQ [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the percentage (-100 to 100 [%]) of the current torque value against the maximum torque command value of axis.
Plus/minus signs indicate the direction.

SAMPLE

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

19 CUT

Terminates another task which is currently being executed

Format

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

Values m: Program number ....0 to 100
n: Task number ....1 to 16

Explanation 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
HALTALL

Program name:SUB_PGM
'TASK2 ROUTINE
*SUBTASK2:
    P100=JTOXY(WHERE)
    IF LOC3(P100) >= 100.0 THEN
        MO(20) = 1
    ELSE
        DELAY 100
    ENDIF
GOTO *SUBTASK2
EXIT TASK
```

Related commands	EXIT TASK, RESTART, START, SUSPEND

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

20 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 programming box.

SAMPLE

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

Related commands	TIME$

21 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 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................1 to 100 (units: %)

Explanation Change the deceleration rate parameter of the specified robot axis to the <*expression*> value.
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
- The acceleration parameter can be changed by using the ACCEL statement.

Functions

Format

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

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)
axis number ....1 to 6

Explanation Acquires the deceleration rate parameter value for the specified axis.

SAMPLE

```
A =50
DECEL A      ·····················Specifies 50 in the deceleration
                                  rate parameter for
                                  all axes of robot 1
DECEL(3)=100 ···················Specifies 100 as the deceleration
                                  rate parameter for the axis 3 of
                                  robot 1
'CYCLE WITH INCREASING DECELERATION
FOR A =10 TO 100 STEP 10
   DECEL A   ···················Specifies the variable A value in
                                  the deceleration rate parameter for
                                  all axes of robot 1
   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 "
```

22 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 characters 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 <*dummy arguments*> are the same type, the difference of variable names does not affect.
- 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.
```

23 DEGRAD

Angle conversion (degree → radian)

Format

```
DEGRAD (expression)
```

Values *expression*............................Angle (units: degrees)

Explanation The <*expression*> value is converted to radians.

MEMO

- To convert radians to degrees, use RADDEG.

SAMPLE

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

Related commands	ATN, COS, RADDEG, SIN, TAN

24 DELAY

Program execution waits for a specified period of time

Format

```
DELAY expression
```

Values *expression*..............................0 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

A-50
DELAY A*10 500ms (0,5 secs) wait
```

25 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 number ....0 to 7, 10 to 17, 20 to 27

b: bit definition.....................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Indicates the parallel input signal status.
Enter "0" if no input port exists.

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".

26 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 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

- The total number of array elements is <*constant*> + 1.
- A "9.300: 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).
```

27 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 (units: mm) between the 2 points (X,Y,Z) specified by <*point expression 1*> and <*point expression 2*>. An error occurs if the 2 points specified by each <*point expression*> do not have Cartesian coordinates.

SAMPLE

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

28 DO

Outputs to parallel port or acquires the output status

Format

```
1.  LET  DOm (b,···,b)   =expression
2.  LET  DO (mb,···,mb)   =expression
```

Values m: port number ....2 to 7, 10 to 17, 20 to 27

b: bit definition......................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Outputs the specified value to the DO port.
No output will occur if a nonexistent DO port is specified.
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.
```

28 DO

Functions

Format

```
LET DOm (b,···,b)
LET DO (mb,···,mb)
```

Values
m: port number ....0 to 7, 10 to 17, 20 to 27
b: bit definition......................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation References the parallel port signal status.

SAMPLE

```
A%= DO2()   ...................... Output status of ports DO(27) to DO(20)
                                   is assigned to variable A%.
A%= DO0(6, 5, 1) ................. Output status of DO(06), DO(05) and
                                   DO(01) is assigned to variable A%.
                                   (If all above signals are 1(ON), then
                                   A%=7.)
A%=DO(37,35,27,10) ............... Output status of DO(37),
                                   DO(35) , DO(27) and DO(10)
                                   is assigned to variable A%.
                                   (If all above signals except DO(27)
                                   are 1 (ON), then A%=13.)
```

Related commands	RESET, SET

29 DRIVE

Executes absolute movement of specified axes

Format

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

Values

robot number .... 1 to 4 (If not input, robot 1 is specified.)
axis number .... 1 to 6
expression .... Motor position (mm, degrees, pulses) or point expression

Explanation Executes absolute movement command for the specified axis
This command is also used in the same way for the auxiliary axes.

- Movement type: PTP movement of specified axis.
- Point setting method: Direct numeric value input, point definition.
- Options: Speed setting, STOPON conditions setting, XY setting.

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.

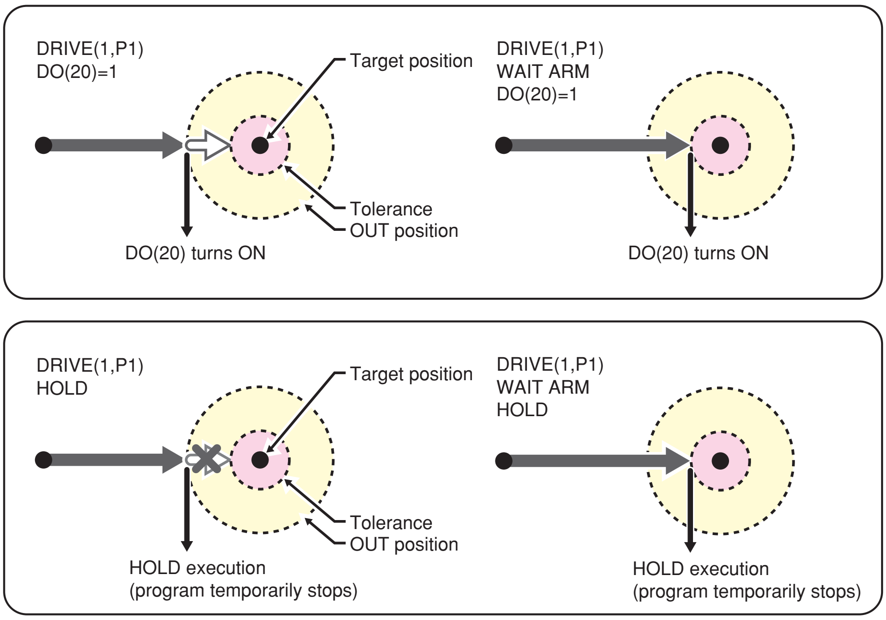
If the next command following the DRIVE command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position tolerance range.

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 of robot 1 moves from its
                               current position to the position
                               specified by P0.
```

Point data setting types

● Direct numeric value input

The target posotion is specified directly in <*expression*>.

If the numeric value is an integer, this is interpreted as "pulse" units. If the numeric value is a real number, this is interpreted as "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 Cartesian 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.
```

29 DRIVE

● 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 Cartesian 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 of robot 1 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
```

Values *expression*.............................1 to 100 (units: %)

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

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 (%) × value of *expression* (%)

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
```

Values *expression*.............................0.01 to 100.00 (units: %)

NOTE
- SPEED option and DSPEED option cannot be used together

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) × value of *expression* (%)

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 condition 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 during program execution.

SAMPLE

```
DRIVE(1,10000),STOPON DI(20)=1
           ··········· Axis 1 of robot 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 with X and Y- axes.
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 of robot 1 move from their
                       current positions to the Cartesian coordinates
                       position specified by P100.
```


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

30 DRIVEI

Moves the specified robot axes in a relative manner

Format

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

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................Target position (mm, deg, pulses) or point expression

Explanation Executes relative movement, 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

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 parameters. (For details, refer to the YRCX user's/ operator'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 target position before interruption is reset.

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.

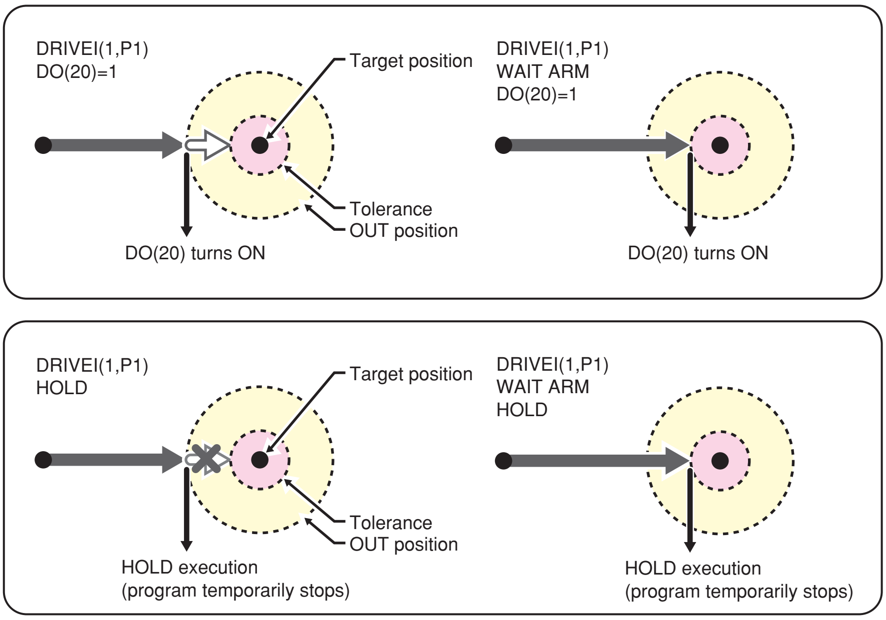
If the next command following the DRIVEI command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position tolerance range.

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.

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: 99,999,999 [pulse]
Minus-direction soft limit: -99,999,999 [pulse]

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

SAMPLE

```
DRIVEI(1,P0) .................... The axis 1 of robot 1 moves from its
                                  current position to the amount of
                                  distance specified by P0.
```

30 DRIVEI

Point data setting types

● Direct numeric value input

The target position is specified in <*expression*>.

If the target position's numeric value is a real number, this is interpreted as a "mm/ deg" units, and each axis will move from its current position to a pulse-converted position.

SAMPLE

```
DRIVEI(1,10000) ··············From its current position, the axis 1
                              of robot 1 moves a distance of "+10000
                              pulses".
DRIVEI(4,90.00) ··············From its current position, the axis 4
                              of robot 1 moves +90°(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.

From its current position, the axis moves the distance specified by the point in a relative manner.

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 of robot 1 moves from its
                              current position the distance specified
                              by P1.
DRIVEI(4,P90)·················The axis 4 of robot 1 moves from its
                              current position the number of degrees
                              specified by P90 (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 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 of robot 1 moves from
                               its current position to the +10000
                               pulses position at 10% of the program
                               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 of robot 1 moves from its
                               current position to the +10000 pulses
                               position at 0.1% of the maximum speed.
```

30 DRIVEI

● STOPON condition 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 of robot 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.
```

31 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, refer to section "104 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

32 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, refer to section "125 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

33 ERR / ERL

Acquires the error code / error line number

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

34 ETHSTS

Acquires the Ethernet port status

Format

```
ETHSTS
```

Explanation Acquires the Ethernet port status.

-2 .... Ethernet port is not opened yet.
-1 .... LAN cable is not connected.
0........ The connection is not established.
1........ The connection is established.
2........ The connection is established / the data is stored in the reception buffer.

SAMPLE

```
A=ETHSTS   ..................... Assigns the the Ethernet port
                                  status to the variable A
```

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

35 EXIT FOR

Terminates the FOR to NEXT statement loop

Format

```
EXIT FOR
```

Explanation 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 "5.212: 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

36 EXIT SUB

Terminates the sub-procedure defined by the SUB to END SUB statement

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 "5.212: 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

37 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
HALTALL

Program name:SUB_PGM
'TASK2 ROUTINE
*SUBTASK2:
    P100=JTOXY(WHERE)
    IF LOCZ(P100)>=100.000 THEN
        MO(20)=1
        EXIT TASK
    ENDIF
    DELAY 100
GOTO *SUBPTASK2
EXIT TASK
```

Related commands	CUT, RESTART, START, SUSPEND, CHGPRI

38 FOR to NEXT

Performs loop processing until the variable exceeds the specified value

Format

```
FOR control variable = start value TO end value STEP step
    command block
NEXT control variable
```

Explanation These 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

39 GEPSTS

Acquires the General Ethernet Port status

Format

```
GEPSTS(General Ethernet Port number)
```

Values *General Ethernet Port number*.................... 0 to 7

Explanation Acquires the specified General Ethernet port status.

-2 ..... The specified General Ethernet port is not opened yet.
-1 ..... LAN cable is not connected.
0 ..... The connection is not established.
1 ..... The connection is established.
2 ..... The connection is established / the data is stored in the reception buffer.

SAMPLE

```
OPEN GP1   ····················· Opens the port which is specified at
                                 the General Ethernet port 1
IF GEPSTS(1) > 0 THEN ·········· Confirms if the connection is
                                 established.
   SEND "ABC" TO GP1············ Sends the character string "123".
   IF GEPSTS(1)=2 THEN·········· Confirms if the data is stored in the
                                 reception buffer.
      SEND GP1 TO RET$ ········· Receives the data and assigns the
                                 received to the variable RET$.
   ENDIF
ENDIF
CLOSE GP1  ····················· Closes the port which is specified at
                                 the General Ethernet port 1.
HALT
```

Related commands	OPEN, CLOSE, SEND, SETGEP

40 GOSUB to RETURN

Jumps to a subroutine

Format

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

Explanation Jumps to the <*label*> subroutine specified by the GOSUB statement.
A RETURN statement within the subroutine 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 "5.212: 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

41 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*>.

SAMPLE

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

42 HALT

Stops the program and performs a reset

Format

```
HALT    expression
        character string
```

Explanation Stops the program and resets it. If restarted after a HALT, the program runs from its beginning.
If an <*expression*> or a <*character string*> is written, the operation result of <*expression*> or the contents of <*character string*> are displayed on the programming box screen, respectively

MEMO

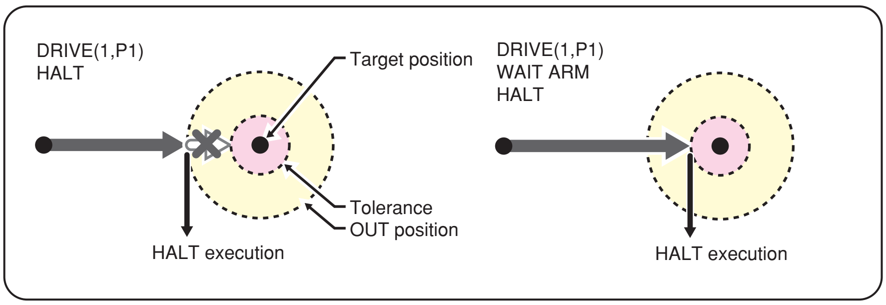
- Variables are not reset by execution of HALT statement. HALTALL is available to reset variables.
- HALT is effective only in the executed task. The programs executed in other tasks continue execution.

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, when specifying CONT options in interpolation movement during MOVE (L or C) command, the next command is executed immediately after movement starts. Therefore, if a HALT command exists immediately after the interpolation movement command during MOVE (L or C) command with CONT options, a HALT command is executed immediately after starting movement.
In either of the above cases, use the WAIT ARM command as shown below if desiring to execute the HALT command after the axis arrives within the target position tolerance range.

HALT command



33821-R7-00

43 HALTALL

Stops all programs and performs reset

Format

```
HALTALL | expression       |
        | character string |
```

Explanation Stops and resets all programs. Dynamic variables, array variables, output variables are also rest.

If a program is restarted after a HALTALL, the program runs from its beginning of the main program or of the last program executed at task 1.

If an <*expression*> or a <*character string*> is written, the calculation result of <*expression*> or the contents of <*character string*> are displayed on the programming box screen, respectively (if variable is written in an <*expression*>, the previous value before clearing is displayed).

MEMO

Output variables (DO/SO/MO/LO/TO/SOW) are reset under the condition as shown below.

- IO parameter "DO output at Program reset" is "IO_RESET".
- Sequence program is in execution and the sequence program execution flag is enabled.

SAMPLE

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

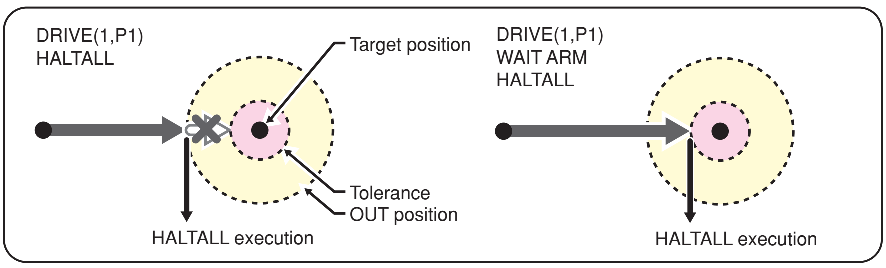
In PTP movement specified by movement commands such as MOVE and DRIVE, the next line's command is executed when the axis enters the OUT position range.

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, when specifying CONT options in interpolation movement during MOVE (L or C) command, the next command is executed immediately after movement starts. Therefore, if a HALTALL command exists immediately after the interpolation movement command during MOVE (L or C) command with CONT options, a HALTALL command is executed immediately after starting movement.

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

HALTALL command



33701-R9-00

44 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 number ....0 to 31
R: Indicates whether a hand is attached to the R-axis.

Explanation The HAND statement only defines the hand. To actually change hands, the CHANGE statement must be used.
For CHANGE statement details, refer to section "12 CHANGE".
If "R" is specified, the hands that are offset from the R-axis rotating center are selected.

MEMO

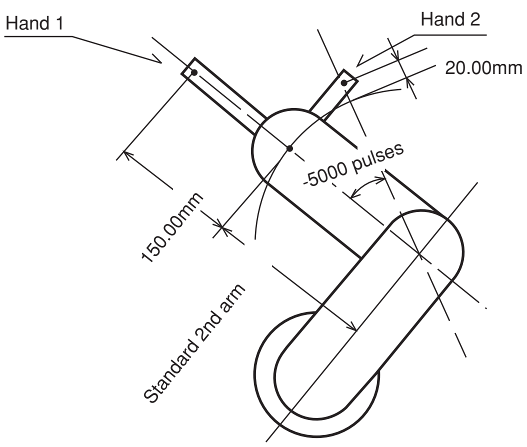
- If a power OFF occurs during execution of the hand definition statement, the "9.707 Hand data destroyed" error may occur.
- If specifying the hand data that was defined by specifying other robots in the CHANGE statement, “6.258: Illegal robot no” error may occur.

44.1 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]

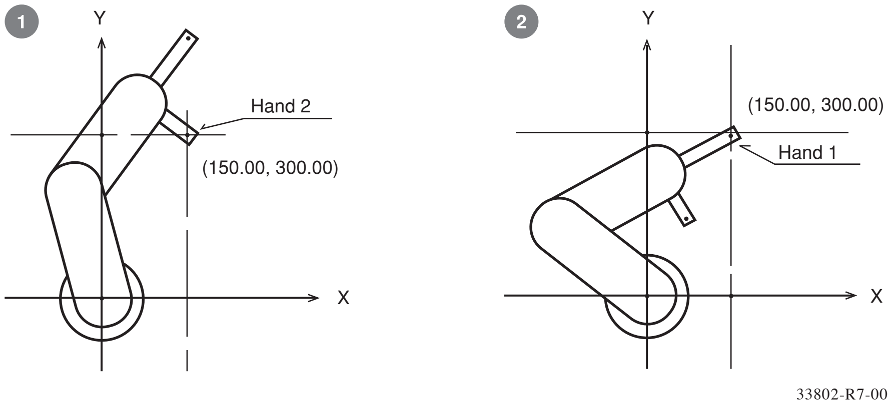


33803-R9-00

SAMPLE

```
HAND H1=          0    150.000    0.0000
HAND H2=      -5000     20.000     0.000
P1=         150.000    300.000     0.000      0.000     0.000     0.000
CHANGE H2   ..................... Hand of robot 1 changes to hand 2.
MOVE P,P1   ..................... Tip of hand 2 of robot 1 moves to P1. ❶
CHANGE H1   ..................... Hand of robot 1 changes to hand 1.
MOVE P,P1   ..................... Tip of hand 1 of robot 1 moves to P1. ❷
HALT
```

SAMPLE:HAND



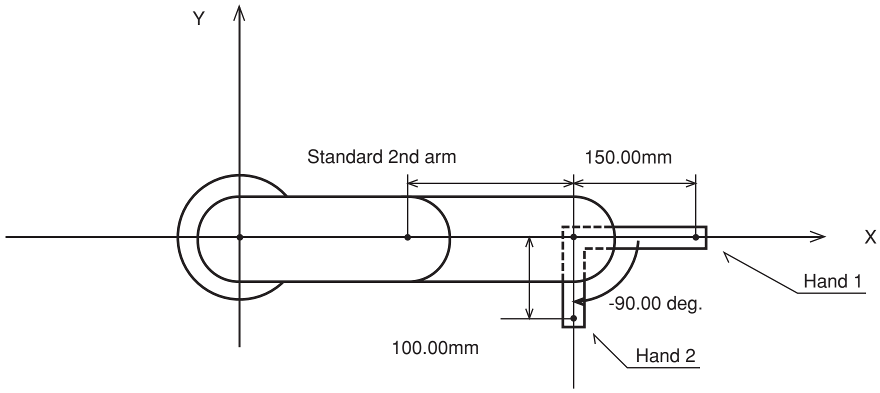
2. When the <*4th parameter*> "R" is specified

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]

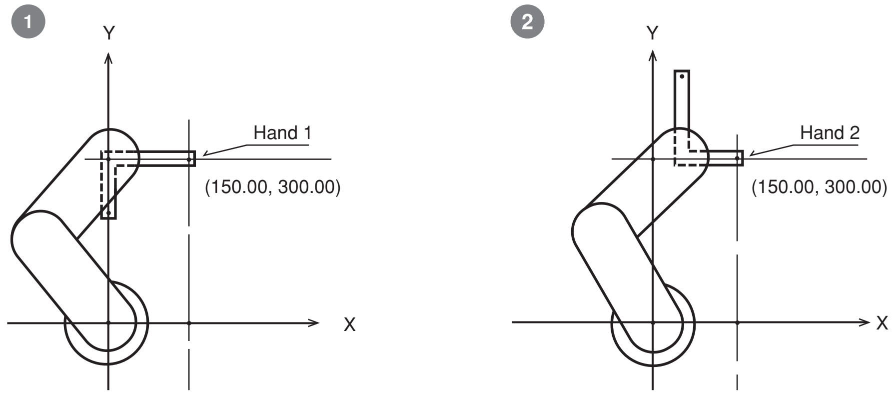


33804-R9-00

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     ..................... Hand of robot 1 changes to hand 1.
MOVE P,P1     ..................... Tip of hand 1 moves to P1. ❶
CHANGE H2     ..................... Hand of robot 1 changes to hand 2.
MOVE P,P1     ..................... Tip of hand 2 moves to P1. ❷
HALT
```

SAMPLE:HAND



33804-R7-00

45 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

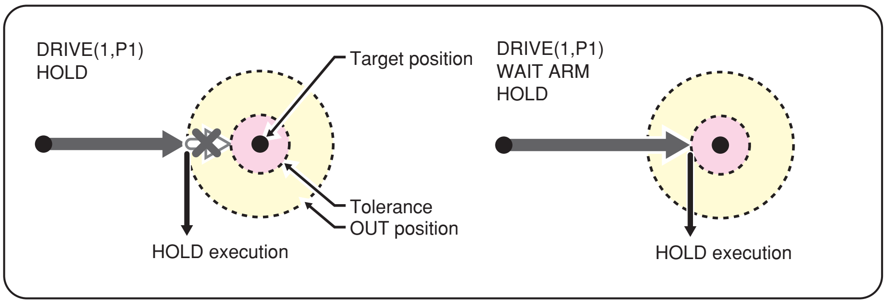
- HOLD is effective only in the task executed. The programs executed in other tasks continue execution.

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, when specifying CONT options in interpolation movement during MOVE (L or C) command, the next command is executed immediately after movement starts. Therefore, if a HOLD command exists immediately after the interpolation movement command during MOVE (L or C) command with CONT options, a HOLD command is executed immediately after starting movement.
In either of the above cases, use the WAIT ARM command as shown below if desiring to execute the HOLD command after the axis arrives within the target position tolerance range.

HOLD command



33822-R7-00

46 HOLDALL

Temporality stops all programs

Format

```
HOLD    expression
        character string
```

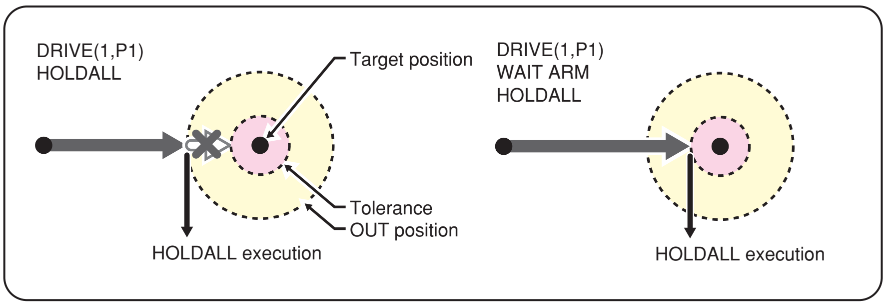
Explanation Temporality stops all programs. When restarted, the program that has executed HOLDALL is executed from the next line after the statement, and other programs are resumed from the line that has interrupted execution. If an <*expression*> or <*character sting*> is written in the statement, the 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, when specifying CONT options in interpolation movement during MOVE (L or C) command, the next command is executed immediately after movement starts. Therefore, if a HOLDALL command exists immediately after the interpolation movement command during MOVE (L or C) command with CONT options, a HOLDALL command is executed immediately after starting movement.
In either of the above cases, use the WAIT ARM command as shown below if desiring to execute the HOLDALL command after the axis arrives within the target position tolerance range.

HOLDALL command



33702-R9-00

47 IF

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

47.1 Simple IF statement

Format

IF *conditional expression* THEN	*label 1* *command statement 1*	ELSE	*label 2* *command statement 2*

Explanation If the condition specified by the <*conditional expression*> is met (true), 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 (false), 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.

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.

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
```

47.2 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 (true), 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 (true), the instructions specified in <*command block 2*> are executed.
If all the conditions specified by the conditional expression are not met (false), <*command block n*> is executed.

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.

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
```

48 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 double 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.

Input	Contents of A$
ABC	ABC
(space)ABC(space)	ABC: space is not entered before and after ABC
" ABC "	ABC : space is entered before and after ABC
ABC,XYZ	ABC is entered, and XYZ is entered when the next INPUT statement is executed.
"ABC,XYZ"	ABC,XYZ
""""ABC""""	"ABC"

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 (").
- When using INPUT statement, the value is assigned to the variable from the channel specified in cotroller parameter "INPUT/PRINT using channel".

48 INPUT

SAMPLE

```
INPUT A     ..................... Converts the enterered character
                                  string to a real number and assigns to
                                  variable A!.
INPUT "INPUT POINT NUMBER";A1
            ..................... Displays INPUT POINT NUMBER on a prompt
                                  of programming box, etc. and converts
                                  the enterered character string to a
                                  real number and assigns to variable A!.
INPUT "INPUT STRING",B$(0),B$(1)
            ..................... Displays INPUT STRING on a prompt of
                                  programming box, etc. If commas are
                                  contained in the enterered character
                                  string, the first character string is
                                  assigned to 0 element of the array
                                  variable B$ and the second character
                                  string is assigned to its 1 element.
INPUT P100  ..................... Assigns the enterered character string
                                  to P100.
HALT
```

49 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.
```

50 JTOXY

Performs axis unit system conversions (pulse → mm)

Format

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

Values *robot number*........................1 to 4 (If not input, robot 1 is specified.)

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*>.

SAMPLE

```
P10=JTOXY(WHERE) ················ Current position data of robot 1 is
                                  converted to Cartesian coordinate data
                                  and assigned to P10.
```

Related commands	XYTOJ

51 LEFT$

Extracts character strings from the left end

Format

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

Values *expression*..............................0 to 255

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 255, otherwise an error will occur.
If the <*expression*> value is 0, then extracted character string will be a null string (empty character string).
If the <*expression*> value has more characters than the <character string expression>, extracted character string 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$, RIGHT$

52 LEFTY

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

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

Format

```
LEFTY  [robot number]
```

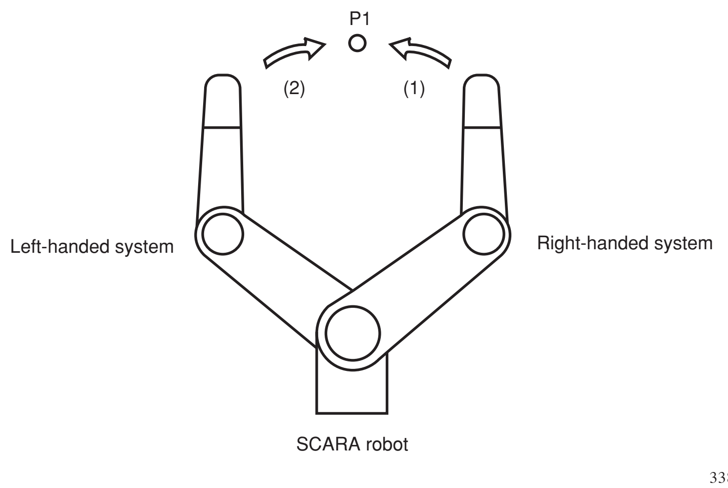
Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Specifies the robot as a left-handed system.
This statement only specifies 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).

SAMPLE

```
RIGHTY      ..................... Specifies the hand system of robot 1 as
                                  a right-handed system.
MOVE P,P1   ..................... (1)
LEFTY       ..................... Specifies the hand system of robot 1 as
                                  a left-handed system.
MOVE P,P1   ..................... (2)
RIGHTY      ..................... Specifies the hand system of robot 1 as
                                  a right-handed system.
HALT
```

SAMPLE:LEFTY/RIGHTY



Related commands	RIGHTY

53

LEN

Acquires a character string length

Format

```
LEN(character string expression)
```

Explanation Returns the *character string* length of the <*character string expression*> as a number of bytes.

SAMPLE

```
A$="OMRON"
B$="OMRON MOTOR"
C$="OMRON CO., LTD."
PRINT LEN(A$) ··················· Indicates "6".
PRINT LEN(B$) ··················· Indicates "12".
PRINT LEN(C$) ··················· Indicates "16".
```

54 LET

Assigns values to variables

Format

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

Explanation 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 "9.702: Point data destroyed" or the "9.706: Shift data destroyed" may occur.

1. Arithmetic assignment statement

Format

```
LET | integer variable                        | =expression
    | real variable
    | parallel output variable
    | internal output variable
    | arm lock output variable
    | timer output variable
    | serial output variable
    | serial word output variable
    | serial double-word output variable
```

Values *expression* ....Variables (except character string variables, point data variables, shift variables)
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$ ="OMRON"
B$ ="ROBOT"
D$ = A$ + "-" + B$

Execution result: OMRON-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 / Division..... Multiplication / 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 rounded to an integer.
- For "mm" units .... Assigned a real number after being rounded off to two decimal places.

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

55 LO

Arm lock output or acquires the output status

REFERENCE

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

Format

```
1.  LET  LOm (b,···,b) =expression
2.  LET  LO (mb,···,mb) =expression
```

Values

m: port number 0, 1

b: bit definition...................... 0 to 7 (If omitted, all 8 bits are processed.) If multiple bits are specified, they are expressed from the left in descending order (high to low).

expression .... Integer value (If real number is specified, rounds to an integer.) Bits beyond the number of bit whom a assignment destination is required are ignored. (If the port number is specified, the lower 8 bits are valid. if the number of bit specified on bit definition is 1 to 8, the lower 1 to 8 bit 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(07) correspond to axes 1 to 8, LO(10) to LO(17) correspond to axes 9 to 16, respectively. An arm lock ON status occurs at axes where bits are set, and axis movement is prohibited.

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, Permits
                                  movement at axis 2.
```

55 LO

Functions

Format

```
LET LOm (b,･･･,b)
LET LO (mb,･･･,mb)
```

Values

m: port number ....0 to 7, 10 to 17, 20 to 27
b: bit definition.....................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Acquires the output status of the specified LO port.
LO(00) to LO(07) correspond to axes 1 to 8, LO(10) to LO(17) correspond to axes 9 to 16, respectively. An arm lock ON status occurs at axes where bits are set, and axis movement is prohibited.

SAMPLE

```
A%= LO0()   ····················· Output status of ports DO(07) to LO(00)
                                  is assigned to variable A%.
A%= LO0(6, 5, 1) ················ Output status of LO(06), LO(05) and
                                  LO(01) is assigned to variable A%.
                                  (If all above signals are 1(ON), then
                                  A%=7.)
A%=LO(17,15,00) ················· Output status of L0(17), LO(15) and
                                  L0(00) is assigned to variable A%.
                                  (If all above signals except L0(15)
                                  are 1 (ON), then A%=5.)
```

Related commands	RESET, SET

56 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 .... Axis or element setting .... coordinate value
Hand system flag setting........ 1 (right-handed system)
2 (left-handed system)
0 (no setting)
First / 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

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).

56 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)

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

57 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

58 MCHREF

Acquires the machine reference value (axes: sensor method / stroke-end method)

Format

```
MCHREF [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation This function returns the return-to-origin or absolute-search machine reference value (unit:%) of axes specified by an <*axis number*>.
This function is valid only for axes whose return-to-origin method is set as "Sensor" or "Stroke-end".

SAMPLE

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

59 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, the exracted value will be a null string (empty character string).

SAMPLE

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

Related commands	LEFT$, RIGHT$

60 MO

Outputs a specified value to the MO port or acquires the output status

Format

```
1.  LET  MOm(b,･･･,b) =expression
2.  LET  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 (If omitted, all 8 bits are processed.) If multiple bits are specified, they are expressed from the left in descending order (high to low).

expression ....Integer value (If real number is specified, rounds to an integer.) Bits beyond the number of bit whom a assignment destination is required are ignored. (If the port number is specified, the lower 8 bits are valid. if the number of bit specified on bit definition is 1 to 8, the lower 1 to 8 bit corresponding to the bits specified on the left side are valid.)

Explanation Outputs a specified value to the MO port.
In order to maintain the origin 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)

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", refer to 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

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

60 MO

Functions

Format

```
MOm (b,···,b)
MO (mb,···,mb)
```

Values m: port number ....2 to 7, 10 to 17, 20 to 27, 30 to 37
b: bit definition......................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Acquires the output status of the specified MO port.

SAMPLE

```
A%= MO0()   ······················ Output status of ports MO(07) to MO(00)
                                   is assigned to variable A%.
A%= MO0(6, 5, 1) ················· Output status of MO(06), MO(05) and
                                   MO(01) is assigned to variable A%.
                                   (If all above signals are 1(ON), then
                                   A%=7.)
A%=MO(17,15,00) ·················· Output status of M0(17), MO(15) and
                                   M0(00) is assigned to variable A%.
                                   (If all above signals except M0(15)
                                   are 1 (ON), then A%=5.)
A%=MO(377,365,255,123) ··········· Output status of M0(377),
                                   MO(365), MO(255) and M0(123)
                                   is assigned to variable A%.
                                   (If all above signals except M0(15)
                                   are 1 (ON), then A%=15.)
```

Related commands	RESET, SET

61 MOTOR

Controls the motor power status

Format

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

Explanation This command 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.
```

62 MOVE

Performs absolute movement of robot axes

Format

```
MOVE [robot number] (axis number,...)  PTP  ,point definition,option,option...
                                       P
                                       L
                                       C
```

Values

robot number .... 1 to 4 (If not input, robot 1 is specified.)
axis number........................... 1 to 6 (• Multiple axes specifiable
• If not input, all axes are specified.)

Explanation Executes absolute movement of the specified axes.
It is not enabled for axes of other robots or for auxiliary axes.

- Movement type : PTP, linear interpolation, circular interpolation
- 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), merged level setting

Options	PTP	Linear interpolation	Arch interpolation	Remarks
Speed setting (SPEED, DSPEED)	✓	✓	✓	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

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:

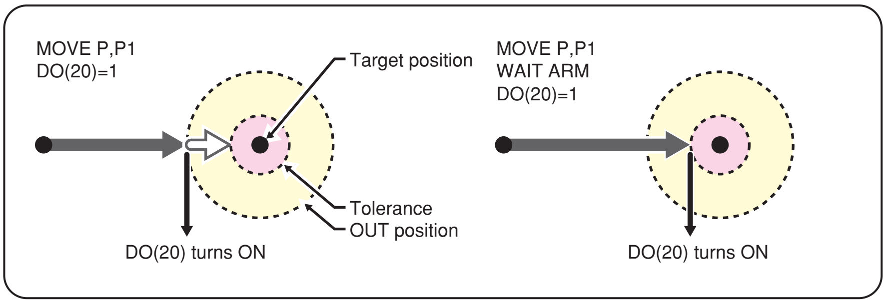
Signal output (DO, etc.)	Signal is output when the axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when the axis enters the OUT position range.
HALT	Program stops and is reset when the axis enters the OUT position range. Therefore, the axis movement also stops.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

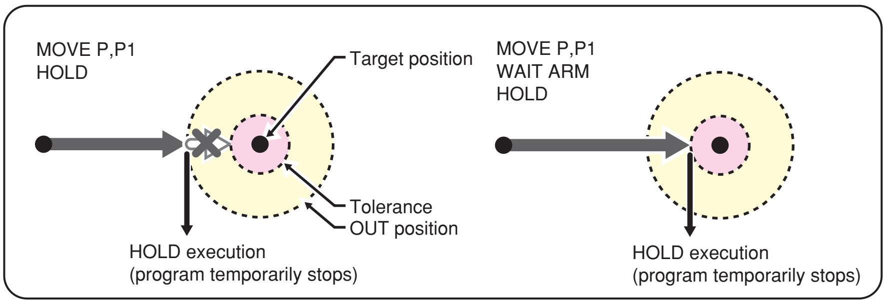
The WAIT ARM statements are used to execute the next command after the axis enters the tolerance range.

MEMO

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

MOVE command





SAMPLE

```
MOVE P,P0 ·····················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.

CAUTION

- In YRCX, the motion of interpolation movement command and END condition are different from conventional model. Addition of the CONT setting to the movement command allows to the equivalent movement and END condition in conventional model.

● Linear 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 (within the tolerance range).
All movement axes arrive at the same time.

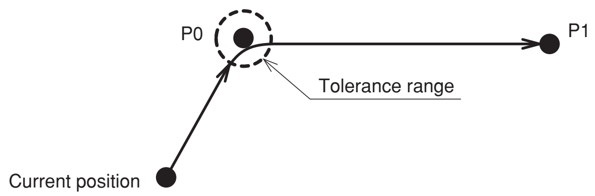
MEMO

- 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.

SAMPLE

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

SAMPLE:MOVE L



33810-R7-00

> **CAUTION**
> - In YRCX, the motion of interpolation movement command and END condition are different from conventional model. Addition of the CONT setting to the movement command allows to the equivalent movement and END condition in conventional model.

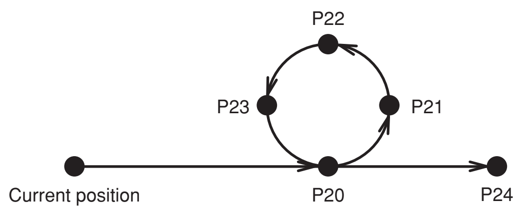
● Circular interpolation movement

Execution START condition: Movement of all specified axes is complete (within the tolerance range).
Execution END condition: Movement of all specified axes has begun.
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.**

SAMPLE

```
MOVE L,P20 ····················Linear interpolation movement of robot 1
                               occurs from the current position to P20.
MOVE C,P21,P22,P23,P20 ········Circular interpolation movement occurs
                               through points P21, P22, P23, P20.
MOVE L,P24 ····················Linear interpolation movement occurs
                               to P24.
```

SAMPLE:MOVE C

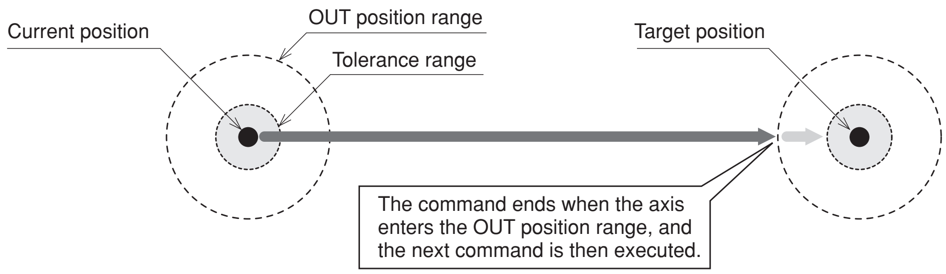


33811-R7-00

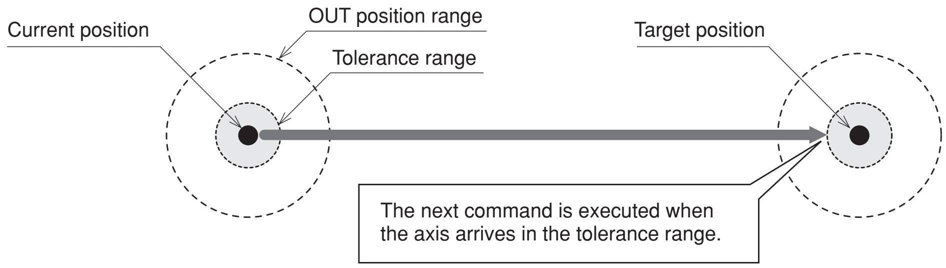
> **MEMO**
> - Circular interpolation is possible within the following range: radius 0.100mm to 5,000.000mm.
> - Circle distortion may occur, depending on the speed, acceleration, and the distance between points.
> - On robots with an R-axis, the R-axis speed may become too fast and cause an error, depending on the R-axis movement distance.

Movement command types and the corresponding movement

1. PTP movement



2. Linear interpolation movement



33703-R9-00

Point data setting types

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

Format

```
p1 p2 p3 p4 p5 p6 f
```

Values p1 to p6 ....Space-separated coordinate values for each axis
f ....Hand system flag)

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 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.

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 hand system are different, 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.

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.

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

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

SAMPLE

```
MOVE P,10000  10000  1000  1000   0   0
          ····················PTP movement of robot 1 occurs from
                              current position to the specified
                              position.
MOVE P,100.0  100.0  50.0  45.0 0.0 0.0 2
          ····················PTP movement of robot 1 occurs from
                              current position to the specified
                              position with Left-handed system.
MOVE P,-180.0 -430.0 50.0 180.0 0.0 0.0 1 -1 1
          ····················PTP movement of robot 1 occurs from
                              current position to the specified
                              position (first arm: -180°to 360°,
                              second arm: 180° to 360°) with right-
                              handed system.
```

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 hand system are different, an error will occur and robot movement will be disabled.

MEMO

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 | 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.

- 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 ····················Robot 1 moves from the current position
                               to the position specified by P1.
MOVE  P,P20,P0,P100 ···········Robot 1 moves in sequence from the
                               current position to positions specified
                               by P20, P0, P100.
```

Option types

● Speed setting 1

PTP | Linear interpolation | Circular interpolation

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 MOVE statement.

SAMPLE

```
MOVE  P,P10,S=10 ··············Robot 1 moves from the current position
                               to the position specified by P10, at
                               10% of the program movement speed.
```

● Speed setting 2

PTP | Linear interpolation | Circular interpolation

Format

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

Values *expression*..............................0.01 to 100.00 (units: %)

NOTE

• SPEED option and DSPEED option cannot be used together.

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

- [Robot max. speed (mm/sec or deg/sec)] × [movement speed (%)].

This option is enabled only for the specified MOVE statement.

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

SAMPLE

```
MOVE  P,P10,DS=0.1 ············Robot 1 moves from the current position
                               to the position specified by P10, at 0.1%
                               of the Robot maximum speed.
```

● **Speed setting 3** 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*............................. 1 to maximum speed depending on the model (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 movement type is linear interpolation or circular interpolation movements.
This option is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE L,P10,VEL=100 ············Robot 1 moves from the current position
                               to the position specified by P10 at the
                               maximum composite speed of 100 mm/sec.
                               of the XYZ axis.
```

● **Arch motion setting** PTP Linear interpolation Circular interpolation

Format

```
x =expression {expression , expression2}
```

Values

x..Specifies an axis from A1 to A6
expression ....Arch position
Integer value: "pulse" units.
Real number (with decimal point): "mm/deg" units.
expression1, expression2.......Arch distance 1, Arch distance 2
Integer value: "pulse" units.
Real number (with decimal point): "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.

NOTE

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

Explanation

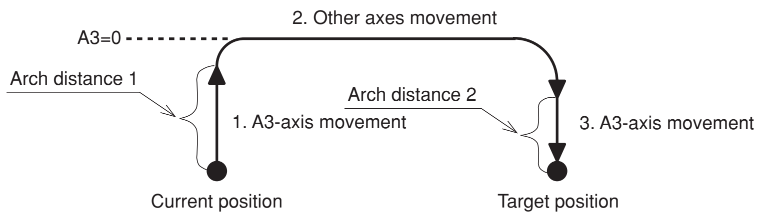
1. The "x" specified axis begins moving toward the position specified by the <*expression*> ("1" shown in the Fig. below).
2. When the axis specified by "x" moves the arch distance 1 or more, other axes move to their target positions ("2" shown in the figure below).
3. The axis specified by "x" moves 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.
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).

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.
```

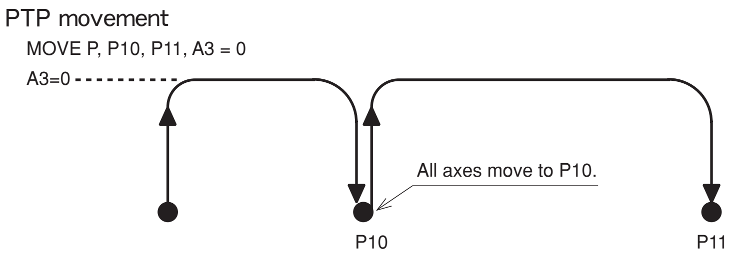
SAMPLE:MOVE A3



33704-R9-00

MEMO

- When multiple points are specified in PTP movement, the axis in arch motion setting also moves to the target position.



CAUTION

- Addition of the STOPON condition setting disables the CONT setting in the PTP movement and the linear interpolation movement.

● STOPON condition setting

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
          ....................Robot 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.

● **CONT setting** PTP Linear interpolation Circular interpolation

Format
CONT

CAUTION

- In YRCX, the motion of interpolation movement command and END condition are different from conventional model. Addition of the CONT setting to the movement command allows to the equivalent movement and END condition in conventional model.

NOTE

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

Explanation When movement is executed with CONT setting option, Movable axes will begin to execute the next command without waiting the completion their movement (entering the tolerance range). If the next command is a movement command, the 2 movement paths are linked by connecting the deceleration and acceleration sections, enabling continuous movement without intermediate stops.

This option is enabled only for the specified MOVE statement.

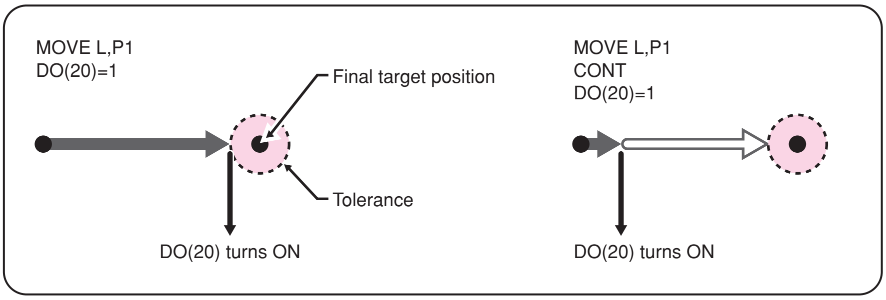
● **Caution regarding MOVE L / MOVE C command with CONT setting:**

If the next command following the MOVE L / MOVE C command with CONT setting is an executable command such as a signal output command, that next command will start immediately after axis movement begins. 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 immediately after movement along the final path begins.
DELAY	DELAY command is executed and standby starts immediately after movement along the final path begins.
HALT	Program stops and is reset immediately after movement along the final path begins. Therefore, axis movement also stops.
HALTALL	All programs in execution stop immediately after movement along the final path begins, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops immediately after movement along the final path begins. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop immediately after movement along the final path begins. Therefore, the movement also stops.
WAIT	WAIT command is executed immediately after movement along the final path begins.

MOVE command

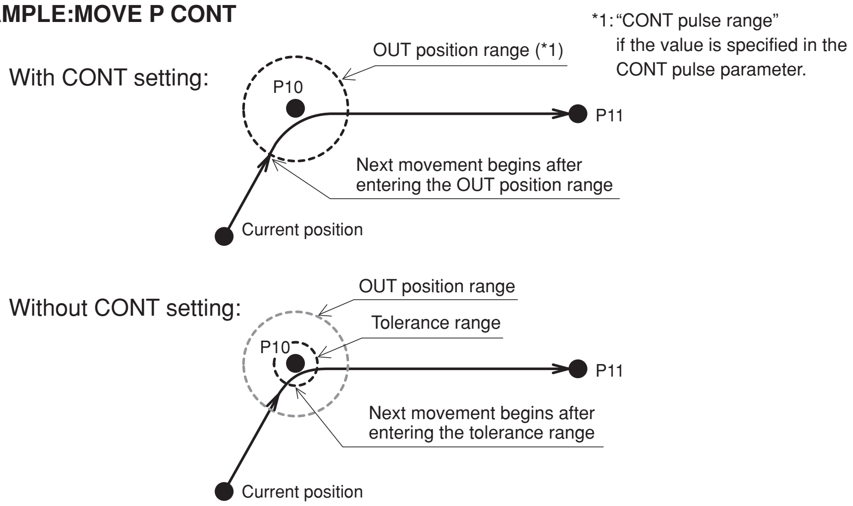


33808-R9-00

SAMPLE

```
MOVE P,P10,P11,CONT
          ....................Robot 1 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 P CONT



33814-R7-00

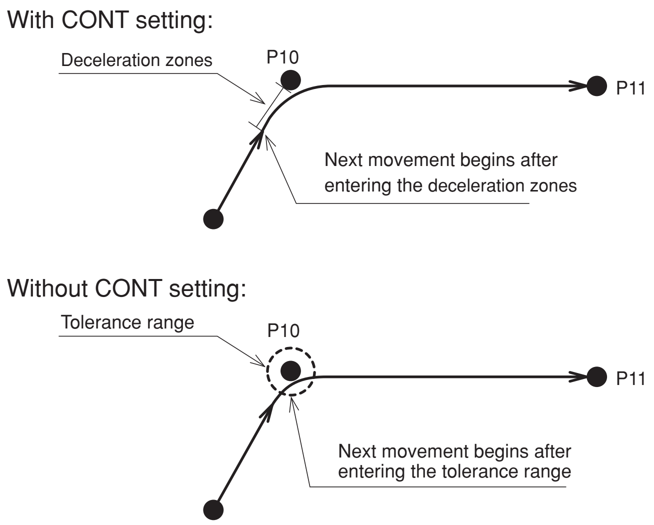
SAMPLE

```
MOVE L,P10,CONT
MOVE L,P11
          ....................Robot 1 Moves from the current position to the
                              position specified by P10, and then moves (linear
                              interpolation movement) to P11 without waiting for
                              the moving axes to arrive in the tolerance range, and
                              completes the movement within the tolerance range.
```

MEMO

- The interpolation movement with CONT setting doesn't stop at intermediate points in the continuous movement.

SAMPLE:MOVE L CONT



33810-R9-00

● **Acceleration setting** PTP Linear interpolation Circular interpolation

Format

```
ACC =expression
```

Values *expression*.............................1 to 100 (units: %)

Explanation Specifies the robot acceleration rate in the <*expression*>. The actual robot acceleration is determined by the acceleration coefficient parameter setting.
This option can only be used for PTP movement and linear interpolation movement and is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE L,P100,ACC=10 ············Robot 1 moves at an acceleration rate of
                               10% from the current position to the
                               position specified by P100.
```

● **Deceleration setting** 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 only be used for PTP movement and linear interpolation movement and is enabled only for the specified MOVE statement.

SAMPLE

```
MOVE L,P100,DEC=20 ·········Robot 1 moves at a deceleration rate of
                            20% from the current position to the
                            position specified by P100.
```

● **Coordinate plane setting** PTP Linear interpolation Circular interpolation

Format

```
|XY|
|YZ|
|ZX|
```

Values
XY..XY coordinate plane
YZ..YZ coordinate plane
ZX..ZX coordinate plane

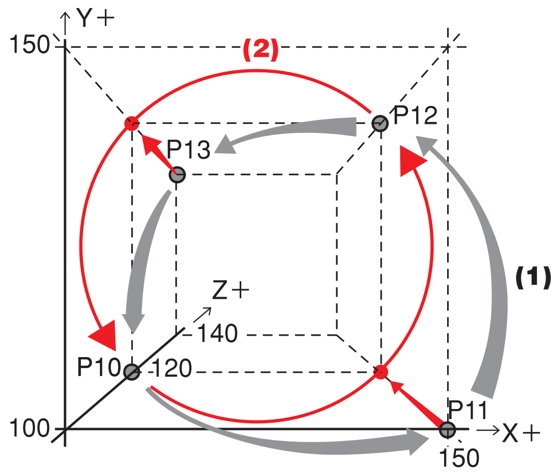
Explanation When circular interpolation is executed by setting coordinates, this option executes circular interpolation so that the projection on the specified coordinate plane becomes a circle.
This option can be used for circular interpolation movement and is enabled only for the specified MOVE statement.

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 =  100.000 100.000 20.000 0.000 0.000 0.000
P11 =  150.000 100.000  0.000 0.000 0.000 0.000
P12 =  150.000 150.000 20.000 0.000 0.000 0.000
P13 =  100.000 150.000 40.000 0.000 0.000 0.000
MOVE P,P10 ···················Robot 1 moves from the current position to the
                              position specified by 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 ········(1)
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)
                              (2)
```

SAMPLE: MOVE C coordinate plane



33822-R9-00

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

● **Port output setting** 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 (If omitted, all 8 bits are processed.) If multiple bits are specified, they are expressed from the left in descending order (high to low).

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 3rd decimal position.

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 (not arc length) 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 no hardware port exists, nothing is output.

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".

SAMPLE 1

```
MOVE P,P0
MOVE L,P1,DO2()=105@25.85
        ··········· During linear interpolation movement of robot 1
                    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 the        1 starts toward P2, MO(22)
                    switches ON when robot 1 leaves a distance of
                    10mm, and switches OFF when robot 1 leaves a
                    distance of 20mm.
```

Related commands	MOVEI, MOVET, DRIVE, DRIVEI, WAIT ARM

63 MOVEI

Performs relative movement of robot axes

Format

```
MOVEI [robot number](axis number,...) PTP ,point definition , option, option…
                                      P
                                      L
```

Values

robot number .... 1 to 4 (If not input, robot 1 is specified.)
axis number........................... 1 to 6 (• Multiple axes specifiable
• If not input, all axes are specified.)

Explanation Executes relative position movement of the specified robot.
It is not enabled for axes of other robots or for auxiliary axes.

- Movement type : PTP, linear interpolation
- Point data setting : Direct coordinate data input, point definition
- Options : Speed setting, STOPON condition setting, CONT setting, acceleration setting, deceleration setting

Options	PTP	Linear interpolation	Remarks
Speed setting (SPEED, DSPEED)	✓	✓	Enabled only for specified MOVEI statement
Speed setting (VEL)	–	✓	Enabled only for specified MOVEI statement
STOPON condition setting	✓	✓	Enabled only by program execution
CONT setting	✓	✓	Enabled only for specified MOVEI statement
Acceleration setting	✓	✓	Enabled only for specified MOVEI statement
Deceleration setting	–	✓	Enabled only for specified MOVEI statement

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 parameter. For details, refer to the YRCX user's or operator'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 P command:

If the next command following the MOVEI 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:

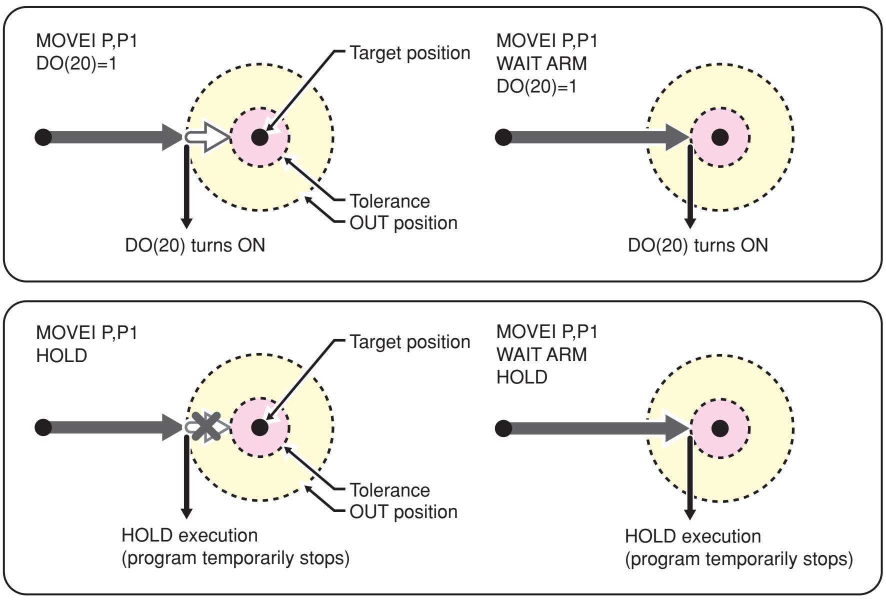
Signal output (DO, etc.)	Signal is output when axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when axis enters the OUT position range.
HALT	Program stops and is reset when axis enters the OUT position range. Therefore, axis movement also stops.
HALTALL	All 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 within the program by using the OUTPOS command.

MOVEI command



33826-R7-00

SAMPLE	
MOVEI P,P0 ·····················	From its current position, the axis of robot 1 moves (PTP movement) the amount specified by P0.

MEMO

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

CAUTION

• In YRCX, the motion of interpolation movement command and END condition are different from conventional model. Addition of the CONT setting to the movement command allows to the equivalent movement and END condition in conventional model.

● Linear interpolation movement

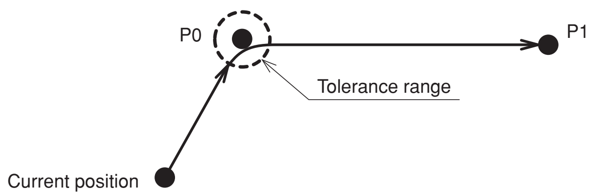
Execution START condition: Movement of all specified axes is complete (within the tolerance range).
Execution END condition: Movement of all specified axes has begun (within the tolerance range).
All movement axes arrive at the same time.

MEMO

• 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.

SAMPLE	
MOVE L,P0,P1 ··················	From its current position, the axis of robot 1 moves (linear interpolation movement) the amount specified by P0, P1.

SAMPLE:MOVEI L



Point data setting types

● Direct numeric value input

PTP Linear interpolation

Format

```
p1 p2 p3 p4 p5 p6 f
```

Values
p1 to p6 ....Space-separated coordinate values for each axis
f ....Hand system flag

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.

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.

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 of robot 1
                      moves (PTP movement) the specified amount (pulse
                      units).
```

63 MOVEI

CAUTION

• When moving the robot by linear 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.

● Point definition

PTP | Linear 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 (,).

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.

CAUTION

• 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.

SAMPLE

```
MOVEI P,P1 ···················From its current position, the axis
                              of robot 1 moves (PTP movement) the
                              amount specified by P1.
```

Option types

● Speed setting 1

PTP Linear interpolation

Format

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

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 MOVEI statement.

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

SAMPLE

```
MOVEI P,P10,S=10 ··············From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount specified by P10, at 10% of the
                               program movement speed.
```

● Speed setting 2

PTP Linear interpolation

Format

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

Values *expression*.............................0.01 to 100.00 (units: %)

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

- [Robot max. speed (mm/sec or deg/sec)] × [movement speed (%)].

This option is enabled only for the specified MOVEI statement.

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

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

SAMPLE

```
MOVEI  P,P10,DS=0.1 ···········From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount specified by P10, at 0.1% of the
                               robot maximum speed.
```

63 MOVEI

● Speed setting 3

PTP Linear interpolation

Format

```
VEL =expression
```

Values *expression* ....1 to maximum speed depending on the model (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 movement type is linear interpolation movements.

This option is enabled only for the specified MOVEI statement.

NOTE

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

SAMPLE

```
MOVEI L,P10,VEL=100 ···········From its current position, the axis of
                               robot 1 moves (linear interpolation
                               movement) the amount specified by P10,
                               at the maximum composite speed of 100
                               mm/sec. of the XYZ axis.
```

● STOPON condition setting

PTP Linear interpolation

CAUTION

• Addition of the STOPON condition setting disables the CONT 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 only possible by program execution.

SAMPLE

```
MOVEI P,P100,STOPON DI(20)=1
            ····················From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount 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.

CAUTION

- In YRCX, the motion of interpolation movement command and END condition are different from conventional model. Addition of the CONT setting to the movement command allows to the equivalent movement and END condition in conventional model.

NOTE

- The CONT setting can be used to reduce the movement START positioning time.

● CONT setting

PTP | Linear interpolation

Format

```
CONT
```

Explanation When movement is executed with CONT setting option, Movable axes will begin to execute the next command without waiting the completion their movement (entering the tolerance range). If the next command is a movement command, the 2 movement paths are linked by connecting the deceleration and acceleration sections, enabling continuous movement without intermediate stops.
This option is enabled only for the specified MOVEI statement.

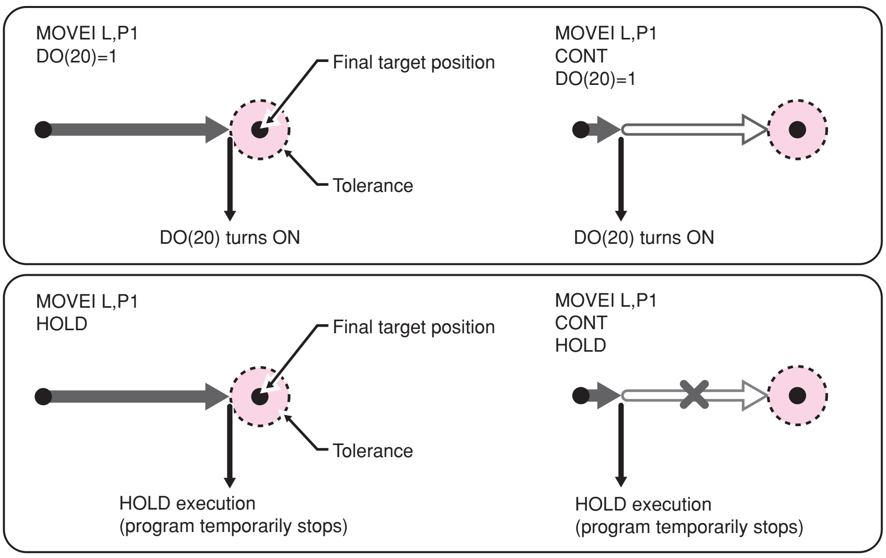
● Caution regarding MOVEI L command with CONT setting:

If the next command following the MOVEI L command with CONT setting is an executable command such as a signal output command, that next command will start immediately after axis movement begins. 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 immediately after movement along the final path begins.
DELAY	DELAY command is executed and standby starts immediately after movement along the final path begins.
HALT	Program stops and is reset immediately after movement along the final path begins. Therefore, axis movement also stops.
HALTALL	All programs in execution stop immediately after movement along the final path begins, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops immediately after movement along the final path begins. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop immediately after movement along the final path begins. Therefore, the movement also stops.
WAIT	WAIT command is executed immediately after movement along the final path begins.

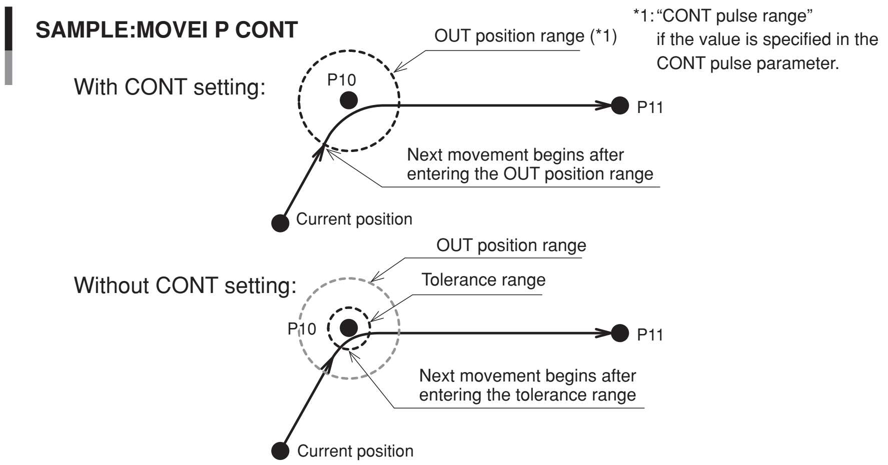
MOVEI command



33814-R9-00

SAMPLE

```
MOVEI P,P10,P11,CONT
        ········ From its current position, the axis of robot 1 moves
                 (PTP movement) the amount specified by P10, and then
                 moves the amount specified by P11 without waiting for
                 the moving axes to arrive in the tolerance range.
```



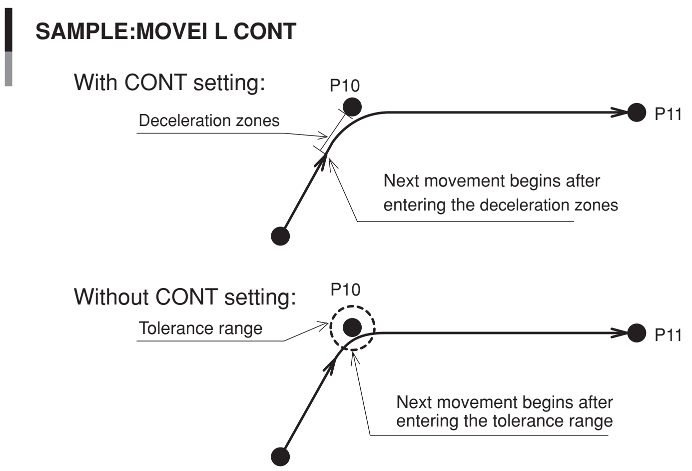
33815-R9-00

SAMPLE

```
MOVEI L,P10,CONT
MOVEI L,P11
        ········ From its current position, the axis of robot 1 moves
                 (linear interpolation movement) the amount specified by
                 P10. and then moves the amount specified by P11 without
                 waiting for the moving axes to arrive in the tolerance
                 range, and completes the movement within the tolerance
                 range.
```

MEMO

- The interpolation movement with CONT setting doesn't stop at intermediate points in the continuous movement.



33810-R9-00

● **Acceleration setting** PTP Linear 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 is enabled only for the specified MOVEI statement.

SAMPLE

```
MOVEI L,P100,ACC=10 ··········· From its current position, the axis of
                                robot 1 moves (linear interpolation
                                movement) the amount specified by P100
                                at an acceleration rate of 10%.
```

● **Deceleration setting** PTP Linear 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 is enabled only for the specified MOVEI statement.

SAMPLE

```
MOVEI L,P100,DEC=20 ··········· From its current position, the axis of
                                robot 1 moves (linear interpolation
                                movement) the amount specified by P100
                                at a deceleration rate of 20%.
```

Related commands	MOVE, MOVET, DRIVE, DRIVEI, WAIT ARM

64 MOVET

Performs relative movement of all robot axes in tool coordinates

Format

```
MOVET [robot number](axis number,...) PTP , point definition , option, option...
                                      P
                                      L
```

Values

robot number .... 1 to 4 (If not input, robot 1 is specified.)

axis number........................... 1 to 6 (• Multiple axes specifiable
• If not input, all axes are specified.)

Explanation Executes relative position movement of the specified axes in the tool coordinates. It is not enabled for axes of other robots or for auxiliary axes.

- Movement type : PTP, linear interpolation
- Point data setting : Direct coordinate data input, point definition
- Options : Speed setting, STOPON condition setting, CONT setting, acceleration setting, deceleration setting

Options	PTP	Linear interpolation	Remarks
Speed setting (SPEED, DSPEED)	✓	✓	Enabled only for specified MOVET statement
Speed setting (VEL)	–	✓	Enabled only for specified MOVET statement
STOPON condition setting	✓	✓	Enabled only by program execution
CONT setting	✓	✓	Enabled only for specified MOVET statement
Acceleration setting	✓	✓	Enabled only for specified MOVET statement
Deceleration setting	–	✓	Enabled only for specified MOVET statement

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 MOVET P command:

If the next command following the MOVET 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:

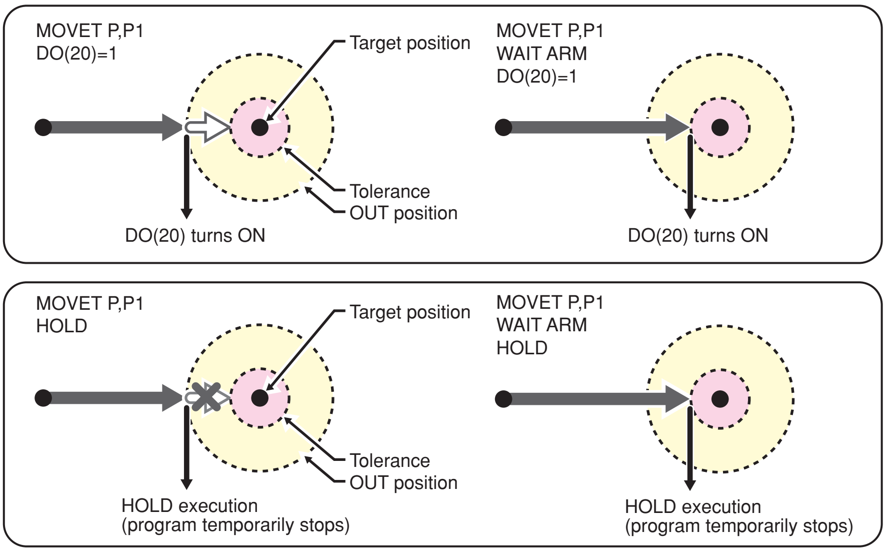
Signal output (DO, etc.)	Signal is output when the axis enters within OUT position range.
DELAY	DELAY command is executed and standby starts, when the axis enters the OUT position range.
HALT	Program stops and is reset when the axis enters the OUT position range. Therefore, the axis movement also stops.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

MOVET command



33826-R7-00

64 MOVET

SAMPLE

```
MOVET P,P0 ··················· From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount specified by P0 in the tool
                               coordinates.
```

MEMO

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

● Linear 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 (within the tolerance range).

All movement axes arrive at the same time.

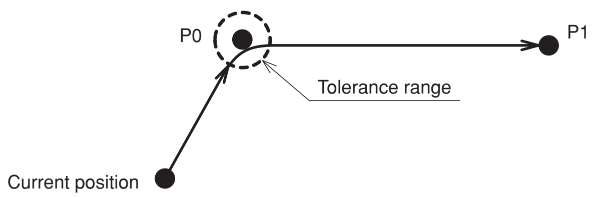
MEMO

- 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.

SAMPLE

```
MOVET L,P0,P1 ················· From its current position, the axis of
                                robot 1 moves (linear interpolation
                                movement) the amount specified by P0,
                                P1 in the tool coordinates.
```

SAMPLE:MOVET L



33810-R7-00

Point data setting types

● Direct numeric value input

PTP Linear interpolation

Format

```
p1 p2 p3 p4 p5 p6 f
```

Values p1 to p6 ....Space-separated coordinate values for each axis
f ....Hand system flag

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.

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.

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

```
MOVET P, 10.000  10.000  10.000  10.000  0.000  0.000
          ....................From its current position, the axis
                              of robot 1 moves (PTP movement) the
                              specified amount (mm units) in the tool
                              coordinates.
```

CAUTION

- When moving the robot by linear 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 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 | Linear 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 (,).

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

```
MOVET P,P1 ····················From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount specified by P1 in the tool
                               coordinates.
```

Option types

● Speed setting 1

PTP Linear interpolation

Format

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

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 MOVET statement.

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

SAMPLE

```
MOVET P,P10,S=10 ··············From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount specified by P10 in the tool
                               coordinates, at 10% of the program
                               movement speed.
```

● Speed setting 2

PTP Linear interpolation

Format

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

Values *expression*.............................0.01 to 100.00 (units: %)

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

- [Robot max. speed (mm/sec or deg/sec)] × [movement speed (%)].

This option is enabled only for the specified MOVET statement.

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

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

SAMPLE

```
MOVET  P,P10,DS=0.1 ···········From its current position, the axis
                               of robot 1 moves (PTP movement) the
                               amount specified by P10 in the tool
                               coordinates, at 0.1% of the robot
                               maximum speed.
```

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

64 MOVET

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

● Speed setting 3

PTP Linear interpolation

Format

```
VEL =expression
```

Values *expression*............................1 to maximum speed depending on the model (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 movement type is linear interpolation movements.
This option is enabled only for the specified MOVET statement.

SAMPLE

```
MOVEI L,P10,VEL=100 ··········From its current position, the axis of
                              robot 1 moves (linear interpolation
                              movement) the amount specified by
                              P10 in the tool coordinates, at the
                              maximum composite speed of 100 mm/sec.
                              of the XYZ axes.
```

CAUTION
- Addition of the STOPON condition setting disables the CONT setting.

● STOPON condition setting

PTP Linear 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 is only possible by program execution.

SAMPLE

```
MOVET P,P100,STOPON DI(20)=1
            ····················From its current position, the axis of
                              robot 1 moves (PTP movement) the amount
                              specified by P100 in the tool coordinates.
                              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.

● CONT setting

PTP Linear interpolation

Format
CONT

NOTE
- The CONT setting can be used to reduce the movement START positioning time.

Explanation

When movement is executed with CONT setting option, Movable axes will begin to execute the next command without waiting the completion their movement (entering the tolerance range). If the next command is a movement command, the 2 movement paths are linked by connecting the deceleration and acceleration sections, enabling continuous movement without intermediate stops.

This option is enabled only for the specified MOVET statement.

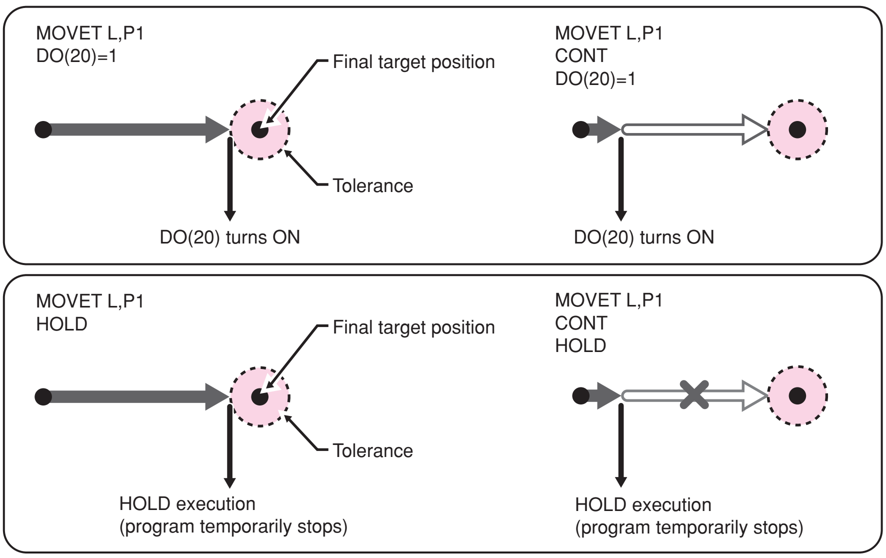
● Caution regarding MOVET L command with CONT setting:

If the next command following the MOVET L command with CONT setting is an executable command such as a signal output command, that next command will start immediately after axis movement begins. 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 immediately after movement along the final path begins.
DELAY	DELAY command is executed and standby starts immediately after movement along the final path begins.
HALT	Program stops and is reset immediately after movement along the final path begins. Therefore, the axis movement also stops.
HALTALL	All programs in execution stop immediately after movement along the final path begins, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops immediately after movement along the final path begins. Therefore, the axis movement also stops.
HOLDALL	All programs in execution temporarily stop immediately after movement along the final path begins. Therefore, the movement also stops.
WAIT	WAIT command is executed immediately after movement along the final path begins.

MOVET command

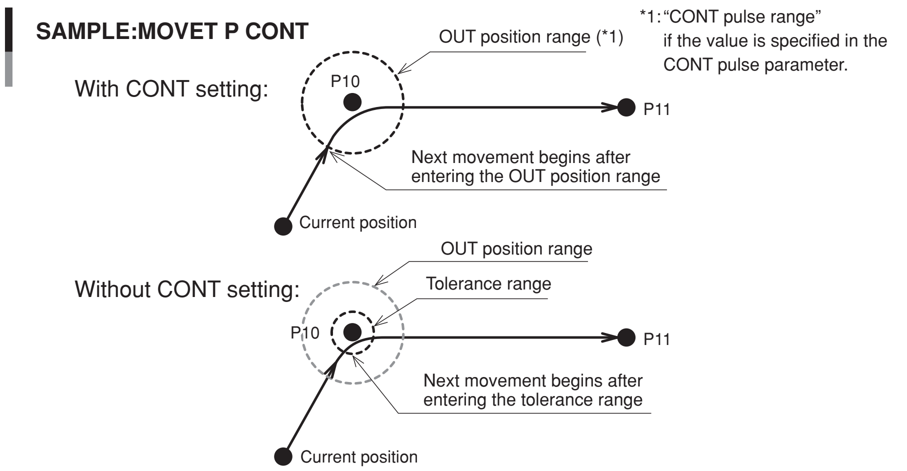


33814-R9-00

64 MOVET

SAMPLE

```
MOVET P,P10,P11,CONT
        ····· From its current position, the axis of robot 1 moves
              (PTP movement) the amount specified by P10 in the tool
              coordinates, and then moves the amount specified by P11 in
              the tool coordinates without waiting for the moving axes
              to arrive in the tolerance range.
```



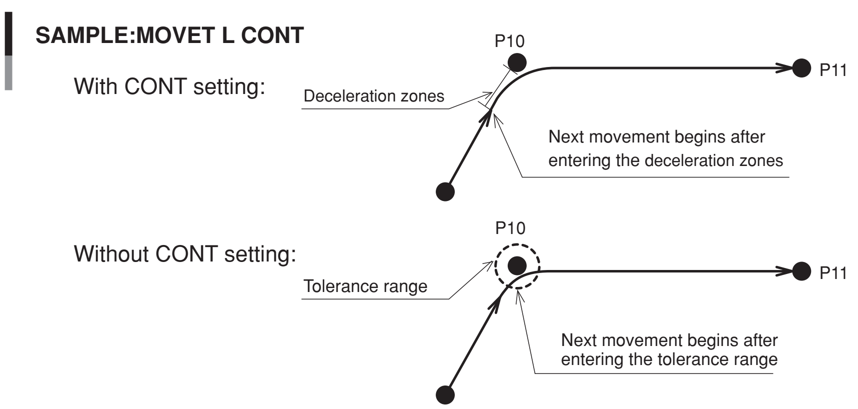
33820-R9-00

SAMPLE

```
MOVET L,P10,CONT
MOVET L,P11
        ····· From its current position, the axis of robot 1 moves
              (linear interpolation movement) the amount specified by
              P10 in the tool coordinates, and then moves the amount
              specified by P11 in the tool coordinates without waiting
              for the moving axes to arrive in the tolerance range, and
              completes the movement within the tolerance range.
```

MEMO

- The interpolation movement with CONT setting doesn't stop at intermediate points in the continuous movement.



33821-R9-00

● Acceleration setting

PTP Linear 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 is enabled only for the specified MOVET statement.

SAMPLE

```
MOVET L,P100,ACC=10 ··········From its current position, the axis of
                               robot 1 moves (linear interpolation
                               movement) the amount specified by
                               P100 in the tool coordinates at an
                               acceleration rate of 10%.
```

● Deceleration setting

PTP Linear 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 is enabled only for the specified MOVET statement.

SAMPLE

```
MOVET L,P100,DEC=20 ··········From its current position, the axis of
                               robot 1 moves (linear interpolation
                               movement) the amount specified by
                               P100 in the tool coordintes at a
                               deceleration rate of 20%.
```

Related commands	MOVE, MOVEI, DRIVE, DRIVEI, WAIT ARM

65 MTRDUTY

Acquires the motor load factor of the specified axis

Format

```
MTRDUTY   [robot number] (axis number)
```

Values

robot number..........................1 to 4 (If not input, robot 1 is specified.)
axis number............................1 to 6

Explanation Acquires the motor load factor (1 to 100 [%]) of the specified axis.

The motor load factor increases when the current value of the specified axis exceeds the rated current value. When the factor reaches 100%, “17.800 Motor overload” error occurs.

SAMPLE

```
A=MTRDUTY(1)  ................... The motor load factor of axis 1 of
                                  robot 1 is assigned to variable A.
```

66 OFFLINE

Sets a specified communication port to the "offline" mode

Format

```
OFFLINE  | ETH |
         | CMU |
```

Explanation Changes the communication mode parameter in order to switch the communication mode to OFFLINE.

ETH	Changes the Ethernet communication mode parameter to OFFLINE, clears the transmission and reception buffers.
CMU	Changes the RS-232C communication mode parameter to OFFLINE, resets the communication error, and clears the transmission and reception buffers.
No setting	Changes the Ethernet and RS-232C communication mode parameter to OFFLINE, resets the communication error (RS-232C only), and clears the transmission and reception buffers.

MEMO

- Online command is invalid in OFFLINE (mode).

SAMPLE

```
OFFLINE
SEND CMU TO A$
SEND CMU TO P10
ONLINE
HALT
```

Related commands	ONLINE

67 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 "17.800: Motor 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 = &H000600CC THEN *NEXT1 · Checks to see if a "Point doesn't
                                  exist" error has occurred.
IF ERR = &H000600CE 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

68

ON to GOSUB

Executes the subroutine specified by the <*expression*> value

8

Format

```
ON expression GOSUB label 1, label 2...
                              * GOSUB can also be expressed as "GO SUB".
```

Values *expression*.............................Expression whose result is 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

69 ON to GOTO

Jumps to the label specified by the <*expression*> value

N
O
P
Q
R
S
T
U
V
W
X
Y
Z

Format

```
ON expression GOTO label 1, label 2...
                    * GOTO can also be expressed as "GO TO".
```

Values *expression*.............................Expression whose result is 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

70 ONLINE

Sets the specified communication port to the "online" mode

Format

```
ONLINE      | ETH |
            | CMU |
```

Explanation Changes the communication mode parameter in order to switch the communication mode to ONLINE..

ETH	Changes the Ethernet communication mode parameter to ONLINE, clears the transmission and reception buffers.
CMU	Changes the RS-232C communication mode parameter to ONLINE, resets the communication error, and clears the transmission and reception buffers.
No setting	Changes the Ethernet and RS-232C communication mode parameter to ONLINE, resets the communication error (RS-232C only), and clears the transmission and reception buffers.

MEMO

- Online command is valid in ONLINE (mode).

SAMPLE

```
OFFLINE
SEND CMU TO A$
SEND CMU TO P10
ONLINE
HALT
```

Related commands	OFFLINE

N
O
P
Q
R
S
T
U
V
W
X
Y
Z

71 OPEN

Opens the specified General Ethernet Port

Format

```
OPEN GPm
```

Values m: General Ethernet Port number .... 0 to 7

Explanation Opens the communication port of the specified General Ethernet Port.

SAMPLE

```
OPEN GP1    ..................... Opens the General Ethernet Port 1.
SEND "123" TO GP1 ............... Sends the character strings "123" from
                                  the General Ethernet Port 1.
SEND GP1 TO A$ .................. Receives the data from the General
                                  Ethernet Port 1 and Saves the received
                                  data in the variable A$.
CLOSE GP1  Closes the General Ethernet Port 1.
```

Related commands	CLOSE, SEND, SETGEP, GEPSTS

72 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$

73 ORGORD

Specifies/acquires the robot's return-to-origin sequence

Format

```
ORGORD [robot number] expression
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)
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 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 (If not input, robot 1 is specified.)

Explanation Acquires the axis sequence parameter for return-to-origin and absolute search operation of the robot specified by the <*robot number*>.

SAMPLE

```
A=3
ORGORD A    ..................... Return-to-origin is executed first for
                                  axis 3 of robot 1.
ORIGIN^     ..................... After the return-to-origin of axis 3 of
                                  robot 1 is completed, return-to-origin
                                  is executed for the remaining axes.
MOVE P,P0
A=ORGORD    ..................... Return-to-origin sequence parameter of
                                  robot 1 is assigned to variable A.
HALT
```

Related commands	ORIGIN

74 ORIGIN

Performs return-to-origin

Format

```
ORIGIN [robot number], motor type
```

Values *robot number*........................0: all robots
1 to 4: specified robot only
motor type.............................0: all types
1: incremental motor only
2: absolute motor only
9: incomplete return-to-origin axis only
(If omitted, 0 (all types) is specified.)

Explanation This statement performs return-to-origin of a robot
If the movement is stopped at an intermediate point, "incomplete return-to-origin" status will occur.
If <*robot number*> is omitted or "0" is specified during multiple robots setting, the return-to-origin and absolute search are first performed for the robot 1 and then for the robots 2 to 4.

SAMPLE

```
ORIGIN 0, 1 ···················· Performs return-to-origin for
                                 incremental motor axes only of all
                                 robots.
```

Related commands	ORGORD, MCHREF

75 OUT

Turns ON the specified port output

Format

```
OUT DOm(b,・・・,b)     ,expression
    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 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

expression ....0 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 "6.225: No sufficient memory for OUT".

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

76 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 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression..............................1 to 9999999 (Unit: pulses)

Explanation Changes the "OUT position" parameter of the specified axis to the value indicated in the <*expression*>.
Format 1: The change is applied to all axes of the specified robot.
Format 2: The change is applied only to the axis specified by <*axis number*>.

Functions

Format

```
OUTPOS [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the "OUT position" parameter's value for the specified axis.

N O P Q R S T U V W X Y Z

76 OUTPOS

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
```

77 PATH

Specifies the motion path

NOTE

- When "R" axis only is specified in the coordinate attribute parameter, an error will occur.

Format

```
PATH [robot number](axis number,...) | L | , point definition , option, option...
                                     | C |
```

Values

robot number ....1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6 (• Multiple axes specifiable
• If not input, all axes are specified.)

Explanation Sets the motion path for the specified axis. This command can only be executed between the PATH SET and PATH END commands. If execution is attempted elsewhere, an error will occur.

- Movement type: Linear interpolation, circular interpolation
- Point setting: Direct numeric value input, point definition
- Options: Speed setting, coordinate plane setting (for circular interpolation only), port output setting

PATH motion types

● Linear interpolation movement

"PATH L…" is set for linear interpolation movement.

● Circular interpolation movement

"PATH C…" is set for circular interpolation movement.

Only the X, Y and Z coordinate values of the specified points are valid for PATH motion. Any other coordinates use the coordinate values of the PATH motion START point.
The motion path can be connected by repeated PATH commands ("PATH L", "PATH C") to allow movement without stopping.

Point data setting types

● Direct numeric value input

Linear interpolation Circular interpolation

Format

```
p1 p2 p3 p4 p5 p6 f
```

Values
p1 to p6 ....Space-separated coordinate values for each axis
f ....Hand system flag

Explanation Directly specifies coordinate data 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" units. If both integers and real numbers are used together (mixed), all coordinate values will be handled in "mm" units.**

With this format, only 1 point can be specified as the movement destination coordinates. The only type of movement specified by this point data setting is linear interpolation.

Hand system flags can be specified for SCARA robots when directly specifying the coordinate 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 set, 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.

The same hand system must always be used between a motion path's START and END points.
The hand system cannot be changed between these points.
Moreover, the first arm and second arm rotation information must be the same throughout the movement path, from the path's START to END points. The first arm and second arm rotation information cannot be changed at any point along the path.

CAUTION
- The hand system used during PATH motion must be the same as the hand system used at the path motion route's start point. The same applies if the path is to pass through points where hand system flags are set. Differing hand systems will cause an error and disable motion.
- The first arm and second arm rotation information during PATH movement must be the same as the first arm and second arm rotation information at the PATH movement's START point. If the two are different, an error will occur and movement will be disabled.

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

```
PATH L,10000 10000 1000 1000 0 0
          ····················Sets the linear interpolation movement
                              path of robot 1 in "pulse" units.
PATH L,150.000 250.000 10.000 30.000 0.000 0.000 1
          ····················The linear interpolation movement path
                              of robot 1 is set in the coordinate
                              values specified by the right-handed
                              system in "mm" units.
```

CAUTION

- The hand system used during PATH motion must be the same as the hand system used at the path motion route's start point. The same applies if the path is to pass through points where hand system flags are set. Differing hand systems will cause an error and disable motion.

● Point definition

Linear interpolation | Circular interpolation

Format

point definition , point definition...

Explanation Specifies the movement destination as <*point expression*> value. Two or more data items can be designated by separating them with a comma (,).
For circular interpolation movement, 2 points must be specified for each arc.

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.

CAUTION

- The first arm and second arm rotation information during PATH movement must be the same as the first arm and second arm rotation information at the PATH movement's START point. If the two are different, an error will occur and movement will be disabled.

SAMPLE

```
PATH L,P1,P2,P3···············Specifies sequential linear
                              interpolation movement of robot 1 from
                              its current position to the positions
                              specified by P1, P2 and P3 from its
                              current position.
PATH C P5,P6,P7,P8············Specifies circular interpolation
                              movement of robot 1 through the
                              following points: current position,
                              P5, P6, and P6, P7, P8.
```

77 PATH

Option types

● **Speed setting** Linear interpolation Circular interpolation

Format

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

Values *expression*.............................1 to 100 (units: %)

NOTE

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

Explanation The program's movement speed is specified as the <*expression*> value (units: %).
The actual speed is determined as shown below.

• Robot's max. speed (mm/sec) × automatic movement speed (%)× program movement speed (%).

This option is enabled only for the specified PATH statement.

SAMPLE

```
PATH L,P5,S=40 ················Movement of robot 1 from its current
                               position to the position specified by P5
                               occurs at 40% of the program movement
                               speed.
```

Format

```
VEL =expression
```

Values *expression*.............................The permissible setting range varies according to the robot type (units: mm/sec).

NOTE

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

Explanation The movement speed is specified by the <*expression*> value (units: mm/sec). An error will occur if the speed is too fast.
This command is enabled only for the specified PATH statement.

SAMPLE

```
PATH L,P10,VEL=150 ············Movement of robot 1 from its current
                               position to the position specified by P10
                               occurs at a speed of 150mm/sec.
```

● **Coordinate plane setting** Linear interpolation Circular interpolation

Format

```
| XY |
| YZ |
| ZX |
```

Values XY..XY coordinate plane
YZ..YZ coordinate plane
ZX..ZX coordinate plane

Explanation Specifies the coordinate plane on which to draw a circular arc for circular interpolation movement. If no coordinate plane is specified, 3-dimensional circular interpolation movement is used.
Only circular interpolation movement can be specified by this coordinate plane setting.
This command is enabled only for the specified PATH statement.

SAMPLE

```
PATH C,P1,P2,XY···············From its current position, circular
                              interpolation movement of robot 1
                              occurs within the XY plane, with
                              the Z-axis moving to the P2 Z-axis
                              coordinates position.
```

77 PATH

● **Port output setting** 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 |
```

CAUTION

- Output to ports "0" and "1" is not allowed at DO, MO, and SO.

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
b: bit definition.....................0 to 7 (If omitted, all 8 bits are processed.) If multiple bits are specified, they are expressed from the left in descending order (high to low).
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 3rd decimal position.

Explanation

During PATH motion, this command option outputs the value of <*expression 1*> to the specified port when the robot reaches the <*expression 2*> distance from the start position.
The <*expression 2*> numeric value represents a circle radius (not arc length) centered on the movement START point.
If no hardware port exists, nothing is output.

SAMPLE

```
PATH SET
PATH L,P1,DO(20)=1@10 ········Specifies to output "1" to DO(20) at a
                              10mm radius position from the START
                              position during linear interpolation
                              movement of robot 1 from its current
                              position to P1.
PATH L,P2,DO(21)=1@12.5 ······Specifies to output "1" to DO(21) at a
                              12.5mm radius position from P1 during
                              linear interpolation movement of robot
                              1 from its current position to P2.
PATH END
PATH START
```

Related commands	PATH SET, PATH END, PATH START

Reference For PATH function details, refer to Chapter 9 "PATH Statements".

78 PATH END

Ends the path setting

Format

```
PATH [robot number] END
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Ends the path setting of specified robot's PATH motion.

The PATH END command must always be paired with a PATH SET command. The PATH motion path end-point is the final point specified by the final PATH command (PATH L, PATH C) which exists between the PATH SET and PATH END commands.

Attempting to execute a PATH END command when no PATH SET command has been executed will result in an error.

SAMPLE

```
PATH END  ....................Ends the path setting of robot
                              1's PATH motion
```

Related commands	PATH, PATH SET, PATH START

Reference For PATH function details, see Chapter 9 "PATH Statements".

79 PATH SET

Starts the path setting

NOTE

- The PATH SET statement is available in software version 1.11 onwards.

Format

```
PATH [point definition] SET point definition
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Starts the path setting of specified robot's PATH motion.

Specifies the <*point definition*> position as the PATH motion start-point. (This only sets the PATH motion start point and does not actually begin robot motion.) If the <*point definition*> value is omitted, the current robot position is set as the start point.

However, if robot movement is in progress, the target position of that movement becomes the start point. (Example: The OUT position range is wider for the MOVE command which precedes the PATH SET command, so the robot is still moving when the PATH SET command is executed, etc.)

The PATH SET command must always be paired with a PATH END.

When a PATH SET command is executed, the previously set PATH motion path data is deleted.

- Point data setting : Direct numeric value input, point definition

● Direct numeric value input

Format

```
p1 p2 p3 p4 p5 p6 f
```

Values p1 to p6 ....Space-separated coordinate values for each axis.
f ....Hand system flag.

Explanation Directly specifies the path's start-point coordinates for PATH motion. If an integer is used, this is interpreted as "pulse" units, and if a real number is used, this is interpreted as "mm" units (valid down to the 3rd decimal position).
Hand system flags can be specified for SCARA robots when directly specifying the coordinate 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 set, 0 will be set to indicate that there is no hand system flag.
1: Indicates that a right-handed system is specified for the PATH motion's start-point.
2: Indicates that a left-handed system is specified for the PATH motion's start-point.

NOTE

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

CAUTION

- The hand system used during PATH motion must be the same hand system as that at the PATH motion's start-point. An error will occur if the hand systems are different.
- The first arm and second arm rotation information during PATH movement must be the same as the first arm and second arm rotation information at the PATH movement's START point. If the two are different, an error will occur and movement will be disabled.

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

```
PATH SET 120 250.000 55.2 20.33 0 0
           ....................The PATH motion's start-point of robot
                               1 is specified in "mm" units as follows:
                               120.000  250.000  55.200  20.330  0.000
                               0.000.
PATH SET -51200 80521 7045 204410 0 0
           ....................The PATH motion's start-point of robot
                               1 is specified in "pulse" units.
```

79 PATH SET

CAUTION

• The hand system used during PATH motion must be the same as the hand system used at the path motion route's start point. Differing hand systems will cause an error and disable motion.

MEMO

CAUTION

• The first arm and second arm rotation information during PATH movement must be the same as the first arm and second arm rotation information at the PATH movement's START point. If the two are different, an error will occur and movement will be disabled.

● Point definition

Format

point expression

Explanation The PATH motion's start-point is specified by the <*point expression*>.

• 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

```
PATH SET P10 ··················The PATH motion's start-point of
                               robot 1 is set as P10.
PATH SET WHERE ················The PATH motion's start-point of
                               robot1 is set as the robot 1's current
                               position.
```

Related commands	PATH, PATH END, PATH START

Reference For PATH function details, see Chapter 9 "PATH Statements".

80 PATH START

Starts the PATH motion

Format

```
PATH  [robot number]  START, option, option...
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Starts PATH motion of specified robot.

Before PATH START can be executed, the PATH motion path must be specified by the PATH SET command, PATH commands (PATH L, PATH C) and the PATH END command. The robot must also be positioned at the motion path's start-point which was specified by the PATH SET command.

The robot's PATH motion speed is the automatic movement speed (%) which was in effect when the PATH START was executed, multiplied by the program movement speed (%) specified by the SPEED command or the (SPEED or S) option of the PATH command. A speed specified by the "VEL" option of the PATH command does not rely on the automatic movement speed.

After PATH motion begins, the PATH START command is terminated when the robot reaches the PATH motion end-point, or when movement is stopped by a stop input, etc.

- Options : STOPON condition setting, CONT setting

80 PATH START

Option types

⚠ CAUTION

• Addition of the STOPON condition setting disables the CONT setting.

● STOPON condition 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 only possible by program execution.

SAMPLE

```
PATH START,STOPON DI(20)=1
                    ···················Robot 1 starts PATH movement, 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.

● **CONT setting**

Format
CONT

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

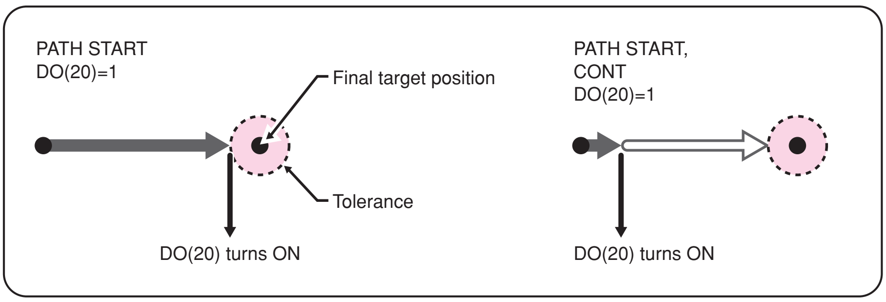
Explanation When PATH movement is executed with CONT setting option, after all movable axes begin to execute the final movement specified by PATH statement, movable axes will begin to execute the next command without waiting the completion their movement (entering the tolerance range). If the next command is a movement command, the 2 movement paths are linked by connecting the deceleration and acceleration sections, enabling continuous movement without intermediate stops.
This option is enabled only for the specified PATH START statement.

● **Caution regarding PATH START command with CONT setting:**

If the next command following the PATH START command with CONT setting is an executable command such as a signal output command, that next command will start immediately after axis movement begins. 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 immediately after movement along the final path begins.
DELAY	DELAY command is executed and standby starts immediately after movement along the final path begins.
HALT	Program stops and is reset immediately after movement along the final path begins. Therefore, axis movement also stops.
HALTALL	All programs in execution stop immediately after movement along the final path begins, task 1 is reset, and other tasks terminate. Therefore, the movement also stops.
HOLD	Program temporarily stops immediately after movement along the final path begins. Therefore, axis movement also stops.
HOLDALL	All programs in execution temporarily stop immediately after movement along the final path begins. Therefore, the movement also stops.
WAIT	WAIT command is executed immediately after movement along the final path begins.

PATH START command



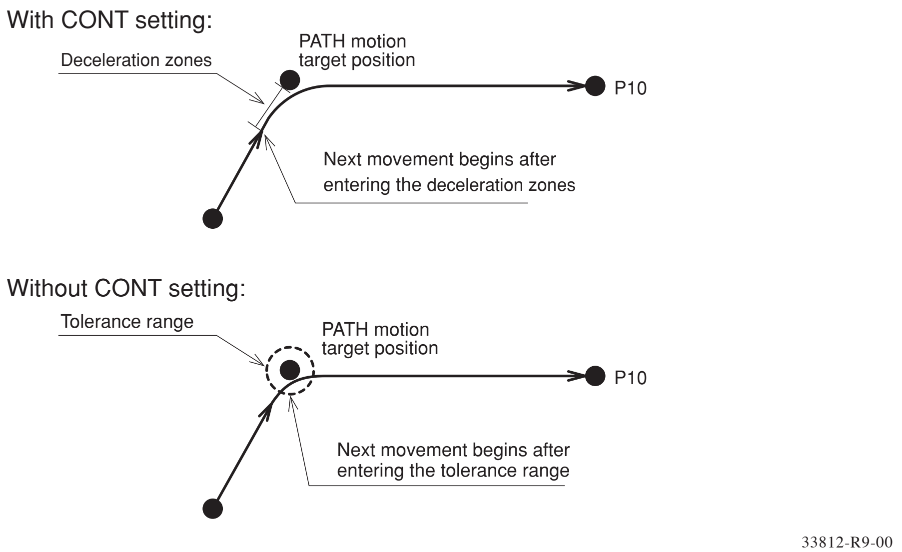
33808-R9-00

80 PATH START

SAMPLE

```
PATH START,CONT
MOVE P,P10
          ....................PATH motion starts, and movement to
                              P10 begins after the moving axes enter
                              the deceleration zone of final PATH
                              motion.
```

SAMPLE:PATH START, CONT



Related commands	PATH, PATH SET, PATH END

Reference For PATH function details, see Chapter 9 "PATH Statements".

81 PDEF

Defines the pallet

Format

```
PDEF(Pallet definition number) =expression 1, expression 2
, expression 3, point definition
```

Values

Pallet definition number	0 to 39
expression 1	Number of elements (NX) between P[1] and P[2].
expression 2	Number of elements (NY) between P[1] and P[3].
expression 3	Number of elements (NZ) between P[1] and P[5]. Total number of elements: must be 32767 or less <expression 1> × <expression 2> × <expression 3> P[1] to P[5] definition: see the figure below.
point definition..................	The point used for a pallet definition. Continuous 5 points starting with the specified point are used.

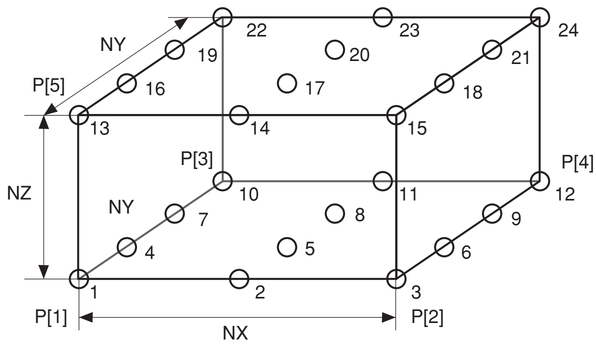
Explanation Defines the pallets to permit execution of the pallet movement command: changes the contents of definition for previously defined pallet data.

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 value becomes "1".

The total number of elemnts defined for a single pallet must not exceed 32,767.

Automatic point calculation



33815-R7-00

SAMPLE

```
PDEF 1 =3,4,2,P3991 ············· Pallet definition 1 is defined as 3 x 4
                                   x 2 by using P3991 to P3995.
```

82 PGMTSK

Acquires the task number in which a specified program is registered

Format

```
PGMTSK (program number)
```

Values *program number*....................1 to 100

Explanation Acquires the task number in which the program specified by <*program number*> is registered.

MEMO

- If the program number which is not registered in the task is specified, "3.203: Program doesn't exist" error occurs

SAMPLE

```
A = PGMTSK(1)                  Assigns the task number in which
                               the program number 1's program is
                               registered to variable A.
```

Related commands	PGN, TSKPGM

83 PGN

Acquires the program number from a specified program name

Format

```
PGN ("program name")
```

Values *program name*........................32 characters or less consisting of alphanumeric characters and underscore (_)

Explanation Acquires the program number of the program specified by <*program name*>.
The program name must be enclosed in double quotation marks (").

SAMPLE

```
A = PGN("PG_SUB") ··············· The program number of PG_SUB is
                                  assigned to variable A.
```

Related commands	PGMTSK, TSKPGM

84 PMOVE

Executes a pallet movement command for the robot

Format

```
PMOVE [robot number] (pallet definition number,
pallet position number),option, option...
```

Values

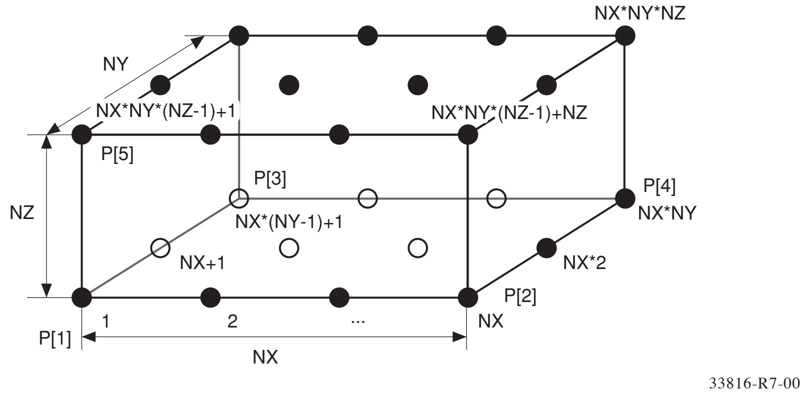
robot number.........................1 to 4 (If not input, robot 1 is specified.)
pallet definition number.........0 to 39
pallet position number...........1 to 32767

Explanation Executes "pallet move" command of the specified axes. (The specified pallet numbers must be registered in advance.)
It is not enabled for axes of other robots or for 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



MEMO

- Acquires the XYZ 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) ..................... Robot 1 moves from its current position
                                  to the position specified by pallet
                                  position number 16 of pallet definition
                                  number 1.
```

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:

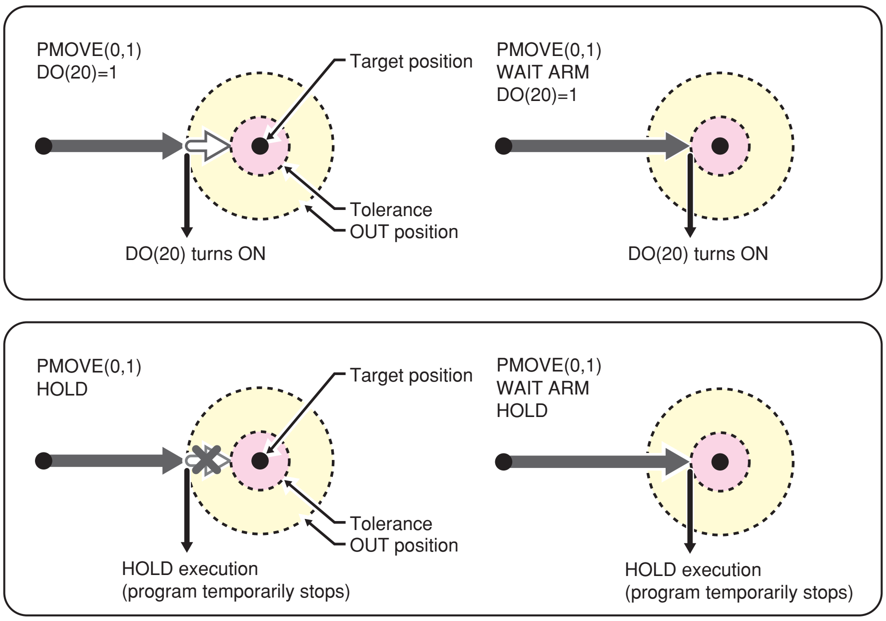
If the next command following the PMOVE command is an executable command such as a signal output command, that next command will start when the movement axis enters the OUT position range. In other words, that next command starts before the axis arrives within the target position OUT position range.

Example:

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

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 movement speed is the automatic movement speed multiplied by the program movement speed.
This option is enabled only for the specified PMOVE statement.

SAMPLE

```
PMOVE(1,3),S=10            Robot 1 moves from its current position
                           to the position specified by pallet
                           position number 3 of pallet definition
                           number 1, at 10% of the program speed.
```

● 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.

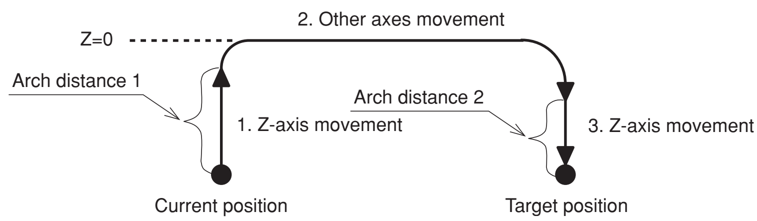
Explanation

1. The "x" specified axis begins moving toward the position specified by the <*expression*> ("1" shown in the figure 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.

SAMPLE

```
PMOVE(1,A),Z=0             First the Z-axis of robot 1 moves from the current
                           position to the "0 pulse" position. Then the other
                           axes of robot 1 move to the position specified by
                           pallet position number A of pallet definition number
                           1. Finally the Z-axis of robot 1 moves to the
                           position specified by pallet position number A.
```

SAMPLE: PMOVE Z



33704-R9-00

● 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
           ....................Robot 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.

85 Pn

Defines points within a program

Format

```
LET Pn = p1 p2 p3 p4 p5 p6 f
```

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.

Explanation Defines the point data.

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 "p1" through "p6", **the movement units are recognized as "mm".**
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

NOTE

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

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.

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 "9.702: Point data destroyed" may occur.

SAMPLE

```
P1  =         0         0         0         0         0         0
P2  =   100.000   200.000    50.000     0.000     0.000     0.000
P3  =    10.000     0.000     0.000     0.000     0.000     0.000
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)

86 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

87 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 is displayed on 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, a line-feed does not occur. 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.
- On the programming box, the PRINT statement is displayed on "Message" space in "Automatic Operation (ALL TASK) screen.

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

88 PSHFRC

Specifies/acquires the pushing force parameter

Format

```
1. PSHFRC [robot number] expression
2. PSHFRC [robot number] (axis number) =expression
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression..............................-1000 to 1000 (unit: %)

Explanation Changes the "push force" parameter of the specified axis to the value of <*expression*>. 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 <*expression*> / 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 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the value of "push force" parameter of the specified axis.

SAMPLE

```
A=PSHFRC (1) .................... The pushing thrust parameter of axis 1
                                  of robot 1 is assigned to variable A.
```

89

PSHJGSP

Specifies/acquires the push judge speed parameter

Format

```
1. PSHJGSP [robot number] expression
2. PSHJGSP [robot number] (axis number) =expression
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression..............................0: Invalid, 1 to 100 (units: %)

Explanation Changes the "push judge speed" parameter of the specified axis to the value of the <*expression*>.

If the push judge speed 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 push judge speed  parameter
                                   of axis 1 of robot 1 to 50%.
```

Functions

Format

```
PSHJGSP [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the value of "push judge speed" parameter of the axis specified by <*axis number*>.

SAMPLE

```
A=PSHJGSP (1) ··················· The pushing detection speed threshold
                                   parameter of axis 1 of robot 1 is
                                   assigned to variable A.
```

8

N O P Q R S T U V W X Y Z

90 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 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................0: Totalizing method, 1: Resetting method

Explanation Changes the "push method" parameter of the specified axis to the value of the <*expression*>.

The pushing type in the PUSH statement can be specified as follows by the <*expression*>.

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 <*axis number*>.

SAMPLE

```
PSHMTD (1) = 1 ·················· Changes the push method parameter of
                                  axis 1 of robot 1 to the resetting
                                  method.
```

Functions

Format

```
PSHMTD  [robot number]  (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the value of "push method" parameter of the axis specified by <*axis number*>.

SAMPLE

```
A=PSHMTD (1) ···················· The pushing method parameter of axis 1
                                  of robot 1 is assigned to variable A.
```

91

PSHRSLT

Acquires the status when PUSH statement ends

Format

```
PSHRSLT [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the end status of PUSH statement executed for the axis specified by <*axis number*>.

0 ....The PUSH statement was ended 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
```

92 PSHSPD

Specifies/acquires the push speed parameter

Format

```
1. PSHSPD [robot number] expression
2. PSHSPD [robot number] (axis number) =expression
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................1 to 100 (units: %)

Explanation Changes the "push speed" parameter of the axis specified by <*robot number*> to the value indicated in <*expression*>.

The motion speed in the PUSH statement is as follows.

- Neither "S" nor "DS" is set as an option in the PUSH statement:
  Maximum speed of a robot (mm/sec. or deg./sec.) x Push speed ratio (%) x Automatic movement speed (%) x Program movement speed (%)

- "S" is set as an option in the PUSH statement:
  Maximum speed of a robot (mm/sec. or deg./sec.) x Push speed ratio (%) x Automatic movement speed (%) x Program movement speed specified by S (%)

- "DS" is set as an option in the PUSH statement:
  Maximum speed of a robot (mm/sec. or deg./sec.) x Push speed ratio (%) x Movement speed of an axis specified by DS (%)

* Refer to ("94 PUSH" in this Chapter/ the YRCX programming manual) for details regarding the option settings of the PUSH statement.

SAMPLE

```
PSHSPD (1) = 50 ················· Changes the push  speed parameter of
                                  axis 1 of robot 1 to 50%.
```

Functions

Format

```
PSHSPD [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the "push speed" parameter value of the axis specified by <*axis number*>.

SAMPLE

```
A=PSHSPD (1) ···················· The push  speed parameter of axis 1 of
                                  robot 1 is assigned to variable A.
```

93 PSHTIME

Specifies/acquires the push time parameter

Format

```
1. PSHTIME [robot number] expression
2. PSHTIME [robot number] (axis number) =expression
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................1 to 32767 (unit: ms)

Explanation Changes the "push time" parameter of the specified axis to the value indicated in <*expression*>.
If the TIM option is omitted in the PUSH statement, the pushing control is executed with the setting of the push 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 push time parameter of
                                  axis 1 of robot 1 to 1000ms
```

Functions

Format

```
PSHTIME [robot number] (axis number)
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the value of "push time" parameter of the axis specified by the <*axis number*>.

SAMPLE

```
A=PSHTIME (1) ··················· The push time parameter of axis 1 of
                                  robot 1 is assigned to variable A.
```

94 PUSH

Executes a pushing operation for specified axes

Format

```
PUSH [robot number](axis number, expression), option, option
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6
expression.............................Motor position (mm, degree, pulse) or point expression

Explanation Executes an absolute position movement of the specified axis with controlling the pushing thrust in the forwarding direction.

- 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 (to cancel the torque value defined by PUSH statement) are as follows.

- When PUSH command finishes and then either of movement commands below is executed. Note that only finishing PUSH command is not enough to cancel the pushing thrust
  1. In push status at PUSH command finish
     When the target axis starts to move in the opposite direction of that specified by PUSH statement
  2. The target axis moves to the target position at PUSH command finish (not in push status)
     When the target axis starts to move either direction
- When a servo off occurs
- When the power source to the controller is interrupted and restartedd

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 the axis enters the OUT position range. Therefore, the 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, the axis movement also stops.
HOLD	Program temporarily stops when the axis enters the OUT position range. Therefore, the 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, the 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 of robot 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](2,90.000) ·············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 *<expression>*. 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.)
```

94 PUSH

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 *<expression>* 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),TIM=5000 ··········Axis 1 of robot 1 moves from its
                                 current position to the 100000 pulse
                                 position with keeping pushing for 5
                                 seconds.
```

● Speed setting

Format

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

Values *expression*.............................1 to 100 (units: %)

Explanation The program movement speed is specified in <expression>.
This option is enabled only for the specified PUSH statement.
The actual speed is determined as shown below.

- Max. speed of a robot (mm/s or deg./s) x Push speed (%) x automatic. movement speed (%) x <expression> (%)

SAMPLE

```
PUSH(1,10000),S=10 ············Axis 1 of robot 1 moves from its
                               current position to the 100000 pulse
                               position with the speed at 10% of the
                               multiplication of the push 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>.
This option is enabled only for the specified PUSH statement.
Movement always occurs at the DSPEED <expression> value (%) without being affected by automatic movement speed value (%).
The actual speed is determined as shown below.

- Max. speed of a robot (mm/s or deg./s) x Push speed (%) x <expression> (%)

SAMPLE

```
PUSH(1,10000),DS=0.1 ··········Axis 1 moves of robot 1 from its
                               current position to the 100000 pulse
                               position with the speed at 0.1% of the
                               pushing movement speed.
```

94 PUSH

● 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 of robot 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, PSHSPD, PSHRSLT, CURTRQ, CURTQST

95 RADDEG

Performs a unit conversion (radians → degrees)

Format

```
RADDEG(expression)
```

Values *expression*..............................Angle (units: radians)

Explanation Converts the <*expression*> value to degrees.

SAMPLE

```
LOC4(P0)=RADDEG(ATN(B)) ········· Converts the variable B arctangent
                                  value to degrees, and assigns it to
                                  4th-axis data of P0.
```

Related commands	ATN, COS, DEGRAD, SIN, TAN

96 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. REM or an apostrophe (') can be entered at any point in the line.

SAMPLE

```
REM *** MAIN PROGRAM ***
    '*** SUBROUTINE ***
HALT   'HALT COMMAND
```

97 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 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation

Format 1: Turns the bits of specified ports OFF.
Format 2: Clears the 1ms counter variables (1ms counter variables are used to measure the time in 1ms units).

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".

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 1ms counter variables.
```

Related commands	SET, DO, MO, SO, TO, LO

98 RESTART

Restarts another task during a temporary stop

Format

```
RESTART   Tn
          <program name>
          PGm
```

Values

n: Task number ....1 to 16
m: Program number ....1 to 100

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.
The program name must be enclosed in < > (angle brackets).

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
'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.

99 RESUME

Resumes program execution after error recovery processing

Format

```
1. RESUME
2. RESUME NEXT
3. RESUME label
```

REFERENCE

• For details, refer to "67 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 "17.800 : Motor overload".

Related commands	ON ERROR GOTO

100 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 "5.212: Stack overflow".

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

101 RIGHT$

Extracts a character string from the right end of another character string

Format

```
RIGHT$(character string expression, expression)
```

Values *expression*..............................0 to 255

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 255, otherwise an error will occur.
If the <*expression*> value is 0, then extracted character string will be a null string (empty character string).
If the <*expression*> value has more characters than the <*character string expression*>, extracted character string 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$, MID$

102 RIGHTY

Sets the SCARA robot hand system as a right-handed system

Format

```
RIGHTY [robot number]
```

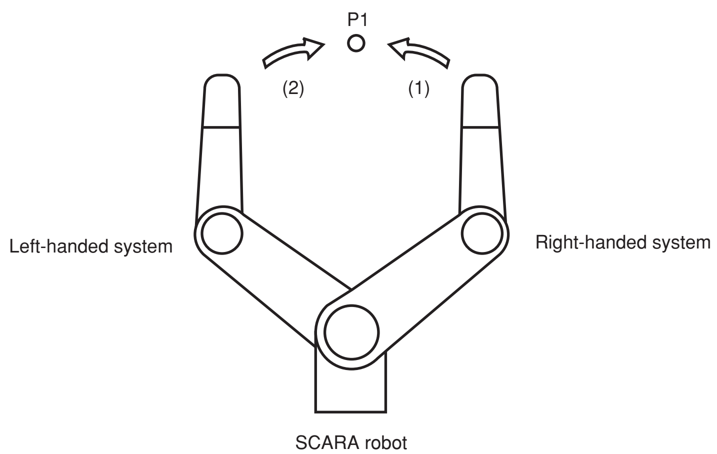
Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Specifies the robot as a roght-handed system. The robot moves to a point specified in the Cartesian coordinates.
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).

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

103 RSHIFT

Shifts a bit value to the right

N O P Q R S T U V W X Y Z

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

104 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
```

105 SEND

Sends readout file data to the write file

Format

```
SEND read-out 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-out 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 10 "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-out/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-out	Write
			All	Individual File		
User	All file		ALL	———	✓	✓
	Program		PGM	<bbbbbbbb> PGn	✓	✓
	Point		PNT	Pn	✓	✓
	Point comment		PCM	PCn	✓	✓
	Point name		PNM	PNn	✓	✓
	Parameter		PRM	/cccccccc/ #cccccccc# \cccccccc\	✓	✓
	Shift definition		SFT	Sn	✓	✓
	Hand definition		HND	Hn	✓	✓
	Pallet definition		PLT	PLn	✓	✓
	General 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	<<bbbbbbbb>>	✓	–
	Parameter directory		DPM	———	✓	–
	Machine reference	sensor, stroke-end	MRF	———	✓	–
		mark	ARP	———	✓	–
	System configuration information		CFG	———	✓	–
	Version information		VER	———	✓	–
	Option board		OPT	———	✓	–
	Self check		SCK	———	✓	–
	Alarm history		LOG	———	✓	–
	Remaining memory size		MEM	———	✓	–

Type	File Name	Definition Format		Read-out	Write
		All	Individual File		
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 (_)
c: Alphanumeric character or special symbol x: Expression (array argument) y: Variable type
i: Input/output type

✓: Permitted –: 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 10 "Data file description".

Related commands OPEN, CLOSE, SETGEP, GEPSTS

106 SERVO

Controls the servo status

Format

```
SERVO [robot number]  ON     (axis number)
                      OFF
                      FREE
```

CAUTION

- Always check that the Emergency Stop is ON and Servo is OFF when working within the robot movement range.
- Electromagnetic brake is the brake to prevent the vertical axis from sliding downward. The vertical axis will slide downward when the servo is FREE, causing a hazardous situation.

Values

robot number ....1 to 4 (If not input, robot 1 is specified.)
axis number...........................1 to 6 (• Multiple axes not specifiable
• If not input, all axes are specified.)

Explanation This command controls the servo ON/OFF at the specified axes or all axes.

SERVO command	SERVO	Motor power	Dynamic brake	Electromagnetic brake
ON	ON	OFF	OFF	OFF
OFF	OFF	OFF (In the case of all axes servo OFF)	ON	ON
FREE	OFF	Continues the previous status	OFF	OFF

MEMO

- This command is executed after the operation of all axes of the specified robot has been complete (after positioned within the tolerance).
- The motor power is a power supply unit for robot (motor) in the controller.
- The dynamic brake controls the motor by using the electric power which is generated in the motor when the servo is turned OFF.

SAMPLE

```
SERVO ON    ..................... Turns servos ON at all axes
                                  of robot 1.
SERVO OFF   ..................... Turns the servo OFF and applies
                                  the dynamic brake at all axes
                                  of robot 1. Axes equipped with brakes are
                                  all locked by the brake.
SERVO FREE(3) ................... Turns servos OFF at axis 3
                                  of robot 1, and releases the brake.
```

107 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 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).
time......................................10 to 3600000 (units: ms)

Explanation

Turns ON the bits of specified ports.
The pulse output time (unit: ms) is specified by the <*time*> value.
The program execution is WAIT status while the output is ON. When the specified time elapses, the output is turned OFF, and the execution ends.
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

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".

108 SETGEP

Sets the General Ethernet Port

Format

```
SETGEP m, n, "IP adress", ppppp, e, t
```

Values

m: General Ethernet Port number ....0 to 7
n: mode ....0: server, 1: client
IP adress...0.0.0.0 to 255.255.255.255
ppppp: port number ....0 to 65535
e: Termination code.............................0: CRLF, 1: CR
t: port type ....0: TCP

Explanation Sets the specified General Ethernet Port. The General Ethernet Port can open/ close the communication port by OPEN/ CLOSE commands.

<*IP adress*> must be enclosed in " " (double quotation marks).

When "0: server" is selected at "n: mode", although <*IP adress*> can be omitted, " " (double quotation marks) must be written.

MEMO

When Server mode is selected,

- IP address: IP address already set on the controller is used to communicate, so IP address setting is unnecessary. (The IP address set by <*IP address*> is invalid in this case.)
- Port number: Set a port number which differs from the one on the controller.

When Client mode is selected,

- IP address and port number: Set the IP address and port number of the connection destination server.

SAMPLE

```
IPADRS$="192.168.0.100" ········· Assigns the IP adress(192.168.0.100) of the
                                  server to connect to variable IPADRS$.
SETGEP 1, 1, IPADRS$, 100, 0, 0
                ····················· Sets the conditions below on General
                                  Ethernet Port 1.
              ■ mode: client
              ■ the IP adress of the server to connect to: 192.168.0.100
              ■ the port number of the server to connect to: 100
              ■ Termination code : CRLF

OPEN GP1    ····················· Connects the server specified at
                                  General Ethernet Port 1.
SEND "123" TO GP1 ··············· Sends the character strings "123" from
                                  General Ethernet Port 1.
CLOSE GP1   ····················· Disconnects from the server specified
                                  at General Ethernet Port 1.
```

Related commands	OPEN, CLOSE, SEND, GEPSTS

109 SGI

Assigns /acquires the value to a specified integer type static variable

Format

```
SGIn=xxxxxx
```

Values n: integer type static variable number..... 0 to 31
xxxxxx .... integer of -2147483648 to 2147483647

Explanation Assigns xxxxxx to the integer type static variable (SGI) specified by "n". If a real number with decimal point is specified at xxxxxx, assigns a value with decimal fractions truncated.

SAMPLE

```
SGI1=300    ..................... Assigns 300 to SGI1.
```

Functions

Format

```
SGIn
```

Values n: integer type static variable number..... 0 to 31

Explanation Acquires the value of the integer type static variable (SGI) specified by "n".

SAMPLE

```
A%=SGI1     ..................... Assigns the value of SGI1 to variable A%.
```

Related commands	SGR

110 SGR

Assigns /acquires the value to a specified real type static variable

Format

```
SGRn=xxxxxx
```

Values

n: real type static variable number.... 0 to 31

xxxxxx... 1. Single-precision real numbers
-999999.9 to +999999.9
- 7 digits including integers and decimals. (For example, ".0000001" may be used.)

2. Single-precision real numbers in exponent form
-1.0×10^{38} to $+1.0\times10^{38}$
- Mantissas should be 7 digits or less, including integers and decimals.

Explanation Assigns xxxxxx to the real type static variable (SGR) specified by "n".

SAMPLE

```
SGR1=1320.355 ··················· Assigns 1320.355 to SGR1.
```

Functions

Format

```
SGRn
```

Values n: real type static variable number .... 0 to 31

Explanation Acquires the value of the real type static variable (SGR) specified by "n".

SAMPLE

```
A!=SGR1     ····················· Assigns the value of SGR1 to variable A!.
```

Related commands	SGI

111 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 dummy argument.
- 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

112 SHIFT

Sets the shift coordinates

Format

```
SHIFT [robot number]   shift variable
                       OFF
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Sets the shift coordinates specified by <*shift variable*> to the robot specified by <*robot number*>.
When OFF is specified, the coordinates shift by <*shift variable*> does not occur.

MEMO

- This statement is executed after axis positioning is complete (within the tolerance range).
- When OFF is specified, it is the same as the setting: 0.000 at each X, Y, Z and rotation direction-offset by the <*shift variable*>.

SAMPLE

```
SHIFT S1     ..................... Shifts the coordinate of robot 1 to
                                   the "shift 1" coordinate.
MOVE P,P10
SHIFT S[A]   ..................... Shifts the coordinate of robot 1 to
                                   the coordinate specified by variable A.
MOVE P,P20
HALT
```

Related commands	Shift definition statement, shift assignment statement

113 SI

Acquires specified SI status

Format

```
LET expression = SIm(b,···,b)
LET expression = SI(mb,···,mb)
```

Values

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

b: bit definition...................... 0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Acquires SI port input status specified by "m".

SAMPLE

```
A%=SI2()    ..................... The input status from SI (27) to SI (20)
                                  is assigned to variable A%.
A%=SI0(6,5,1)  .................. The SI (06), SI (05), SI (01) input
                                  status is assigned to variable A% (when
                                  all the above signals are "1" (ON), A% = 7).
A%=SI(37,35,27,10) .............. The SI (37), SI (35), SI (27) SI(10) input status
                                  is assigned to variable A% (when all the above
                                  signals except SI (27) are "1" (ON), A% = 13).
```

114

SID

Acquires a specified serial input's double-word information

Format

```
LET SID(m)
```

Values m: port number......................2, 4, 6, 8, 10, 12, 14

Explanation Acquires the value at the SID port specified by "m".
The acquisition range is -2,147,483,648 (&H80000000) to 2,147,483,647 (&H7FFFFFFF).

MEMO

- The information is handled as signed double word data.
- "0" is input if the specified port does not exist.
- The lower port number data is placed at the lower address.
  For example, if SIW(2) =&H2345,SIW(3) =&H0001, then SID(2) =&H00012345.

SAMPLE

```
A%=SID(2)   ..................... The input status of SIW(2), SIW(3) is
                                  assigned to variable A%.
A%=SID(14)  ..................... The input status of SIW(14), SIW(15)
                                  is assigned to variable A%.
```

Related commands	SIW

115 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

116 SIW

Acquires a specified serial input's word information

Format

```
LET SIW(m)
```

Values m: port number......................2 to 15

Explanation Acquires the value at the SIW port specified by "m".
The acquisition range is 0 (&H0000) to 65535 (&HFFFF).

MEMO

- The information is handled as unsigned word data
- "0" is input if the specified port does not exist.

SAMPLE

```
A%=SIW(2)   ..................... The input status of SIW (2) is assigned
                                  to variable A%.
A%=SIW(15)  ..................... The input status of SIW (15) is
                                  assigned to variable A%.
```

Related commands	SID

117 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 "9.706: Shift data destroyed" 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  =     0.000    0.000    0.000    0.000
S1  =   100.000  200.000   50.000   90.000
P3  =   100.000    0.000    0.000    0.000    0.000    0.000
SHIFT S0
MOVE P,P3
SHIFT S1
MOVE P,P3
HALT
```

Related commands	Shift assignment statement, SHIFT

118 SO

Outputs a specified value to serial port or acquires the output status

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 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

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 the port which does not exist is specified, nothing is output.

CAUTION

- Outputs to SO0() and SO1() are not possible.

REFERENCE

- For bit setting details, refer to Chapter 3 "10 Bit Settings".

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

118 SO

Functions

Format

```
LET SOm (b,･･･,b)
LET SO (mb,･･･,mb)
```

Values m: port number ....0 to 7, 10 to 17, 20 to 27
b: bit definition.....................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Indicates SO port output status.

SAMPLE

```
A%= SO2()  ····················· Output status of ports SO(27) to
                                 SO(20) is assigned to variable A%.
A%= SO0(6, 5, 1) ··············· Output status of SO(06), SO(05) and
                                 SO(01) is assigned to variable A%.
                                 (If all above signals are 1(ON), then
                                 A%=7.)
A%= SO(37,35,27,10) ············ Output status of SO(37),
                                 SO(35) , SO(27) and SO(10)
                                 is assigned to variable A%.
                                 (If all above signals except SO(27)
                                 are 1 (ON), then A%=13.)
```

Related commands	RESET, SET

119 SOD

Outputs a specified serial output's double-word information or acquires the output status

Format

```
LET SOD(m)=expression
```

Values m: port number......................2, 4, 6, 8, 10, 12, 14

Explanation Outputs the value to the SOD port specified by "m".
The output range is -2,147,483,648 (&H80000000) to 2,147,483,647 (&H7FFFFFFF).

MEMO

- The information is handled as signed double word data.
- If a serial port does not actually exist, the information is not output externally
- The lower port number data is placed at the lower address.
  For example, if SOW(2) =&H2345,SOW(3) =&H0001, then SOD(2) =&H00012345.

SAMPLE

```
SOD(2)=&H12345678  ............... Outputs &H12345678 to SOD(2).
SOD(4)=1048575   .................. Outputs 1048575(&HFFFFF) to SOD(4).
SOD(2)=A%   ...................... Outputs the value of variable A% to
                                   SOD(2).
```

Functions

Format

```
LET SOD(m)
```

Values m: port number......................2, 4, 6, 8, 10, 12, 14

Explanation Acquires the SOD port output status specified by "m".

SAMPLE

```
A%=SOD(2)   ...................... The output status of SOD(2) is assigned
                                   to variable A%.
```

Related commands	SOW

120 SOW

Outputs a specified serial output's word information or acquires the output status

Format

```
LET SOW(m)=expression
```

Values m: port number ....2 to 15

Explanation Outputs the value to the SOW port specified by "m".
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.
Example: SOW(2)=-255
The contents of -255 (&HFFFFFF01) are assigned to SOW (2).
-255 is a negative value, so the low-order word information (&HFF01) is assigned.

MEMO

- The information is handled as unsigned word data.
- If a serial port 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.

SAMPLE

```
SOW(2)=&H0001  .................. Outputs &H0001 to SOW(2).
SOW(3)=255  .................... Outputs 255(&H00FF) to SOW(3).
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.
```

Functions

Format

```
LET SOW(m)
```

Values m: port number ....2 to 15

Explanation Acquires the SOW port output status specified by "m".

SAMPLE

```
A%=SOW(2)  ..................... The output status of SOW (2) is
                                 assigned to variable A%.
```

Related commands	SOW

121 SPEED

Changes the program movement speed

Format

```
SPEED [robot number] expression
```

Values

robot number.........................1 to 4 (If not input, robot 1 is specified.)
expression.............................1 to 100 (units: %)

Explanation Changes the program movement speed to the value indicated by <*expression*>.
This speed change applies to all 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).

Operation speed = automatic movement speed × 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.

SAMPLE

```
ASPEED 100   ...................... Changes the Automatic movement speed
                                    of robot 1 to 100%
SPEED 70     ...................... Changes the Program movement speed of
                                    robot 1 to 70%
MOVE P,P0    ...................... Moves robot 1 from current position to
                                    P0 at a speed of 70% (=100 × 70).
SPEED 50     ...................... Changes the Program movement speed of
                                    robot 1 to 50%
MOVE P, P1   ...................... Moves robot 1 from current position to
                                    P1 at a speed of 50% (=100 × 50).
MOVE P,P2, S=10 ................... Moves robot 1 from current position to
                                    P2 at a speed of 10% (=100 × 10).
HALT
```

Related commands	ASPEED

122 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.
```

123 START

Starts a new task

Format

```
START  <program name>  ,Tn, p
       PGm
```

Values

m: Program number ....1 to 100
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 RUNNING status occurs at a task with higher priority, all tasks with lower priority also remain in READY status.
The program name must be enclosed in < > (angle brackets).

SAMPLE

```
START <SUB_PGM>,T2,33
*ST:
    MOVE P,P0,P1
GOTO *ST
HALT

Program name:SUB_PGM
'SUBTASK ROUTINE
*SUBTASK:
    P100 = WHERE
    IF LOCZ(P100) > 10000 THEN
        DO(20) = 1
    ELSE
        DO(20) = 0
    ENDIF
GOTO *SUBTASK
EXIT TASK
```

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

N O P Q R S T U V W X Y Z

124 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 value.

SAMPLE

```
B$=STR$(10.01)
```

Related commands	VAL

125 SUB to END SUB

Defines a sub-procedure

8

N O P Q R S T U V W X Y Z

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".

125 SUB to END SUB

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

126

SUSPEND

Temporarily stops another task which is being executed

8

N O P Q R S T U V W X Y Z

Format

```
SUSPEND  Tn
         <program name>
         PGm
```

Values n: Task number ....1 to 16
m: Program number ....1 to 100

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.
The program name must be enclosed in < > (angle brackets).

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
'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

127 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 must be enclosed in < > (angle brackets).

MEMO

- If the program specified as the switching target does not exist, a "3.203: Program doesn't exist" (code: &H0003 &H00CB) error occurs and operation stops.

SAMPLE

```
SWI <ABC>   ..................... Switches the execution program to
                                  "ABC".
```

128 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

129 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 2,147,483,647, 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
                                 until the axis enters the tolerance
                                 range on the programming box.
```

Related commands	RESET

130 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

131 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 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$

132 TO

Outputs a specified value to the TO port or acquires the output status

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 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

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.
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
```

Functions

Format

```
LET TOm (b,···,b)
LET TO (mb,···,mb)
```

Values
m: port number 0, 1
b: bit definition.....................0 to 7 (If omitted, all 8 bits are processed.)
If multiple bits are specified, they are expressed from the left in descending order (high to low).

Explanation Indicates the parallel port signal status.

SAMPLE

```
A%= TO0()   ..................... Output status of ports TO(07) to
                                  TO(00) is assigned to variable A%.
A%= TO0(6, 5, 1) ................ Output status of TO(06), TO(05) and
                                  TO(01) is assigned to variable A%.
                                  (If all above signals are 1(ON), then A%=7.)
A%=TO(17, 15, 00) ............... Output status of TO(17), TO(15) and
                                  TO(00) is assigned to variable A%.
                                  (If all above signals except TO(15)
                                  are 1 (ON), then A%=5.)
```

Related commands	RESET, SET

133 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 (If not input, robot 1 is specified.)
axis number................1 to 6
expression..................Varies according to the motor which has been specified (unit: pulse)

Explanation Change the "tolerance" parameter of the specified axis to the <*expression*> value (unit: pulse).
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
- 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 (If not input, robot 1 is specified.)
axis number...........................1 to 6

Explanation Acquires the "tolerance" parameter values for the axis specified by <*axis number*>.

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
```

134 TORQUE

Specifies/acquires the maximum torque command value

Format

```
TORQUE [robot number] (axis number) =expression
```

Values *robot number* ....1 to 4 (If not input, robot 1 is specified.)
axis number ....1 to 6
expression.............................1 to 100 (units: %)

Explanation Changes the maximum torque command value of the specified axis to the <*expression*> value. The new value is valid when the next movement command (MOVE or DRIVE statement, etc.) is executed. The 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 the parameter is initialized.
- The servo is turned off.

The maximum torque command value becomes temporarily invalid in execution below.

- Return- to-origin is in execution.
- The PUSH statement is in execution.

(only the torque value in the moving direction is changed to the value specified by the PUSH statement, the value in the opposite direction is hold and not changed.)

After these movements, the value backs to the maximum torque command value when a next movement command (MOVE statement, for example) is executed.

CAUTION

- If the specified torque limit is too small, the axis may not move. Never enter within the robot movement range to avoid danger even though the robot is in stop status. 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

- The TORQUE statement limits the torque in the both (rotation and opposite) direction of axis, whereas the PUSH statement limits the torque in its rotation direction only.

Functions

Format

```
TORQUE [robot number] (axis number)
```

Values *robot number* ....1 to 4 (If not input, robot 1 is specified.)
axis number ....1 to 6

Explanation Acquires the maximum torque command value for the axis specified by <*axis number*>.

134 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.
                                  (Changes 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

135 TSKPGM

Acquires the program number which is registered in a specified task number

Format

```
TSKPGM(task number)
```

Values *task number* ....Task number which acquires the program number

Explanation Acquires the program number which is registered in the task specified by the task number.

SAMPLE

```
A=TSKPGM(1) ····················· Assignes a program number registered
                                  in task 1 to variable A.
```

Related commands	PGMTSK, PGN

136 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.

Hexadecimal number..................... &Hnnnn
Decimal number .... nnnn
Binary number .... &Bnnnn

Decimal point .... nnn.nnn
Exponents...................................... nnEmm

SAMPLE

```
A=VAL("&B100001")
```

137 WAIT

Waits until the conditional expression is met

Format

```
WAIT conditional expression , expression
```

Values *expression*..............................0 to 2147483647 (units: ms)

Explanation Establishes "wait" status until the condition specified by the <*conditional expression*> is met. Specify the time-out period (unit: ms) in the <*expression*>.
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 TRUE status, and "0" indicates FALSE status.

SAMPLE

```
WAIT A=10  ·····················  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 ·······  Wait status continues until either DI
                                  (31) or DO(21) turns ON.
WAIT DI(20)=1,1000 ··············  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, MOVET

138 WAIT ARM

Waits until the robot axis operation is completed

Format

```
WAIT ARM [robot number] (axis number)
```

Values

robot number .... 1 to 4 (If not input, robot 1 is specified.)
axis number........................... 1 to 6 (• Multiple axes not specifiable
• If not input, all axes are specified.)

Explanation Establishes "wait" status until the axis movement is completed (is positioned within the tolerance range).

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, MOVET

139 WEIGHT

Specifies/acquires the tip weight (kg) parameter

Format

```
WEIGHT [robot number] expression
```

Values

robot number........................1 to 4 (If not input, robot 1 is specified.)

expression.............................The range varies according to the robot which has been specified.

Explanation Changes the "tip weight" parameter of the robot to the <*expression*> value.
This change does not apply to auxiliary axes.

Functions

Format

```
WEIGHT [robot number]
```

Values

robot number........................1 to 4 (If not input, robot 1 is specified.)

Explanation Acquires the "tip weight" parameter value of the robot specified by <*robot number*>.

SAMPLE

```
A=5
B=2
C=WEIGHT    ..................... The tip weight parameter of robot 1 is
                                  assigned to variable C.
WEIGHT A    ..................... The tip weight parameter of robot 1 is
                                  changed to value (5) of variable A.
MOVE P,P0
WEIGHT B    ..................... The tip weight parameter of robot 1 is
                                  changed to value (2) of variable B.
MOVE P,P1
WEIGHT C    ..................... The tip weight parameter of robot 1
                                  is replaced to the origin value (the
                                  value of variable C).
D=WEIGHT    ..................... The tip weight parameter of robot 1 is
                                  assigned to variable D.
HALT
```

MEMO

- If both of Tip weight parameters; <WEIGHT> and <WEIGHTG> are set, a total value will be set.

Related commands	WEIGHTG

140 WEIGHTG

Specifies/acquires the tip weight (g) parameter

Format

```
WEIGHTG [robot number] expression
```

Values

robot number........................1 to 4 (If not input, robot 1 is specified.)
expression.............................The range varies according to the robot which has been specified.

Explanation Changes the "tip weight (g)" parameter of the robot to the <*expression*> value. This change does not apply to auxiliary axes.

Functions

Format

```
WEIGHTG [robot number]
```

Values

robot number........................1 to 4 (If not input, robot 1 is specified.)

Explanation Acquires the "tip weight (g)" parameter value of the robot specified by <*robot number*>.

SAMPLE

```
A=5
B=2
C=WEIGHTG    ..................... The tip weight (g) parameter of robot 1
                                   is assigned to variable C.
WEIGHTG A    ..................... The tip weight (G) parameter of robot 1
                                   is changed to value (5) of variable A.
MOVE P,P0
WEIGHTG B    ..................... The tip weight (g) parameter of robot 1
                                   is changed to value (2) of variable B.
MOVE P,P1
WEIGHTG C    ..................... The tip weight (g) parameter of robot 1
                                   is replaced to the origin value (the
                                   value of variable C).
D=WEIGHTG    ..................... The tip weight (g) parameter of
                                   robot 1 is assigned to variable D.
HALT
```

MEMO

- If both of Tip weight parameters; <WEIGHT> and <WEIGHTG> are set, a total value will be set.

Related commands	WEIGHT

141

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

142 WHERE

Acquires the arm's current position (pulse coordinates)

Format

```
WHERE [robot number]
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Acquires the arm's current position in the joint coordinates.

SAMPLE

```
P10=WHERE   ..................... The current position's pulse coordinate
                                  value of robot 1 is assigned to P10.
```

Related commands	WHRXY

143

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 TRUE status, and "0" indicates 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
```

144 WHRXY

Acquires the arm's current position in Cartesian coordinates

Format

```
WHRXY [robot number]
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation Acquires the arm’s current position in the Cartesian coordinates.

SAMPLE

```
P10=WHRXY   ..................... The current position Cartesian
                                  coordinate value of robot 1 is assigned
                                  to P10.
```

Related commands	WHERE

145 XYTOJ

Converts the Cartesian coordinate data ("mm") to joint coordinate data ("pulse")

Format

```
XYTOJ [robot number] (point expression)
```

Values *robot number*.........................1 to 4 (If not input, robot 1 is specified.)

Explanation This function converts the Cartesian coordinate data (unit: mm, deg.) specified by the <*point expression*> to the joint coordinate data (unit: pulse) of the robot specified by the <*robot number*>.

- 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.
- To convert joint coordinate data to Cartesian coordinate data, use the JTOXY statement.

SAMPLE

```
P10=XYTOJ(P10) ·················· P10 is converted to joint coordinate
                                  data of robot 1.
```

Chapter 9

PATH Statements

1 Overview

This function moves the robot at a specified speed along a path composed of linear and circular segments. Because speed fluctuations during movement are minimal, the PATH function is ideal for applications such as sealing, etc.

2 Features

- Moves the robot at a constant speed along the entire movement path (except during acceleration from a stop, and during deceleration just prior to the operation end).
- Permits easy point teaching because the robot speed is not affected by the point teaching positions' level of precision.
- Permits movement speed changes for the entire movement path, or speed changes for only one portion of the path (using the speed option).
- Using the DO option permits signal outputs to a specified port at any desired position during movement.

3 How to use

The following robot language commands must be used as a set in order to use the PATH function.

- PATH SET .... Starts path setting.
- PATH (PATH L, PATCH C) .... Specifies the path to be used.
- PATH END .... Ends path setting.
- PATH START .... Starts actual movement along the path.

As shown below, the motion path is specified between the PATH SET and PATH END statements. Simply specifying a path, however, does not begin robot motion.
Robot motion only occurs when the PATH START statement is executed after the path setting procedure has been completed.

SAMPLE

```
MOVE P,P0,Z=0
PATH SET   ····················· Start of robot 1's path setting
PATH L,P1,DO(20)=1@10.0
PATH L,P2
    .
    .
    .
PATH C,P12,P13
PATH L,P14,DO(20)=0@20.0
PATH END   ····················· End of robot 1's path setting
MOVE P,P1,Z=0
    .
    .
    .
MOVE P,P0,Z=0
PATH START ····················· Path motion of robot 1 is executed
HALT
```

4 Cautions when using this function

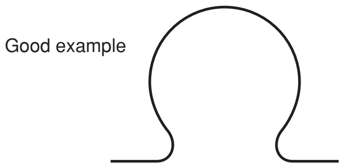
- Paths may comprise no more than 1000 points (total) linear and circular segments. 1 point forms 1 linear segment by PATH L command and 2 points form 1 circular segment by PATH C command.

$$\text{Number of points specified by PATH L} + \frac{\text{Number of points specified by PATH C}}{2} \leqq 1000$$

- The robot must be positioned at the path start point when PATH motion is executed (by PATH START statement).

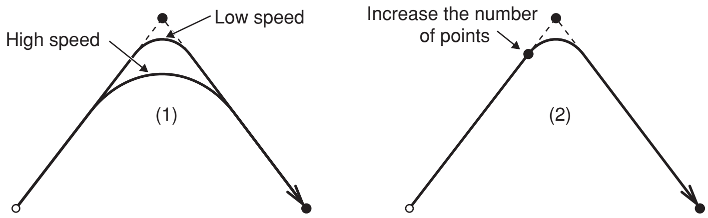
- At points where circular and linear segments connect, the motion direction of the two connecting segments should be a close match (as close as possible). An excessive difference in their motion directions could cause vibration and robot errors.

Circular and linear segment connection point:
if there is a large difference between the motion directions of the connecting segments



- Where a linear segment connects to another linear segment, the motion path passes to the inner side of the connection point. Moreover, as shown in fig. (1) below, the faster the speed, the further to the inner side the path becomes. To prevent significant speed-related path shifts, add more points as shown in fig. (2). Note also, that in some cases, the speed may have to be reduced in order to prevent errors from occurring.

Linear segments connection point: Prevents a deviation



- If an error occurs due the robot's inability to move at the specified speed:
Robot acceleration/deceleration occurs if the speed setting is changed when PATH motion begins, stops, or at some point along the path. At such times, an error may occur before motion begins if the distance between points is too short for the specified speed to be reached. In such cases, a slower speed must be specified. If the error still occurs after the speed is lowered, adjust the PATH points to increase the length of the linear or circular segments which contain acceleration or deceleration zones.

- The hand system used during PATH motion must be the same as the hand system used at the path's start point. The same applies if the path is to pass through points where hand flags are set. Differing hand systems will cause an error and disable motion.

- The first arm and second arm rotation information during PATH movement must be the same as the first arm and second arm rotation information at the PATH movement's START point. If the two are different, an error will occur and movement will be disabled.

- If the robot is stopped by a stop signal, etc., during PATH motion, this is interpreted as an execution termination, and the remaining path motion will not be completed even if a restart is executed.

Chapter 10
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 36 different types of data files.

Type	File Name		Definition Format: All	Definition Format: Individual File	Read-out	Write
User	All file		ALL	———	✓	✓
	Program		PGM	<bbbbbbbb> PGn	✓	✓
	Point		PNT	Pn	✓	✓
	Point comment		PCM	PCn	✓	✓
	Point name		PNM	PNn	✓	✓
	Parameter		PRM	/cccccccc/ #cccccccc# \cccccccc\	✓	✓
	Shift definition		SFT	Sn	✓	✓
	Hand definition		HND	Hn	✓	✓
	Pallet definition		PLT	PLn	✓	✓
	General 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	<<bbbbbbbb>>	✓	–
	Parameter directory		DPM	———	✓	–
	Machine reference	sensor, stroke-end	MRF	———	✓	–
		mark	ARP	———	✓	–
	System configuration information		CFG	———	✓	–
	Version information		VER	———	✓	–
	Option board		OPT	———	✓	–
	Self check		SCK	———	✓	–
	Alarm history		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 (_)

c: Alphanumeric character or special symbol　x: Expression (array argument)　y: Variable type

i: Input/output type

✓: Permitted　–: Not Permitted

1.2 Cautions

Observe the following cautions when handling data files.

- Only one-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 out" and "writing" used in this manual indicate the following data flow;
  Reading out: Controller → external communication device
  Writing: External communication device → controller

2 Program file

2.1 All programs

Read-out	✓	When used as a read-out file, all programs currently stored are read out.
Write	✓	Write files are registered at the controller under the program name indicated at the "NAME = *program name*" line.

Format

```
PGM
```

Meaning

- Expresses all programs.
- 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 older program is deleted.

DATA FORMAT

```
NAME = program name [cr/lf]
PGN=mmm[cr/lf]
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
        :
aaaaa ...aaaaaaaaaaaaaa[cr/lf]
        :
NAME = program name [cr/lf]
PGN=mmm[cr/lf]
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 the
                                  communication port.
Response:
RUN [cr/lf]
NAME=TEST[cr/lf]
PGN=1[cr/lf]
PGN=1
A=1[cr/lf]
RESET DO2()[cr/lf]
        :
HALT[cr/lf]
[cr/lf]
END [cr/lf]
```

2.2 One program

Read-out	✓
Write	✓

Format

```
1.<program name>
2.PGmmm
```

Meaning
- Expresses a specified program.
- "mmm" represents a number from 1 to 100.
- *Program names* are shown with 32 characters or less consisting of alphanumeric characters and _ (underscore), and must be enclosed in < > (angle brackets).
- If a program name is omitted and written as <> in format 1, the current program is specified.
- In the case of write file, an error occurs if the specified program name (<*program name*>) differs from one on the data (NAME=program name).

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 specify the program name after NAME=. Program writing cannot occur if the program name is not specified.
- When there is a program number with the different program, the older one will be overwritten.
- When there is no program number specified, the smallest free number will be specified automatically.
- 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 program TEST1 from the
                                  communication port.
Response:
RUN [cr/lf]
NAME=TEST1[cr/lf]
PGN=1[cr/lf]
A=1[cr/lf]
RESET DO2()[cr/lf]
       :
HALT[cr/lf]
[cr/lf]
END [cr/lf]
```

3 Point file

3.1 All points

Read-out	✓	When used as a read-out file, all points currently stored are read out.
Write	✓	When used as a write file, writing is performed with a point number.

Format

```
PNT
```

Meaning • Expresses all point data.

DATA FORMAT

```
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 number: 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 2:LEFT

- 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 is processed as "0" if a numeral other than 0, 1, -1 has been specified, or if no numeral has been specified.

3 Point file

- A line containing only [cr/lf] is added at the end of the file to indicate the end of the file.

SAMPLE

```
SEND PNT TO CMU ················· Outputs all points from the
                                  communication port.
Response:
RUN [cr/lf]
P0 = 1 2 3 4 5 6 [cr/lf]
P1 = 426.200 -160.770 0.001 337.210 0.000 0.000 0 1 0 [cr/lf]
P2 = -27.570 -377.840 0.360 193.220 0.000 0.000 0 -1 0 [cr/lf]
       :
P29999= -251.660 -419.510 0.000 -127.790 0.000 0.000 2 -1 -1 [cr/lf]
[cr/lf]
END [cr/lf]
```

3.2 One point

Read-out	✓
Write	✓

Format

```
Pmmmm
```

Meaning
- Expresses a specified point.
- "mmmm" represents a number from 0 to 29999.

DATA FORMAT

```
Pmmmm= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t xr yr [cr/lf]
```

▪

Values
mmmm.................. Point number: 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 2:LEFT

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 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, indicating the end of the file.

SAMPLE

```
SEND P100 TO CMU ················ Outputs the specified point from the
                                  communication port.
Response:
RUN [cr/lf]
P100= 1.000 2.000 3.000 4.000 5.000 6.000 0 1 0 [cr/lf]
END [cr/lf]
```

7
8
9
10
11
12
13

4 Point comment file

4.1 All point comments

Read-out	✓	When used as a read-out file, all point comments currently stored are read out.
Write	✓	When used as a write file, writing is performed with a point comment number.

Format

```
PCM
```

Meaning • Expresses all point comments.

DATA FORMAT

```
PCmmmm= sssssssssssssss[cr/lf]
PCmmmm= sssssssssssssss[cr/lf]
       :
PCmmmm= sssssssssssssss[cr/lf]
PCmmmm= sssssssssssssss[cr/lf]
[cr/lf]
```

Values mmmm ....Point comment number: 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 onward 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 the
                                  communication port.
Response:
RUN [cr/lf]
PC1 = ORIGIN POS[cr/lf]
PC3 = WAIT POS[cr/lf]
    :
PC3999 =  WORK100[cr/lf]
[cr/lf]
END [cr/lf]
```

4.2 One point comment

Read-out	✓
Write	✓

Format

```
PCmmmm
```

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: 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 onward will be deleted.

SAMPLE

```
SEND PC1 TO CMU ················· Outputs the specified point comment
                                  from the communication port.
Response:
RUN [cr/lf]
PC1 = ORIGIN POS[cr/lf]
END [cr/lf]
```

5 Point name file

5.1 All point names

Read-out	✓	When used as a read-out file, all point names currently stored are read out.
Write	✓	When used as a write file, writing is performed with a point name number.

Format

```
PNM
```

Meaning • Expresses all point names.

DATA FORMAT

```
PNmmmm= asssssssss [cr/lf]
PNmmmm= asssssssss [cr/lf]
:
PNmmmm= asssssssss [cr/lf]
PNmmmm= asssssssss [cr/lf]
[cr/lf]
```

Values

mmmm .... Point comment number: 0 to 29999

a .... Name data (the first character): Use only one-byte alphabetic character. Otherwise, "4.202: Input format error" occurs.

ss...ss..................................... Name data (the second character onward): Use one-byte alphanumeric characters and _ (underscore). Otherwise, "4.202: Input format error" occurs. If name data exceeds 16 characters, then the 17th character onward will be deleted.

MEMO

Name data must not be duplicate. If name data were duplicate, delete the name data with the ealier point name number and save the name data to newly specified point name number.

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND PNM TO CMU ················· Outputs all point names from    t h e
                                  communication port.
Response:
RUN [cr/lf]
PN1=ORIGIN_POS [cr/lf]
PN3=WAIT_POS [cr/lf]
    :
PN 3999=WORK 100 [cr/lf]
[cr/lf]
END [cr/lf]
```

5.2 One point name

Read-out	✓
Write	✓

Format

```
PNmmmm
```

Meaning
- Expresses a specified point name.
- "mmmm" represents a number from 0 to 29999.

DATA FORMAT

```
PNmmmm= asssssssssssssss [cr/lf]
```

Values

mmmm ....Point name number: 0 to 29999

a ....Name data (the first character): Use only one-byte alphabetic character. Otherwise, "4.202: Input format error" occurs.

ss...ss.....................................Name data (the second character onward): Use one-byte alphanumeric characters and _ (underscore). Otherwise, "4.202: Input format error" occurs. If name data exceeds 16 characters, then the 17th character onward will be deleted.

SAMPLE

```
SEND PN1 TO CMU ················· Outputs the specified point name from
                                  the communication port.
Response:
RUN [cr/lf]
PN1=ORIGIN_POS [cr/lf]
END [cr/lf]
```


7 8 9 10 11 12 13

6 Parameter file

6.1 All parameters

Read-out	✓	When used as a read-out file, all parameters currently stored are read out.
Write	✓	When used as a write file, only the parameters specified by labels are written.

Format

```
PRM
```

Meaning • Expresses all parameters.

DATA FORMAT

```
/parameter label/ [cr/lf]
RC=xxxxxx [cr/lf]
/parameter label/ [cr/lf]
R?=xxxxxx[cr/lf]
/parameter label/ [cr/lf]
R?A=xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
\parameter label\ [cr/lf]
C?=xxxxxx [cr/lf]
\parameter label\ [cr/lf]
R?=xxxxxx[cr/lf]
\parameter label\ [cr/lf]
R?A=xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
#parameter label# [cr/lf]
R?=xxxxxx[cr/lf]
#parameter label# [cr/lf]
R?A=xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
/parameter label/ [cr/lf]
C?O=xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
        :
[cr/lf]
```

Values

RC..Indicates the entire controller.
R?..Robot setting (?: Robot number)
C? ....Controller setting (?: Controller number)
A..Represents an axis parameter. Each data is separated by a comma.
O ....Represents an option board parameter. Each data is separated by a comma.

- 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, "10.214: Undefined parameter found" error may occur. In this case, you may load the parameter by setting the "PRM SKIP" parameter to "VALID".
- As parameters whose labels are enclosed in "\" are controller configuration parameters, take care when editing them.
- As parameters whose labels are enclosed in "#" affect robot control, take care when editing them.
- "\" symbols may be displayed as "¥" depending on the computer environment.

SAMPLE

```
SEND PRM TO CMU ················· Outputs all parameters from the
                                  communication port.
Response:
RUN [cr/lf]
` V1.22,R0191-V1.000-V1.09,R0015/V1.09,R0015 [cr/lf]
` Gripper,V0.32/Gripper,V0.32///[cr/lf]
` PRM(0)[cr/lf]
\CNTTYP\[cr/lf]
C1=340[cr/lf]
\YCEADR\[cr/lf]
C1=0[cr/lf]
\DRVASGN\[cr/lf]
R1A=101,102,103,104,0,0[cr/lf]
R2A=0,0,0,0,0,0[cr/lf]
R3A=0,0,0,0,0,0[cr/lf]
R4A=0,0,0,0,0,0[cr/lf]
\RBTNUM\[cr/lf]
R1=2203[cr/lf]
       :
[cr/lf]
END [cr/lf]
```

6.2 One parameter

Read-out	✓	When used as a read-out file, only the parameter specified by a label is read out.
Write	✓	When used as a write file, only the parameter specified by a label is written.

Format

```
/parameter label/, \parameter label\, #parameter label#
```

Meaning
- Parameter labels are shown with 8 alphabetic characters.

DATA FORMAT 1

```
/parameter label/ [cr/lf]
RC= xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT 2

```
/parameter label/ [cr/lf]
R?= xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT 3

```
/parameter label/ [cr/lf]
R?A=xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
[cr/lf]]
```

DATA FORMAT 4

```
\parameter label\ [cr/lf]
C?=xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT5

```
\parameter label\ [cr/lf]
R?=xxxxxx[cr/lf]
[cr/lf]
```

DATA FORMAT 6

```
\parameter label\ [cr/lf]
R?A=xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT 7

```
#parameter label# [cr/lf]
R?=xxxxxx[cr/lf]
[cr/lf]
```

DATA FORMAT 8

```
#parameter label# [cr/lf]
R?A=xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
[cr/lf]
```

DATA FORMAT 9

```
/parameter label/ [cr/lf]
C?O=xxxxxx,xxxxxx,xxxxxx,xxxxxx [cr/lf]
[cr/lf]
```

Values

RC..Indicates the entire controller.
R?..Robot setting (?: Robot number)
C? ....Controller setting (?: Controller number)
A..Represents an axis parameter. Each data is separated by a comma.
O ....Represents an option board parameter. Each data is separated by a comma.

- 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, "10.214: Undefined parameter found" error may occur. In this case, you may load the parameter by setting the "PRM SKIP" to "VALID". (For detail, refer to the YRCX operator's manual.
- As parameters whose labels are enclosed in "\" are controller configuration parameters, take care when editing them.
- As parameters whose labels are enclosed in "#" affect robot control, take care when editing them.
- "\" symbols may be displayed as "¥" depending on the computer environment.

SAMPLE

```
SEND /ACCEL / TO CMU ············ Outputs the acceleration parameter
                                  from the communication port.
Response:
RUN [cr/lf]
/ACCEL / [cr/lf]
R1A=100, 100, 100, 100 [cr/lf]
[cr/lf]
END [cr/lf]
```

7 Shift coordinate definition file

7.1 All shift data

Read-out	✓	When used as a read-out file, all shift data currently stored are read out.
Write	✓	When used as a write file, writing is performed with a shift number.

Format

```
SFT
```

Meaning • Expresses all shift data.

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 number: 0 to 39
f .... Coordinate sign: + / - / space
xxxxxx/yyyyyy/../rrrrrr .... Represent a numeric value of 7 digits or less, having 3 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 the
                                  communication port.
Response:
RUN [cr/lf]
S0 = 0.000 0.000 0.000 0.000 [cr/lf]
SP0= 0.000 0.000 0.000 0.000 [cr/lf]
SM0= 0.000 0.000 0.000 0.000 [cr/lf]
S1 = 1.000 1.000 1.000 1.000 [cr/lf]
       :
SM39= 9.000 9.000 9.000 9.000 [cr/lf]
[cr/lf]
END [cr/lf]
```

7.2 One shift definition

Read-out	✓
Write	✓

Format

```
Sm
```

Meaning • Expresses a specified shift definition.

DATA FORMAT

```
Sm  = fxxxxxx  fyyyyyy  fzzzzzz  frrrrrr[cr/lf]
```

Values

m ....Shift number: 0 to 39
f ....Coordinate sign: + / - / space
xxxxxx/yyyyyy/../rrrrrr ....Represent a numeric value of 7 digits or less, having 3 or less places below the decimal point.

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND S0 TO CMU ·················· Outputs the specified shift coordinate
                                  from the communication port.
Response:
RUN [cr/lf]
S0 = 0.000 0.000 0.000 0.000[cr/lf]
SP0= 0.000 0.000 0.000 0.000[cr/lf]
SM0= 0.000 0.000 0.000 0.000[cr/lf]
[cr/lf]
END [cr/lf]
```

8 Hand definition file

8.1 All hand data

Read-out	✓	When used as a read-out file, all hand data currently stored are read out.
Write	✓	When used as a write file, writing is performed with a hand number.

Format

```
HND
```

Meaning • Expresses all hand data.

DATA FORMAT

```
Hm  = n,fxxxxxxx, fyyyyyyy, fzzzzzz ,{R}[cr/lf]
     :
Hm  = n,fxxxxxxx, fyyyyyyy, 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 real numeric value of 7 digits or less, having 3 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 the
                                  communication port.
Response:
RUN [cr/lf]
H0 = 1, 0.000, 0.000, 0.000 [cr/lf]
H1 = 1, 1.000, 1.000, 1.000 [cr/lf]
H2 = 2, 2.000, 2.000, 2.000 [cr/lf]
H3 = 2, 3.000, 3.000, 3.000 [cr/lf]
H4 = 3, 4.000, 4.000, 4.000 [cr/lf]
H5 = 3, 5.000, 5.000, 5.000 [cr/lf]
H6 = 4, 6.000, 6.000, 6.000 [cr/lf]
H7 = 4, 7.000, 7.000, 7.000 [cr/lf]
[cr/lf]
END [cr/lf]
```

8.2 One hand definition

Read-out	✓
Write	✓

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 real numeric value of 7 digits or less, having 3 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 H3 TO CMU .................. Outputs the specified hand definition
                                  data from the communication port.
Response:
RUN [cr/lf]
H3=2, 3.000, 3.000, 3.000, R [cr/lf]
[cr/lf]
END [cr/lf]
```

9 Work definition file

9.1 All work data

Read-out	✓	When used as a read-out file, all work data currently stored are read out.
Write	✓	When used as a write file, writing is performed with a work number.

Format

```
WRKDEF
```

Meaning • Expresses all work data.

DATA FORMAT

```
Wm = fxxxx.xxx fyyyy.yyy fzzzz.zzz frrrr.rrr [cr/lf]
    :
Wm = fxxxx.xxx fyyyy.yyy fzzzz.zzz frrrr.rrr [cr/lf]
[cr/lf]
```

Notation	Value	Range
m	Work number	0 to 39
f	Coordinate sign	+ / - / space
xxxx.xxx/yyyyy.yy zzzz.zzz/rrrr.rrr	Numeric value consisting of an integer portion of up to 4 digits and having 3 or less places below the decimal point [Unit] xxxx.xxx/yyyyy.yy/zzzz.zzz: mm rrrr.rrr: degree	

■ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND WRKDEF TO CMU ············· Outputs all work data from the
                                 communication port.
SEND CMU TO WRKDEF ············· Intputs all work data from the
                                 communication port.
Response:
RUN [cr/lf]
W0 = 0.000, 0.000, 0.000, 0.000 [cr/lf]
W1 = 1.000, 1.000, 1.000, 1.000 [cr/lf]
W2 = 2.000, 2.000, 2.000, 2.000 [cr/lf]
W3 = 3.000, 3.000, 3.000, 3.000 [cr/lf]
W4 = 4.000, 4.000, 4.000, 4.000 [cr/lf]
W5 = 5.000, 5.000, 5.000, 5.000 [cr/lf]
[cr/lf]
END [cr/lf]
```

9.2 One work definition

Read-out	✓
Write	✓

Format

```
Wm
```

Meaning
- Expresses a specified work definition.

DATA FORMAT

```
Wm = fxxxx.xxx fyyyy.yyy fzzzz.zzz frrrr.rrr [cr/lf]
```

Notation	Value	Range
m	Work number	0 to 39
f	Coordinate sign	+ / - / space
xxxx.xxx/yyyyy.yy zzzz.zzz/rrrr.rrr	Numeric value consisting of an integer portion of up to 4 digits and having 3 or less places below the decimal point [Unit] xxxx.xxx/yyyyy.yy/zzzz.zzz: mm rrrr.rrr: degree	

SAMPLE

```
SEND W3 TO CMU .................. Outputs the specified work data from
                                  the communication port.
SEND CMU TO W3 .................. nputs the specified work data from
                                  communication port.
Response:
RUN [cr/lf]
W3= 3.000, 3.000, 3.000, 3.000 [cr/lf]
END [cr/lf]
```

10 Pallet definition file

10.1 All pallet definitions

Read-out	✓	When used as a read-out file, all pallet definitions currently stored are read out.
Write	✓	When used as a write file, writing is performed with a pallet number.

Format

```
PLT
```

Meaning
- Expresses all pallet definitions.

DATA FORMAT

```
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

mmmm.......................... Pallet number: 0 to 39

XY .... Coordinate plane setting: XY coordinate plane

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/../bbbbbbxr....... 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 2: LEFT

- 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.
- 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 the
                                  communication port.
Response:
RUN [cr/lf]
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]= 0.000 0.000 0.000 0.000 0.000 0.000 [cr/lf]
P[2]= 100.000 0.000 0.000 0.000 0.000 0.000 [cr/lf]
P[3]= 0.000 100.000 0.000 0.000 0.000 0.000 [cr/lf]
P[4]= 100.000 100.000 0.000 0.000 0.000 0.000 [cr/lf]
P[5]= 0.000 0.000 50.000 0.000 0.000 0.000 [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]= 0.000 0.000 0.000 0.000 0.000 0.000 [cr/lf]
P[2]= 100.000 100.000 0.000 0.000 0.000 0.000 [cr/lf]
P[3]= 0.000 200.000 0.000 0.000 0.000 0.000 [cr/lf]
P[4]= 100.000 200.000 0.000 0.000 0.000 0.000 [cr/lf]
P[5]= 0.000 0.000 100.000 0.000 0.000 0.000 [cr/lf]
[cr/lf]
END [cr/lf]
```

10.2 One pallet definition

Read-out	✓
Write	✓

Format

```
PLm
```

Meaning
- Expresses a specified pallet definition.
- "m" represents a number from 0 to 39.

DATA FORMAT

```
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/../bbbbbbxr....... 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 2: LEFT

- 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.
- 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
                                  the communication port as shown below.
Response:
RUN [cr/lf]
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]= 100.000 100.000 50.000 90.000 0.000 0.000 [cr/lf]
P[2]= 200.000 100.000 50.000 90.000 0.000 0.000 [cr/lf]
P[3]= 100.000 200.000 50.000 90.000 0.000 0.000 [cr/lf]
P[4]= 200.000 200.000 50.000 90.000 0.000 0.000 [cr/lf]
P[5]= 100.000 10.000 100.000 90.000 0.000 0.000 [cr/lf]
[cr/lf]
END [cr/lf]
```

11 General Ethernet port file

Read-out	✓	When used as a read-out file, all general Ethernet port definitions are read out.
Write	✓	When used as a write file, writing is performed with a general Ethernet port number.

Format

```
GEP
```

Meaning • Expresses all general Ethernet port definitions.

DATA FORMAT

```
GPm [cr/lf]
MODE=n [cr/lf]
IPADRS= aaa.aaa.aaa.aaa [cr/lf]
PORT=ppppp [cr/lf]
EOL=e [cr/lf]
TYPE=t [cr/lf]
       :
TYPE=t [cr/lf]
[cr/lf]
```

Values

m.................................. General Ethernet port number: 0 to 7
n.................................. Mode
0: Server 1: Client
aaa .... IP address: 0 to 255
ppppp............................ Port number: 0 to 65535
e.................................. Termination character code
0: CRLF 1: CR
t.................................... Port type (0: TCP)

MEMO

When Client mode is selected in the write file,

- IP address and port number: Set the IP address and port number of the connection destination server.

When Server mode is selected in the write file,

- IP address: IP address already set on the controller is used to communicate, so IP address setting is unnecessary.
- Port number: Set a port number which differs from the one on the controller.

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND GEP TO CMU ················· Outputs all files of the general Ethernet
                                  port from the communication port.
Response:
RUN [cr/lf]
GP0 [cr/lf]
MODE=1 [cr/lf]
IPADRS=192.168.0.1 [cr/lf]
PORT=100 [cr/lf]
EOL=0 [cr/lf]
TYPE=0 [cr/lf]
GP1 [cr/lf]
MODE=1 [cr/lf]
IPADRS=192.168.0.100 [cr/lf]
PORT=200 [cr/lf]
EOL=0 [cr/lf]
TYPE=0 [cr/lf]
[cr/lf]
END [cr/lf]
```

7
8
9
10
11
12
13

12 Input/output name file

12.1 All input/output name data

Read-out	✓	When used as a read-out file, all input/output data currently stored are read out.
Write	✓	When used as a write file, writing is performed with a input/output number.

Format

```
ION
```

Meaning • Expresses all input/output name data.

DATA FORMAT

```
ioNMpp(b)=asssssssssssssss [cr/lf]
ioNMpp(b)=asssssssssssssss [cr/lf]
    :
ioNMpp(b)=asssssssssssssss [cr/lf]
ioNMpp(b)=asssssssssssssss [cr/lf]
[cr/lf]
```

Notation	Value	Range
io	Input/output type	DI / DO / SI / SO
pp	Port number	2 to 7 / 10 to 15
b	Bit number	0 to 7
a	Name data (the first character)	Use only one-byte alphabetic character. Otherwise, "4.202: Input format error" occurs.
ss...ss	Name data (the second character onward)	Use one-byte alphanumeric characters and _ (underscore). Otherwise, "4.202: Input format error" occurs. If name data exceeds 16 characters, then the 17th character onward 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 ION TO CMU ................. Outputs all input/output name data
                                  from the communication port.
SEND CMU TO ION ................. Inputs all input/output name data
                                  from the communication port.
Response:
RUN [cr/lf]
DONM2(0)=DO_PORT2_0 [cr/lf]
DONM2(1)=DO_PORT2_1 [cr/lf]
        :
SINM15(6)=SI_PORT15_6 [cr/lf]
SINM15(7)=SI_PORT15_7 [cr/lf]
[cr/lf]
END [cr/lf]
```

MEMO

Name data must not be duplicate. When duplicate name data is saved, delete the name data with the earlier point number and save the name data to the point number which is specified as the new destination to save to.

12.2 One input/output type

Read-out	✓
Write	Restricted*

Format

```
ioNM()
```

Meaning • Expresses a specified input/output type.

DATA FORMAT

```
ioNMpp(b)=asssssssssssssss [cr/lf]
    :
ioNMpp(b)=asssssssssssssss [cr/lf]
[cr/lf]
```

Notation	Value	Range
io	Input/output type	DI / DO / SI / SO
pp	Port number	2 to 7 / 10 to 15 *Readable input/output type and Port number DI: Up to Port14 / DO: Up to Port 10 / SI, SO: Up to Port 15
b	Bit number	0 to 7
a	Name data (the first character)	Use only one-byte alphabetic character. Otherwise, "4.202: Input format error" occurs.
ss...ss	Name data (the second character onward)	Use one-byte alphanumeric characters and _ (underscore). Otherwise, "4.202: Input format error" occurs. If name data exceeds 16 characters, then the 17th character onward 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 DONM() TO CMU ·············· Outputs the specified input/output name
                                  data from the communication port.
Response:
RUN [cr/lf]
DONM2(0)=DO_PORT2_0 [cr/lf]
DONM2(1)=DO_PORT2_1 [cr/lf]
        :
DONM10(6)=DO_PORT10_6 [cr/lf]
DONM10(7)=DO_PORT10_7 [cr/lf]
[cr/lf]
END [cr/lf]
```

12.3 One input/output port

Read-out	✓
Write	Restricted*

Format

```
ioNMpp()
```

Meaning • Expresses a specified input/output type and port number.

DATA FORMAT

```
ioNMpp(b)=assssssssssssssss [cr/lf]
        :
ioNMpp(b)=assssssssssssssss [cr/lf]
[cr/lf]
```

Notation	Value	Range
io	Input/output type	DI / DO / SI / SO
pp	Port number	2 to 7 / 10 to 15 *Readable input/output type and Port number DI: Up to Port14 / DO: Up to Port 10 / SI, SO: Up to Port 15
b	Bit number	0 to 7
a	Name data (the first character)	Use only one-byte alphabetic character. Otherwise, "4.202: Input format error" occurs.
ss...ss	Name data (the second character onward)	Use one-byte alphanumeric characters and _ (underscore). Otherwise, "4.202: Input format error" occurs. If name data exceeds 16 characters, then the 17th character onward 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 DONM2() TO CMU ············· Outputs the specified input/output name
                                  data from the communication port.
Response:
RUN [cr/lf]
DONM2(0)=DO_PORT2_0 [cr/lf]
DONM2(1)=DO_PORT2_1 [cr/lf]
        :
DONM10(6)=DO_PORT10_6 [cr/lf]
DONM10(7)=DO_PORT10_7 [cr/lf]
[cr/lf]
END [cr/lf]
```

12.4 One input/output bit

Read-out	✓
Write	Restricted*

Format

```
ioNMpp(b)
```

Meaning • Expresses a specified input/output type and bit number.

DATA FORMAT

```
ioNMpp(b)=asssssssssssssss [cr/lf]
[cr/lf]
```

Notation	Value	Range
io	Input/output type	DI / DO / SI / SO
pp	Port number	2 to 7 / 10 to 15 *Readable input/output type and Port number DI: Up to Port14 / DO: Up to Port 10 / SI, SO: Up to Port 15
b	Bit number	0 to 7
a	Name data (the first character)	Use only one-byte alphabetic character. Otherwise, "4.202: Input format error" occurs.
ss...ss	Name data (the second character onward)	Use one-byte alphanumeric characters and _ (underscore). Otherwise, "4.202: Input format error" occurs. If name data exceeds 16 characters, then the 17th character onward will be deleted.

SAMPLE

```
SEND DONM2(0) TO CMU ············ Outputs the specified input/output name
                                  data from the communication port.
Response:
RUN [cr/lf]
DONM2(0)=DO_PORT2_0 [cr/lf]
END [cr/lf]
```

7
8
9
10
11
12
13

13 Area check output file

13.1 All area check output data

Read-out	✓	When used as a read-out file, all area check output data currently stored are read out.
Write	✓	When used as a write file, writing is performed with an area check output number.

Format

```
ACO
```

Meaning • Expresses all area check output data.

DATA FORMAT

```
ACm=r,p1,p2,t,n,l [cr/lf]
ACm=r,p1,p2,t,n,l [cr/lf]
       :
ACm=r,p1,p2,t,n,l [cr/lf]
ACm=r,p1,p2,t,n,l [cr/lf]
[cr/lf]
```

Values

m.................................... Area check output number: 0 to 31
r .... Robot number: 0 to 4 (0: Invalid)
p1.................................. Comparison point number 1: 0 to 29999
p2.................................. Comparison point number 2: 0 to 29999
t...................................... Port type
0: DO/SO 1: DO 2: SO 3: MO
n..................................... Port number: 20 to 277
l...................................... Logic
0: OFF 1: ON

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND ACO TO CMU ················· Outputs all area check output data
                                  from the communication port.
Response:
RUN [cr/lf]
AC0=1,0,1,0,20,0 [cr/lf]
AC1=2,100,110,0,50,0 [cr/lf]
       :
AC30=1,20,21,0,20,0 [cr/lf]
AC31=1,50,51,0,100,0 [cr/lf]
[cr/lf]
END[cr/lf]
```

13.2 One area check output definition

Read-out	✓	
Write	✓	When used as a write file, writing is performed with an area check output number.

Format

```
ACm
```

Meaning • Expresses a specified area check output definition.

DATA FORMAT

```
ACm=r,p1,p2,t,n,l [cr/lf]
```

Values

m.................................... Area check output number: 0 to 31
r .... Robot number: 0 to 4 (0: Invalid)
p1.................................. Comparison point number 1: 0 to 29999
p2.................................. Comparison point number 2: 0 to 29999
t..................................... Port type
0: DO/SO 1: DO 2: SO 3: MO
n..................................... Port number: 20 to 277
l...................................... Logic
0: OFF 1: ON

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND AC0 TO CMU ················· Outputs specified area check output
                                  data from the communication port.
Response:
RUN [cr/lf]
AC0=1,0,1,0,20,0 [cr/lf]
END[cr/lf]
```

14 All file

14.1 All file

Read-out	✓
Write	✓

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>
PGN=mmm
aaaa ····aaaaaaaa [cr/lf]
        :
aaaa ····aaaaaaaa [cr/lf]
[cr/lf]
[PNT] ····All point format
Pmmmm=fxxxxxx fyyyyyy fzzzzzz faaaaaa fbbbbbb t [cr/lf]
        :
Pmmmm=fxxxxxx fyyyyyy fzzzzzz faaaaaa fbbbbbb t [cr/lf]
[cr/lf]
[PCM] All point comment format
PCmmmm= ssssssssssssssss [cr/lf]
        :
PCmmmm= ssssssssssssssss [cr/lf]
[cr/lf]
[PNM] ····All point name format
PNmmmm= assssssssssssssss [cr/lf]
        :
PNmmmm= assssssssssssssss [cr/lf]
[cr/lf]
[PRM] ····All parameter format
/parameter label/ [cr/lf]
RC=xxxxxx [cr/lf]
        :
#parameter label# [cr/lf]
R?=xxxxxx [cr/lf]
[cr/lf]
[SFT] ····All shift format
Sm= fxxxxxx fyyyyyy fzzzzzz frrrrrr [cr/lf]
        :
SMm= fxxxxxx fyyyyyy fzzzzzz frrrrrr [cr/lf]
[cr/lf]
[HND] ····All hand format
Hm= n, fxxxxxx, fyyyyyy, fzzzzzz ,{R} [cr/lf]
        :
Hm= n, fxxxxxx, fyyyyyy, fzzzzzz ,{R} [cr/lf]
[cr/lf]
```

14 All file

```
[PLT] ····All pallet format
PLm [cr/lf]
        :
P[5]= fxxxxxx fyyyyyy fzzzzzz frrrrrr faaaaaa fbbbbbb t [cr/lf]
[cr/lf]
[GEP] ····All general Ethernet port format
MODE=n [cr/lf]
        :
TYPE=t [cr/lf]
[cr/lf]
[ION] ····All input/output name format
ioNMpp(b)=asssssssssssssss [cr/lf]
        :
ioNMpp(b)=asssssssssssssss [cr/lf]
[cr/lf]
[ACO] ····All area check output format
ACm=r,p1,p2,t,n,l [cr/lf]
        :
ACm=r,p1,p2,t,n,l [cr/lf]
[cr/lf]
[END] ····All file end
```

MEMO

- In readout files, only items whose data is saved in the controller is readout.
- 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
                                  the communication port.
SEND CMU TO ALL ················· Inputs all files of the entire system from the
                                  communication port.
```

7
8
9
10
11
12
13

15 Program directory file

15.1 Entire program directory

Read-out	✓	When used as a read-out file, information on entire program directory is read out.
Write	–	This file cannot be used as a write file.

Format

```
DIR
```

Meaning • Expresses entire program directory.

DATA FORMAT

```
nnn, yy/mm/dd, hh:mm, bbbbbbb, llll, xx, ff, sssss…sssssssss [cr/lf]
       :
nnn, yy/mm/dd, hh:mm, bbbbbbb, llll, xx, ff, sssss…sssssssss [cr/lf]
[cr/lf]
```

Values

nnn................................ Program number: 1 to 100
yy/mm/dd...................... Date when the program was updated
hh:mm........................... Time when the program was updated
bbbbbb.......................... Byte size of program: 7 digits
xx.................................. File attribute
RW: Readable/writable
RO: Not writable (read only)
H: Hidden file
ff.................................... Flag
m: Main program
c: Current program
s: Sequence program
sss…ssssss .... Program name: 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 DIR TO CMU ················· Outputs information on all program
                                  directory from the communication port.
Response:
RUN [cr/lf]
1, 15/01/10,10:14,100,24,RW,m,SAMPLE1 [cr/lf]
2, 15/01/18,18:00,50,18,RO,,SAMPLE2 [cr/lf]
3, 15/02/11,20:15,200,58,RW,c,SAMPLE3 [cr/lf]
4, 15/02/11,19:03,28,15,H,,SAMPLE4 [cr/lf]
10, 15/03/02, 20:21,592,288,RW,,SAMPLE10 [cr/lf]
24, 15/01/18,13:19,10,3,RW,,SAMPLE24 [cr/lf]
[cr/lf]
END [cr/lf]
```

15.2 One program directory

Read-out	✓
Write	–

Format

```
<<program name>>
```

Meaning
- Expresses information on one program.
- The program name is enclosed in << >> (double brackets).

DATA FORMAT

```
nnn, yy/mm/dd, hh:mm, bbbbbbb, llll, xx, ff, sssss…sssssssss [cr/lf]
```

Values

nnn................................ Program number: 1 to 100
yy/mm/dd....................... Date when the program was updated
hh:mm........................... Time when the program was updated
bbbbbb.......................... Byte size of program: 7 digits
xx.................................. File attribute
RW: Readable/writable
RO: Not writable (read only)
H: Hidden file
ff.................................... Flag
m: Main program
c: Current program
s: Sequence program
sss…ssssss .... Program name: shown with 32 characters or less consisting of alphanumeric characters and _ (underscore)

SAMPLE

```
SEND <<TEST>> TO CMU ············ Outputs information on the specified
                                  program from the communication port.
Response:
RUN [cr/lf]
1, 15/01/10,10:14,100,24,RW,m,SAMPLE1 [cr/lf]
END [cr/lf]
```

16 Parameter directory file

16.1 Entire parameter directory

Read-out	✓	When used as a read-out file, information on entire parameter directory is read out.
Write	–	This file cannot be used as a write file.

Format

```
DPM
```

Meaning
- Expresses entire parameter directory.

DATA FORMAT

```
\mmmmmm\ a m n1 n2 n3 … n10 n11 n12 uuuuuu [cr/lf]
/mmmmmm/ a m n1 n2 n3 … n10 n11 n12 uuuuuu [cr/lf]
#mmmmmm# a m n1 n2 n3 … n10 n11 n12 uuuuuu [cr/lf]
[cr/lf]
```

Values

mmmmmm.................... Parameter label: 8 characters or less having some symbols
a.................................... Attribute
m.................................... Input method
0: Direct input
1 to 12: Selective input
n*.................................. Input range
n1: Minimum value
n2: Maximum value
Selective input value (n1 to n12)
uuuuuu.......................... Units

- 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.

MEMO
"\" symbols may be shown as "¥" depending on the computer environment.

SAMPLE

```
SEND DPM TO CMU ················· Outputs information on all parameter
                                  directory from the communication port.
Response:
RUN [cr/lf]
'PRM(0) [cr/lf]
\CNTTYP\ 16460 0 0 2147493647 [cr/lf]
\YCEADR\ 16396 0 0 99 [cr/lf]
\DRVASGN\ 16398 0 0 9906 [cr/lf]
       :
/IOORGOUT/ 2052 0 0 27 [cr/lf]
/IOSRVOUT/ 2052 0 0 27 [cr/lf]
/GRPORGIN/ 2052 0 0 27 [cr/lf]
[cr/lf]
END [cr/lf]
```

17 Machine reference file

17.1 Machine reference (axes: sensor method, stroke-end method)

Read-out	✓
Write	–

Format

```
MRF
```

Meaning

• Expresses all machine reference values of axes whose return-to-origin method is set as "Sensor" or "Stroke-end".

DATA FORMAT

```
RnA=mmm,mmm,mmm,mmm,mmm,mmm [cr/lf]
       :
RnA= mmm,mmm,mmm,mmm,mmm,mmm [cr/lf]
[cr/lf]
```

Values

n ....Robot number: 1 to 4
mmm ....Machine reference value: 0 to 100

MEMO

This file reads out the machine reference values of the axes set to the robots.
Example: When the 1st through 6th axes of the robot 1 and 1st and 3rd axes of the robot 2 are connected, the data is shown as follows.
R1A = mmm, mmm, mmm, mmm, mmm, mmm
R2A = mmm, mmm

■ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND MRF TO CMU ················· Outputs all machine reference data
                                  from the communication port.
Response:
RUN[cr/lf]
R1A=53,47,58,25,55,59 [cr/lf]
       :
R4A=52,58,41,38,61,50 [cr/lf]
[cr/lf]
END[cr/lf]
```

17.2 Machine reference (axes: mark method)

Read-out	✓
Write	–

Format

```
ARP
```

Meaning

- Expresses all machine reference values of axes whose return-to-origin method is set as "Mark".

DATA FORMAT

```
RnA=mmm,mmm,mmm,mmm,mmm,mmm [cr/lf]
       :
RnA= mmm,mmm,mmm,mmm,mmm,mmm [cr/lf]
[cr/lf]
```

Values

n ....Robot number: 1 to 4
mmm ....Machine reference value: 0 to 100

MEMO

This file reads out the machine reference values of the axes set to the robots.
Example: When the 1st through 6th axes of the robot 1 and 1st and 3rd axes of the robot 2 are connected, the data is shown as follows.
R1A = mmm, mmm, mmm, mmm, mmm, mmm
R2A = mmm, mmm

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND ARP TO CMU ················· Outputs all machine reference data
                                  from the communication port.
Response:
RUN[cr/lf]
R1A=53,47,58,25,55,59 [cr/lf]
       :
R4A=52,58,41,38,61,50 [cr/lf]
[cr/lf]
END[cr/lf]
```

18 System configuration information file

Read-out	✓
Write	–

Format

```
CFG
```

Meaning • Expresses all system configuration information.

DATA FORMAT

```
Cm:nnnn, s, b, kkkkk, ff-ff-ff-ff-ff-ff [cr/lf]
Cm:nnnn, s, b, kkkkk, ff-ff-ff-ff-ff-ff [cr/lf]
       :
Rr:aaaa,hhhhhh [cr/lf]
Rr:aaaa,hhhhhh [cr/lf]
[cr/lf]
```

Values

m	Controllr number: 1 onward
nnn	Controller ID number
s	Specification G: CE specification L: Normal specification
b	Brake power I: Internal E: External
kkkkkk	Memory size
ff	MAC address
r	Robot number: 1 to 4
aaaa	Robot ID number
hhhhhh	Connected axis number

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND CFG TO CMU ················· Outputs all the system configuration
                                  file from the communication port.
Response:
RUN [cr/lf]
C1:340,L,I,2.1MB,00-04-C6-FF-83-12[cr/lf]
R1:MULTI,1234[cr/lf]
[cr/lf]
END [cr/lf]
```

19 Version information file

Read-out	✓
Write	–

Format

```
VER
```

Meaning • Expresses version information.

DATA FORMAT

```
Cm:cv, cr-mv-dv1, dr1/dv2, dr2 [cr/lf]
    :
Cm:cv, cr-mv-dv1, dr1/dv2, dr2 [cr/lf]
[cr/lf]
```

Values

m ....Controllr number: 1 onward
cv..Host version
cr ....Host revision (Rxxxx)
mv ....PLD version (Vx.xx)
dv? (?: 1,2)..............................Driver version (Vx.xx)
dr? (?: 1,2) ....Driver revision (Rxxx)

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND VER TO CMU ················· Outputs all files of the version
                                  information from the communication port.
Response:
RUN [cr/lf]
C1:V1.22,R0191-V1.000-V1.09,R0015/V1.09,R0015 [cr/lf]
C2:V1.22,R0191-V1.000-V1.09,R0015/V1.09,R0015 [cr/lf]
C3:V1.22,R0191-V1.000-V1.09,R0015/V1.09,R0015 [cr/lf]
C4:V1.22,R0191-V1.000-V1.09,R0015/V1.09,R0015 [cr/lf]
[cr/lf]
END [cr/lf]
```

20 Option board file

Read-out	✓
Write	–

Format

```
OPT
```

Meaning • Expresses all option boards.

DATA FORMAT

```
CmOn:aaaaaa,Vb.bb [cr/lf]
CmOn:aaaaaa,Vb.bb [cr/lf]
      :
CmOn:aaaaaa,Vb.bb [cr/lf]
CmOn:aaaaaa,Vb.bb [cr/lf]
[cr/lf]
```

Values

m.................................... Controllr number: 1 onward
n.................................... Option board number inside the controller
Slot number: 1 to 4
aaaaaa............................ Option board name
b.bb................................ Option board version

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND OPT TO CMU ················· Outputs all files of the option boards
                                  from the communication port.
Response:
RUN [cr/lf]
C1O1:Gripper,V0.32 [cr/lf]
C1O2:Gripper,V0.32 [cr/lf]
[cr/lf]
END [cr/lf]
```

21 Self check file

Read-out	✓
Write	–

Format

```
SCK
```

Meaning • Expresses self check file.

DATA FORMAT

```
gg.bbb:mmmm [cr/lf]
gg.bbb:mmmm [cr/lf]
:
gg.bbb:mmmm [cr/lf]
gg.bbb:mmmm [cr/lf]
[cr/lf]
```

Values

gg.................................. Alarm group number
bbb................................ Alarm classification number
mmmm.......................... Alarm occurrence location
RC: Entire controller
R?: Robot (?: Robot number)
C?: Controller (?: Controller number)
A?: Axis (?: Axis number)
M?: Driver (?: Driver number)
R?: Option board
(?: Option board number inside the controller)
T?: Task (?: Task number)
ETH: Ethernet
CMU: RS-232CBrake power

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SCK TO CMU ················· Outputs all files of the self check
                                  information from the communication port.
Response:
RUN [cr/lf]
12.600:C1M1 [cr/lf]
12.600:C1M2 [cr/lf]
12.600:C1M3 [cr/lf]
12.600:C1M4 [cr/lf]
[cr/lf]
END [cr/lf]
```

22 Alarm history file

Read-out	✓
Write	–

Format

```
LOG
```

Meaning • Expresses all alarm history.

DATA FORMAT

```
nnn:yy/mm/dd, hh:mm:ss, gg.bbb : aaaa,c, eee : ffff,
iiiii, jjjjjjjj, kkkkkkkk, llllllll, oooooooo, pppppppp,
pppppppp, pppppppp, pppppppp, pppppppp, pppppppp, q [cr/lf]
nnn:yy/mm/dd, hh:mm:ss, gg.bbb : aaaa,c, eee : ffff,
iiiii, jjjjjjjj, kkkkkkkk, llllllll, oooooooo, pppppppp,
pppppppp, pppppppp, pppppppp, pppppppp, pppppppp, q [cr/lf]
        :
nnn:yy/mm/dd, hh:mm:ss, gg.bbb : aaaa,c, eee : ffff,
iiiii, jjjjjjjj, kkkkkkkk, llllllll, oooooooo, pppppppp,
pppppppp, pppppppp, pppppppp, pppppppp, pppppppp, q [cr/lf]
[cr/lf]
```

22 Alarm history file

Values

nnn	Alarm history number: 1 to 500
yy/mm/dd	Alarm occurrence date
hh:mm:ss	Alarm occurrence time
gg	Alarm group number
bbb	Alarm classification number
aaaa	Alarm occurrence location RC: Entire controller R?: Robot (?: Robot number) C?: Controller (?: Controller number) A?: Axis (?: Axis number) M?: Driver (?: Driver number) R?: Option board (?: Option board number inside the controller) T?: Task (?: Task number) ETH: Ethernet CMU: RS-232C Brake power
c	Operation mode I: IllegalM: Manual mode A: Automatic mode (with programming box) O: Automatic mode (with other devices) CMU: RS-232C
eee	Program number
ffff	Program execution line
iiiii	Point number
jjjjjjjj	Parallel input: Port o to 3 (hexadecimal)
kkkkkkkk	Parallel output: Port o to 3 (hexadecimal)
llllllll	Serial input: Port o to 3 (hexadecimal)
oooooooo	Serial output: Port o to 3 (hexadecimal)
jjjjjj	Alarm occurrence location: A1 to A6
q	Hand system 0: NONE 1: RIGHT 2: LEFT

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND LOG TO CMU ················· Outputs all files of the alarm history
                                  from the communication port.
Response:
RUN [cr/lf]
1:15/03/30,08:23:05,1.100:RC,O,:,0,00000000,00000012,00000000,00000112,,,,,,, [cr/lf]
2:15/03/30,08:23:05,5.288: RC,O,:,0,00000000,00000010,00000000,00000110,,,,,,, [cr/lf]
       :
500:15/03/18,10:23:04,5.228:T01,O,17:3,,00000000,00000010,00000000,00000110,
    40119,100000,99996,39375,0,0,0 [cr/lf]
[cr/lf]
END [cr/lf]
```

23 Remaining memory size file

Read-out	✓
Write	–

Format

```
MEM
```

Meaning • Expresses remaining memory size

DATA FORMAT

```
PGM+PNT AREA=mmmmmmm/nnnnnnnn[cr/lf]
VAR AREA=xxxxx/yyyyy[cr/lf]
[cr/lf]
```

Values

mmmmmmmm ....Remaining memory size of program and point area
nnnnnnn ....Total memory size of program and point area
xxxxx ....Remaining memory size of variable area
yyyyy ....Total memory size of variable area

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND MEM TO CMU ················· Outputs all files of the remaining memory
                                  size from the communication port.
Response:
RUN [cr/lf]
PGM+PNT AREA=2088547 / 2100000 [cr/lf]
VAR AREA=23220 / 24000 [cr/lf]
[cr/lf]
END [cr/lf]
```

24 Variable file

24.1 Dynamic variables

All dynamic variables

Read-out	✓	When used as a read-out file, all dynamic variables currently stored are read out.
Write	✓	When used as a write file, a specified dynamic variable is written.

Format

```
VAR
```

Meaning
- Expresses all dynamic variables.

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 number, %: Integer, $: Character string

xxxxxx.......................Value of variable
Integer type: Integer of -2147483647 to 2147483647
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

```
SEND VAR TO CMU ················· Outputs all global variables from the
                                  communication port.
Response:
RUN [cr/lf]
A%=150 [cr/lf]
B!=1.0234E1 [cr/lf]
C1$="SAMPLE1" [cr/lf]
C2$="SAMPLE2" [cr/lf]
[cr/lf]
END [cr/lf]C1$="CNS_1"[cr/lf]
C2$="CNS_2"[cr/lf]
[cr/lf]
END [cr/lf]
```

One dynamic variable

Read-out	✓
Write	✓

Format

```
variable name t
```

Meaning • Expresses one dynamic 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 number, %: Integer, $: Character string

xxxxxx......................Value 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 A% TO CMU [cr/lf] ·········· Outputs the specified variable A% from
                                  the communication port.
Response:
150 [cr/lf]
```

SAMPLE 2

```
SEND CMU TO A% [cr/lf] ·········· Inputs the specified variable A% from
                                  the communication port.
Response:
300 [cr/lf] ····················· Data input to the controller.
OK [cr/lf]  ····················· Result output from the controller.
```

24.2 Static variables

24.2.1 Integer type static variables (SGI)

All integer type static variables

Read-out	✓	When used as a read-out file, all integer type static variables currently stored are read out.
Write	✓	When used as a write file, a specified integer type static variable is written.

Format

```
SGI
```

Meaning
- Expresses all integer static variables.

DATA FORMAT

```
SGIn=xxxxxx [cr/lf]
SGIn=xxxxxx [cr/lf]
       :
SGIn=xxxxxx [cr/lf]
[cr/lf]
```

Values
n ....Integer type static variable number: 0 to 31
xxxxxx ....Integer of -2147483647 to 2147483647

- A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SGI TO CMU ················· Outputs all integer type static
                                  variables from the communication port.
Response:
RUN [cr/lf]
SGR0=0 [cr/lf]
SGR1=0 [cr/lf]
       :
SGR31=0 [cr/lf]
[cr/lf]
END [cr/lf]
```

One integer type static variables

Read-out	✓
Write	✓

Format

```
SGIm
```

Meaning

- Expresses a specified integer type static variable.
- "m" represents a number from 0 to 31.

DATA FORMAT

```
xxxxxx [cr/lf]
```

Values xxxxxx ....Integer of -2147483647 to 2147483647

SAMPLE

```
SEND SGI1 TO CMU ················ Outputs the specified integer type
                                  static variables (SGI1) from the
                                  communication port.
Response:
RUN [cr/lf]
0 [cr/lf]
END [cr/lf]
```

24.2.2 Real type static variables (SGR)

All real type static variables

Read-out	✓	When used as a read-out file, all real type static variables currently stored are read out.
Write	✓	When used as a write file, a specified real type static variable is written.

Format

```
SGR
```

Meaning • Expresses all real type static variables.

DATA FORMAT

```
SGRn=xxxxxx [cr/lf]
SGRn=xxxxxx [cr/lf]
       :
SGRn=xxxxxx [cr/lf]
[cr/lf]
```

Values n .... Real type static variable number: 0 to 31
xxxxxx .... Real number of 7 digits or less including decimal fractions

■ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE

```
SEND SGR TO CMU ················· Outputs all real type static variables
                                  from the communication port.
Response:
RUN [cr/lf]
SGI0=0 [cr/lf]
SGI1=0 [cr/lf]
       :
SGI31=0 [cr/lf]
[cr/lf]
END [cr/lf]
```

One real type static variables

Read-out	✓
Write	✓

Format

```
SGRm
```

Meaning

- Expresses a specified real type static variable.
- "m" represents a number from 0 to 31.

DATA FORMAT

```
xxxxxx [cr/lf]
```

Values

xxxxxx .... Real number of 7 digits or less including decimal fractions

SAMPLE

```
SEND SGR1 TO CMU ················ Outputs the specified real type static
                                  variables (SGR1) from the communication
                                  port.
Response:
RUN [cr/lf]
0 [cr/lf]
END [cr/lf]
```

25 Constant file

25.1 One character string

Read-out	✓	When used as a read-out file, the specified character string is read out.
Write	–	This file cannot be used as a write file.

Format

```
"character string"
```

Meaning • Expresses a specified character string.

DATA FORMAT

```
sssss...ssssss[cr/lf]
```

Values sssss...ssssss............................Character string: 255 characters or less

- Output of " symbol (double quotation) is shown with successive " symbol.

SAMPLE

```
SEND """OMRON ROBOT""" TO CMU
            ..................... Outputs the specified character string
                                  from the communication port.
Response:
"OMRON ROBOT"[cr/lf]
```

26 Array variable file

26.1 All array variables

Read-out	✓	When used as a read-out file, all array variables are read out.
Write	✓	When used as a write file, a specified array variable is written.

Format

```
ARY
```

Meaning • Expresses all array variables.

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 by the DIM statement in the program. Variable name is shown with 32 characters or less consisting of alphanumeric characters and _ (underscore).

t.............................Type of variable
!: Real number, %: Integer, $: Character string

l, m, n...................Indicate array arguments

xxxxxx...................Differs depending on the type of array variable.
Integer type: Integer of -2147483647 to 2147483647
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

```
SEND ARY TO CMU ················· Outputs all global array variables
                                  from the communication port.
Response:
RUN [cr/lf]
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%(1,0)=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]
END [cr/lf]
```

26.2 One array variable

Read-out	✓
Write	✓

Format

```
variable name t(l {,m {,n }})
```

Meaning • Expresses one array variable.

DATA FORMAT

```
xxxxxx [cr/lf]
```

Values

Variable name.......Global variable defined by the DIM statement in the program. Variable name is shown with 32 characters or less consisting of alphanumeric characters and _ (underscore).

t............................Type of variable
!: Real number, %: Integer, $: Character string

l, m, n...................Indicate array arguments

xxxxxx..................Differs depending on the type of array variable.
Integer type: Integer of -2147483647 to 2147483647
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

```
SEND C1$(2) TO CMU ·············· Outputs the specified array variable
                                  C1$(2) from the communication port.
Response:
RUN [cr/lf]
OMRON ROBOT [cr/lf]
END [cr/lf]
```

27 DI file

27.1 All DI information

Read-out	✓	When used as a read-out file, all DI information is read out.
Write	–	This file cannot be used as a write file.

Format

```
DI()
```

Meaning • Expresses all DI (parallel input variable) information.

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).

- 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 the
                                  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]
END [cr/lf]
```

27.2 One DI port

Read-out	✓	When used as a read-out file, the specified DI port status is read out.
Write	–	This file cannot be used as a write file.

Format

```
DIm()
```

Meaning • Expresses the status of one DI port.

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 number).

SAMPLE

```
SEND DI5() TO CMU ··············· Outputs the DI5 port status from the
                                  communication port.
Response:
RUN [cr/lf]
DI15()=&B00000000 [cr/lf]
END [cr/lf]
```

28 DO file

28.1 All DO information

Read-out	✓	When used as a read-out file, all DO information is read out.
Write	✓	When used as a write file, the value is written to the specified DO port.

Format

```
DO()
```

Meaning
- Expresses all DO (parallel output variable) information.
- Writing to DO0() and DO1() is prohibited.

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).

- 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 the
                                  communication port.
Response:
RUN [cr/lf]
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]
END [cr/lf]
```

28.2 One DO port

Read-out	✓	When used as a read-out file, the specified DO port status is read out.
Write	✓	When used as a write file, the value is written to the specified DO port.

Format

```
DOm()
```

Meaning

- Expresses the status of one DO port.
- 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 number).
k .... Integer from 0 to 255

MEMO

Writing to DO0() and DO1() is prohibited. Only referencing is permitted.

SAMPLE

```
SEND DO5() TO CMU ··············· Outputs the DO5 port status from the
                                  communication port.
Response:
RUN [cr/lf]
DO5()=&B00000000[cr/lf]
END [cr/lf]
```

29 MO file

29.1 All MO information

Read-out	✓	When used as a read-out file, all MO information is read out.
Write	✓	When used as a write file, the value is written to the specified MO port.

Format

```
MO()
```

Meaning
- Expresses all MO (internal output variable) information.
- Writing to MO30() and DO37() is prohibited.

DATA FORMAT

```
MO0()=&Bnnnnnnnn [cr/lf]
MO1()=&Bnnnnnnnn [cr/lf]
       :
MO37()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values n ...."0" or "1" (total of 8 digits).

- 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 the
                                  communication port.
Response:
RUN [cr/lf]
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]
:
MO27()=&B00000000 [cr/lf]
MO30()=&B00000000 [cr/lf]
:
MO36()=&B00000000 [cr/lf]
MO37()=&B00000000 [cr/lf]
[cr/lf]
END [cr/lf]
```

29.2 One MO port

Read-out	✓	When used as a read-out file, the specified MO port status is read out.
Write	✓	When used as a write file, the value is written to the specified MO port.

Format

```
MOm()
```

Meaning
- Expresses the status of one MO port.
- 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 number).
k ....Integer from 0 to 255

MEMO Writing to MO30() to MO37() is prohibited. Only reference is permitted.

SAMPLE

```
SEND MO5() TO CMU ··············· Outputs the MO5 port status from the
                                  communication port.
Response:
RUN [cr/lf]
MO5()=&B00000000[cr/lf]
END [cr/lf]
```

30 LO file

30.1 All LO information

Read-out	✓	When used as a read-out file, all LO information is read out.
Write	✓	When used as a write file, the value is written to the specified LO port.

Format

```
LO()
```

Meaning • Expresses all LO (internal output variable) information.

DATA FOMAT

```
LO0()=&Bnnnnnnnn [cr/lf]
LO1()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values n ...."0" or "1" (total of 8 digits).

- 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 the
                                  communication port.
Response:
RUN [cr/lf]
LO0()=&B10001001 [cr/lf]
LO1()=&B00100100 [cr/lf]
[cr/lf]
END [cr/lf]
```

30.2 One LO port

Read-out	✓	When used as a read-out file, the specified LO port status is read out.
Write	✓	When used as a write file, the value is written to the specified LO port.

Format

```
LOm()
```

Meaning

- Expresses the status of one 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 m7, m6, …, m0, reading from the left ("m" is the port number)
k .... Integer from 0 to 255

SAMPLE

```
SEND LO0() TO CMU ··············· Outputs the LO0 port status from the
                                  communication port.
Response:
RUN [cr/lf]
LO0()=&B00000000 [cr/lf]
END [cr/lf]
```

31 TO file

31.1 All TO information

Read-out	✓	When used as a read-out file, all TO information is read out.
Write	✓	When used as a write file, the value is written to the specified TO port.

Format

```
TO()
```

Meaning • Expresses all TO (timer output variable) information.

DATA FORMAT

```
TO0()=&Bnnnnnnnn [cr/lf]
TO1()=&Bnnnnnnnn [cr/lf]
[cr/lf]
```

Values n ...."0" or "1" (total of 8 digits).

- 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 the
                                  communication port.
Response:
RUN [cr/lf]
TO0()=&B10001001 [cr/lf]
TO1()=&B10001001 [cr/lf]
[cr/lf]
END [cr/lf]
```

31.2 One TO port

Read-out	✓	When used as a read-out file, the specified TO port status is read out.
Write	✓	When used as a write file, the value is written to the specified TO port.

Format

```
TOm()
```

Meaning • Expresses the status of one 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 m7, m6, …, m0, reading from the left ("m" is the port number).
k .... Integer from 0 to 255

SAMPLE 1

```
SEND TO0() TO CMU ··············· Outputs the TO0 port status from the
                                   communication port.
Response:
RUN [cr/lf]
TO0()=&B00000000 [cr/lf]
END [cr/lf]
```

32 SI file

32.1 All SI information

Read-out	✓	When used as a read-out file, all SI information is read out.
Write	–	This file cannot be used as a write file.

Format

```
SI()
```

Meaning • Expresses all SI (serial input variable) information.

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).

- 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 the
                                  communication port.
Response:
RUN [cr/lf]
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]
END [cr/lf]
```

32 SI file

32.2 One SI port

Read-out	✓	When used as a read-out file, the specified SI port status is read out.
Write	–	This file cannot be used as a write file.

Format

```
SIm()
```

Meaning • Expresses the status of one SI port.

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 number).

SAMPLE

```
SEND SI5() TO CMU ················ Outputs the SI5 port status from the
                                   communication port.
Response:
RUN [cr/lf]
SI5()=&B00000000 [cr/lf]
END [cr/lf]
```

33 SO file

33.1 All SO information

Read-out	✓	When used as a read-out file, all SO information is read out.
Write	✓	When used as a write file, the value is written to the specified SO port.

Format

```
SO()
```

Meaning
- Expresses all SO (serial output variable) information.
- Writing to SO0() and SO1() is prohibited.

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).

■ 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 the
                                  communication port.
Response:
RUN [cr/lf]
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]
END [cr/lf]
```

33 SO file

33.2 One SO port

Read-out	✓	When used as a read-out file, the specified SO port status is read out.
Write	✓	When used as a write file, the value is written to the specified SO port.

Format

```
SOm()
```

Meaning
- Expresses the output status of one SO port.
- 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 number).
k ....Integer from 0 to 255

MEMO

Writing to SO0() and SO1() is prohibited. Only reference is permitted.

SAMPLE

```
SEND SO5() TO CMU ··············· Outputs the SO5 port status from the
                                  communication port.
Response:
RUN [cr/lf]
SO5()=&B00000000 [cr/lf]
END [cr/lf]
```

34 SIW file

34.1 All SIW data

Read-out	✓	When used as a read-out file, all SIW information is read out in hexadecimal digit.
Write	–	This file cannot be used as a write file.

Format

```
SIW()
```

Meaning • Expresses all SIW (serial word input) data.

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 the
                                  communication port.
Response:
RUN [cr/lf]
SIW(0)=&H1001[cr/lf]
SIW(1)=&H0010[cr/lf]
SIW(2)=&H0000[cr/lf]
       :
SIW(15)=&H0000[cr/lf]
[cr/lf]
END [cr/lf]
```

34.2 One SIW data

Read-out	✓	When used as a read-out file, the specified SIW status is read out in hexadecimal digit.
Write	–	This file cannot be used as a write file.

Format

```
SIW(m)
```

Meaning • Expresses one SIW status.

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 the communication
                                  port.
Response:
RUN [cr/lf]
SIW(5)=&H1001[cr/lf]
END [cr/lf]
```

7
8
9
10
11
12
13

35 SOW file

35.1 All SOW

Read-out	✓	When used as a read-out file, all SOW information is read out in hexadecimal digit.
Write	✓	When used as a write file, the value is written to the specified SOW port.

Format

```
SOW()
```

Meaning
- Expresses all SOW (serial word output) data.
- Writing to SOW(0) and SOW(1) is prohibited.

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 the
                                  communication port.
Response:
RUN [cr/lf]
SOW(0)=&H1001[cr/lf]
SOW(1)=&H0010[cr/lf]
SOW(2)=&H0000[cr/lf]
       :
SOW(15)=&H0000[cr/lf]
[cr/lf]
END [cr/lf]
```

35.2 One SOW data

Read-out	✓	When used as a read-out file, the specified SOW port status is read out in hexadecimal digit.
Write	✓	When used as a write file, the value is written to the specified SOW port.

Format

```
SOW(m)
```

Meaning
- Expresses one SOW status.
- Writing to SOW(0) and SOW(1) 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)

▪ A line containing only [cr/lf] is added at the end of the file, indicating the end of the file.

SAMPLE 1

```
SEND SOW(5) TO CMU ·············· Outputs SOW(5) from the communication
                                  port.
Response:
RUN [cr/lf]
SOW(5)=&H1001[cr/lf]
END [cr/lf]
```

36 EOF file

36.1 EOF data

Read-out	✓	When used as a read-out file, ^Z (=1Ah) is read out.
Write	–	This file cannot be used as a write file.

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.

DATA FORMAT

```
^Z (=1Ah)
```

SAMPLE

```
SEND PGM TO CMU
SEND EOF TO CMU ················· Outputs EOF data from the 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.

37 Serial port communication file

37.1 Serial port communication file

Read-out	✓
Write	✓

Format

```
CMU
```

Meaning

- Expresses the serial communication port.
- Depends on the various data formats.

SAMPLE

```
SEND PNT TO CMU ················· Outputs all point data from the
                                  communication port.
SEND CMU TO PNT ················· Inputs all point data from the
                                  communication port.
```

38 Ethernet port communication file

38.1 Ethernet port communication file

Read-out	✓
Write	✓

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 11
User program examples

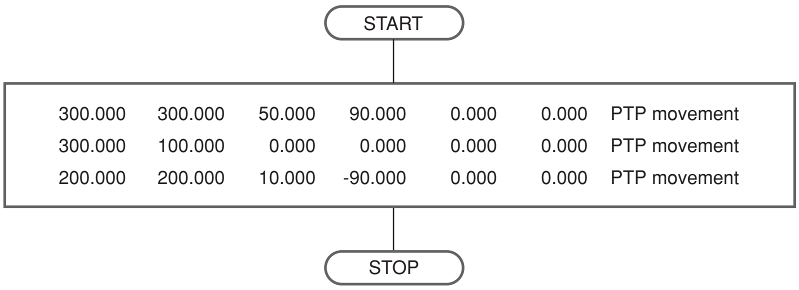
1 Basic operation

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



33C01-R7-00

SAMPLE

```
MOVE P,    300.000   300.000   50.000    90.000   0.000   0.000
MOVE P,    300.000   100.000    0.000     0.000   0.000   0.000
MOVE P,    200.000   200.000   10.000   -90.000   0.000   0.000
HALT
```

1.2 Using point numbers

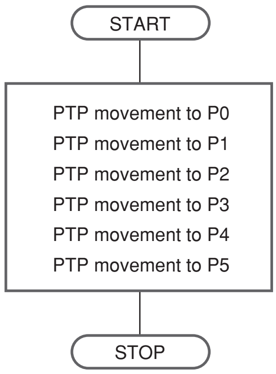
■ **Overview**

Coordinate data can be specified by using point numbers in a program. Coordinate data should be entered beforehand from the programming box or the support software "SCARA-YRCX Studio", for example as shown below (For details, refer to the YRCX operator's manual or the SCARA-YRCX Studio manual).

POINT DATA

```
P0= 0.000 0.000     0.000 ··· 0.000     0.000     0.000
P1= 100.0000.000  150.000 ·· 30.000     0.000     0.000
P2= 0.000100.000   50.000 ··· 0.000     0.000     0.000
P3= 300.000300.000  0.000 ··· 0.000     0.000     0.000
P4= 300.000100.000100.000·· 90.000     0.000     0.000
P5= 200.000200.000  0.000 ··· 0.000     0.000     0.000
```

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 numbers and the FOR statement.

1.3 Using shift coordinates

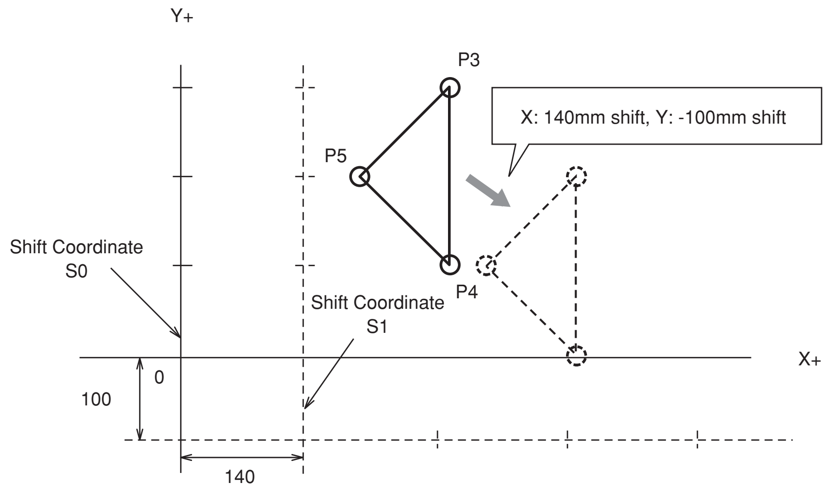
- **Overview**

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= 0.000 0.000     0.000 ··· 0.000
S1= 140.000-100.000 0.000 ··· 0.000
```

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
HALT
```

1.4 Palletizing

1.4.1 Calculating point coordinates

■ **Overview**

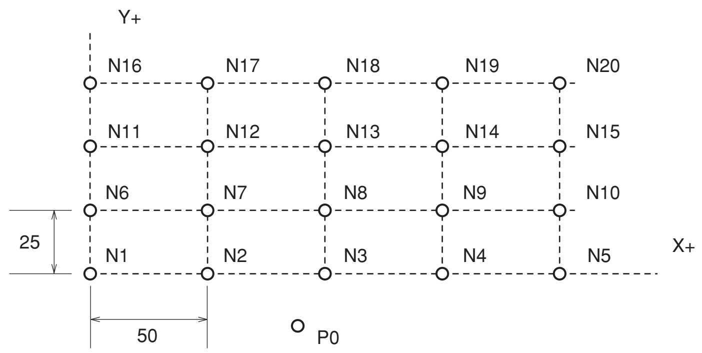
Repetitive movement between a fixed work supply position P0 and each of the equally spaced points on a pallet can be performed with the following program.

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

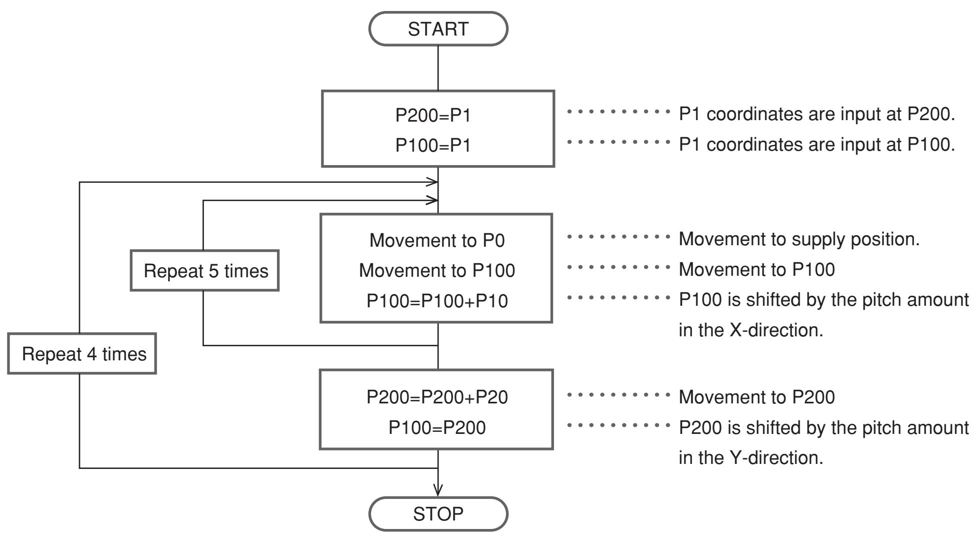
```
Work supply position:
   P0=        0.000     0.000     0.000      0.000     0.000     0.000
X-axis pitch:
   P10=      50.000     0.000     0.000      0.000     0.000     0.000
Y-axis pitch:
   P20=       0.000    25.000     0.000      0.000     0.000     0.000
N1 position:
   P1 =     100.000    50.000     0.000      0.000     0.000     0.000
```

Calculating point coordinates



33C04-R7-00

Processing flow



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
HALT
```

7 8 9 10 11 12 13

1.4.2 Utilizing pallet movement

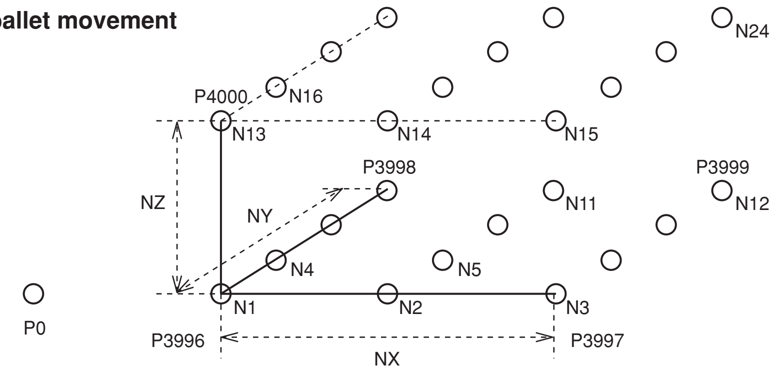
- **Overview**

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

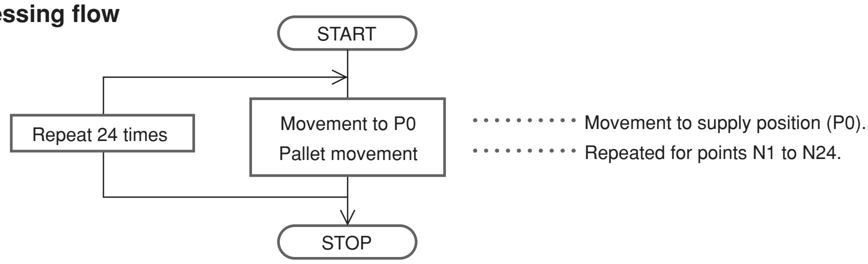
```
Work supply position:
P0=           0.000     0.000   200.000      0.000     0.000     0.000
Pallet definition:
PL0
NX= 3
NY= 4
NZ= 2
PLP=   3996:(P3996 to P4000 are used)
P[1]=       100.000    50.000   200.000      0.000     0.000     0.000
P[2]=       200.000    50.000   200.000      0.000     0.000     0.000
P[3]=       100.000   200.000   200.000      0.000     0.000     0.000
P[4]=       200.000   200.000   200.000      0.000     0.000     0.000
P[5]=       100.000    50.000   100.000      0.000     0.000     0.000
```

Utilizing pallet movement



33C06-R9-00

Processing flow



33C07-R7-00

SAMPLE

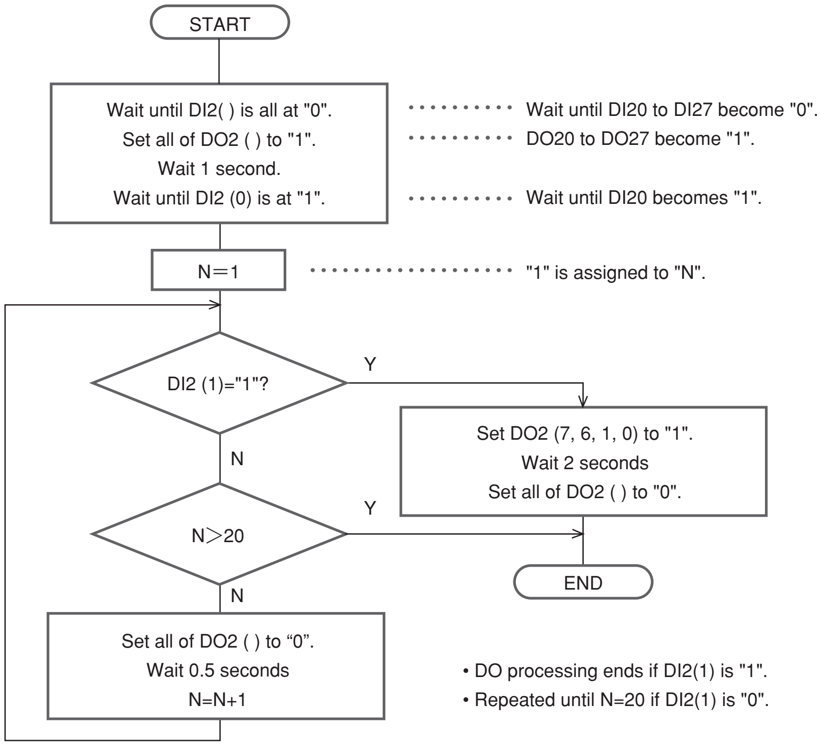
```
FOR I=1 TO 24 ··················· Repeated for I = 1 to 24.
    MOVE P,P0,Z=0.000··········· Movement of robot 1 to supply position.
    PMOVE (0,I),Z=0.000········· Movement of robot 1 to pallet point.
NEXT I
MOVE P,P0,Z=0.000
HALT
```

1.5 DI/DO (digital input and output) operation

■ **Overview**

The following example shows input/ output signal operations through the general-purpose input/ output device.

Processing flow



33C08-R7-00

SAMPLE

```
WAIT DI2()=0 ···················· Waits until DI20 to DI27 become "0".
DO2()=&B11111111 ················ DO20 to DO27 become "1".
DELAY 1000
WAIT DI2(0)=1 ··················· Waits until DI20 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 ············· Sets DO27, 26, 21, 20 to "1".
DELAY 2000  ····················· Waits 2 seconds
DO2()=0     ····················· Sets DO20 to "0".
*ALLEND:
HALT
```

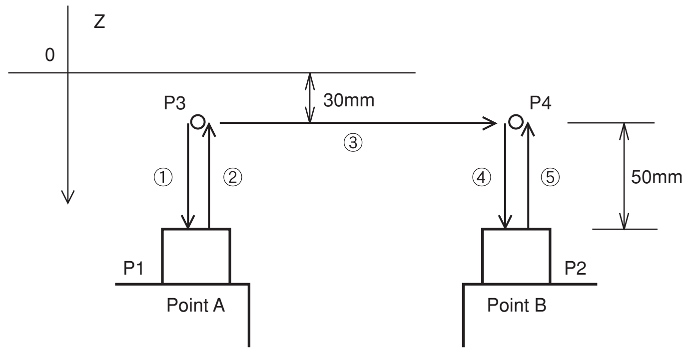
2 Application

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)-50.000..... Axis 3 data of P3 is shifted 50mm in
                                  upper direction.
    LOC3(P4)=LOC3(P4)-50.000..... Axis 3 data of P4 is shifted 50mm in
                                  upper 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

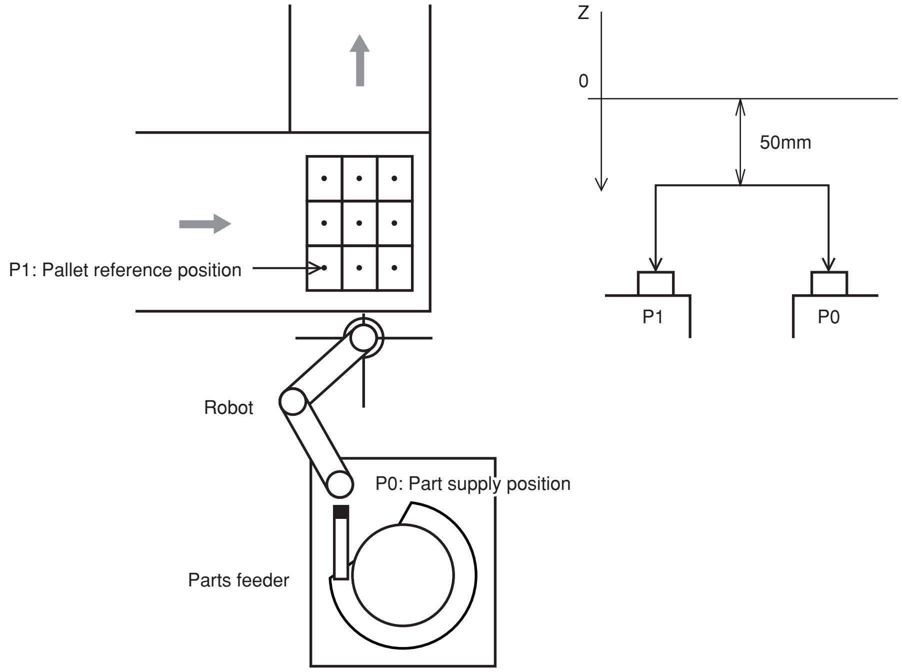
```
    P4=P2   ····················· P2 coordinates are assigned to P4.
    LOC3(P4)=LOC3(P4)-50.000····· Axis 3 data of P4 is shifted 50mm in
                                  upper direction.
    GOSUB *OPEN
    MOVE P,P1,A3=30.000·········· Arch motion at A3 = 30mm.
    GOSUB *CLOSE
    MOVE P,P2,A3=30.000·········· 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 seconds 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=50.000 ······ Movement of robot 1 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=50.000 ···· Movement of robot 1 to calculated point.
        DO(30)=0 ················· Robot hand OPENS.
        DELAY 100
    NEXT
  NEXT
  DRIVE (3,0) ··················· Only 3 axis of robot 1 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

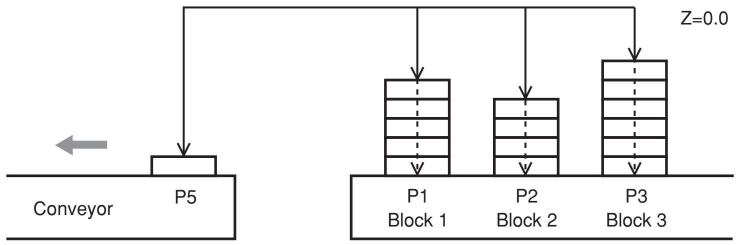
```
* Precondition: Must be defined at pallet "0".
WHILE -1    ····················· All repeated.
    FOR A=1 TO 9
        WAIT DI(31)=1 ············ Wait until a pallet "present" status
                                   occurs.
        WAIT DI(30)=1 ············ Wait until the supplied component
                                   "present" status occurs.
        DO(30)=0 ················· Robot hand OPENS.
        DELAY 100
        MOVE P,P0,A3=50.000 ······ Movement of robot 1 to supply position.
        DO(30)=1 ················· Robot hand CLOSES.
        DELAY 100
        PMOVE(0,A),A3=50.000 ····· Movement of robot 1 to pallet point.
        DO(30)=0 ················· Robot hand OPENS.
        DELAY 100
    NEXT
    DRIVE(3,0)··················· Only axis 3 of robot 1 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

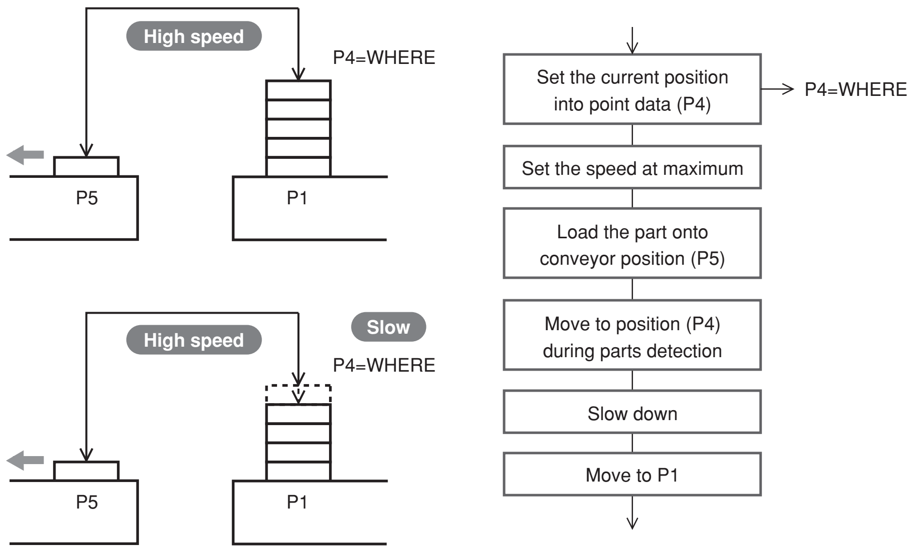
• Robot hand open/close time is 0.1 seconds.

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



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)=0.000
MOVE P,P6,A3=0.000
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=0.000
    GOSUB *OPEN
    MOVE P,P4,A3=0.000
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**

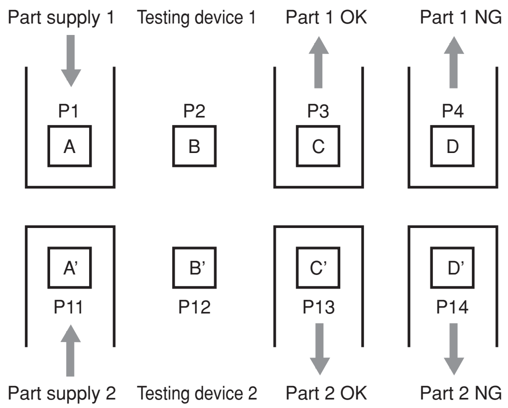
One robot is used to inspect two different parts and sort them according to the OK/NG results judged by a testing device.

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)

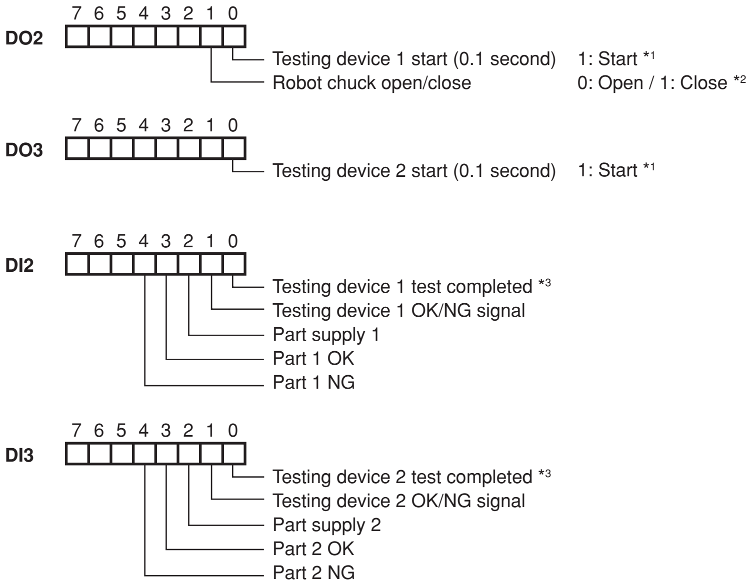


33C13-R7-00

■ **Precondition**

1. I/O signal

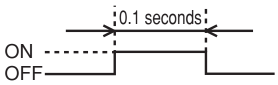
I/O signal



33C14-R7-00

NOTE

- *1: As the start signal, supply a 0.1 second pulse signal to the testing device.



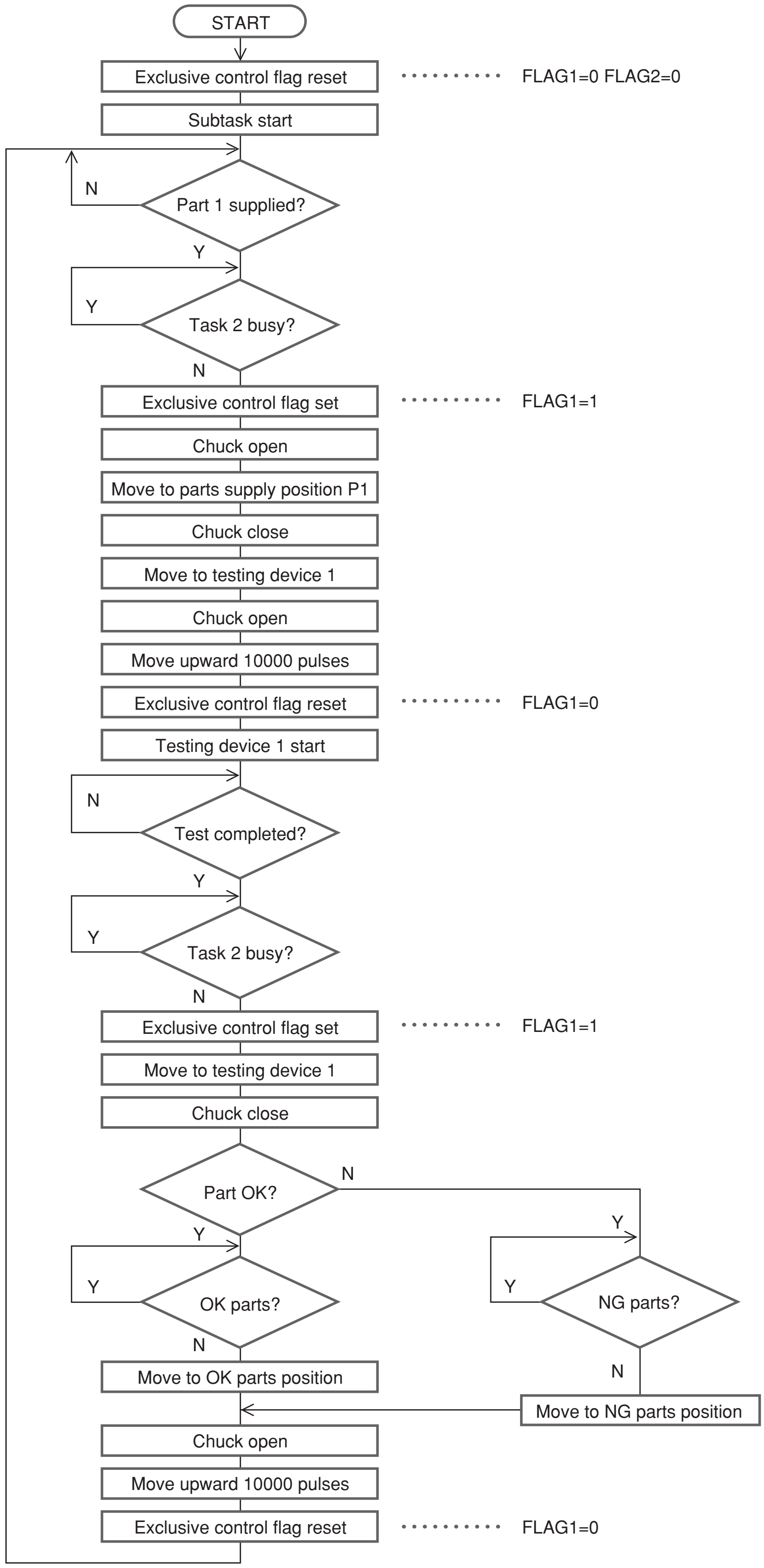
- *2: Chuck open and close time is 0.1 seconds.
- *3: Each time a test is finished, the test completion signal and OK/NG signal are sent from the testing device. After testing, the test completion signal turns ON (=1), and the OK/ NG signal turns ON (=1) when the result is OK and turns OFF (=0) when NG.

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: Task 1 standby	(Task 2 execution enabled)
	1: Executing Task 1	(Task 2 execution disabled)
FLAG2	0: Task 2 standby	(Task 1 execution enabled)
	1: Executing Task 2	(Task 1 execution disabled)

4. Flow chart

Processing flow



33C15-R7-00

Task 2 (subtask) runs in the same flow.

Program example

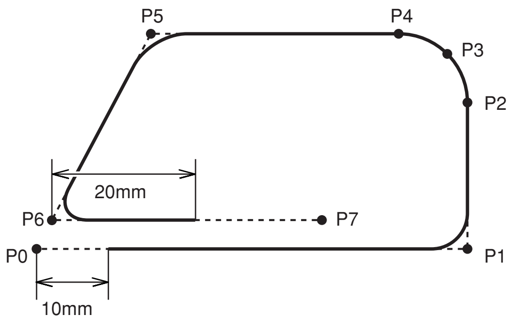
```
SAMPLE
<Main task>                 <Subtask>
FLAG1=0                     Program name:SUB_PGM
FLAG2=0
UPPOS=0.000
START <SUB_PGM>,T2                                      ······ Subtask Start
*L1:                        *S1:
WAIT DI2(2)=1               WAIT DI3(2)=1               ······· Part supply standby
WAIT FLAG2=0                WAIT FLAG1=0                ······ Other tasks waiting for standby status
FLAG1=1                     FLAG2=1                     ······ Exclusive control flag set
GOSUB *OPEN                 GOSUB *OPEN                 ······ Chuck open
MOVE P,P1,Z=UPPOS           MOVE P,P11,Z=UPPOS          ······ Move to part supply position
GOSUB *CLOSE                GOSUB *CLOSE                ······ Chuck close
MOVE P,P2,Z=UPPOS           MOVE P,P12,Z=UPPOS          ······ Move to testing device
GOSUB *OPEN                 GOSUB *OPEN                 ······ Chuck open
DRIVEI (3,-10000)           DRIVEI (3,-10000)           ······ Move axis 3 upward 10,000 pulses
FLAG1=0                     FLAG2=0                     ······ Exclusive control flag reset
DO2(0)=1                    DO3(0)=1                    ······ Testing device start
DELAY 100                   DELAY 100
DO2(0)=0                    DO3(0)=0
WAIT DI2(0)=1               WAIT DI3(0)=1               ······· Test completion standby
WAIT FLAG2=0                WAIT FLAG1=0                ······· Task completion standby
FLAG1=1                     FLAG2=1                     ······· Exclusive control flag set
MOVE P,P2,Z=UPPOS           MOVE P,P12,Z=UPPOS          ······· Move to testing device
GOSUB *CLOSE                GOSUB *CLOSE                ······ Chuck close
IF DI2(1)=1 THEN            IF DI3(1)=1 THEN            ······ Test
'GOOD                       'GOOD
WAIT DI4(2)=0               WAIT DI3(3)=0               ······· Part movement standby
MOVE P,P3,Z=UPPOS           MOVE P,P13,Z=UPPOS          ······· Move to OK parts position
ELSE                        ELSE
'NG                         'NG
WAIT DI2(4)=0               WAIT DI3(4)=0               ······· Part movement standby
MOVE P,P4,Z=UPPOS           MOVE P,P14,Z=UPPOS          ······· Move to NG parts position
ENDIF                       ENDIF
GOSUB *OPEN                 GOSUB *OPEN                 ······ Chuck open
DRIVEI (3,-10000)           DRIVEI (3,-10000)           ········ Move axis 3 upward 10,000 pulses
FLAG1=0                     FLAG2=0                     ········ Exclusive control flag reset
GOTO *L1                    GOTO *S1

          <common routine>
          Program name:COMMON
          *OPEN:
          DO2(1)=0
          DELAY 100
          RETURN
          *CLOSE:
          DO2(1)=1
          DELAY 100
          RETURN
```

2.5 Sealing

■ **Overview**

The following is an example for sealing a part.

Sealing



33C11-R9-00

■ **Precondition**

1. I/O signal

DO (20)	Valve open/close	1: Open / 0: Close

2. Positions of P0 to P7 are set by teaching.

SAMPLE

```
MOVE P,P0,Z=0
SPEED 40
PATH SET    ······················ Start of robot 1's path setting
PATH L,P1,DO(20)=1@10.000·········· Start of sealing
                                    at a 10mm position        ⎫
PATH L,P2                                                     ⎪
PATH C,P3,P4                                                  ⎪
PATH L,P5                                                     ⎪ Setting of the
PATH L,P6,S=30                                                ⎬ motion path
PATH L,P7,DO(20)=0@20.000·········· End of sealing at a       ⎪ (Robot does
                                    20mm position             ⎪  not move.)
PATH END    ······················ End of robot 1's           ⎪
                                    path setting              ⎭
PATH START Path motion of robot 1 is executed (Robot 1 starts moving from P0
                                    and stops at P7).
HALT
```

2.6 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 YRCX 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= 156.420243.910      0.000 ····· 0.000      0.000      0.000     [cr/lf]
```

SAMPLE

```
'INIT
   VCMD$="SDATA/X/Y"············ Command:Requiring the Movement position.
   P0=   0.000    0.000 ··· 0.000    0.000    0.000    0.000
          ····················· An initial position
'MAIN ROUTINE
   MOVE P, P0··················· Moves to the initial position.
*ST:
   SEND VCMD$ TO CMU············ Sends the command.
   SEND CMU TO P10·············· Receives the destination point to move
                                 to.
   MOVE P, P10·················· Moves to the reception position.
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.

2.7 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 YRCX 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=156.420, Y=243.910 [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

```
'INIT
   VCMD$="SDATA/X/Y"············ Command: Requiring the Movement
                                  position.
   P0=   0.000    0.000 ··· 0.000    0.000    0.000    0.000
          ····················· An initial position
   P11=100.000       100.000     0.000    0.000     0.000     0.000
          ····················· A reception position
'MAIN ROUTINE
   MOVE P,P0···················· Moves to the initial position.
*ST:SEND VCMD$ TO CMU ·········· Sends the command.
   SEND CMU TO VIN$············· Receives the Response:
                                  "X=156.420,Y=243.910".
   FOR I%=1 TO LEN(VIN$)-2
      IF MID$(VIN$,I%,2)="X=" THEN EXIT FOR
          ····················· If "X=", then exits from the roop.
   NEXT I%
   LOC1(P11)=VAL(MID$(VIN$,I%+2))
          ····················· Converts "X=" downward to numeric value
                                  and assigns to axis 1 of P11.
   FOR I%=1 TO LEN(VIN$)-2
      IF MID$(VIN$,I%,2)="Y=" THEN EXIT FOR
          ····················· If "Y=", then exits from the roop.
   NEXT I%
   LOC2(P11)=VAL(MID$(VIN$,I%+2))
          ····················· Converts "Y=" downward to numeric value
                                  and assigns to axis 2 of P11.
   MOVE P,P11··················· Moves to the reception position.
   GOTO *ST
```

SAMPLE

```
'INT
    VCMD$="SDATA/X/Y"
    VIN$=""
    VX$=""
    VY$=""
    P0=        0.000      0.000      0.000       0.000      0.000      0.000
    P11=     100.000    100.000      0.000       0.000      0.000      0.000
'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
HALT
```

Chapter 12
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 "YRCX 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: Operation-specific

Key operation

Operation type		Command	Option	Condition
Register program in the task		LOAD	<program name> ,Tn , p PGm (m: 1-100, n: 1-16, p: 1-64)	2
Program	Reset program Execute program Stop program	RESET RUN STOP	Tn <program name> PGm (m: 1-100, n: 1-16)	2
Program	Execute one line Skip one line Execute to next line	STEP SKIP NEXT	Tn <program name> PGm (m: 1-100, n: 1-16)	2
Program	Execute before specified line Skip before specified line	RUNTO SKIPTO	Tn ,k <program name> PGm (m: 1-100, n: 1-16, k: 1-9999)	2
Set break point		BREAK	<program name> (n, n, n,...), k PGm 0 0 (m: 1-100, n: 1-9999, k: 0/1)	2
Change manual movement speed		MSPEED	[robot number] k (robot number: 1-4, k: 1-100)	2
Move to absolute reset position		ABSADJ	[robot number] k, f (robot number: 1-4, k: 1-6, f: 0/1)	3
Absolute reset		MRKSET	[robot number] k (robot number: 1-4, k: 1-6)	3
Return-to-origin		ORGRTN	[robot number] k (robot number: 1-4, k: 1-6)	3
Change inching movement amount		IDIST	[robot number] k (robot number: 1-4, k: 1-10000)	2
Manual movement (inching)		INCH INCHXY INCHT	[robot number] km (robot number: 1-4, k: 1-6, m: +/-)	3
Manual movement (jog)		JOG JOGXY JOGT	[robot number] km (robot number: 1-4, k: 1-6, m: +/-)	3
Point data teaching		TEACH TCHXY	[robot number] m (robot number: 1-4, m: 0-29999)	2

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.

Utility

Operation type		Command	Option	Condition
Copy program		COPY	<program name1> / PGm TO <program name2> / PGm (m: 1-100)	2
Copy points "m - n" to point "k"		COPY	Pm-Pn TO Pk (m: 0-29999, n: 0-29999, k: 0-29999)	2
Copy point comments "m - n" to point comment "k"		COPY	PCm-PCn TO PCk (m: 0-29999, n: 0-29999, k: 0-29999)	2
Delete program		ERA	<program name> / PGm (m: 1-100)	2
Delete points "m - n"		ERA	Pm-Pn (m: 0-29999, n: 0-29999)	2
Delete point comments "m - n"		ERA	PCm-PCn (m: 0-29999, n: 0-29999)	2
Delete point names "m - n"		ERA	PNm-PNn (m: 0-29999, n: 0-29999)	2
Delete pallet "m"		ERA	PLm (m: 0-39)	2
Rename "program 1" to "program 2"		REN	<program 1> TO <program 2>	2
Check program syntax		SYNCHK	<program name> / PGm , k (m: 1-100, k: 1-100)	2
Compile sequence program		SEQCMPL		2
Change program attribute		ATTR	<program name> / PGm TO s (m: 1-100, s: RW/RO/H)	2
Setting main program		MAINPG	m (m: 1-100)	2
Initialize data	Program	INIT	PGM	3
	Point	INIT	PNT	3
	Point comment	INIT	PCM	3
	Point name	INIT	PNM	3
	Shift	INIT	SFT	3
	Hand	INIT	HND	3
	Pallet	INIT	PLT	3
	General Ethernet Port	INIT	GEP	3
	Input/output name	INIT	ION	3
	Area check output	INIT	ACO	3
	All data except parameters	INIT	MEM	3
	Parameter	INIT	PRM	3
	All data (MEM+PRM)	INIT	ALL	3
Initialize data	Communication parameter	INIT	CMU / ETH	3
Initialize data	Alarm history	INIT	LOG	3
Setting	Input data	INPUT	SET d / CAN / CLR (d: input data)	2
Buffer clear	Output message	MSGCLR		2
Change access level		ACCESS	k , pppppppp (k: 0/1, p: alphanumeric characters of 8 characters or less)	2
Setting password		SETPW		2
Setting Sequence execution flag		SEQUENCE	k (k: 0/1/3)	2
Reset alarm		ALMRST		2
Check or set date		DATE	yy/mm/dd (yy: 00-99, mm: 01-12, dd: 00-31)	2
Check or set time		TIME	hh: mm: ss (hh: 00-23, mm: 00-59. ss: 00-59)	2

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 k , pppppppp (k: 0/1, p: alphanumeric characters of 8 characters or less)	1
	Alarm status		ALM	
	Break point status		BREAK <program name> PGm (m: 1-100)	
	Last (Current) point number reference		CURPNT	
	Emergency stop status		EMG	
	Selected hand status		HAND [*robot number*] (robot number: 1-4)	
	Inching movement amount status		IDIST [*robot number*] (robot number: 1-4)	
	Input data		INPUT	
	Online/offline status		LINEMODE ETH CMU	
	Main program number		MAINPG	
	Remaining memory capacity		MEM	
	Mode status		MODE	
	Motor power status		MOTOR	
	Output message		MSG	
	Manual movement speed		MSPEED [*robot number*] (robot number: 1-4)	
	Return-to-origin status		ORIGIN [*robot number*] (robot number: 1-4)	
	Sequence program execution status		SEQUENCE	
	Servo status		SERVO [*robot number*] (robot number: 1-4)	
	Selected shift status		SHIFT [*robot number*] (robot number: 1-4)	
	Acquire task in RUN or SUSPEND status		TASKS	
	Task end condition		TSKECD Tk (k: 1-16)	
	Task operation status		TSKMON Tk (k: 1-16)	
	Version information		VER	
	Numerical data		*numerical expression*	
	Character string data		*character string expression*	
	Point data		*point expression*	
	Shift data		*shift expression*	
Read-out data		READ	*read-out file*	2
Write data		WRITE	*write file*	2

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.

Robot language independent execution

The Robot languages executable independently are the commands/functions with "✓" at "Online" column in Chapter 8 "robot language table".

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 k , pppppppp (k: 0/1, p: alphanumeric characters of 8 characters or less)	Acquire access level	1
	ALM	Acquire alarm status	
	BREAK <program name> PGm (m: 1-100)	Acquire break point status	
	CURPNT	Acquire Last (Current) point number reference	
	EMG	Acquire emergency stop status	
	HAND [*robot number*] (robot number: 1-4)	Acquire selected hand status	
	IDIST [*robot number*] (robot number: 1-4)	Acquire inching movement amount status	
	INPUT	Acquire input data status	
	LINEMODE ETH CMU	Acquire online/offline status	
	MAINPG	Acquire main program number	
	MEM	Acquire remaining memory capacity	
	MODE	Acquire mode status	
	MOTOR	Acquire motor power status	
	MSG	Acquire output message	
	MSPEED [*robot number*] (robot number: 1-4)	Acquire manual movement speed	
	ORIGIN [*robot number*] (robot number: 1-4)	Acquire return-to-origin status	
	SEQUENCE	Acquire sequence program execution status	
	SERVO [*robot number*] (robot number: 1-4)	Acquire servo status	
	SHIFT [*robot number*] (robot number: 1-4)	Acquire selected shift status	
	TASKS	Acquire task in RUN or SUSPEND status	
	TSKECD Tk (k: 1-16)	Acquire task end condition	
	TSKMON Tk (k: 1-16)	Acquire task operation status	
	VER	Acquire version	
	numerical expression	Acquire numerical data	
	character string expression	Acquire character string data	
	point expression	Acquire point data	
	shift expression	Acquire shift data	
^C (=03H)		Execution language interruption	1
ABSADJ	[*robot number*] k, f (robot number: 1-4, k: 1-6, f: 0/1)	Move to absolute reset position	3
ACCESS	k , pppppppp (k: 0/1, p: alphanumeric characters of 8 characters or less)	Change access level	2
ARMRST		Reset alarm	
ATTR	<program name> TO s PGm (m: 1-100, s: RW/RO/H)	Change program attribute	2
BREAK	<program name> (n, n, n,...), k PGm 0 0 (m: 1-100, n: 1-9999, k: 0/1)	Set break point	2

Command	Option	Meaning	Condition
COPY	<program name1> TO <program name2> PGm (m: 1-100)	Copy program	2
	Pm-Pn TO Pk (m: 0-29999, n: 0-29999, k: 0-29999)	Copy points "m - n" to point "k"	
	PCm-PCn TO PCk (m: 0-29999, n: 0-29999, k: 0-29999)	Copy point comments "m - n" to point comment "k"	
DATE	yy/mm/dd (yy: 00-99, mm: 01-12, dd: 00-31)	Check or set the date	2
ERA	<program name> PGm (m: 1-100)	Delete program	2
	Pm-Pn (m: 0-29999, n: 0-29999)	Delete points "m - n"	
	PCm-PCn (m: 0-29999, n: 0-29999)	Delete point comments "m - n"	
	PNm-PNn (m: 0-29999, n: 0-29999)	Delete point names "m - n"	
	PLm (m: 0-39)	Delete pallet "m"	
IDIST	[robot number] k (robot number: 1-4, k: 1-10000)	Change inching movement amount	3
INCH INCHXY INCHT	[robot number] km (robot number: 1-4, k: 1-6, m: +/-)	Manual movement (inching)	3
INIT	ACO	Initialize area check output)	3
	ALL	Initialize all data (MEM+PRM)	
	CMU	Initialize communication parameter (RS-232C)	
	ETH	Initialize communication parameter (Ethernet)	
	GEP	Initialize General Ethernet Port	
	HND	Initialize hand data	
	ION	Initialize input/output name	
	LOG	Initialize alarm history	
	MEM	Initialize all data except parameters	
	PCM	Initialize point comment data	
	PGM	Initialize program data	
	PLT	Initialize pallet data	
	PNM	Initialize point name	
	PNT	Initialize point data	
	PRM	Initialize parameter data	
	SFT	Initialize shift data	
INPUT	SET d CAN CLR (d: input data)	Sets the input data to the data request by the INPUT statement	2
JOG JOGXY JOGT	[robot number] km (m: 1-4, k: 1-6, m: +/-)	Manual movement (jog)	3
LOAD	<program name> ,Tn , p PGm (m: 1-100, n: 1-16, p: 1-64)	Register program in the task	2
MAINPG	m (m: 1-100)	Setting main program	2
MRKSET	[robot number] k (robot number: 1-4, k: 1-6)	Absolute reset	3
MSGCLR		Buffer clear Output message	1
MSPEED	[robot number] k (robot number: 1-4, k: 1-100)	Change manual movement speed	2

Command	Option	Meaning	Condition
NEXT	Tn *<program name>* PGm (m: 1-100, n: 1-16)	Execute program to next line	4
ORGRTN	[*robot number*] k (robot number: 1-4, k: 1-6)	Return-to-origin	3
READ	*read-out file*	Read-out data	2
REN	*<program 1>* TO *<program 2>*	Change program name from "1" to "2"	2
RESET	Tn *<program name>* PGm (m: 1-100, n: 1-16)	Reset program	2
RUN	Tn *<program name>* PGm (m: 1-100, n: 1-16)	Execute program	4
RUNTO	Tn *<program name>* ,k PGm (m: 1-100, n: 1-16, k: 1-9999)	Execute program before specified line	2
SEQCMPL		Compile sequence program	
SEQUENCE	k (k: 0/1/3)	Set sequence execution flag	2
SETPW		Setting password	
SKIP	Tn *<program name>* PGm (m: 1-100, n: 1-16)	Program: Skip one line	4
SKIPTO	Tn *<program name>* ,k PGm (m: 1-100, n: 1-16, k: 1-9999)	Program: Skip before specified line	2
STEP	Tn *<program name>* PGm (m: 1-100, n: 1-16)	Program: Execute one line	4
STOP	Tn *<program name>* PGm (m: 1-100, n: 1-16)	Stop program	2
SYNCHK	*<program name>* , k PGm (m: 1-100, k: 1-100)	Check program syntax	2
TEACH TCHXY	[*robot number*] m (robot number: 1-4, m: 0-29999)	Point data teaching	3
TIME	hh: mm: ss (hh: 00-23, mm: 00-59. ss: 00-59)	Check or set time	2
WRITE	*write file*	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> PGm	,Tn, p [cr/lf]

Response format

```
OK[cr/lf]
```

Values
m ....Program number: 1 to 100
n ....Task number: 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 <program name> PGm	[cr/lf]

Response format

```
OK[cr/lf]
```

Values
n ....Task number: 1 to 16
m ....Program number: 1 to 100

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 number: 1 to 16
m ....Program number: 1 to 100

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]
```

Values n .... Task number: 1 to 16
m .... Program number: 1 to 100

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

```
@STOP | Tn             | [cr/lf]
      | <program name> |
      | PGm            |
```

Command format

```
OK[cr/lf]
```

Values n .... Task number: 1 to 16
m .... Program number: 1 to 100

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 <program name> PGm	[cr/lf]

Response format

```
OK[cr/lf]
```

Values n ....Task number: 1 to 16
m ....Program number: 1 to 100

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 the next line

Command format		
@NEXT	Tn <program name> PGm	[cr/lf]

Response format

```
OK[cr/lf]
```

Values n ....Task number: 1 to 16
m ....Program number: 1 to 100

Meaning Executes the specified program to the next line. Executing @NEXT on the line in the GOSUB or in the CALL statement make the program execute and return through the sub-procedure processing, then stop at the next line.

MEMO

- This is a same processing as setting the breakpoint on the next line in the program currently suspended and executing the program (@RUN).
  @STEP stops the program at the beginning line of the sub-procedure called by GOSUB or CALL statement.

```
SAMPLE
Command:   @NEXT T3 [cr/lf] ·········· Executes the program in execution
                                       at T3 until the next line.
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 number: 1 to 16
m .... Program number: 1 to 100
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 14th line
                                       and stops at 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 number: 1 to 16
m .... Program number: 1 to 100
k .... Specified line number: 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 and stops
                                       at the 15th line.
Response:  OK [cr/lf]
```

7

8

9

10

11

12

13

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: 1 to 100
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]
```

11. Check program syntax

Command format

```
@SYNCHK  <program name>   ,k [cr/lf]
         PGm
```

Command format

```
RUN [cr/lf]
nnnn:gg.bbb [cr/lf]
nnnn:gg.bbb [cr/lf]
:
nnnn:gg.bbb [cr/lf]
nnnn:gg.bbb [cr/lf]
END [cr/lf]
```

Values

m .... Program number: 1 to 100
k .... Maximum number of error: 1 to 100
nnnn .... Line number where error occurred: 1 to 9999
gg.. Alarm group number
bbb .... Alarm classification number

Meaning

Checks syntax of the program specified by <*program name*> or program number.
If there are syntax errors in the specified program, line number where error occurred, alarm group number and alarm classification number are output. For details regarding alarm group number and alarm classification number, refer to the "YRCX Controller User's Manual" or "YRCX Controller Operator’s Manual".

SAMPLE

```
Command:   @SYNCHK PG1, 100 [cr/lf] ······· Sets a Maximum number of error
                                            at 100 and checks syntax of the
                                            program 1.
Response:  RUN [cr/lf]
           1:5.239 [cr/lf] ··········· Detects syntax errors "5.239:
                                            Illegal identifier" at 1th, 2nd,
                                            3rd and 8th lines.
           2:5.239 [cr/lf]
           3:5.239 [cr/lf]
           8:5.239 [cr/lf]
           6:5.222 [cr/lf] ··········· Detects syntax error "5.222 IF
                                            without ENDIF" at 6th line.
END [cr/lf]
```

12. Set main program

Command format

```
@MAINPG[cr/lf]
```

Response format

```
OK[cr/lf]
```

NOTE

• "Main program" corresponds conventional function "_SELECT" of YRC, etc.

Values m: Program number ....1 to 100

Meaning Specifies the program which is always selected when all programs are reset. When "0" is specified at the main program number or program specified at the main program number doesn't exist, the program that has been executed last (current program) in the task 1 is selected after resetting all programs.

SAMPLE

```
Command:   @MAINPG 1[cr/lf] ·········· Sets program number 1 at the main
                                       program.
Response:  OK[cr/lf]
```

13. Compile sequence program

Command format

```
@SEQCMPL[cr/lf]
```

Response format

```
RUN[cr/lf]
END[cr/lf]
```

Meaning Compiles the sequence program.
When the program named "SEQUENCE" doesn't exist or syntax errors exist in the program, an error message appears.
The execution program is created after successful termination of compiling and the letter "s" appears in Flag.
For details, refer to Chapter 7 "Sequence function".

SAMPLE

```
Command:   @SEQCMPL[cr/lf] ·········· Compiles the sequence program.
Response:  RUN[cr/lf]
           END[cr/lf]
```

2.2 MANUAL mode operation

1. Change the MANUAL mode speed

Command format

```
@MSPEED [robot number] k[cr/lf]
```

Response format

```
OK[cr/lf]
```

Values *robot number*........................1 to 4 (If not input, robot 1 is specified.)
k ....Manual movement speed: 1 to 100

Meaning Changes the manual mode movement speed of the robot specified by the <*robot number*>.

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 (If not input, robot 1 is specified.)
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[2] 100[cr/lf]
Response:  OK[cr/lf]
```

3. Change inching movement amount

Command format

```
@IDIST [robot number] mmmmm [cr/lf]
```

Response format

```
OK[cr/lf]
```

Values *Robot number*.................................1 to 4 (If not input, robot 1 is specified.)
mmmmm: inching movement amount....1 to 10000

Meaning Changes the inching movement amount of the robot specified by the <*robot number*>.

The unit of the movement amount may vary depending on the command.
INCH ...."pulse" units: 1 to 10000 pulse
INCHXY ...."mm" units: 0.001 to 10.000mm
INCHXT ...."mm" units: 0.001 to 10.000mm

SAMPLE

```
Command:   @IDIST[2] 100[cr/lf]
Response:  OK[cr/lf]
```

2.3 Alarm reset

Command format

```
@ALMRST [cr/lf]
```

Response format

```
RUN[cr/lf]
END[cr/lf]
```

Meaning Resets the alarm.
However, this command cannot be used for the alarms which require the restart of system. In this case, turn off the controller and turn it on again.

SAMPLE

```
Command:   @ALMRST [cr/lf]
Response:  RUN[cr/lf]
           END[cr/lf]
```

2.4 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]
```

7
8
9
10
11
12
13

2.5 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 for responding to a data request by INPUT statement of robot program.
The controller parameter "INPUT/PRINT using channel" should be set a current communication channel (CMU, ETH or iVY).

SET........Sets the data which is input to the variable when INPUT statement is executed.
CAN......Cancels the data request by INPUT statement.
CLR.......Clears the data specified @INPUT SET downward.

SAMPLE

```
<Online command>                         <Robot program>
@INPUT SET 10[cr/lf]
@INPUT SET 5[cr/lf]
OK[cr/lf]  INPUT A%[cr/lf]
@?MSG[cr/lf]                             PRINT A%[cr/lf]
10[cr/lf]
OK[cr/lf]

<Online command>                         <Robot program>
@INPUT SET 10[cr/lf]
OK[cr/lf]
@INPUT CLR[cr/lf]
OK[cr/lf]
@INPUT SET 5[cr/lf]
OK[cr/lf]  INPUT A%[cr/lf]
@?MSG[cr/lf]                             PRINT A%[cr/lf]
5[cr/lf]
OK[cr/lf]
```

2.6 Change access level

Command format

```
@ACCESS k , pppppppp [cr/lf]
```

Response format

```
OK[cr/lf]
```

Values k: Access level ....0: Maintainer level, 1: Operator level
pppppppp: Password.............Alphanumeric characters of 8 characters or less

Meaning Changes access level. If password is omitted, sets without password.
When changes access level to the maintainer level and entered password is incorrect, "6.235: Password error" will occur.

REFERENCE
- For details regarding access level, refer to the YRCX user's manual or operator's manual.

SAMPLE

```
Command:   @ACCESS 0,password [cr/lf]········ Sets "password" as password, and
                                              changes the level to "maintainer
                                              level".
Response:  OK [cr/lf]
```

7

8

9

10

11

12

13

2.7 Setting input data

Command format

```
@SETPW [cr/lf]
```

Response format

```
READY[cr/lf]
pppppppp[cr/lf]
kkkkkkkk[cr/lf]
nnnnnnnn[cr/lf]
[cr/lf]    ························line-feed
OK[cr/lf]
```

Values

pppppppp: old password (current password)........ Alphanumeric characters of 8 characters or less
kkkkkkkk: new password .... Alphanumeric characters of 8 characters or less
nnnnnnnn: new password (confirmation)............ Alphanumeric characters of 8 characters or less

Meaning

Changes the password for the access level changing to the maintainer level.
The current password is input for the old password, and the revised password is input for the new password and for the new password of confirmation. In the next line of the new password (confirmation), inserts line feeds only.
When input password as the old password is different from the current password or new password and new password (confirmation) are not same, "6.235: Password error" will occur.

REFERENCE

- For details regarding access level, refer to the YRCX user's manual or operator's manual.

SAMPLE

```
Command:   @SETPW[cr/lf
Response:  READY [cr/lf]
           oldpass [cr/lf] ··········· Inputs "oldpass" as old password.
           newpass [cr/lf] ··········· Inputs "newpass" as new password.
           newpass [cr/lf] ··········· Inputs "newpass" as new password
                                       (confirmation).
           [cr/lf] ··················· line-feed
           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 (If not input, robot 1 is specified.)
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 (If not input, robot 1 is specified.)

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.

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

```
k[cr/lf]
OK[cr/lf]
```

Values k: Access level ....0 to 1

Meaning Acquires the access level.

REFERENCE

• For details regarding access level, refer to the YRCX user's manual or operator's manual.

SAMPLE

```
Command:   @?ACCESS[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.5 Acquiring the break point status

Command format

@?BREAK	*<program name>* PGm	[cr/lf]

Response format

```
n{,n{,{...}}} [cr/lf]
OK [cr/lf]
```

Values n: Line number on which break point "n" is set ..... 1 to 9999
Program name... Program name intended to delete
m: Program number .... 1 to 100

Meaning Acquires the break point status.

SAMPLE

```
Command:   @?BREAK <TEST>[cr/lf]
Response:  12,35[cr/lf]
           OK[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 .... 0: MANUAL mode
1: AUTO mode (Control source: Programming box)
2: AUTO mode (Control source release)
-1: Restricted mode

Meaning Acquires the controller mode status.

SAMPLE

```
Command:   @?MODE[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.7 Acquiring the communication port status

Command format

@?LINEMODE	ETH CMU	[cr/lf]

Response format

```
k[cr/lf]
OK[cr/lf]
```

Values k .... 0: OFFLINE, 1: ONLINE

Meaning Acquires the specified communication port status.
ONLINE / OFFLINE commands allow to change a specified communication port to the "online" / "offline" mode, respectively.

SAMPLE

```
Command:   @?LINEMODE ETH [cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.8 Acquiring the main program number

Command format

```
@?MAINPG[cr/lf]
```

Response format

```
m[cr/lf]
OK[cr/lf]
```

Values m: Program number.............. 0 to 100
(If not registered in the main program, acquires 0.)

Meaning Acquires the program number which is registered in the main program.

SAMPLE

```
Command:   @?MAINPG[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.9 Acquiring the sequence program execution status

Command format

```
@?SEQUENCE[cr/lf]
```

Response format

```
1.  1,s[cr/lf]
    OK[cr/lf]
2.  3,s[cr/lf]
    OK[cr/lf]
3.  0[cr/lf]
    OK[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]
           OK[cr/lf]
```

7

8

9

10

11

12

13

3.10 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]
           OK[cr/lf]
```

3.11 Acquiring the tasks in RUN or SUSPEND status

Command format

```
@?TASKS[cr/lf]
```

Response format

```
n{,n{,{...}}}[cr/lf]
OK[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]
           OK[cr/lf]
```

3.12 Acquiring the tasks operation status

Command format

```
@?TSKMON Tk[cr/lf]
```

Response format

```
m,n,f,p[cr/lf]
OK [cr/lf]
```

Values k : Task number.. 1 to 16
m : Execution program number .... 1 to 100
n : Task execution line number.................. 1 to 9999
f : Each task status...................................... R: RUN
U: SUSPEND
S: STOP
W: WAIT
p : Priority level of each task .... 17 to 47

Meaning Acquires the status of specified task.

SAMPLE

```
Command:   @?TSKMON T3[cr/lf]
Response:  5,11,R,32[cr/lf]
           OK[cr/lf]
```

3.13 Acquiring the task end condition

Command format

```
@?TSKECD Tk[cr/lf]
```

Response format

```
gg.bbb[cr/lf]
OK[cr/lf]
```

Values k : Task number ....1 to 16
gg : Alarm group number of the task end condition
bbb : Alarm classification number of the task end condition

Meaning Acquires the specified task end condition.
For details about alarm group number and classification number of the task end condition, refer to YRCX user's or operator's manual.

MEMO
- When the specified task ends by error, acquires this alarm number.

SAMPLE

```
Command:   @?TSKECD T1[cr/lf] ········ Acquires the end condition of task 1.
Response:  1.5[cr/lf] ··········· The end condition of task 1: 1.5:
                                  Program ended by "HALT".
           OK[cr/lf]
```

7
8
9
10
11
12
13

3.14 Acquiring the shift status

Command format

```
@?SHIFT [robot number] [cr/lf]
```

Response format

```
m[cr/lf]
OK[cr/lf]
```

Values *Robot number* ....1 to 4 (If not input, robot 1 is specified.)
m: ....Shift number selected for the specified robot: 0 to 39
Shift not selected: -1

Meaning Acquires the shift status of the robot specified by the <*robot number*>.

SAMPLE

```
Command:   @?SHIFT[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.15 Acquiring the hand status

Command format

```
@?HAND [robot number] [cr/lf]
```

Response format

```
m[cr/lf]
```

Values *Robot number* ....1 to 4 (If not input, robot 1 is specified.)
m...........................Hand number selected for the specified robot: 0 to 31
Hand not selected: -1

Meaning Acquires the hand status of the robot specified by the <*robot number*>.

SAMPLE

```
Command:   @?HAND[cr/lf]
Response:  1[cr/lf]
           OK[cr/lf]
```

3.16 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]
```

3.17 Acquiring the alarm status

Command format

```
@?ALM[cr/lf]
```

Response format

```
gg.bbb[cr/lf]
OK[cr/lf]
```

Values

gg...Alarm group number
bbb ....Alarm classification number

Meaning Acquires the alarm which occurs in the controller.
For details regarding the alarm group number and alarm classification number, refer to the YRCX user's or operator's manual.

MEMO

- The requirable alarms are number 400 or more of alarm classification number. If multiple alarms occur, the alarm with larger alarm classification number (more serious alarm) is acquired.

SAMPLE

```
Command:   @?ALM[cr/lf]
Response:  12.600[cr/lf]
           OK[cr/lf]
```

7
8
9
10
11
12
13

3.18 Acquiring the emergency stop status

Command format

```
@?EMG[cr/lf]
```

Response format

```
k[cr/lf]
OK[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]
```

3.19 Acquiring the manual movement speed

Command format

```
@?MSPEED [robot number] [cr/lf]
```

Response format

```
k[cr/lf]
OK[cr/lf]
```

Values *Robot number*........................1 to 4 (If not input, robot 1 is specified.)
k: manual movement speed ...1 to 100 (unit: %)

Meaning Acquires the value of the manual movement speed specified by <*Robot number*>.

SAMPLE

```
Command:   @?MSPEED[cr/lf]
Response:  50[cr/lf]
           OK[cr/lf]
```

3.20 Acquiring the inching movement amount

Command format

```
@?IDIST [robot number] [cr/lf]
```

Response format

```
mmmmm[cr/lf]
OK[cr/lf]
```

Values *Robot number*.. 1 to 4 (If not input, robot 1 is specified.)
mmmmm: Inching movement amount .... 1 to 10000

Meaning Acquires the inching movement amount specified by <*Robot number*>.

SAMPLE

```
Command:   @?IDIST[2][cr/lf]
Response:  100[c/lf]
           OK[cr/lf]
```

3.21 Acquiring the last reference point number (current point number)

Command format

```
@?CURPNT[cr/lf]
```

Response format

```
k[cr/lf]
OK[cr/lf]
```

Values k: Current point number.........0 to 29999

Meaning Acquires the point number which is referred last. The current point number (the point number of last reference) is renewed by operations which uses the point data (point edit, for example).

MEMO

- The current point number is renewed by following operations: the point reference and the point setting movement by remote commands, the trace movement or teaching by programming box or SCARA-YRCX Studio, etc.

SAMPLE

```
Command:   @?CURPNT[cr/lf]
Response:  100[cr/lf]
           OK[cr/lf]
```

7

8

9

10

11

12

13

3.22 Acquiring the output message

Command format

```
@?MSG[cr/lf]
```

Response format

```
sssss ··· ssssss[cr/lf]
OK[cr/lf]
```

Values s: Message character string

Meaning Acquires one line of message which is input from the output message buffer of the controller by the PRINT statement, etc.

SAMPLE

```
Command:   @?MSG[cr/lf]
Response:  MESSAGE[cr/lf] ············ PRINT "MESSAGE" is executed in a
                                       program.
           OK[cr/lf]
```

MEMO

- For executing this command, it is required that the "INPUT/PRINT using channel" parameter is set at the port to execute command.
- When the output message buffer is empty, only "OK" is output as the response.

3.23 Acquiring the input data

Command format

```
@?INPUT[cr/lf]
```

Response format

```
d[cr/lf]
OK[cr/lf]
```

Values d: Input data

Meaning Acquires the input data by the INPUT statement.

SAMPLE

```
Command:   @?INPUT[cr/lf]
Response:  INPUT_SAMPLE[cr/lf]
           OK[cr/lf]
```

3.24 Acquiring various values

1. Acquiring the value of a numerical expression

Command format

```
@?numerical expression[cr/lf]
OK[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:  2.236067E01[cr/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]
OK[cr/lf]
```

Meaning Acquires the value (character string) of the specified character string expression.

SAMPLE

```
The case of 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 expression[cr/lf]
```

Response format

```
point data[cr/lf]
OK[cr/lf]
```

Meaning Acquires the value (point data) of the specified point expression.

SAMPLE

```
Command:   @?P1+WHRXY[cr/lf]
Response:  10.410 -1.600 52.150 3.000 0.000 0.000 0 0 0[cr/lf]
           OK[cr/lf]
```

4. Acquiring the value of a shift expression

Command format

```
@?shift expression[cr/lf]
OK[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:  25.000  12.600  10.000  0.000[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

```
RUN[cr/lf] ········ At movement start
END[cr/lf] ········ At movement end
```

Values

Robot number........................1 to 4 (If not input, robot 1 is specified.)
k ....Axis number: 1 to 6
f ....Movement direction / 0: + direction, 1: - direction

Meaning

Performs the absolute reset operation of the specified axis of the robot specified by the <*robot number*>.
This command is available only to axes whose return-to-origin method is set as "Mark".
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] ················ Movement start
           END[cr/lf] ················ Movement end
```

7
8
9
10
11
12
13

4.2 Return-to-origin operation

Command format

```
@ORGRTN [robot number] k[cr/lf]
```

Response format

```
RUN[cr/lf] ········ At movement start
END[cr/lf] ········ At movement end
```

Values *Robot number*........................1 to 4 (If not input, robot 1 is specified.)
k ....Axis number: 1 to 6

Meaning Performs the return-to-origin operation of the specified axis of the robot specified by the <*robot number*>.

SAMPLE

```
Command:   @ORGRTN 1[cr/lf]
Response:  RUN[cr/lf] ················ Movement start
           END[cr/lf] ················ Movement end
```

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

```
RUN[cr/lf] ········ At movement start
END[cr/lf] ········ At movement end
```

Values

Robot number........................1 to 4 (If not input, robot 1 is specified.)
k ....Axis number: 1 to 6
m ....Movement direction / +, -

Meaning

Manually moves (inching motion) the specified axis of the robot specified by the <*robot number*>.
The robot performs the same motion as when moved manually in inching motion with the programming box's 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] ················ Movement start
           END[cr/lf] ················ Movement end
```

7
8
9
10
11
12
13

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

```
RUN[cr/lf] ········ At movement start
END[cr/lf] ········ At movement end
```

Values

Robot number........................1 to 4 (If not input, robot 1 is specified.)
k ....Axis number: 1 to 6
m ....Movement direction / +, -

Meaning

Manually moves (jog motion) the specified axis of the robot specified by the <*robot number*>.
The robot performs the same motion as when holding down the programming box's 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 stop 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] ················ Movement start
           END[cr/lf] ················ Movement end
```

7 8 9 10 11 12 13

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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 ....1 to 100

Meaning Copies the program specified by <*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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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
```

7
8
9
10
11
12
13

3. Copying point comments

Command format

```
@COPY  PCmmmmm-PCnnnnn  TO  PCkkkkk[cr/lf]
```

Response format

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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>* PGn	[cr/lf]

Response format

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

Values *Program name* ....Program name to be erased (32 characters or less consisting of alphanumeric characters and underscore)
n: Program number ....1 to 100

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

Values m ....Pallet number to be erased: 0 to 39

Meaning Erases the PLm pallet data.

SAMPLE

```
Command:   @ERA PL1[cr/lf]
Response:  RUN [cr/lf] ··············· Process start
           END [cr/lf] ··············· Process end
```

6. Erasing hand

Command format

```
@ERA Hm [cr/lf]
```

Response format

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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

Command format

@REN	*<program name 1>* PGn	TO *<program name 2>* [cr/lf]

Response format

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

Values

Program name 1 .... Program name before renaming: shown with 32 characters or less consisting of alphanumeric characters and _ (underscore)

Program name 2 .... Program name after renaming: shown with 32 characters or less consisting of alphanumeric characters and _ (underscore)

n: Program number.... 1 to 100

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>* PGn	TO s [cr/lf]

Response format

```
OK[cr/lf]
```

Values

Program name .... Program name to change the attribute: shown with 32 characters or less consisting of alphanumeric characters and _ (underscore)

s: Attribute................ RW: Readable/writable
RO: Not writable (read only)
H: Hidden

n: Program number.... 1 to 100

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]
```

7
8
9
10
11
12
13

5.5 Initialization process

1. Initializing the memory area

Command format

```
@INIT memory area[cr/lf]
```

Response format

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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.
ION ....Initializes the input/output 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 GEP).
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

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

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 YRCX user's or operator's manual.

SAMPLE

```
Command:   @INIT CMU [cr/lf]
Response:  RUN [cr/lf] ················ Process start
           END [cr/lf] ················ Process end
```

3. Initializing the alarm history

Command format

```
@INIT LOG[cr/lf]
```

Response format

```
RUN[cr/lf] ········ At prosess start
END[cr/lf] ········ At prosess end
```

Meaning Initializes the alarm history.

SAMPLE

```
Command:   @INIT LOG[cr/lf]
Response:  RUN [cr/lf] ················ Process start
           END [cr/lf] ················ Process end
```

5.6 Data readout processing

Command format

```
@READ read-out file[cr/lf]
```

Response format

```
BEGIN [cr/lf]····················At process start
(Data output: The contents may vary depending on the read-out file.)
END [cr/lf]······················At process end
```

Values *Read-out file* .... Designates a read-out 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 <*read-out file*> TO CMU

Commands via Ethernet have the same meaning as the following command.

- SEND <*read-out file*> TO ETH

n NOTE

- For more information about files, refer to the earlier Chapter 10 "Data file description".

Type	Read-out file name	Definition format	
		All	Individual file
User memory	All file	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 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 (sensor or stroke-end)	MRF	———
	Machine reference (mark)	ARP	———
	System configuration information	CFG	———
	Controller	CNT	———
	Robot	RBT	———
	Driver	DRV	———
	Option board	OPT	———
	Self check	SCK	———
	Alarm history	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()
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] ········· Reads out all programs.
          @READ P100 [cr/lf] ········ Reads out the point 100.
          @READ DINM2(0) [cr/lf] ···· Reads out the input/output name
                                      of DI2(0).
```

5.7 Data write processing

Command format

```
@WRITE write file[cr/lf]
```

Response format

```
READY[cr/lf]······· Input request display
OK [cr/lf] ········ After input is completed
```

NOTE
- For more information about files, refer to the earlier Chapter 10 "Data file description".

Values *Write file*............................... Designates 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, refer to Chapter 10 "Data file description".

Type	Write file name	Definition format	
		All	Separate file
User memory	All file	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 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] ········ Writes the label specified
                                      parameter.
          @WRITE P100 [cr/lf] ······· Writes the point 100.
          @WRITE DINM2(0)[cr/lf] ········Writes the input/output name of
                                      DI2(0).
```

7

8

9

10

11

12

13

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]
```

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.

NOTE

- To change only the year or month, the slash (/) following it can be omitted.

  Example:

  To set the year to 2016, enter 16(cr/lf).

  To set the month to June, enter /06(cr/lf).

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 "5.202: Data error" occurs.

SAMPLE 1

```
To change only the day,
 //15[cr/lf] ···················· Day is set to 15th.
```

SAMPLE 2

```
Command:   @DATE 16/01/14[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 "5.202: Data error" occurs.

SAMPLE 1

```
To change only the minute,
 :20:[cr/lf] .................... Minute is set to 20.
```

SAMPLE 2

```
Command:   @TIME 10:21:35[cr/lf]
Response:  OK[cr/lf]
```

7 Individual execution of robot language

Command format

```
@robot language[cr/lf]
```

Response format 1

```
OK[cr/lf] or NG=gg.bbb [cr/lf]
```

Response format 2

```
RUN[cr/lf] or NG=gg.bbb[cr/lf]················· At process start
END[cr/lf] or NG=gg.bbb[cr/lf]················· At process end
```

Values

OK, END.. Command ended correctly.
NG.. An error occurred.
RUN .... Command starts correctly.
gg: Alarm group number .... 0 to 99
bbb: Alarm classification number...... 0 to 999

Meaning

Robot language commands can be executed.

- Only independently executable commands are 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 13

Appendix

1 Reserved word list

The words shown below are reserved for robot language and cannot be used as identifiers (variables, etc.).

A	DATE	HND	MOVET
ABS	DBP	HOLD	MRF
ABSADJ	DEC	HOLDALL	MRKSET
ABSRPOS	DECEL	**I**	MSG
ACC	DEF	IDIST	MSGCLR
ACCEL	DEGRAD	IF	MSPEED
ACCESS	DELAY	IMP	MTRDUTY
ACO	DI	INCH	**N**
ALL	DIM	INCHT	NAME
ALM	DIR	INCHXY	NEXT
ALMRST	DIST	INIT	NOT
AND	DO	INPUT	**O**
ARCHP1	DPM	INT	OFF
ARCHP2	DRIVE	ION	OFFLINE
ARM	DRIVEI	**J**	ON
ARMCND	DRV	JOG	ONLINE
ARMSEL	**E**	JOGT	OPEN
ARMTYP	ELSE	JOGXY	OPT
ARP	ELSEIF	JTOXY	OR
ARY	EMG	**L**	ORD
ASPEED	END	LEFT	ORGORD
ATN	ENDIF	LEFTY	ORGRTN
ATN2	EOF	LEN	ORIGIN
ATTR	EQV	LET	OUT
AXWGHT	ERA	LINEMODE	OUTPOS
B	ERL	LO	**P**
BIN	ERR	LOAD	P
BREAK	ERROR	LOC1	PATH
C	ETH	LOC2	PC
CALL	ETHSTS	LOC3	PCM
CASE	EXIT	LOC4	PDEF
CFG	EXITTASK	LOC5	PGM
CHANGE	**F**	LOC6	PGMTSK
CHGPRI	FN	LOCF	PGN
CHR	FOR	LOG	PLT
CLOSE	FREE	LSHIFT	PMOVE
CMU	**G**	**M**	PNM
CNT	GEP	MAINPG	PNT
CONT	GEPSTS	MCHREF	PPNT
CONTPLS	GO	MEM	PRINT
COPY	GOSUB	MID	PRM
COS	GOTO	MO	PSHFRC
CURPNT	**H**	MOD	PSHJGSP
CURTQST	HALT	MODE	PSHMTD
CURTRQ	HALTALL	MOTOR	PSHRSLT
CUT	HAND	MOVE	PSHSPD
D	HEX	MOVEI	PSHTIME

PUSH	SET	SWI	WEIGHT
PWR	SETGEP	SYNCHK	WEIGHTG
R	SETPW	**T**	WEND
RADDEG	SFT	TAG	WHERE
RBT	SGI	TAN	WHILE
READ	SGR	TASKS	WHRXY
REF	SHARED	TCHXY	WRITE
REM	SHIFT	TCOUNTER	**X**
REN	SI	TEACH	XOR
RESET	SID	THEN	XY
RESTART	SIN	TIM	XYTOJ
RESUME	SIW	TIME	**Y**
RETURN	SKIP	TIMER	YZ
RIGHT	SKIPTO	TO	**Z**
RIGHTY	SO	TOLE	ZX
RSHIFT	SOD	TORQUE	
RUN	SOW	TSKECD	
RUNTO	SPEED	TSKMON	
S	SQR	TSKPGM	
S	START	**V**	
SCK	STEP	VAL	
SELECT	STOP	VAR	
SEND	STOPON	VEL	
SEQCMPL	STR	VER	
SEQUENCE	SUB	**W**	
SERVO	SUSPEND	WAIT	

Because the following names are used as system variable names, they cannot be used at the beginning of other variable names (n: numeric value).

ARMSEL	CHGWRK	CLOSE	CREWRK
CURTQST	ETHSTS	GEPSTS	HALTALL
HOLDALL	MOTOR	MOVET	MTRDUTY
OPEN	PGMTSK	PGN	PSHFRC
PSHJGSP	PSHMTD	PSHRSLT	PSHSPD
PSHTIME	PUSH	SETGEP	TSKPGM
WRKDEF	WEIGHTG		

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 Changes from conventional models

1 Program name

For YRCX, the following two program names which have been special for conventional models (YRC, etc.) don't have a special meaning.

A) FUNCTION
B) _SELECT

A) FUNCTION

In conventional models (YRC, etc.), "FUNCTION" has been special program for registering a user function. YRCX doesn't have a user function and "FUNCTION" doesn't have a special meaning.

B) _SELECT

In conventional models (YRC, etc.), the "_SELECT" program has been selected and executed every time robot programs were reset.
In YRCX, the program specified at the main program number (or the program executed last if there is no specified program there) is selected and executed when robot programs are reset.
For details regarding the main program, refer to "12. Set main program" in "2.1 Program operations" in Chapter 12.

2 Multiple Robot Control

In conventional models (YRC, etc.), robot has consisted of a main group (one main robot, main auxiliary axes) and a sub group (one sub robot, sub auxiliary axes).
In YRCX, robot consists of robot 1 to 4 (normal axes, auxiliary axes).
Due to this change, commands for each group have changed to ones for each robot.
For details regarding the command for each robot, refer to "2. Command list with a robot setting" in Chapter 5 of this manual for YRCX, and regarding the command for each group, refer to "Command list for each group" of the programming manual for conventional models (YRC, etc.), respectively.

SAMPLE

Command for each group: conventional model (YRC, etc.)

```
MOVE P, P1··················· Axes of a main group move to the
                              position specified by P1.
MOVE2 P, P5·················· Axes of a sub group move to the
                              position specified by P5.
```

Command for each robot: YRCX

```
MOVE P, P1··················· Axes of the robot 1 move to the
                              position specified by P1.
MOVE[2] P, P5················ Axes of the robot 2 move to the
                              position specified by P5.
```

MEMO

- The command with robot setting can be omitted a robot number. If it is omitted, robot 1 is specified.

3 Multi-tasking

The differences between YRCX and conventional models (YRC, etc.) are shown below.

	Conventional models	YRCX
Maximum number of task	8	16
Priority	17 to 47	1 to 63
Task definition	During the program	In another program
Starting tasks	Task is assigned in Task 1 automatically and placed in RUN status	Task is assigned in a specified task number and placed in RUN status
Command execution for Task 1 (restart, etc.)	Not executable	Executable

For details regarding the multi-tasking, refer to Chapter 6 "Multi-tasking" in this manual or in a programming manual for conventional models (YRC, etc.).

4 Robot Language

1. In YRCX, the robot languages shown below are added to ones of conventional models (YRC, etc.).

ARMSEL	CHGWRK	CLOSE	CREWRK
CURTQST	ETHSTS	GEPSTS	HALTALL
HOLDALL	MOTOR	MOVET	MTRDUTY
OPEN	PGMTSK	PGN	PSHFRC
PSHJGSP	PSHMTD	PSHRSLT	PSHSPD
PSHTIME	PUSH	SETGEP	TSKPGM
WRKDEF	WEIGHTG		

For details regarding the robot Language, refer to Chapter 8 "Robot Language Lists".

2. These robot languages for conventional models (YRC, etc.) became unavailable in YRCX.

ABSINIT	ABSINIT2	ABSRST	ABSRPOS2
ACCEL2	ARMCND2	ARMTYP2	ASPEED2
AXWGHT2	CHANGE2	CURTRQ2	DECEL2
DECLARE	DRIVE2	DRIVEI2	HAND2
JTOXY2	LEFTY2	MCHREF2	MOVE2
MOVEI2	ORGORD2	OUTPOS2	PMOVE2
RIGHTY2	SERVO2	SHIFT2	SPEED2
TOLE2	TORQUE2	TRQSTS	TRQSTS2
TRQTIME	TRQTIME2	WAIT ARM2	WEIGHT2
WHERE2	WHRXY2	XYTOJ2	_SYSFLG

For details regarding the robot Language, refer to "Robot Language Lists" of a programming manual for conventional models (YRC, etc.).

5 Online commands

1. In YRCX, the online commands shown below are added to ones of conventional models (YRC, etc.).

RUNTO	SKIPTO	MRKSET	IDIST
INCHXY	INCHT	JOGXY	JOGT
TCHXY	SYNCHK	SEQCMPL	LOAD
MAINPG	MSGCLR	SETPW	ALMRST
? ALM	? CURPNT	? IDIST	? INPUT
? LONEMODE	? MAINPG	? MODE	? MSG
? MSPEED	? TSKECD		

For details regarding the online commands, refer to Chapter 12 "Online commands".

2. These online commands for conventional models (YRC, etc.) became unavailable in YRCX.

AUTO	EMGRST	EXELV	MANUAL
? ARM	? CONFIG	? EXELVL	? OPSLOT
? SELFCHK	? WHRXYEX		

For details regarding the online commands, refer to "Online commands" of a programming manual for conventional models (YRC, etc.).

6 Data file

In YRCX, the data files shown below are added to ones of conventional models (YRC, etc.).

1. Point name file
2. General Ethernet port file
3. Input/output name file
4. Area check output file
5. System configuration information file
6. Version information file
7. Option board file
8. Self check file
9. Remaining memory size file

For details regarding the data files, refer to Chapter 10 "Data file description".

MEMO

- "Alarm history file" replaced "Error message history file" and "Error message history details file" of conventional models.
- In YRCX, the point number ranges from 0 to 29999 (0 to 9999: Conventional models).

Index

Index

A

B

C

D

E

R

S

T

U

V

W

X

Revision history

A manual revision code appears as a suffix to the catalog number on the front cover manual.

Cat. No. I232E-EN-02

Revision code

The following table outlines the changes made to the manual during each revision.

Revision code	Date	Description
01	June 2016	Original production
01A	February 2018	Small corrections
02	April 2020	A new section “140. WEIGHTG” has been included to the “Chapter 8. Robot Language Lists”. The sections “17. CURTQST”, “18. CURTRQ”, “65. MTRDUTY”, “66. OFFLINE”, “70. ONLINE” and “94. PUSH” from the “Chapter 8. Robot Language Lists” were updated. A new section “9. Work definition file” has been included to the “Chapter 10. Data file description”. The section “12. Input/output name file” from the “Chapter 10. Data file description” was updated.

OMRON

Authorized Distributor:

Printed in Europe