Robostar Robot Controller

N2 Series
Programming Manual

Version: N2-OM-E04
Issued Data: May 07, 2020

Robostar

Copyright 2020, ROBOSTAR Co., Ltd. All right reserved.

The copyright of this instruction manual belongs to Robostar Co., Ltd.
no part of this manual can be used in any form or by any means without the permission of Robostar Co., Ltd.

Specifications are subject to change without prior notice.

Warranty Policy

All products from Robostar Co., Ltd. are manufactured under strict quality management and 12 months of warranty from the production date is provided for all products. During the warranty period, Robostar provides free services only for the products failure due to faults of Robostar or problems caused by any defect in design or manufacturing under normal operation.

No free service applies to the following cases.

(1) Out of warranty period
(2) Product failures due to unauthorized repair, modification or other mishandling of products without permission from Robostar
(3) Product failures due to use of wrong components or grease other than specified by Robostar
(4) Product failures due to force majeure such as fire, calamity, earthquake or damages by storm and flood
(5) Product failures due to use in terrible circumstances beyond the environmental conditions specified by Robostar such as flooding
(6) Product failures due to depletion of consumables
(7) Product failures due to violation of instructions for repair, maintenance & inspection guidelines specified in either the operational manual or maintenance manual
(8) Damages other than the cost for a robot repair

Address & Contact for Robostar

- HQ & Factory
  700, Suin-ro, Sangnok-gu, Ansan-City,
  Gyeonggi-do, Republic of South Korea
  (15523)

- Request for service or Questions for products
  - Sales request
  TEL. 031-400-3600
  FAX. 031-419-4249
  - Service request
  TEL. 1588-4428

- Suwon Factory
  37, Saneop-ro 155beon-gil,
  Gwonseon-gu, Suwon-City,
  Gyeonggi-do,
  Republic of South Korea (16648)

Robostar
www.robostar.co.kr

Table of Contents

Chapter 1 Program Design 11
1. Basic Structure of Program 11
1.1 Declaration of Main Function 12
1.2 Declaration of Common Function 12
2. Program Call & Sub-routine 13
3. Multithreading 14
Chapter 2 Construction Elements of Language & Syntax 16
1. Construction Elements of Language 16
2. Name 16
3. Variable 16
3.1 Global Variable 17
3.2 Local Variable 18
3.3 System Variable 19
4. Function 19
5. Branch Point 19
6. Program 19
7. Type and Conversion of Data 20
7.1 Data Type 20
7.2 Data Type Conversion 23
8. Constant 24
9. Expression and Operator 25
9.1 Priority in Operators 25
9.2 Assignment Operator 25
9.3 Arithmetic Operator 26
9.4 Relational Operator 26
9.5 Logical and Bit Operators 26
9.6 Array 27
9.7 Use of String 27
9.8 Use of Positional Variable 28
Chapter 3 Command List 30
1. Variable & Constant 30
1.1 Common variable 30
1.2 System Variable 31
1.3 Constant 31
2. Preprocessor and Comment 32
3. Operator 32
4. Program Control 33
5. Robot Motion 34
6. Robot Motion Condition 35
7. Input & Output Control 37

7.1 General Digital Input & Output 37
7.2 Field Bus Input & Output 38
7.3 CNET Input/output 39
8. Multithreading Control 41
9. Function 42
9.1 Coordinate Conversion 42
9.2 Mathematics 45
9.3 String 46
10. System 48
10.1 Error 48
10.2 Status Information 49
10.3 Log and Data Storage 50
11. Commands dedicated to System Mode 51
12. Commands TCP IP Client 52
Chapter 4 Commend Reference 53
1. Format of Command Reference 53
2. Variable 54
2.1 INT 54
2.2 REAL 55
2.3 DEFSTR 56
2.4 POS 57
2.5 XPOS 58
2.6 UPOS 60
2.7 Program Position P 62
2.8 Global Integer I 63
2.9 Global Real F 64
2.10 Global Position GP 65
2.11 TMR 66
2.12 MVR 67
2.13 HERE 69
2.14 HERE_REF 70
3. Preprocessor & Comments 71
3.1 INCLUDE 71
3.2 Comment 72
4. Operator 73
4.1 Assignment Operator 73
4.2 Arithmetic Operator 74
4.3 Relational Operator 75
4.4 Logical Operator 76
4.5 Bit Operator 77
4.6 Parenthesis 79

5. Program Control 80
5.1 MAIN ··· EOP (Program Start/End) 80
5.2 FUNC ··· END (Program Start/End) 81
5.3 RET 85
5.4 FOR ··· TO ··· [BY] ··· NEXT (Iterative Execution Statement) 86
5.5 WHILE ··· ENDWL (Conditional Iterative Execution Statement) 87
5.6 IF ···THEN ··· ENDIF (Conditional Branch Command) 88
5.7 LABL ··· GOTO (Branch Command) 89
5.8 CONTINUE 90
5.9 BREAK 91
5.10 JCALL (Program Call) 92
5.11 STOP (Robot Stop) 93
5.12 EXIT (JOB Program Stop) 94
6. Robot Motion 95
6.1 SVON (Servo on) 95
6.2 SVOF (Servo off) 96
6.3 JMOV (PTP Movement) 97
6.4 LMOV (Linear Interpolation Movement) 100
6.5 CMOV (Circular Interpolation Movement) 103
6.6 AMOV (Arc Interpolation Movement) 105
7. Robot Motion Condition 107
7.1 WITH ··· ENDWT (Simultaneous Processing Commands) 107
7.2 VEL (Axis Moving Velocity Setting) 109
7.3 ACC (Axis Acceleration Time Setting) 110
7.4 DEC (Axis Deceleration Time Setting) 111
7.5 FOS (Setting Factor of smooth path) 112
7.6 DLAY (Delay Time Setting) 116
7.7 OFFS (Offset Designation) 117
7.8 OFFSCLR (Offset Clear) 119
7.9 LIMT (Axis Moving Range Limitation) 120
7.10 TRQ (Collision Detection Command) 121
7.11 GAIN (Converting Gain Group) 122
7.12 IPE (Setting In-position Allowable Time) 123
8. Input/Output Control 124
8.1 IN (Digital Bit Input) 124
8.2 OUT (Digital Bit Output) 126
8.3 PIN (Digital Port Input) 128
8.4 POUT (Digital Port Output) 130
8.5 CIN (Field Bus Bit Input) 132
8.6 COUT (Field Bus Bit Output) 134

8.7 CBIN (Field Bus Byte Input) 136
8.8 CBOUT (Field Bus Byte Output) 138
8.9 CDIN (Inputting Fieldbus Double Word) 139
8.10 CDOUT (Outputting Fieldbus Double Word) 140
8.11 CWIN (Field Bus Word Input) 141
8.12 CWOUT (Field Bus Word Output) 142
8.13 CFIN (Field Bus Double Word Input) 144
8.14 CFOUT (Field Bus Double Word Output) 145
8.15 XIN (Inputting CNET bit) 146
8.16 XOUT (CNET bit output) 147
8.17 XBIN (Inputting CNET Byte) 148
8.18 XBOUT (Outputting CNET Byte) 149
8.19 XWIN (Inputting CNET Word) 150
8.20 XWOUT (Outputting CNET Word) 151
8.21 XDIN (Inputting CNET Double Word) 152
8.22 XDOUT (Outputting CNET Double Word) 153
8.23 XFIN (Inputting CNET real number) 154
8.24 XFOUT (Outputting CNET real number) 155
9. Multithreading Control 156
9.1 CTHREAD (Thread Creation) 156
9.2 ETHREAD (Thread End) 158
9.3 TH_STATE (Thread Status Reading) 159
9.4 TH_SUSPEND (Thread Stop) 161
9.5 TH_RESUME (Thread Re-Start) 162
10. Function 163
10.1 TOOL (Tool Coordinate System Selection) 163
10.2 USER (User Coordinate System Selection) 164
10.3 JTOB (Coordinate Data Calculation, JOINT → BASE) 165
10.4 BTOJ (Coordinate Data Calculation, BASE → JOINT) 166
10.5 JTOU (Coordinate Data Calculation, JOINT → USER) 167
10.6 UTOJ (Coordinate Data Calculation, USER → JOINT) 168
10.7 BTOU (Coordinate Data Calculation, BASE → USER) 169
10.8 UTOB (Coordinate Data Calculation, USER → BASE) 170
10.9 RELBASE (Relative move based on Base Coordinate System) 171
10.10 RELTOOL (TOOL Coordinate System Based Relative Movement) 173
10.11 RELUSER (Relative move based on User Coordinate System) 175
10.12 GBASEDISP (Displacement detection based on Base Coordinate System) 177
10.13 GTOOLDISP (Displacement detection based on Tool Coordinate System) 179
10.14 GUSERDISP (Displacement detection based on User Coordinate System) 181
10.15 ABS (Absolute Value) 183

10.16 DEG (Conversion of Radian to Angle) 184
10.17 RAD (Conversion of Angle to Radian) 185
10.18 POW (Exponential Function) 186
10.19 RND (Round-off) 187
10.20 EXP (Exponential e^x) 188
10.21 LN (Natural Logarithm) 189
10.22 LOG (Common Logarithm) 190
10.23 SQRT (Square Root) 191
10.24 SIN (Sine Function) 192
10.25 ASIN (Arc Sine Function) 193
10.26 COS (CoSine Function) 194
10.27 ACOS (Arc CoSine Function) 195
10.28 TAN (Tangent Function) 196
10.29 ATAN (Arc Tangent Function) 197
10.30 ATAN2 (The 2^{nd} Arc Tangent Function) 198
10.31 MIN (Minimum Value) 199
10.32 MAX (Maximum Value) 200
10.33 RAND (Random Number Creation) 201
10.34 ASC (Returning of First Character Code in String) 202
10.35 BINS (Conversion of Binary Number to String) 203
10.36 CHR (Conversion of Integer into ASCII Text) 204
10.37 FLUSH (Clearing Communication Input/output Buffer) 205
10.38 FTOS (Conversion of Integer/Real Number Value into String) 206
10.39 HTOS (Conversion of Integer into Hexadecimal String) 207
10.40 SLEFT (Left String Extraction) 208
10.41 SLEN (Returning of String Length) 209
10.42 SMID (Extraction of String in Specified Position) 210
10.43 SPOS (Returning of Start Position in String Matching) 211
10.44 SRIGHT (Extraction of Right Side String) 212
10.45 STRIN (Receiving of Communication String) 213
10.46 STROUT (Sending of Communication Strings) 215
10.47 SVAL (Conversion of String into Number) 217
11. System 218
11.1 SETERR (SYSTEM EMG Alarm Occurrence) 218
11.2 RESET (Alarm Clear) 219
11.3 RERROR (Returning of Last Alarm Code) 220
11.4 RERRCNT (Returning of Alarm Count Occurred) 221
11.5 RERRCODE (Returning of Alarm Coe of Index Number) 222
11.6 RERRTEXT (Returning of Alarm Content) 223

11.7 READ_DERR (Returning of Driver Alarm Code) 224
11.8 RSTATE (Acquiring of Robot Status Information) 225
11.9 SWLIMIT (Reading of S/W LIMIT Parameter Value) 227
11.10 READ_RPM (Reading of RPM Value) 228
11.11 READ_LOAD (Load Rate Reading) 229
11.12 READ_TRQMAX (Reading of Max Torque Value) 230
11.13 READ_VEL (Reading of Velocity) 231
11.14 READ_TRQ (Reading of Torque Value) 232
11.15 READ_MTEMP (Reading of Motor Encoder temperature Value) 233
11.16 READ_CTEMP (Reading of controller Internal Temperature Value) 234
11.17 PRINT (Monitoring Log) 235
11.18 WLOG (User Log) 237
11.19 CLOG (Communication Log) ※ Not currently supported. 239
11.20 WGPNT (Global Point Variable File Store) 239
11.21 WGINT (Global Variable File Store) 240
11.22 WGFLT (Global Real Number Variable File Store) 241
11.23 TACT_START (Setting Time measurement start position) 242
11.24 TACT_STOP (Setting Time measurement end position) 243
12. System Mode Dedicated Commands 244
12.1 PSEL (Program Selection) 244
12.2 PSTART (Program Start) 246
12.3 PSTOP (Program Execution Stop) 247
12.4 PSTATE (Program State Check) 248
12.5 PRESTART (Program Re-start) 250
12.6 PEXIT (Program Clear) 252
13. External Communications 254
13.1 NET_OPEN (Creating TCP I/P Client Socket and requesting for connection) 254
13.2 NET_CLOSE (Termination of TCP I/P Communication) 256
13.3 NET_STATUS (Checking TCP I/P Communication status) 257
13.4 NET_KEEPALIVE (Option setting to check the TCP I/P Client Session) 258
13.5 NET_FD (TCP I/P Server FD Return) 260
13.6 NET_STRIN (Receiving Communication String) 261
13.7 NET_STROUT (Sending Communication String) 262
Chapter 5 Revision 263
A. Document Reference 264
B. Hazard Stages & Signs 265
C. Command List (Alphabetical Order) 266

Table of Figures

Figure 1-1 Basic Structure of Program 11
Figure 1-2 Example of Main Function Creation 12
Figure 1-3 Example of Common Function Creation 12
Figure 1-4 Difference between Subroutine Call and Program Call 13
Figure 1-5 Difference between Single-Thread and Multi-Thread 14
Figure 1-6 Thread Structure in System Mode & Robot Mode 15
Figure 1-7 Example of Multi-Threading Execution 15
Figure 4-1 Format of Command Reference 53
Figure 4-2 Example of Comparison between Common Use and MVR Use 67
Figure 4-3 Example of Function Motion 82
Figure 4-4 PTP Movement Example 97
Figure 4-5 Linear Interpolation Movement Example 100
Figure 4-6 Circular Interpolation Example 103
Figure 4-7 Arc Interpolation Example 105
Figure 4-8 WITH Statement Use Example 107
Figure 4-9 FOS Rate 113
Figure 4-10 Basic Examples of using FOS 114
Figure 4-11 Example of Signal Output while Moving 114
Figure 4-12 Example of checking Signal Input while moving 115
Figure 4-13 OFFS Command Example 117

Chapter 1 Program Design

1. Basic Structure of Program

Robostar Robot Language (RRL) is an exclusive language that was developed by Robostar for creating Robot Job Programs in the system.

Programs are typically created in the structure shown below. This is a recommendation not a mandatory.

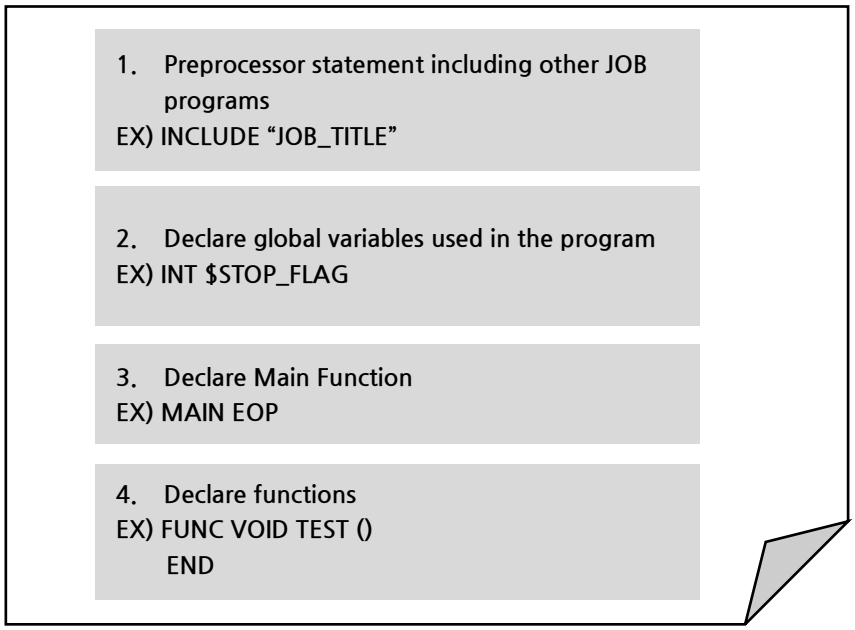


Figure 1-1 Basic Structure of Program

1.1 Declaration of Main Function

A program with Robostar Robot Language must have one Main Function. As shown in Figure 1-2 Example of Main Function Creation, the end of Main Function must be followed by the EOP command that notices the end of a program after declaring the Main Function.

※ Refer to chapter 4, section 5.1 MAIN … EOP (Program Start/End) for detail description and examples.

1	MAIN	1. Declare Main Function Start
2	SVON	2. Servo on
3	JMOV GP1	3. Motion Execution & Time Delay
4	DLAY 2000	
5	JMOV GP2	
6	DLAY 2000	
7	SVOF	7. Servo off
8	EOP	8. Main Function End

Figure 1-2 Example of Main Function Creation

1.2 Declaration of Common Function

Besides Main Function, it is possible to call other functions after declaring it.

※ Refer to chapter 4, section 5.2 FUNC … END(Function Start/End) for detail description and examples.

Syntax

```
FUNC <variable> <function name> (parameter, parameter, …)
                ⋮
    (RET)
END
```

1	FUNC INT TEST()	1. Declare Function Start, Return Value, Function Name, Parameter
2	SVON	2. Servo on
3	JMOV GP1	3. Motion Execution & Time Delay
4	DLAY 2000	
5	JMOV GP2	
6	DLAY 2000	
7	SVOF	7. Servo off
8	RET 1	8. Return Value return
9	END	9. Function End

Figure 1-3 Example of Common Function Creation

2. Program Call & Sub-routine

When a specific motion or function are needed to be used repeatedly, you can create a routine which repeats the motion or function in program. And then it can be called and used in MAIN or other routines. This kind of method is called a sub-routine. if this routine is saved in other separate program and called, it is called a program call.

If a series of job is composed as sub-routine or a unit of other programs, there's no need to write the same contents repeatedly. this is very efficient to reduce time for writing program and to improve the easiness of program reading

A sub-routine must be included in the same file together with a calling program. When the sub-routine ends. the process returns to the next line of the program that called the sub-routine. A sub-routine can be called by other sub-routines. Global variable can be used and Argument can be transferred

Independent individual file can be called from various programs. When calling a program, a name of program must be nominated in JCALL statement. if JCALL statement calls programs, execution line moves to the called program. After executing the last line of the called program, the execution line returns to the next line of the calling program. The called program can call another program. However it cannot call itself and the program which calls it. It can generally use the global variable but cannot transfer the arguments.

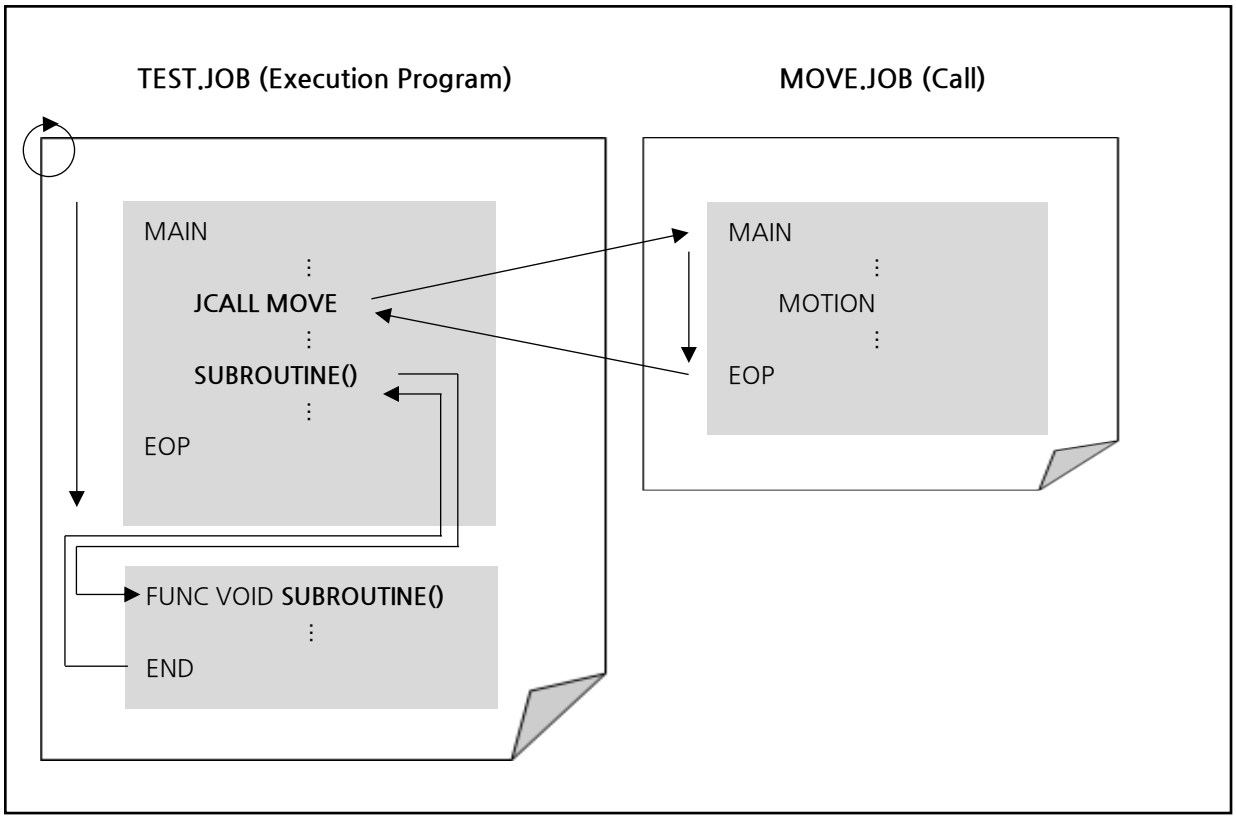


Figure 1-4 Difference between Subroutine Call and Program Call

3. Multithreading

Robostar Robot Language (RRL, Robostar Robot Language) provides the multi-threading function that one program processes multiple jobs simultaneously not one job at a time.

When processing two jobs with the single thread that processes one job at a time, it processes two jobs serially, one after another. However, the multi-thread method enables to process multiple jobs simultaneously. In case that a Robot is controlled at the multi-threading, the performance can be optimized by executing the individual JOB program or functions in parallel. Even if one part of the program is stopped or along job is executed, other programs can continue their jobs, thus improving the responsiveness to users.

※ When two different commands use the same data, a caution is required because executions result is not predicted or execution sequence is not secured.
※ Commands to drive motors such as JMOV and LMOV must be used in one thread

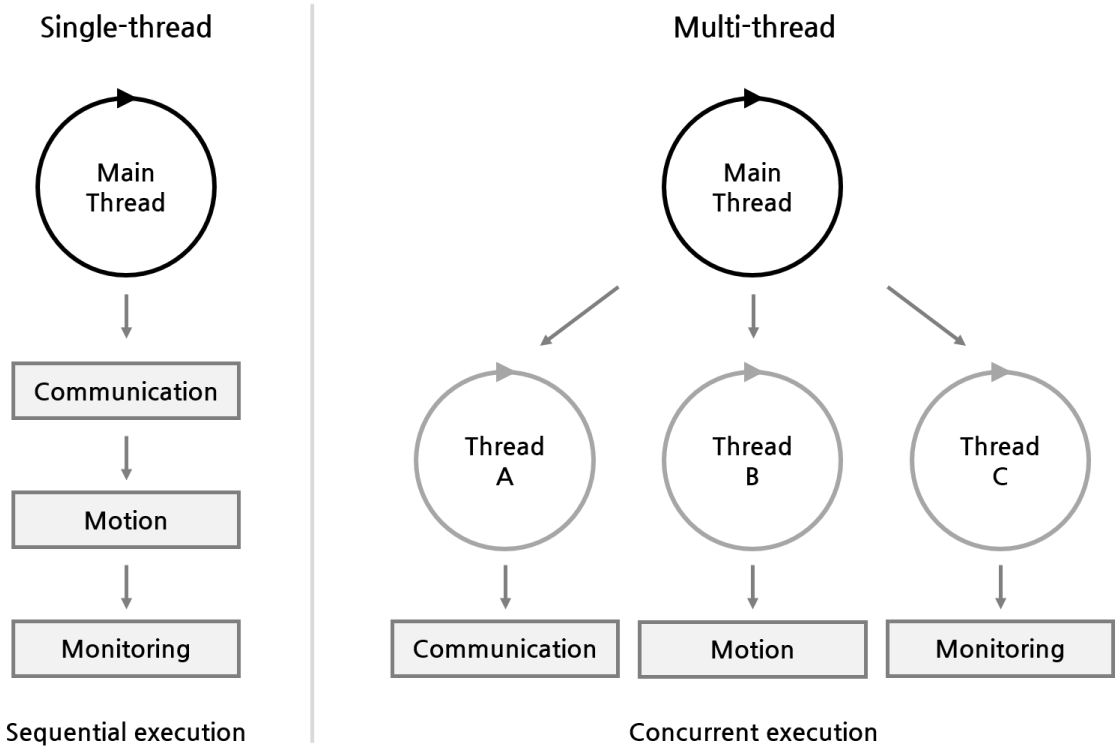


Figure 1-5 Difference between Single-Thread and Multi-Thread

System Mode and Robot Mode can additionally assign up to 3 threads each to operate simultaneously except the main thread, implying that total 4 threads can be used at the same time.

In general, a separate program is created for each job, and the main program is selectively assigned to the thread according to the specific conditions and this can be released after completing the job. As such, various jobs can be processed.

	System Mode	Robot Mode
TH1	SYSTEM THREAD 1(Main Thread)	ROBOT THREAD 1(Main Thread)
TH2	SYSTEM THREAD 2	ROBOT THREAD 2
TH3	SYSTEM THREAD 3	ROBOT THREAD 3
TH4	SYSTEM THREAD 4	ROBOT THREAD 4

Figure 1-6 Thread Structure in System Mode & Robot Mode

※ It is possible to check the multi-threading through the monitoring function in execution window. Refer to chapter 4, section 9. Multithreading for the functions to use the multi-threading and detail description.

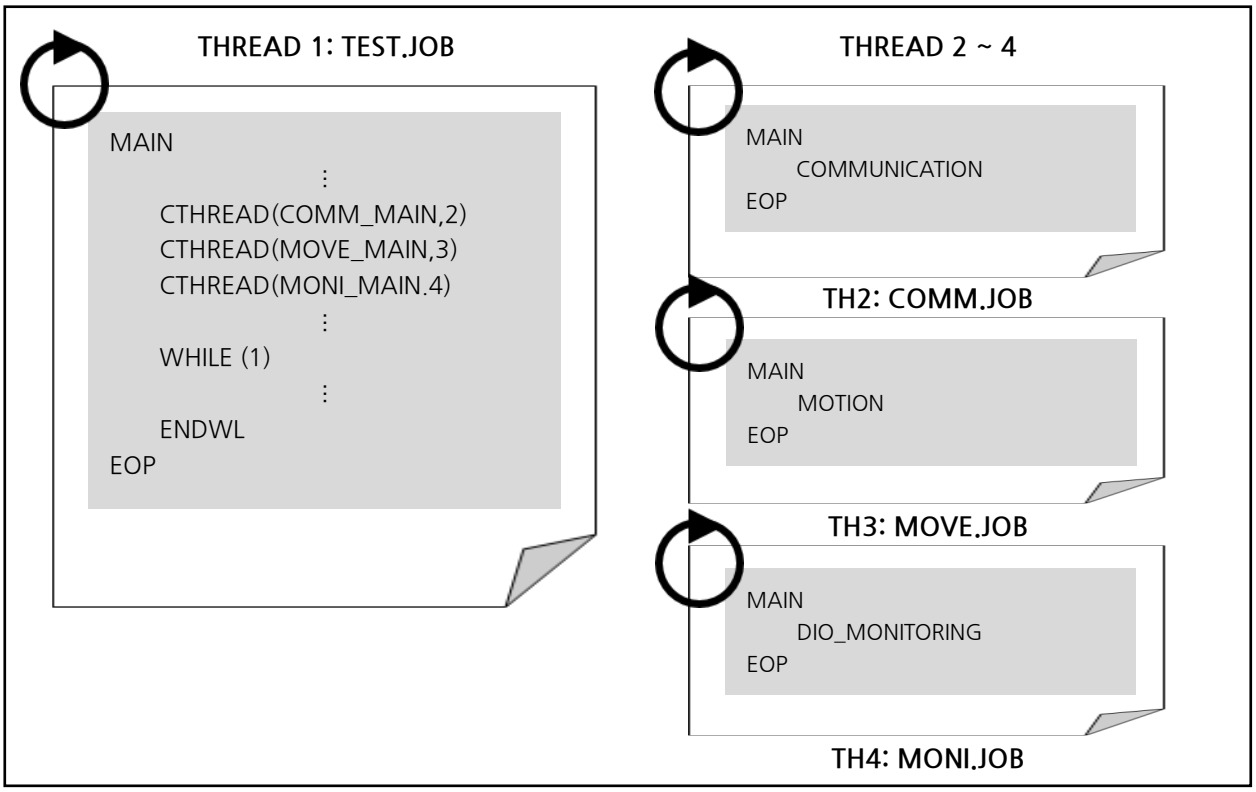


Figure 1-7 Example of Multi-Threading Execution

1) The main thread in system mode of N2 Controller, currently can run the BGT(Background Task) program according to set of BGT parameters.
2) Since System mode and Robot mode are independent each other, the main JOB program for System mode (BGTD.JOB, BGTF.JOB) cannot be used in Robot mode.

Chapter 2 Construction Elements of Language & Syntax

1. Construction Elements of Language

Robostar Robot Language (RRL, Robostar Robot Language) is composed of the elements as shown below.

- Identifier: A name that identifies a construction element
- Variable: A space consisting of characters and numbers for temporary store data
- Constant: A value that cannot be changed during a program execution
- Operator: A symbol that instructs to execute a specific mathematical or logical operation
- Expression: A combination of construction elements to obtain an unique value
- Command: A built-in instruction to perform a specific function

2. Name

Name that represents commands, variables, functions, labels and programs follow the rules below.

- The name must begin with a capitlal letter of English alphabet or underscore.
- A capital letter of the English alphabet, number and underscore can be used in any position of the name except the beginning.
- The characters used as operators are not allowed when creating the name.

3. Variable

Variables are used for making a space to store data in a program. Variables include global, local and system variables. System variables can be used all programs. Global variables fall into two categories, one that can be used in all programs and another that can be used dependently only in the corresponding program. Local variables are effective only in the block where the variable was declared.

3.1 Global Variable

Global variables include integer variable, real number variable and positional variable in joint coordinate system. Global variables are expressed in a format with alphabet characters followed by an integer expression. The name of global variables are defined by the system and they can be used without declaration accordingly. Global variables include following formats.

Item	Global variables that can be commonly used in all programs
I	Integer (4 bytes, range: -2,147,483,648 ~ 2,147,438,647) Max usable number: 1,000 (0 ~ 999)
	Examples) I0, I[1]
F	Real type (4 bytes, range: 3.402,823E-38 ~ 3.402,823E+38) Max usable number: 1,000 (0 ~ 999)
	Examples) F100, F[101]
GP	Global joint position type, Max usable number: 2,000 (0 ~ 1,999)
	Examples) GP0, GP[1]

Item	Global variables depended on a program
P	Program joint position type (unable to alter the value within the program) Max usable number: 1000 (0 ~ 999)
	Examples) P0, P[1]

Of variables above, P variable has a feature additionally that cannot be altered within the program.

Once a global variable is assigned, the variable number can be nominated by using an expression. This is an indirect reference. If an indirect reference is executed, the variable number is displayed with a parenthesis “[“, “]”.

〈Examples〉	
I0 = I[2*3]	// put I6 variable into I0 variable
F1 = F[I12]	// put F variable in I12 into F1 variable
GP1 = GP[I5+2]	// put GP variable of I5 adding 2 into GP1
JMOV P[0]	// JMOV movement from current position to P0

※ Refer to the chapter 4, section 2.7 Program Position P ~ chapter 4, section 2.10 Global Position GP for the detail.

3.2 Local Variable

Local variables include integer variable, real number variable, string variable, position variable of joint coordinate system, position variable of base coordinate system and position variable of user coordinate system. Local variables can be used after declaration. They must be declared because the name of local variable is not defined by the system. When referring to a certain local variable, a specific value must be assigned to the local variable (initialization of variable). If they are referred without an initialization, error occurs in execution. Refer to chapter 4, section 2. Variable regarding the commands and a method for type declaration.

<Examples>	
INT AA	// Declaration of variable
AA = 0	// Initialization of variable
INT BB = 4	// Error occurs(initiailized and declared at the same time)

※ it is not possible to initialize them with declaration simultaneously.

If declaring a variable using **"$" symbol** as shown below**, the variable can be used as a global variable**. In other words, it can be used in the program. In addition, it is possible to use the variable in other programs if referring to the program that declared the variable using INCLUDE command.

<Examples>	
INT $AA	// Declaration of the global integer variable
REAL $BB	// Declaration of the global real number variable
MAIN	// Declaration of Main Function
$AA = 10	// Initialization of global integer variable
$BB = 12.4	// Initialization of global real number variable
INIT()	// Function Call
EOP	// End of Main Function
POS $ACUR	// Declaration of global position variable of joint coordinate //system
FUNC VOID INIT()	// Define a function with No Return Value (VOID)
$ACUR = <0,0,0,0,0,0>	// Initialization of global position variable of joint coordinate //system
END	// End of Function

Here, in case of utilizing a local variable as a global variable, the variable must declared at outside of the Main Function or the function, and initialization must be performed within the function.

3.3 System Variable

System variables are used for checking the system status. Since the variable name is guaranteed by the system, declaration of variable is not required. Refer to the chapter 4, section 2.11TMR ~ 2.14 HERE_REF for detail description and method.

Item	Description
TMR	This is used to read or specify the current count value
MVR	This is a variable that was predefined in the system and returns the moving distance ratio of the total moving distance.
HERE	This is a predefined variable in the system and alters the current position with the joint angle value.
HERE_REF	This is a predefined variable in the system and returns the current reference position with the joint angle value.

4. Function

Function is used for getting the results of transfer arguments using the predefined motion types. There exist some functions that have neither a return value or nor a transfer arguments according to function types. It is possible for a user to define a function. In such case, all data types can be used as a return value and a transfer arguments. VOID must be used in case of no return value when declaring a function.

5. Branch Point

Branch Point indicates a specific position in a program using a text. By assigning the position of a Branch Point and using a command, it is possible to move the position of Branch Point. However, the same name cannot be used in one file.

6. Program

It is possible to call and use other program names within a program.

〈Examples〉	
MAIN	// Start of Main Function
JCALL MOTION	// Call & execute MOTION.JOB
EOP	// End of Main Function

7. Type and Conversion of Data

7.1 Data Type

The table below shows data types that a user can declare to use.

Variable Type		Description	Usable Count
		Size & Range	
Number	INT	Integer type	Within memory range
		4 bytes, -2,147,483,648 ~ 2,147,438,647	
	REAL	Real type	Within memory range
		4 bytes, 3.402,823E-38 ~ 3.402,823E+38	
String	DEFSTR	Integer type	1,000
		Within 100 characters	
Position	POS	Joint position type	Within memory range
		48 bytes (J1, J2, J3, J4, J5, J6, E1, E2)	
	XPOS	Base position type	Within memory range
		48 bytes (X, Y, Z, RX, RY, RZ, E1, E2, ARM, T)	
	UPOS	User position type	Within memory range
		48 bytes (Xu, Yu, Zu, RX, RY, RZ, E1, E2, ARM, T, U)	

The number of local variables that a user can declare, when using the local variables, is limited for the memory size. In case of using the local variables as shown in the table below, it is allowed to declare 1Mbytes of number or position variables as well as 1,000 of string variables. When using “$” symbol for using them as a global variable, it uses another memory area with the same size. Thus, it is possible to use max 2Mbyes of number or position variables as well as 2,000 of string variables.

Item	Local Variables	Global Variables	Sum
Number, Position Variables	1Mbytes	1Mbytes	2Mbytes
String Variables	1,000 (less than 100 characters)	1,000 (less than 100 characters)	2,000



1) When over-using the memory, an alarm occurs.

In the case of the position variables, it is possible to access an individual item by using the "." symbol next to the variable name. The number of positional data that can access varies depending on the number of axes and a degree of freedom (DOF). Unlike when initializing the variable using a constant, all items in the table below can be accessed to read or write.

Classification	Example
J1 axis (or X axis)	<Variable Name>.1
J2 axis (or Y axis)	<Variable Name>.2
J3 axis (or Z axis)	<Variable Name>.3
J4 axis (or RX axis)	<Variable Name>.4
J5 axis (or RY axis)	<Variable Name>.5
J6 axis (or RZ axis)	<Variable Name>.6
J7 axis (E1 axis)	<Variable Name>.7
J8 axis (E2 axis)	<Variable Name>.8
Posture of Robot	<Variable Name>.ARM
Tool Number	<Variable Name>.T
User Number	<Variable Name>.U
Initialization of Variable	<Variable Name>.USED

When using the 6-axis robot, each axis 1 ~ 6 indicates each joint of the robot. Axis 7 and 8 are used for the case of requiring an additional axis depending on the installation environment of Robot.

Posture of robot can be input as shown in the table below. In case of inputting 0 value, the posture of robot is set based on current joint value of the robot. Refer to the Operation manual (N2-OM-E□□) for more detail description on the each posture.

Value	Shoulder	Elbow	Wrist
000	NOT DEFINED	NOT DEFINED	NOT DEFINED
001	NOT DEFINED	NOT DEFINED	FLIP
002	NOT DEFINED	NOT DEFINED	NON FLIP
010	NOT DEFINED	ABOVE	NOT DEFINED
011	NOT DEFINED	ABOVE	FLIP
012	NOT DEFINED	ABOVE	NON FLIP
020	NOT DEFINED	BELOW	NOT DEFINED
021	NOT DEFINED	BELOW	FLIP
022	NOT DEFINED	BELOW	NON FLIP
100	REAR	NOT DEFINED	NOT DEFINED
101	REAR	NOT DEFINED	FLIP
102	REAR	NOT DEFINED	NON FLIP
110	REAR	ABOVE	NOT DEFINED
111	REAR	ABOVE	FLIP
112	REAR	ABOVE	NON FLIP
120	REAR	BELOW	NOT DEFINED
121	REAR	BELOW	FLIP
122	REAR	BELOW	NON FLIP
200	FRONT	NOT DEFINED	NOT DEFINED
201	FRONT	NOT DEFINED	FLIP
202	FRONT	NOT DEFINED	NON FLIP
210	FRONT	ABOVE	NOT DEFINED
211	FRONT	ABOVE	FLIP
212	FRONT	ABOVE	NON FLIP
220	FRONT	BELOW	NOT DEFINED
221	FRONT	BELOW	FLIP
222	FRONT	BELOW	NON FLIP

Tool and User number can be entered up to 16 in total from 0 ~ 15.

Lastly, an item for variable initialization can be entered (0: no initialization, 1: initialization completed). However, when a variable is initialized as a constant, the value is automatically set to 1. Alarm occurs when the move command refers to a positional variable with variable initialization set to 0.



1) When performing an initialization by access each individual item, a completion of initialization must be informed by inputting the initialization completion value (1) into the USED item.

7.2 Data Type Conversion

Data types can be converted each other following the rules below. Alarm may occur if the following rules are breached.

1) Integer and Real Number Values

- Number value is converted so that it can meet the type if assigned to a different type.
- When a real number is converted to integer, a value with round-off is outputted as a result.

2) Character and Number Values

Post-Conversion / Pre-Conversion	Character	Integer
Character		ASC
Integer	BINS, CHR	

- Ex) Character data are converted to integer data using ASC function.

3) Positional Value

Post-Conversion / Pre-Conversion	POS	XPOS	UPOS
POS		JTOB	JTOU
XPOS	BTOJ		BTOU
UPOS	UTOJ	UTOB	

- Ex) POS data are converted to XPOS data using JTOB function.

8. Constant

Constant is an expression that has fixed value. It is classified as shown in the table below.

Classification		Description
Integer	Binary	Only 0 or 1 is used to represent the value. 0B is used as a prefix in writing.
		Example) 0B11
	Decimal	Number 0 ~ 9 are used to represent.
		Example) 100
	Hexadecimal	Total 16 such as 0 ~ 9, A ~ F are used to represent. 0H is used as a prefix in writing.
		Example) 0HFF
Real Number		The "." symbol is used to represent the digits below a decimal point.
		Example) 2.3456
String		String is represented using the quotation marks.
		Example) "String statement"
Positional	POS	The "<" and ">" symbols are used to represent.
		Example) <100, 100, 100, 100, 100, 100>
	XPOS	ARM (Posture), T (Tool Number) are used to represent.
		Example) <100, 100, 100, -30, 30, 60, ARM = 0, T = 2>
	UPOS	ARM (Posture), T (Tool Number), U (User Number) are used to represent.
		Example) <0, 0, 0, -30, 30, 60, ARM = 0, T = 1, U = 3>

9. Expression and Operator

An expression is used to return the value. There are two types of expression, one expression that has an independent value and the other expression that consists of various elements with the operators.

9.1 Priority in Operators

Operator	Description	Priority
(,)	Parenthesis	High
~, !	Bit and Logical NOT	↑
*, /, %	Multiplication, Division, Remainder	
+, -	Addition, Subtraction	
<<,>>	Shift to Left, Shift to Right	
<, <=,>,>=	Smaller, Smaller or Equal, Bigger, Bigger or Equal	
==, !=	Equal, Unequal	
&	Bit AND	
^	Bit XOR	
|	Bit OR	
&&	Logic AND	
||	Logic OR	↓
=	Assignment	Low

9.2 Assignment Operator

The result or value of the expression is put into the left variables using the “=” symbol. Type conversion may appear according to variable types.

<Examples>

```
INT A, B, C, D
IN0 = 1                    // IN Command
A = 100
B = I5
C = F1
D = A*(B+C)
```

9.3 Arithmetic Operator

Addition (+) and Subtraction (-), Multiplication (*), Division (/), Remainder of Division (%) are acquired. The sign is determined as the rules below when obtaining the remainder of division.

Dividend \ Divisor	(+)	0	(-)
(+)	(+)	Error	(+)
0	0	Error	0
(-)	(-)	Error	(+)

1) An alarm occurs if divided by 0 or obtaining the remainder using 0.
2) An overflow may occur due to a use of arithmetic operators. No alarm occurs. Pay attention to the max allowed value.

9.4 Relational Operator

Relational operators are used for comparing two number values. Results are logical values (1: true, 0: false). These are typically used in a conditional expression of instruction statement for flow control.

Relational Operator	Description
<	Smaller
>	Bigger
<=	Smaller or Equal
>=	Bigger or Equal
==	Equal
!=	Unequal

9.5 Logical and Bit Operators

Logical operators are used for connecting one or many conditional expressions. Bit operators are used for logical responses by Bit. The implication for each symbol is in the table below.

	Logic	Bit
Negation	!	~
Product	&&	&
Addition	||	|
Exclusive addition		^

9.6 Array

Robostar Robot Language (RRL) allows to use the one dimensional array. Array implies a set of variables in same type. Each value consisting of an array is called an element and the number indicating a position in array is called an index. Index always begins with 0 and can have only the position integer numbers including 0.

〈Syntax〉
〈Variable Type〉 〈Array Name〉"["〈Array Length〉"]"
〈Examples〉
INT AA[10] AA[0] = 10

1) It is not possible to use array in a way like a pointer in C-language.
2) As soon as declared as an array, an initialization is not allowed.
3) Array must be used within the total memory size (1MB).

9.7 Use of String

Two strings can be added together using the addition symbol "+".
The "+" can be used in either constants or arrays.

〈Syntax〉
〈Test String Variable〉 = 〈Test String〉 + 〈Test String〉 〈Test String Variable〉 = 〈Test String Variable〉 + 〈Test String〉 〈Test String Variable〉 = 〈Test String Variable〉 + 〈Test String Variable〉
〈Examples〉
DEFSTR STR1, STR2, STR3 STR1 = "TEST " + "STATEMENT " STR2 = STR1 + "COMPLETE" STR3 = STR1 + STR2

1) The added string shall not exceed 100 characters.

9.8 Use of Positional Variable

Positional data enable to perform the following jobs.

〈Syntax〉
〈Positional Variable〉 = 〈Positional Constant or Variable〉 + 〈Positional Constant or Variable〉 〈Positional Variable〉 = 〈Positional Constant or Variable〉 - 〈Positional Constant or Variable〉 〈Positional Variable〉 = 〈Positional Constant or Variable〉 * 〈Number Variable or Constant〉 〈Positional Variable〉 = 〈Positional Constant or Variable〉 / 〈Number Variable or Constant〉
〈Examples〉

```
POS RES_P, AP, BP
REAL A, B

A = 10.1
B = -7.2
AP = 〈50, 50, 50, 50, 50, 50〉
BP = 〈30, 30, 30, 30, 30, 30〉

RES_P = 〈100, 100, 100, 100, 100, 100〉 + 〈100, 100, 100, 100, 100, 100〉
PRINT(0,RES_P)
// 〈200.000,200.000,200.000,200.000,200.000,200.000,0.000,0.000〉

RES_P = AP + 〈100, 100, 100, 100, 100, 100〉
PRINT(1,RES_P)
// 〈150.000,150.000,150.000,150.000,150.000,150.000,0.000,0.000〉

RES_P = 〈100, 100, 100, 100, 100, 100〉 + BP
PRINT(2,RES_P)
// 〈130.000,130.000,130.000,130.000,130.000,130.000,0.000,0.000〉

RES_P = AP + BP
PRINT(3,RES_P)
// 〈80.000,80.000,80.000,80.000,80.000,80.000,0.000,0.000〉

RES_P = 〈100, 100, 100, 100, 100, 100〉 * A
PRINT(4,RES_P)
// 〈1010.000,1010.000,1010.000,1010.000,1010.000,1010.000,0.000,0.000〉

RES_P = 〈100, 100, 100, 100, 100, 100〉 / B
PRINT(5,RES_P)
// 〈-13.889,-13.889,-13.889,-13.889,-13.889,-13.889,-0.000,-0.000〉

RES_P = AP * A
PRINT(6,RES_P)
```

```
// <505.000,505.000,505.000,505.000,505.000,505.000,0.000,0.000>

RES_P = BP / B
PRINT(7,RES_P)
// <-4.167,-4.167,-4.167,-4.167,-4.167,-4.167,-0.000,-0.000>
```

1) Different types of positional variables cannot be used together.
   Identical type of positional variables must be used.
2) In case of converting to different types of positional variables, refer to the Conversion Function of chapter 4, Section 10.3 JTOB ~ 10.8 UTOB.

Chapter 3 Command List

1. Variable & Constant

1.1 Common variable

Classification		Description Format	Example in Use	Ref.
Number	INT	Declare an integer variable (4 bytes) INT <Variable Name>	INT AA AA = 0	2.1
	REAL	Declare a real number variable (4 bytes) REAL <Variable Name>	REAL BB BB = 0.0	2.2
DEFSTR		Declare a string variable DEFSTR <Variable Name>	DEFSTR SS SS = "STRING"	2.3
POS		Declare a position variable in joint coordinate system POS <Variable Name>	POS AP AP = <10,10,10,10,10,10>	2.4
XPOS		Declare a position variable in base coordinate system XPOS <Variable Name>	XPOS BP BP = <0,0,0,0,0,0,ARM = 0,T = 0>	2.5
UPOS		Declare a position variable in user coordinate system UPOS <Variable Name>	UPOS CP CP = <0,0,0,0,0,0,ARM = 0,T = 0,USED = 0>	2.6
P		This is a predefined positional variable in joint coordinate system. This is stored in the program file and is used as a global variable within the corresponding program. ※ It is not possible to directly edit the positional Variable in program joint P that is stored in a JOB file. P<Number> or P[Number]	JMOV P0 JMOV P[1]	2.7
I		Predefined global integer variable I<Number> or I[Number]	I0 = 100 I[1] = 50	2.8
F		Predefined global real number variable F<Number> or F[Number]	F0 = 3.14 F[1] = 10.54	2.9
GP		Predefined global position variable of joint coordinate system GP<Number> or GP[Number]	GP0 = <0,0,0,0,0,0> GP[1] = <10,10,10,10,10,10>	2.10

1) Variable name cannot begin with numbers.
2) Variable name consists of the capital alphabet letters and numbers.
3) It is possible to approach the positional variables individually by designating the axis number or options after using "." symbol right behind the variable name.
4) For detail description on the use of variables, refer to the chapter 2, section 3 Variable and chapter 2, section 7. Type & Conversion of Data in Chapter 2 as well as chapter 4, section 2. Variable
5) By attaching the symbol $ as a prefix at the variable name, the variable can be used as a global variable.

1.2 System Variable

Classification	Description Format	Example in Use	Ref.
TMR(0) TMR(1)	A value is assigned as soon as inputting integer value, thus increases by 1 per 10ms interval. (Timer Function) <Number>: Select Timer Number (0 or 1) TMR(<Number>) = <Value> <Integer Variable> = TMR(<Number>)	INT AA TMR(0) = -50 TMR(1) = 0 AA = TMR(0)	2.11

MVR	Checking the percentage of moving distance
	<Robot Number>: Robot designation by entering the number. Information stored in JOB program is used in case of no input (1 ~ 3).
	<Real Number Variable> = MVR[(<Robot Number>)]
	MVR
		VALUE1 = MVR
		VALUE2 = MVR(2)
		IF MVR <50 THEN
		ENDIF
			2.12

HERE	Current position is returned as a joint value. <Robot Number>: Robot designation by entering the number. Information stored in JOB program is used in case of no input (1 ~ 3) <Position Variable> = HERE[(<Robot Number>)]	POS AP, BP AP = HERE BP = HERE(2)	2.13
HERE_REF	Current reference position is returned as a joint value. <Robot Number>: Robot designation by entering the Number. Information stored in JOB program is used in case of no input (1 ~ 3) <Position Variable> = HERE_REF[(<Robot Number>)]	POS AP, BP AP = HERE_REF BP = HERE_REF(2)	2.14

1.3 Constant

Classification	Description Format	Example in Use	Ref.

Number Constant	This is expressed by integer, real number, binary integer, and hexadecimal integer.
	[0H/0B]<Number>
	If no 0H or 0B as a prefix in the number, it is a decimal number.
		INT AA
		AA = 0H12 // Hexadecimal
		AA = 0B11 // Binary number
			-

String Constant	This expresses a string. Less than 100 characters in max "<String>"	DEFSTR STR STR = "STRING Constant"	-
Positional Constant	It expresses an individual value of positional variable. When using XPOS, the values of ARM (Posture), T (Tool Number) are required and when using UPOS, the values of ARM (Posture) and T (Tool Number) are required. U (User Number) values are additionally required. **※ When designation of <Variable Value Setting> is Required, USED is used for designating the value.** **※ When initializing the positional variable value by using the position constant, <Variable Value Setting> can be skipped.** **(Default: 1, 0: No value was set, 1: Value was set)** <Position Value(DOF), [<Arm form>], [<Tool Number>], [<User Number>], [<Variable Value Setting>]>	XPOS XP XP = <0,0,0,0,0,0,ARM = 0,T = 1,USED = 1> UPOS UP UP = <1,2,3,4,5,6,ARM = 0,T = 1,USED = 1>	-

2. Preprocessor and Comment

Classification	Description Format	Example in Use	Ref.
INCLUDE	This enables a specific program file to be used as if it is a one program by attaching it at current position. INCLUDE “<JOB File Name>”	INCLUDE “TEST”	3.1
/*, */	Many Lines of foot note /* <Content> */	/* JMOV GP0 DLAY 1000 */	3.2
//	one line of foot note // <Content>	// JMOV GP0 DLAY 1000	

3. Operator

Classification	Description Format	Example in Use	Ref.
=	Assignment Operator <Variable> = <Variable> or <Expression>	B = A	4.1
+, -, *, /, %	Arithmetic Operator (Addition, Subtraction, Multiplication, Division, modulus) <Variable> = <A Term> <Arithmetic Operator> <B Term>	C = A+B	4.2
>, <,>=, <=, !=, ==	Relational Operator (Over, Less, More Than, Less Then, Unequal, Equal) [<Number Variable> =]<A Term> <Relational Operator> <B Term>	IF A> B THEN ENDIF	4.3

&&, ||, !	Logical Operator
	(AND, OR, Negation)
	[<Number Variable> =] <A Term> <Logic Operator> <B Term>
		C = !A
		IF (A == 1) && (B == 0) THEN
		ENDIF
			4.4

&, I , ~, ^, <<,>>	Bit Operator
	(BAND, BOR, Complement, BXOR, left shift, right shift)
	[<Number Variable> =]<A Term> <Bit Operator> <B Term>
		C = A <<0H01
		IF A & B THEN
		ENDIF
			4.5

()	Parenthesis (<Expression>)	D = (A+B)*C	4.6

4. Program Control

Classification	Description Format	Example in Use	Ref.
MAIN … EOP	Start & End of main program MAIN … EOP	MAIN JMOV P1 JMOV P2 EOP	5.1
FUNC … END	Declaration & End of function, Return type (including VOID) & Enable to declare parameters. Enable to return specific values FUNC <Return> <Function Name>(<Parameter>) [RET <Variable>] END	FUNC VOID TEST() VEL 200 JMOV P0 END	5.2
RET	End of function & Return of value RET <Return Value>	FUNC INT TEST() RET 1 END	5.3
FOR … TO … [BY …] NEXT	This creates a loop that executes as defined turns. When skipping BY <Increment Size>, then it is defined as “1”. FOR <Initialization Statement> TO <Ending Condition> [BY <Increment Size>] … NEXT	… FOR AA = 1 TO 5 [BY 2] BB = BB+AA NEXT …	5.4
WHILE … ENDWL	Conditional iteration execution WHILE <Conditional Expression> … ENDWL	… WHILE IN0 == 1 JMOV P1 ENDWL …	5.5

IF … THEN … [ELSE IF … THEN] [ELSE …] ENDIF	Conditional judgment execution
	IF <Conditional Expression> THEN
	…
	[ELSEIF <Conditional Expression> THEN]
	…
	[ELSE]
	…
	ENDIF
		…
		IF IN3 == 1 THEN
		GOTO A0
		ELSE
		JMOV P1
		ENDIF
		LABL A0
		…
			5.6

LABL … GOTO	This designates the position of a branch point and move to the branch point. LABL <Branch Point Name> … GOTO <Branch Point Name>	… LABL A1 JMOV P1 JMOV P2 GOTO A1 …	5.7

CONTINUE	This returns to the beginning of an iterative statement.
	WHILE < Conditional Expression >
	…
	CONTINUE
	ENDWL
		WHILE 1
		IF IN(1) == 1 THEN
		CONTINUE
		ENDIF
		JMOV P0
		ENDWL
			5.8

Classification	Description Format	Example in Use	Ref.

BREAK	This stops the iterative statement and escapes from the statement.
	WHILE <Conditional Expression>
	…
	BREAK
	ENDWL
		WHILE 1
		IF IN(1) == 1 THEN
		BREAK
		ENDIF
		JMOV P0
		ENDWL
			5.9

JCALL	This execute the command by immediately calling a JOB file. JCALL <Job File Name>	JCALL INC	5.10

STOP	Robot motion stop
	STOP
		IF IN0 == 1 THEN
		STOP
		ENDIF
			5.11

EXIT	Execution stop of JOB program
	EXIT
		IF IN0 == 1 THEN
		EXIT
		ENDIF
			5.12

5. Robot Motion

Classification	Description Format	Example in Use	Ref.
SVON	Servo power on, <Robot Number>: Robot is nominated by entering the robot number. If a robot number is not entered, the info stored in a JOB program is used. (1 ~ 3) SVON[(<Robot Number>)]	SVON SVON(2)	6.1
SVOF	Servo power off, <Robot Number>: Robot is nominated by entering the robot number. If a robot number is not entered, the info stored in a job Program is used. (1 ~ 3) SVOF[(<Robot Number>)]	SVOF SVOF(2)	6.2
JMOV	This moves PTP from current position to target point. JMOV <Point Variable>[, <VEL = Value>, <ACC = Value>,<DEC = Value>, <FOS = Value>, <T = Value>, <U = Value>]	JMOV P1 JMOV P2, VEL = 50, ACC = 30, DEC = 30	6.3
LMOV	This enables a linear interpolation movement from current position to target point. LMOV <Point Variable>[, <VEL = Value>, <ACC = Value>, <DEC = Value>, <FOS = Value>, <T = Value>, <U = Value>]	LMOV P1 LMOV P2, VEL = 50, ACC = 50, DEC = 50	6.4
CMOV	This enables to move in a circle along the via point 1, 2 from current position. CMOV <Via point 1> <Via point 2> [, <VEL = Value>, <ACC = Value>, <DEC = Value>, <T = Value>, <U = Value>]	CMOV P1 P2 CMOV P1 P2, VEL = 100	6.5
AMOV	This enables to move in an arc between via point 1 and target point from current position. AMOV <Via point 1> <Target Point> [, <VEL = Value>, <ACC = Value>, <DEC = Value>, <T = Value>, <U =	AMOV P1 P2, VEL = 100	6.6

	Value>]		

6. Robot Motion Condition

Classification	Description Format	Example in Use	Ref.
WITH … ENDWT	This processes an execution statement within a block simultaneously while a robot is in motion. WITH … ENDWT	WITH JMOV P1 WHILE MVR <80 ENDWL OUT0 = 1 ENDWT	7.1
VEL	This sets a permiliage (%) of axis moving speed. Axis moving speed = Initial Speed X Rated Speed X Per Millage Value X 0.001 **※ Rated Speed & Initial Speed are set in the Parameters.** VEL <Per Millage Value (%)>	VEL 200 JMOV P1	7.2
ACC	This sets a percentage (%) of acceleration time. Acceleration Time = Rated Acceleration Time X Percentage Value X 0.01 **※ Rated Deceleration Time is set in the parameters.** ACC <Percentage Value (%)>	ACC 50 JMOV P1	7.3
DEC	This sets a percentage (%) of deceleration time. Deceleration Time = Rated Deceleration Time X Percentage Value X 0.01 **※ Rated Deceleration Time is set in the parameters.** DEC <Percentage Value (%)>	DEC 200 JMOV P1	7.4
FOS	This alters a trajectory by referring to the distance ratio (%) to next target point before the axis end reaching the target. The distance ratio is a percentage of total moving distance (%). This is applicable to JMOV & LMOV. FOS <Distance Ratio (%)>	FOS 5 JMOV P1 FOS 0 // Release	7.5
DLAY	This sets a delay time. The unit of delay time is 1ms. DLAY <Delay Time>	JMOV P1 DLAY 1000	7.6

Classification	Description Format	Example in Use	Ref.
OFFS	This moves the target point by the specified value. Position variable is only allowed to enter. (A constant cannot be assigned directly) OFFS <Deviation Value>	OFFS P1 JMOV P2	7.7
OFFSCLR	This releases the defined OFFS value. OFFSCLR	OFFSCLR	7.8
LIMT	This limits the moving range for each axis. Position variable is only allowed to enter. (A constant cannot be assigned directly) LIMT <(-) Moving Range , (+) Moving Range>	LIMT P0 P1	7.9
TRQ	This sets the torque limit value. An alarm occurs if exceeding the defined torque limit value. TRQ <Robot Number> <Axis Number> <Torque Limit Value>	TRQ 1 2 100	7.10
GAIN	Converts Gain Group GAIN <Group Number>	GAIN 1 JMOV P0	7.11
IPE	Sets Max Allowable Time value until reaching target position, unit 1ms IPE <Allowable Time>	IPE 100	7.12

7. Input & Output Control

7.1 General Digital Input & Output

Classification	Description Format	Example in Use	Ref.
IN	This reads the value of bit number specified by 1 bit unit 〈Bit Number〉: 0 ~ 19 (up to 83 in max when connecting an expansion board)	INT AA IN10 = 1 AA = IN(0)	8.1
	IN 〈Bit Number〉 or IN(〈Bit Number〉) = 〈0/1〉 - Standby until the input conditions are met 〈Integer Variable〉 = IN〈Bit Number〉 - Input status value is stored in the variable		
OUT	This turns on (= 1) or off (= 0) the output of bit number specified by 1 bit unit. 〈Bit Number〉: 0 ~ 19 (up to 83 in max when connecting an expansion board) 〈Pulse Width〉: Unit of 1ms, This returns to the former state after the effective time. 〈Pulse Cycle〉: Unit of 1ms, Pulse cycle time is entered that is always bigger than Pulse Width.	INT AA OUT0 = 1 OUT0 = 1 100 OUT0 = 1 100 200 AA = OUT(0)	8.2
	OUT〈Bit Number〉 or OUT(〈Bit Number〉) = 〈0/1〉 OUT〈Bit Number〉 = 〈0/1〉[〈Pulse Width〉 〈Pulse Cycle〉] OUT〈Bit Number〉 = 〈0/1〉[〈Pulse Width〉 〈Pulse Cycle〉] - Output value is sent out. 〈Integer Variable〉 = OUT〈Bit Number〉 - Data value being outputted is checked.		
PIN	This reads the value of input port specified by a port unit. 〈Port Number〉: 0 ~ 1 (up to 5 in max when connecting an expansion board)	INT AA, BB AA = PIN(0) BB = PIN1	8.3
	〈Integer Variable〉 = PIN〈Port Number〉 or 〈Integer Variable〉 = PIN(〈Port Number〉)		
POUT	This outputs the defined value into the port specified by a port unit. 〈Port Number〉: 0 ~ 1 (up to 5 in max when connecting an expansion board)	INT AA POUT0 = 0 POUT0 = 20 POUT1 = 0H000F AA = POUT(0)	8.4
	POUT〈Port Number〉 = 〈Output Specified Value〉 or POUT(〈Port Number〉) = 〈Output Specified Value〉 〈Integer Variable〉 = POUT〈Port Number〉		



1) When creating impulse signals or pulse cycle waves using **OUT** or **COUT** commands, the input unit for 〈Pulse Width〉 and 〈Pulse Cycle〉 is 1ms. However, the input value must be multiple of 2 because the value below 2ms is ignored.

7.2 Field Bus Input & Output

Classification	Description Format	Example in Use	Ref.
CIN	This reads the value of field bus bit number specified by 1 bit unit. 〈Integer Variable〉 = CIN〈Bit Number〉 or 〈Integer Variable〉 = CIN(〈Bit Number〉)	INT AA, BB AA = CIN0 BB = CIN(1)	8.5
COUT	This turns on (= 1) or off (= 0) the output of bit number specified by 1 bit unit. 〈Pulse Width〉: Unit of 1ms, This returns to the former state after the effective time. 〈Pulse Cycle〉: Unit of 1ms, Pulse cycle time is entered that is always bigger than Pulse Width. OUT〈Bit Number〉 or OUT(〈Bit Number〉) = 〈0/1〉 OUT〈Bit Number〉 = 〈0/1〉[〈Pulse Width〉 〈Pulse Cycle〉] OUT〈Bit Number〉 = 〈0/1〉[〈Pulse Width〉 〈Pulse Cycle〉] - Output value is sent out. 〈Integer Variable〉 = OUT〈Bit Number〉 - Data value being outputted is checked.	INT AA, BB COUT0 = 1 COUT0 = 1 100 COUT0 = 1 100 200 AA = COUT0 BB = COUT(1)	8.6
CBIN	This reads the value of field bus byte number specified by 1 byte unit. 〈Integer Variable〉 = CBIN〈Port Number〉 or 〈Integer Variable〉 = CBIN(〈Port Number〉)	INT AA, BB AA = CBIN1 BB = CBIN(0)	8.7
CBOUT	This outputs the defined value into the field bus byte specified by a byte unit. CBOUT〈Port Number〉 = 〈Output Specified Value〉 or CBOUT(〈Port Number〉) = 〈Output Specified Value〉) 〈Integer Variable〉 = CBOUT〈Port Number〉	INT AA CBOUT0 = 0HFF AA = CBOUT(1)	8.8
CDIN	Reads Double Word Input by integer unit 〈Integer Variable〉 = CDIN〈Double Word Number〉 or 〈Integer Variable〉 = CDIN(〈Word Number〉)	INT AA, BB AA = CDIN0 BB = CDIN(0)	8.9
CDOUT	Output Double Word Output by integer unit. CDOUT〈Double Word Number〉 = 〈Output designated value〉 or CDOUT(〈Double Word Number〉) = 〈Output designated value) 〈Integer Variable〉 = CDOUT〈Double Word Number〉	INT AA CDOUT1 = 0HFFFFFFFF AA = CDOUT(0)	8.10

Classification	Description Format	Example in Use	Ref.
CWIN	This reads the value of field bus word number specified by 1 word unit. 〈Integer Variable〉 = CWIN〈Word Number〉 or 〈Integer Variable〉 = CWIN(〈Word Number〉)	INT AA, BB AA = CWIN0 BB = CWIN(1)	8.11
CWOUT	This outputs the defined value into the word number of field bus specified by a word unit. CWOUT〈Word Number〉 = 〈Output Specified Value〉 or CWOUT(〈Word Number〉) = 〈Output Specified Value〉) 〈Integer Variable〉 = CWOUT〈Word Number〉	INT AA CWOUT0 = 0HFFFF AA = CWOUT(1)	8.12
CFIN	This reads double word input by a real number unit. 〈Real number Variable〉 = CFIN〈Double Word Number〉 or 〈 Real number Variable 〉 = CFIN(〈Word Number〉)	REAL AA, BB AA = CFIN0 BB = CFIN(0)	8.13
CFOUT	This outputs double word output by a real number unit. CFOUT〈Double Word Number〉 = 〈Output Specified Value〉 or CFOUT(〈Double Word Number〉) = 〈Output Specified Value〉) 〈Real Number Variable〉 = CFOUT〈Double Word Number〉	REAL AA CFOUT1 = 0HFFFFFF AA = CFOUT(0)	8.14

7.3 CNET Input/output

Classification	Description Format	Example in Use	Ref.
XIN	Reads bit input by designating CNET Device Area and bit Number 〈Integer Variable〉 = XIN(〈Device Area〉, 〈bit Number〉)	INT DATA DATA = XIN(“M”, 0)	8.15
XOUT	Designates CNET Device Area and bit Number to output bit output or read output bit data value XOUT(〈Device Area〉, 〈bit Number〉) = 〈0/1〉 or 〈Integer Variable〉 = XOUT(〈Device Area〉, 〈bit Number〉)	INT DATA XOUT(“M”, 0) = 1 DATA = XOUT(“M”, 0)	8.16
XBIN	Reads Byte input by designating CNET Device Area and Byte Number 〈Integer Variable〉 = XBIN(〈Device Area〉, 〈Byte Number〉)	INT DATA DATA = XBIN(“D”, 0)	8.17
XBOUT	Designates CNET Device Area and Byte Number to output Byte Output or read output Byte data value XBOUT(〈Device Area〉, 〈Byte Number〉) = 〈Output Value〉, or 〈Integer Variable〉 = XBOUT(〈Device Area〉, 〈Byte Number〉)	INT DATA XBOUT(“D”, 0) = 0HFF DATA = XBOUT(“D”, 0)	8.18

Classification	Description Format	Example in Use	Ref.
XWIN	Reads Word Input by designating CNET Device Area and Word Number <Integer Variable> = XWIN(<Device Area>, <Word Number>)	INT DATA DATA = XWIN("M", 0)	8.19
XWOUT	Designates CNET Device Area and Word Number to output Word output or read output Word data value XWOUT(<Device Area>, <Word Number>) = <Output Value>, or <Integer Variable> = XWOUT(<Device Area>, <Word Number>)	INT DATA XWOUT("M", 0) = 0HFFFF DATA = XWOUT("M", 0)	8.20
XDIN	Reads input value in an integer form by designating CNET Device Area and Double Word Number <Integer Variable> = XDIN(<Device Area>, <Double Word Number>)	INT DATA DATA = XDIN("D", 0)	8.21
XDOUT	Designates CNET Device Area and Double Word Number to output Double Word Output or read output Double Word data value XDOUT(<Device Area>, <Double Word Number>) = <Output Value>, or <Integer Variable> = XDOUT(<Device Area>, <Double Word Number>)	INT DATA XDOUT("D", 0)=0HFFFFFFFF DATA = XDOUT("D", 0)	8.22
XFIN	Reads input value in a real number form by designating CNET Device Area and Double Word Number <Real Type Variable> = XFIN(<Device Area>, <Double Word Number>)	REAL DATA DATA = XFIN("M", 0)	8.23
XFOUT	Designates CNET Device Area and Double Word Number to output real number value or read output real number data value XFOUT(<Device Area>, <Double Word Number>) = <real number value>, or <Real Type Variable> = XFOUT(<Device Area>, <Double Word Number>)	REAL DATA XFOUT("M", 0) = 12.45 DATA = XFOUT("M", 0)	8.24

8. Multithreading Control

Classification	Description Format	Example in Use	Ref.
CTHREAD	This creates a thread. 〈Thread Number〉: only 2 ~ 4 is allowed to use. 〈thread Alarm Stop〉: This is a function to stop a thread when an alarm occurs (1: on, 0: off) [〈Integer Variable〉 =] CTHREAD(〈Function Name〉, 〈Thread Number〉, [〈Thread Alarm Stop〉])	INCLUDE “AA” MAIN INT T_RET T_RET = CTHREAD(AA_MAIN, 2) EOP	9.1
ETHREAD	This ends a thread. 〈Thread Number〉: only 2 ~ 4 is allowed to use. [〈Integer Variable〉 =] ETHREAD(〈thread Number〉)	ETHREAD(2)	9.2
TH_STATE	This reads thread status information. 〈Execution Mode〉: System Mode (1) or Robot Mode (0) can be entered. 〈Thread Number〉: only 1 ~ 4 is allowed to use. – Return Value: 0 (TH_ERROR): Error in thread number 1 (TH_IDLE): Thread available 2 (TH_STOP): Thread in stop 3 (TH_RUN): Thread in operation 〈Integer Variable〉 = TH_STATE(〈Execution Mode〉, 〈Thread Number〉)	INT T_STATE T_STATE = TH_STATE(1,1)	9.3
TH_SUSPEND	This stops the thread in operation. 〈Thread Number〉: only 2 ~ 4 is allowed to use. [〈Integer Variable〉 =] TH_SUSPEND(〈Thread Number〉)	TH_SUSPEND(2)	9.4
TH_RESUME	This operates the Thread in stop 〈Thread Number〉: only 2~4 is allowed to use. [〈Integer Variable〉 =] TH_RESUME(〈Thread Number〉)	TH_RESUME(2)	9.5

9. Function

9.1 Coordinate Conversion

Classification	Description Format	Example in Use	Ref.
TOOL	This enables to select a tool number to use (0 ~ 15). TOOL <Number>	TOOL 3 LMOV P1	10.1
USER	This enables to select the number of user coordinate system to be used (0 ~ 15) USER <Number>	USER 10 LMOV P1	10.2
JTOB	This alters a position variable of joint coordinate system to a position variable of base coordinate system. <Robot Number>: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. <Position Variable of Base Coordinate System> = JTOB([<Robot Number>,] <Position Variable of Joint Coordinate System>)	POS AP XPOS XP1, XP2 AP = <0,0,0,0,0,0> AP.T = 2 XP1 = JTOB(AP) XP2 = JTOB(2,AP)	10.3
BTOJ	This converts a position variable of base coordinate system to a position variable of joint coordinate system. <Robot Number>: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. <Position Variable of Joint Coordinate System> = BTOJ([<Robot Number>,] <Position Variable of Base Coordinate System>)	POS AP1, AP2 XPOS XP XP = <10,10,20,0,5,0,ARM = 0,T = 1> AP1 = BTOJ(XP) AP2 = BTOJ(2,XP)	10.4
JTOU	This converts a position variable of joint coordinate system to a position variable of user coordinate system. <Robot Number>: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. <Position Variable of User Coordinate System> = JTOU([<Robot Number>,] <Position Variable of Joint Coordinate System>)	POS AP UPOS UP1, UP2 AP = <0,0,0,0,0,0> AP.U = 1 UP1 = JTOU(AP) UP2 = JTOU(2, AP)	10.5
UTOJ	This converts a position variable of user coordinate system to a position variable of joint coordinate system. <Robot Number>: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. <Position Variable of Joint Coordinate System> = UTOJ([<Robot Number>,] <Position Variable of User Coordinate System>)	POS AP1, AP2 UPOS UP UP = <10,10,10,3,6,0,ARM = 0,T = 0,USED = 1> AP1 = UTOJ(UP) AP2 = UTOJ(2, UP)	10.6
BTOU	This converts a position variable of base coordinate system to a position variable of user coordinate system. <Robot Number>: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. <Position Variable of User Coordinate System> = BTOU([<Robot Number>,] <Position Variable of Base Coordinate System>)	XPOS XP UPOS UP1, UP2 XP = <100,100,200,30,5,45,ARM = 0,T = 1> XP.U = 2 UP1 = BTOU(XP) UP2 = BTOU(2, XP)	10.7

Classification	Description Format	Example in Use	Ref.
UTOB	This converts a position variable of user coordinate system to a position variable of base coordinate system. 〈Robot Number〉: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. 〈Base Coordinate System Position Variable〉 = UTOB([〈Robot Number〉,] 〈User Coordinate System Position Variable〉)	XPOS XP1, XP2 UPOS UP UP = 〈10,10,10,0,0,0,ARM = 0,T = 0,USED = 1〉 XP1 = UTOB(UP) XP2 = UTOB(2, UP)	10.8
RELBASE	Calculate relatively moved position value based on Base Coordinate System as much as displacement substituted at Reference Position. 〈Robot Number〉: A Robot can be selected by designating Number. If not entered, use data saved in the JOB Program. 〈Reference Position〉: Base Coordinate System Position Variable 〈Displacement〉: Position Variable of All Coordinate Systems 〈Base Coordinate System Positional Variable〉 = RELBASE([〈Robot Number〉,] 〈Reference Position〉, 〈Displacement〉)	POS AP, DATA XPOS XP, RP1, RP2 DATA = 〈100,0,0,0,0,50〉 AP = HERE AP.T = 0 XP = JTOB(AP) RP1 = RELBASE(XP, DATA) RP2 = RELBASE(1,XP, DATA)	10.9
RELTOOL	This is a function that calculates the relatively moved position value with the assigned movement from the reference position based on a tool coordinate system. 〈Robot Number〉: Robot is selectable by nominating the number. If not entered, the information stored in a JOB program is used. 〈Reference Position〉: Position variable of base coordinate system 〈Movement〉: Position variable of all coordinate systems 〈Base Coordinate System Position Variable〉 = RELTOOL([〈Robot Number〉,] 〈Reference Position〉, 〈Movement〉)	POS AP, DATA XPOS XP1, XP2, RP1, RP2 DATA = 〈100,0,0,0,0,50〉 TOOL 1 AP = HERE XP1 = JTOB(AP) AP = HERE(2) XP2 = JTOB(AP) RP1 = RELTOOL(XP1, DATA) RP2 = RELTOOL(2, XP2, DATA)	10.10
RELUSER	Calculate relatively moved position value based on User Coordinate System as much as displacement substituted at Reference Position. 〈Robot Number〉: A Robot can be selected by designating Number. If not entered, use data saved in the JOB Program. 〈Reference Position〉: User Coordinate System Position Variable 〈Displacement〉: Position Variable of All Coordinate Systems 〈User Coordinate System Positional Variable〉 = RELUSER([〈Robot Number〉,] 〈Reference Position〉, 〈Displacement〉)	POS AP, DATA UPOS UP, RP1, RP2 DATA = 〈100,0,0,0,0,50〉 AP = HERE AP.T = 0 AP.USED = 1 UP = JTOU(AP) RP1 = RELUSER(UP, DATA) RP2 = RELUSER(1,UP, DATA)	10.11
GBASEDISP	Function to Calculates displacement based on Base Coordinate System between 2 Coordinates. 〈Position 1〉 and 〈Position 2〉 must be same type to input coordinate. 〈Robot Number〉: A Robot can be selected by designating Number. If not entered, use data saved in the JOB Program. 〈Position 1〉: All Positional Variable 〈Position 2〉: All Positional Variable 〈Base Coordinate System Positional Variable〉 = GBASEDISP([〈Robot Number〉,] 〈Position 1〉, 〈Position 2〉)	POS AP1, AP2 XPOS XP1, XP2, D_X1, D_X2 AP1 = GP0 AP1.T = 0 AP2 = GP1 AP2.T = 0 D_X1 = GBASEDISP(AP1, AP2) D_X2 = GBASEDISP(1,AP1, AP2)	10.12

Classification	Description Format	Example in Use	Ref.
GTOOLDISP	Calculates displacement based on Tool Coordinate System between 2 Coordinates. <Position 1> and <Position 2> must be same type to input coordinate <Robot Number>: A Robot can be selected by designating Number. If not entered, use data saved in the JOB Program. <Position 1>: All Positional Variables <Position 2>: All Positional Variables <Base Coordinate System Positional Variable> = GTOOLDISP([<Robot Number>,] <Position 1>, <Position 2>)	POS AP1, AP2 XPOS XP1, XP2, D_X1, D_X2 AP1 = GP0 AP1.T = 1 XP1 = JTOB(AP1) AP2 = GP1 AP2.T = 1 XP2 = JTOB(AP2) D_X1 = GTOOLDISP(XP1, XP2) D_X2 = GTOOLDISP(1,XP1, XP2)	10.13
GUSERDISP	Calculates displacement based on User Coordinate System between 2 Coordinates. <Position 1> and <Position 2> must be same type to input coordinate <Robot Number>: A Robot can be selected by designating Number. If not entered, use data saved in the JOB Program. <Position 1>: All Positional Variables <Position 2>: All Positional Variables <User Coordinate System Positional Variable> = GUSERDISP([<Robot Number>,] <Position 1>, <Position 2>)	POS AP1, AP2 UPOS UP1, UP2, D_U1, D_U2 AP1 = GP0 AP1.T = 1 AP1.U = 1 UP1 = JTOU(AP1) AP2 = GP1 AP2.T = 1 AP2.U = 1 UP2 = JTOU(AP2) D_U1 = GUSERDISP(UP1, UP2) D_U2 = GUSERDISP(1,UP1, UP2)	10.14

9.2 Mathematics

Classification	Description Format	Example in Use	Ref.
ABS	This takes the absolute value. 〈Real Number Variable〉 = ABS(〈Input Value〉)	REAL AA AA = ABS(-1.5)	10.15
DEG	This converts Radian angle to Degree angle. 〈Real Number Variable〉 = DEG(〈Input Value〉)	REAL AA AA = DEG(3.14)	10.16
RAD	This converts Degree angle to Radian angle. 〈Real Number Variable〉 = RAD(〈Input Value〉)	REAL AA AA = RAD(180)	10.17
POW	Power Function 〈Real Number Variable〉 = POW (〈Base Value〉, 〈Exponent Value〉)	REAL AA AA = POW(2,4)	10.18
RND	This converts a real number value to an integer value by rounding off the decimal part. 〈Real Number Variable〉 = RND (〈Input Value〉)	REAL AA AA = RND(14.8)	10.19
EXP	Exponential Function, e^x 〈Real Number Variable〉 = EXP(〈Input Value〉)	REAL AA AA = EXP(3)	10.20
LN	Natural Logarithm $\log_e X$ 〈Real Number Variable〉 = LN(〈Input Value〉)	REAL AA AA = LN(15)	10.21
LOG	Common Logarithm $\log_{10} X$ 〈Real Number Variable〉 = LOG(〈Input Value〉)	REAL AA AA = LOG(100)	10.22
SQRT	This calculates a square root. 〈Real Number Variable〉 = SQRT(〈Input Value〉)	REAL AA AA = SQRT(16)	10.23
SIN	Sine Function 〈Real Number Variable〉 = SIN(〈Input Value〉)	REAL AA AA = SIN(RAD(30))	10.24
ASIN	Arcsine Function 〈Real Number Variable〉 = ASIN(〈Input Value〉)	REAL AA AA = ASIN(0.5)	10.25
COS	Cosine Function 〈Real Number Variable〉 = COS(〈Input Value〉)	REAL AA AA = COS(0)	10.26
ACOS	ArccoSine Function 〈Real Number Variable〉 = ACOS(〈Input Value〉)	REAL AA AA = ACOS(0.5)	10.27
TAN	Tangent Function 〈Real Number Variable〉 = TAN(〈Input Value〉)	REAL AA AA = TAN(RAD(45))	10.28
ATAN	Arctangent Function 〈Real Number Variable〉 = ATAN(〈Input Value〉)	REAL AA AA = ATAN(-1.0)	10.29

Classification	Description Format	Example in Use	Ref.
ATAN2	2nd Arc Tangent Function <Real Number Variable> = ATAN2(<Y ratio>,<X Ratio>)	REAL AA AA = ATAN2(1,-1)	10.30
MIN	Min Value Function <Real Number Variable> = MIN(<Input Value 1>, <Input Value 2>)	REAL AA AA = MIN(1,2)	10.31
MAX	Max Value Function <Real Number Variable> = MAX(<Input Value 1>, <Input Value 2>)	REAL AA AA = MAX(1.1,2.2)	10.32
RAND	Random Number Generation Function <Number Variable> = RAND()	REAL AA AA = RAND()	10.33

9.3 String

Classification	Description Format	Example in Use	Ref.
ASC	This returns the first character of string into character code. <Integer Variable> = ASC(<String>)	INT VAL VAL = ASC("STATEMENT")	10.34
BINS	This converts an integer to a binary string. <String Variable> = BINS(<Integer Value>)	DEFSTR STR STR = BINS(16)	10.35
CHR	This converts an integer to a character. A character with ASCII code is stored in the string variable. <String Variable> = CHR(<Integer Value>)	DEFSTR STR STR = CHR(65)	10.36
FLUSH	This clears the input/output buffers for serial communication. - Buffer selection range: 1 ~ 3 1: Input Buffer Clear 2: Output Buffer Clear 3: Input/output Buffer Clear FLUSH <Clear Buffer Selection>	FLUSH 1	10.37

Classification	Description Format	Example in Use	Ref.
FTOS	This alters integer or real number to string. <String Variable> = FTOS(<Input Value>)	DEFSTR STR STR = FTOS(10.5)	10.38
HTOS	This alters an integer to a hexadecimal string. <String Variable> = HTOS(<Input Value>)	DEFSTR STR STR = HTOS(100)	10.39
SLEFT	This extracts the characters as many as the number in the left side of string and stores it in the string variable. <String Variable> = SLEFT(<String>, <Extraction Count>)	DEFSTR STR STR = SLEFT("TEST",2)	10.40
SLEN	This returns string length. <Integer Variable> = SLEN(<String>)	INT LEN LEN = SLEN("STATEMENT")	10.41
SMID	This extracts string as the number of digits from the specified position in the string. <String Variable> = SMID(<String>, <Specified Position>, <Extraction Character Count>)	DEFSTR STR STR = SMID("STATEMENT", 2, 3)	10.42
SPOS	This returns the position of string 2 within string 1. <Integer Variable> = SPOS(<String 1>, <String 2>)	INT VAL VAL = SPOS("TEST LINE", "LINE")	10.43
SRIGHT	This extracts the characters as many as the number in the right side of string and stores it in the string variable. <String Variable> = SRIGHT(<String>, <Extraction Count>)	DEFSTR STR STR = SRIGHT("TEST",2)	10.44
STRIN	This receives the string until the discriminator. <Port Number>: RS-232(0), Ethernet(10, 11, 12) <Standby Time>: Unit of 1ms <String Variable> = STRIN(<Port Number>, <Standby Time>)	DEFSTR STR STR = STRIN(0,1000)	10.45
STROUT	This returns the number of string that was transmitted or not transmitted. <Port Number>: RS-232(0), Ethernet(10, 11, 12) <Integer Variable> = STROUT(<Port Number>, <String>)	INT VAL VAL = STROUT(0,"TEST")	10.46
SVAL	This alters string to number <Number Variable> = SVAL(<String>)	INT VAL VAL = SVAL("1234")	10.47

10. System

10.1 Error

Classification	Description Format	Example in Use	Ref.

SETERR	User arbitrarily creates an alarm.
	SETERR(<User Alarm Number> [, <User Alarm Message>])
		IF IN(0) == 1 THEN
		SETERR(1, "User Err 1")
		ENDIF
			11.1

RESET	This clears the alarm. RESET[(<Robot Number>)]	RESET RESET(1)	11.2
RERROR	This returns the last alarm code. <Robot Number>: Robot is selected by nominating the number (1 ~ 3), 0 indicates the selection of all robots. <Integer Variable> = RERROR([<Robot Number>])	INT ECODE ECODE = RERROR(0) ECODE = RERROR(1)	11.3
RERRCNT	This returns the number of alarms currently occurred. <Robot Number>: Robot is selected by nominating the number (1 ~ 3), 0 indicates the selection of all robots. <Integer Variable> = RERRCNT([<Robot Number>])	INT ECOUNT ECOUNT = RERRCNT(0) ECOUNT = RERRCNT(1)	11.4
RERRCODE	This returns the alarm code corresponding the index number. <Robot Number>: Robot is selected by nominating the number (1 ~ 3), 0 indicates the selection of all robots. <Index Number> indicates alarm occurrence sequence. <Integer Variable> = RERRCODE([<Robot Number>,] <Index Number>)	INT ECODE ECODE = RERRCODE(0,1) ECODE = RERRCODE(1,1)	11.5
RERRTEXT	This returns the alarm code corresponding the index number. <Robot Number>: Robot is selected by nominating the Number (1 ~ 3), 0 indicates the selection of all robots. <Index Number> indicates alarm occurrence sequence. <String Variable> = RERRTEXT([<Robot Number>,] <Index Number>)	DEFSTR ETEXT ETEXT = RERRTEXT(0,1) ETEXT = RERRTEXT(1,2)	11.6
READ_DERR	This reads Driver Alarm Code. <Robot Number>: Robot is selected by nominating the number (1 ~ 3) <Integer Variable> = READ_DERR([<Robot Number>,] <Axis Number>)	INT VALUE VALUE = READ_DERR(1,1)	11.7

10.2 Status Information

Classification	Description Format	Example in Use	Ref.
RSTATE	This checks the status of a robot. - Meaning of <Status Index> 0: ALL, this returns entire status information. 1: ALARM status 2: MC Status (Ready) (0: off, 1: on) 3: Origin completion status 4: In position status 5: Reserved 6: SERVO state (on/off) status 7 ~ 8: Reserved 9: System Mode status (1: System, 0: Robot) 10: Reserved 11: Emergency status 12 ~ 14: Reserved 15: Performing of robot motion execution 16: Checking of fan alarm <Integer Variable> = RSTATE(<Robot Number>, <Status Index>)	INT R_ST R_ST = RSTATE(1,0)	11.8
SWLIMIT	This checks the min or max moving range of each axis. <Max/Min>: 0: Min Value of Moving Range, 1: Max Value of Moving Range <Real Number Variable> = SWLIMIT(<Robot Number>, <Axis Number>, <Maximum/Minimum>)	REAL L_LIMIT, R_LIMIT L_LIMIT = SWLIMIT(1,1,0) R_LIMIT = SWLIMIT(1,1,1)	11.9
READ_RPM	This checks current RPM value. <Real Number Variable> = READ_RPM(<Robot Number>, <Axis Number>)	REAL VALUE VALUE = READ_RPM(1,1)	11.10
READ_LOAD	This checks current load rate value. <Real Number Variable> = READ_LOAD(<Robot Number>, <Axis Number>)	REAL VALUE VALUE = READ_LOAD(1,2)	11.11
READ_TRQMAX	This checks max load rate value. <Real Number Variable> = READ_TRQMAX(<Robot Number>, <Axis Number>)	REAL VALUE VALUE = READ_TRQMAX(1,3)	11.12
READ_VEL	This checks current velocity value. <Real Number Variable> = READ_VEL(<Robot Number>, <Axis Number>)	REAL VALUE VALUE = READ_VEL(1,4)	11.13
READ_TRQ	This checks current torque value. <Real Number Variable> = READ_TRQ(<Robot Number>, <Axis Number>)	REAL VALUE VALUE = READ_TRQ(1,1)	11.14

Classification	Description Format	Example in Use	Ref.
READ_MTEMP	This checks current motor encoder temperature value. <Real Number Variable> = READ_MTEMP(<Robot Number>, <Axis Number>)	REAL VALUE VALUE = READ_MTEMP(1,1)	11.15
READ_CTEMP	This checks current internal temperature value of controller. <Real Number Variable> = READ_CTEMP()	REAL VALUE VALUE = READ_CTEMP()	11.16

10.3 Log and Data Storage

Classification	Description Format	Example in Use	Ref.
PRINT	This leaves logs can be checked from monitoring screen. <Index> Range: 0 ~ 999 PRINT(<Index>, <Log Content 1> [{, <Log Content 2>, ··· <Log Content N>}])	PRINT(0,"PRINT LOG")	11.17
WLOG	This stores the user logs including time information in sequence. WLOG(<Log Content 1> [{, <Log Content 2>, ··· <Log Content N>}]))	WLOG("WLOG TEST")	11.18
CLOG	※ **Not currently supported.**	-	11.19
WGPNT	This stores Global Point Variable as a file. <Synchronization>: This selects standby status until a file is synchronized to a disk. (0: No Standby, 1: Standby) - Return Value: 0(Normal), 1(Writing Error) [<Integer Variable> =] WGPNT(<Synchronization>)	WGPNT(1) INT VAL VAL = WGPNT(1)	11.20
WGINT	This stores Global Integer Variable as a file. <Synchronization>: This selects standby status until a file is synchronized to a disk. (0: No Standby, 1: Standby) - Return Value: 0(Normal), 1(Writing Error) [<Integer Variable> =] WGINT(<Synchronization>)	WGINT(1) INT VAL VAL = WGINT (1)	11.21
WGFLT	This stores global real number variable as a file. <Synchronization>: This selects standby status until a file is synchronized to a disk. (0: No Standby, 1: Standby) - Return Value: 0(Normal), 1(Writing Error) [<Integer Variable> =] WGFLT(<Synchronization>)	WGFLT(1) INT VAL VAL = WGFLT (1)	11.22
TACT_START	Sets Start position for Tact Time measurement TACT_START()	INT VAL TACT_START() DELAY 1000 VAL = TACT_STOP()	11.23
TACT_STOP	Sets End position for Tact Time measurement. [<Variable> =] TACT_STOP()		11.24

11. Commands dedicated to System Mode

Classification	Description Format	Example in Use	Ref.
PSEL	This enables a matching of the JOB program that was entered in thread number of specified robot. <Thread Number>: only 1 ~ 4 can be used. [<Integer Variable> =] PSEL(<Thread Number>, "<JOB File Name>")	INT R_VAL R_VAL = PSEL(1,"TEST")	12.1
PSTART	This executes the JOB program of thread number of specified robot. <Thread Number>: only 1 ~ 4 can be used. [<Integer Variable> =] PSTART(<Thread Number>)	PSTART(1)	12.2
PSTOP	This stops the JOB program of thread number of specified robot. <Thread Number>: only 1 ~ 4 can be used. [<Integer Variable> =] PSTOP(<Thread Number>)	PSTOP(1)	12.3
PSTATE	This reads the status of thread number of specified robot. <Thread Number>: only 1 ~ 4 can be used. <Integer Variable> = PSTATE(<Thread Number>)	INT STATE STATE = PSTATE(1)	12.4
PRESTART	This re-starts the JOB program of thread number of specified robot. <Thread Number>: only 1 ~ 4 can be used. [<Integer Variable> =] PRESTART(<Thread Number>)	PRESTART(1)	12.5
PEXIT	This releases the JOB program of thread number of specified robot. <Thread Number>: only 1 ~ 4 can be used. [<Integer Variable> =] PEXIT(<Thread Number>)	PEXIT(1)	12.6

12. Commands TCP IP Client

Classification	Description Format	Example in Use	Ref.
NET_OPEN	creates TCP I/P Client Socket of the Controller and requests for connection to the Server <IP Number>: input IP Number of the Server <Port Number>: input Port Number of the Server <Wait Time>: input Receiving Wait Time (msec) after requesting <Integer Variable> = NET_OPEN(<IP Number>, <Port Number>, <Wait Time>)	INT FD_NUM DEFSTR IP_NUM INT PORT_NUM INT WAIT_TIME IP_NUM = "192.168.1.196" PORT_NUM = 10000 WAIT_TIME = 1000 FD_NUM = NET_OPEN(IP_NUM, PORT_NUM, WAIT_TIME)	13.1
NET_CLOSE	Ends TCP I/P Communication of the Controller and removes the Client Socket <File Descriptor Number>: File Descriptor (File descriptor) Number for Access to socket information [<Integer Variable> =] NET_CLOSE(<File Descriptor Number>)	INT FD_NUM ..... RET_VEL = NET_CLOSE(FD_NUM)	13.2
NET_STATUS	Function to check TCP I/P Communication status of the Controller <File Descriptor Number>: File Descriptor (File descriptor) Number for Access to socket information <Integer Variable> = NET_STATUS(<File Descriptor Number>)	INT FD_NUM ..... RET_VEL = NET_STATUS(FD_NUM)	13.3
NET_KEEPALIVE	Sets option to check Communication status of the TCP I/P Client Socket of the Controller <File Descriptor Number>: File Descriptor (File descriptor) Number for Access to socket information <Holding Time>: Time(sec) from time when the last Packet is transmitted to Time when Search Packet is started to be sent <Max search count>: Max repeat count in case of no Response of the Search Packet <Search period>: interval (sec) to send Search Packet <Integer Variable> = NET_KEEPALIVE(<File Descriptor Number>, <Holding Time>, <Max search count>, <Search period>)	INT FD_NUM ..... INT RET_VEL INT KEEP_IDLE INT KEEP_CNT INT KEEP_INTERVAL FD_NUM = 10000 KEEP_IDEL = 10 KEEP_CNT = 20 KEEP_INTERVAL = 3 RET_VEL = NET_KEEPALIVE(FD_NUM, KEEP_IDLE, KEEP_CNT, KEEP_INTERVAL)	13.4
NET_FD	Returns Socket File Descriptor of the Client connected to the TCP I/P Server <Index>: Port Number Designation Index of the Server <Integer Variable> = NET_FD(<Index>)	INT FD_NUM INT INDEX INDEX = 10 FD_NUM = NET_FD(<INDEX>)	13.5
NET_STRIN	Receives Character String to Separator <File Descriptor Number>: File Descriptor (File descriptor) Number for Access to socket information <Wait Time>: unit 1ms <Character String Variable> = NET_STRIN(<File Descriptor Number>, <Wait Time>)	DEFSTR STR INT FD_NUM ..... STR = NET_STRIN(FD_NUM,1000)	13.6
NET_STROUT	Sends Character String, returns number of unsent Character Receives Character String to Separator <File Descriptor Number>: Access to socket information <Character String Variable>: Character String to be output <Integer Variable> = NET_STRIN(<Port Number>, <Wait Time>)	INT VAL DEFSTR STR INT FD_NUM ..... VAL = NET_STROUT(FD_NUM, STR)	13.7

Chapter 4 Commend Reference

1. Format of Command Reference

This describes in detail for meanings and a method to use as the format shown below.

(1)<Section> (2)<Command Name>	
(3) Usage	
	This describes the purpose of a command.
4) Syntax	
	This describes the right method of using commands. The “<” and “>” are represented in case that the transfer factors are included in the commands. The “,” is used in case that there are multiple transfer factors. Big parentheses “[“ and “]” are used for elements that can be ignored. Array is displayed by ‘[‘ and ‘]’. When only one element is required in the element set, the set is described in the parenthesis “{” and “}” and each transfer factor is separated by the symbol ”|”. In case of requiring a transfer factor inevitably or the special cases such as functions, the transfer factor is described within small parentheses “(“ and “)” after a command.
(5) Description	
	This describes characteristics and functions of the commands.
(6) Return value	
	In case that there exists a return value when using the commands, the shape and purpose of the corresponding returned value are described.
(7) Arguments	
	The shape and purpose of transfer factors included are described.
(9) Examples	
	This shows examples about how to use commands.
(10) Related terms	
	This shows other related commands or explanations.

Figure 4-1 Format of Command Reference

2. Variable

2.1 INT

Usage

This declares an integer variable that can be used in the program inside.

Syntax

INT [$]<Variable Name>['[' <Array Count> ']'] [, <Variable Name>['[' <Array Count> ']'], …]

Description

This declares the integer variable in size of 4 bytes.
After declaration of variables, an initialization is required, but declaration and initialization cannot be performed simultaneously.
Range: -2,147,483,648 ~ 2,147,438,647

1) When using the "$" symbol as prefix of a variable, the variable can be used as a global variable. When used as a global variable in such case, it must be declared in out of the function.
2) Both English characters and numbers can be used in <Variable Name> and the first character must be an English character or "_" symbol.
3) If using two of this symbol ",", more than two variables can be declared.
4) Multiple data can be tied up together and used by allocating "["<Array Count>"]".

Examples

```
INT $AA                          // Declare a global variable

MAIN                             // Declare Main Function Start
    INT D_1, D_2, D_3[2]         // Declare local variables
    $AA = 15                     // Initialize the global variable
    D_1 = 0                      // Assign a constant value to D_1
    D_2 = I100                   // Assign the global variable I100 value to D_2
    D_3[0] = $AA                 // Assign the global variable $AA value to D_3
EOP                              // Main Function End
```

Related terms

I, REAL, DEFSTR



1) When declaring the variables, the characters starting with "P", "GP", "I" and "F" cannot be used because those are duplicated with the global variables that were already declared.
2) Caution is required because variable declaration and initialization cannot be done simultaneously.

2.2 REAL

Usage

This declares a real number variable that can be used in a program inside.

Syntax

REAL [$]<Variable Name>['[' <Array Count> ']'] [, <Variable Name>>['[' <Array Count> ']'], ···]

Description

This declares a real number variable in size of 4 bytes.
After declaration of variables, an initialization is required, but declaration and initialization cannot be performed simultaneously.
Range: 3.402,823E-38 ~ 3.402,823E+38

1) When using the "$" symbol as prefix of a variable, the variable can be used as a global variable. When used as a global variable in such case, it must be declared in out of the function "$".
2) Both English characters and numbers can be used in <Variable Name> and the first character must be an English character or "_" symbol.
3) If using two of this symbol ",", more than two variables can be declared.
4) Multiple data can be tied up together and used by allocating "["<Array Count>"]".

Examples

```
REAL $AA                    // Declare a global variable

MAIN                        // Declare Main Function Start
    REAL D_1, D_2, D_3[2]   // Declare local variables
    $AA = 15.15             // Initialize the global variable
    D_1 = 0.58              // Assign a constant value to D_1
    D_2 = F100              // Assign the global variable I100 value to D_2
    D_3[0] = $AA            // Assign the global variable $AA value to D_3
EOP                         // Main Function End
```

Related terms

F, INT, DEFSTR



1) When declaring the variables, the characters starting with "P", "GP", "I" and "F" cannot be used because those are duplicated with the global variables that were already declared.
2) Caution is required because variable declaration and initialization cannot be done simultaneously.

2.3 DEFSTR

Usage

This declares a string variable that can be used in a program inside.

Syntax

DEFSTR [$]<Variable Name>['[' <Array Count>']'] [, <Variable Name>['[' <Array Count> ']'],…]

Description

This declares a string variable.
Initialization must be performed after declaration, however initialization and Declaration cannot be done simultaneously.
Max number of characters should not exceed 100.

1) When using the "$" symbol as prefix of a variable, the variable can be used as a global variable. When used as a global variable in such case, it must be declared in out of the function "$".
2) Both English characters and numbers can be used in <Variable Name> and the first character must be an English character or "_" symbol.
3) If using two of this symbol ",", more than two variables can be declared.
4) Multiple data can be tied up together and used by allocating "["<Array Count>"]".

Examples

```
DEFSTR $STR                     // Declare a global variable

MAIN                            // Declare Main Function Start
    DEFSTR D_1, D_2[2]          // Declare local variables
    $STR = "STRING"             // Initialize the global variable
    D_1 = "CHECK"               // Assign a constant value to D_1
    D_2[0] = $STR               // Assign the global variable I100 value to D_2
EOP                             // Main Function End
```

Related terms

INT, REAL



1) When declaring the variables, the characters starting with "P", "GP", "I" and "F" cannot be used because those are duplicated with the global variables that were already declared.
2) Caution is required because variable declaration and initialization cannot be done simultaneously.

2.4 POS

Usage

This declares a position variable in the joint coordinate system.
This can be used in a program inside.

Syntax

POS [$]<Variable Name>['['<Array Count>']'][,<Variable Name>['['<Array Count>']'],…]

Description

This declares a position variable in the joint coordinate system. Position value can be stored upto DOF (degree of freedom). Initialization must be done after declaration. However, initialization and declaration cannot be done simultaneously. Initialization can be achieved by an access of position constant or individual element.

1) When using the "$" symbol as prefix of a variable, the variable can be used as a global variable. When used as a global variable in such case, it must be declared in out of the function.
2) Both English characters and numbers can be used in <Variable Name> and the first character must be an English character or "_" symbol.
3) If using two of this symbol ",", more than two variables can be declared.
4) Multiple data can be tied up together and used by allocating "["<Array Count>"]".

Examples

```
POS $AP                                  // Declare a global variable
MAIN                                     // Declare Main Function Start
    POS D_1, D_2[2]                      // Declare local variables
    $AP = <100,100,100,100,100,100>      // Initialize the global variable, 6 axes
    D_1.1 = 100.0                        // Local variable Individual Element Access Method
    D_1.2 = 120.1
    D_1.3 = 130.2
    D_1.4 = 140.3
    D_1.5 = 150.4
    D_1.6 = 160.5
    D_1.USED = 1                         // D_1 Initialization Completion
    D_2[0] =$AP                          // Assign the global variable $AP value to D_2
EOP                                      // Main Function End
```

Related terms

XPOS, UPOS, P, GP



1) When declaring the variables, the characters starting with "P", "GP", "I" and "F" cannot be used because those are duplicated with the global variables that were already declared.
2) Caution is required because variable declaration and initialization cannot be done simultaneously.

2.5 XPOS

Usage

This declares a position variable of base coordinate system that can be used in a program inside.

Syntax

XPOS [$]<Variable Name>['['<Array Count> ']'], <Variable Name>['[' <Array Count> ']'], ···]

Description

This declares a position variable of base coordinate system. Position value can be stored upto DOF (degree of freedom) in max. Initialization must be done after declaration. However, initialization and declaration cannot be done simultaneously. Initialization can be achieved by an access of positional constant or individual element.

1) When using the "$" symbol as prefix of a variable, the variable can be used as a global variable. When used as a global variable in such case, it must be declared in out of the function "$".
2) Both English characters and numbers can be used in <Variable Name> and the first character must be an English character or "_" symbol.
3) If using two of this symbol ",", more than two variables can be declared.
4) Multiple data can be tied up together and used by allocating "["<Array Count>"]".

Examples

```
XPOS $AP                                          // Declare a global variable
MAIN                                              // Declare Main Function Start
    XPOS D_1, D_2[2]                              // Declare local variables
    $AP = <100,100,100,100,100,100,ARM = 0,T = 1> // Initialize the global variable (6axis)
    D_1.1 = 100.0                                 // Local variable Individual Element
    D_1.2 = 120.1                                 //Access Method
    D_1.3 = 130.2
    D_1.4 = 140.3
    D_1.5 = 150.4
    D_1.6 = 160.5
    D_1.ARM = 111                                 // Assign a Robot Posture
    D_1.T = 2                                     // Assign a Robot Tool Number
    D_1.USED = 1                                  // D_1 Initialization Completion
    D_2[0] = $AP                                  // Assign the variable $AP value to
                                                  // D_2
EOP                                               // Main Function End
```

Related terms

POS, UPOS, P, GP



1) When declaring the variables, the characters starting with “P”, “GP”, “I” and “F” cannot be used because those are duplicated with the global variables that were already declared.
2) Caution is required because variable declaration and initialization cannot be done simultaneously.

2.6 UPOS

Usage

This declares a position variable of user coordinate system that can be used in a program inside.

Syntax

UPOS [$]<Variable Name>['['<Array Count>']'] [, <Variable Name>['['<Array Count>']'], ···]

Description

This declares a position variable of user coordinate system. Position value can be stored upto DOF (degree of freedom) in max. Initialization must be done after declaration. However, initialization and declaration cannot be done simultaneously. Initialization can be achieved by an access of positional constant or individual element.

1) When using the "$" symbol as prefix of a variable, the variable can be used as a global variable. When used as a global variable in such case, it must be declared in out of the function "$".
2) Both English characters and numbers can be used in <Variable Name> and the first character must be an English character or "_" symbol.
3) If using two of this symbol ",", more than two variables can be declared.
4) Multiple data can be tied up together and used by allocating "["<Array Count>"]".

Examples

```
UPOS $AP                                  // Declare a global variable
MAIN                                      // Declare Main Function Start
    UPOS D_1, D_2[2]                      // Declare local variables
    $AP = <100,100,100,100,100,100>       // Initialize the global variable, 6 axes
    D_1.1 = 100.0                         // Local variable Individual Element Access
    D_1.2 = 120.1                         // Method
    D_1.3 = 130.2
    D_1.4 = 140.3
    D_1.5 = 150.4
    D_1.6 = 160.5
    D_1.ARM = 111                         // Assign a Robot Posture
    D_1.T = 2                             // Assign a Robot Tool Number
    D_1.U = 3                             // Assign a Robot User Number
    D_1.USED = 1                          // D_1 Initialization Completion
    D_2[0] = $AP                          // Assign the variable $AP value to D_2
EOP                                       // Main Function End
```

Related terms

POS, XPOS, P, GP

1) When declaring the variables, the characters starting with “P”, “GP”, “I” and “F” cannot be used because those are duplicated with the global variables that were already declared.
2) Caution is required because variable declaration and initialization cannot be done simultaneously.

2.7 Program Position P

Usage

This is a position variable of joint coordinate system that was predefined in the system and the position value is stored in a file. This has a characteristic that is dependent on a program. This is used as a global variable in the corresponding program.

Syntax

P<Number> or P'['<Number>']'

Description

This enables to store the position value upto max DOF (degree of freedom) same as a position variable of joint coordinate system that was defined as POS command.
Unlike GP, this enables to store different values in each program file and is a global variable that can be used in a program file only.
Unlike a local variable, this can be used without declaring the program.
When executing a program, an editing is not possible and only reading a value is available with individual element access.

1) The <Number> can be entered just like Array and immediately created to use. No space is allowed between character P and <Number>.
2) The range of <Number> is 0 ~ 999 and total 1,000 numbers can be used.

Examples

```
REAL AA, BB          // Declare a real number variables
AA = P0.1            // Assign P0.1 value to AA variable
BB = P[0].2          // Assign P[0].2 value to BB variable
SVON                 // Servo on
JMOV P0              // Move JMOV to P0 position.
```

Related terms

POS, XPOS, UPOS, GP



1) It is not possible to alter the value during a program execution. Data editing on variable is available in only teaching mode.

2.8 Global Integer I

Usage

This is an integer global variable that was predefined and can be commonly used in the entire program.

Syntax

I<Number> or I'['<Number>']'

Description

This enables to store the integer value in a size of 4 bytes same as an integer variable declared as INT command.
Unlike a local variable, this can be used without declaration in a program.
The value can be altered during the execution of a program. After altering the value, it can be stored in a file using WGINT function.

1) The <Number> can be entered just like array and immediately created to use. No blank is allowed between the character I and <Number>.
2) The range of <Number> is 0 ~ 999 and total 1,000 numbers can be used.

Examples

```
I[0] = 100          // Assign 100 to 0th I variable
I[999] = -1         // Assign -1 to 999th I variable
```

Related terms

INT, F, WGINT



1) WGINT function must be used to store the value that was altered during a program execution.
2) If a store command is not executed after altering a global variable during a program execution, the altered data are not secured in case of power cut-off or re-booting.
3) There is a possibility to lose data if power is abruptly turned off during file storing.

2.9 Global Real F

Usage

This is a real number global variable that was predefined in the system and is commonly used in the entire program.

Syntax

F<Number> or F['<Number>']

Description

This enables to store the real number value in the size of 4 bytes same as the real variable as REAL.
Unlike the local variables, this can be used without declaration in a program.
The value can be altered during the execution of a program. After altering the value, it can be stored in a file using WGFLT function.

1) The <Number> can be entered just like an array and immediately created to use. No blank is allowed between the character F and <Number>.
2) The range of <Number> is 0 ~ 999 and total 1,000 numbers can be used.

Examples

```
F[0] = 100      // Assign a constant value to the 0th F variable
F[999] = -1     // Assign a constant value to the 999th F variable
```

Related terms

REAL, I, WGFLT



1) WGFLT function must be used to store the value that was altered during a program execution.
2) If a store command is not executed after altering a global variable during a program execution, the altered data are not secured in case of power cut-off or re-booting.
3) There is a possibility to lose data if power is abruptly turned off during file storing.

2.10 Global Position GP

Usage

This is a position global variable of joint coordinate system that was predefined in the system and is commonly used in the entire program.

Syntax

GP<Number> or GP'['<Number>']'

Description

This enables to store the position value upto DOF (degree of freedom) same as a position variable of joint coordinate system that was defined as POS command.
Unlike the local variables, this can be used without declaration in a program.
The value can be altered during the execution of a program. After altering the value, it can be stored in a file using WGPNT function.
It is possible to assign the value through the position variables and to alter the value with individual element access.

1) The <Number> can be entered just like an array and immediately created to use. No blank is allowed between character GP and <Number>.
2) The range of <Number> is 0 ~ 1,999 and total 2,000 numbers can be used.

Examples

```
GP[20] = <100,100,100,100,100,100>     // Assign a constant value to the 20th GP variable
GP[20].1 = 200
GP[20].2 = 50
GP[20].3 = GP[20].1+GP[20].2           // The 20th Individual Element Access Method
GP[20].USED = 1
```

Related terms

POS, XPOS, UPOS, P, WGPNT



1) **※ WGPNT function must be used to store the value that was altered during a program execution.**
2) If a store command is not executed after altering a global variable during a program execution, the altered data are not secured in case of power cut-off or re-booting.
3) There is a possibility to lose data if power is abruptly turned off during file storing.

2.11 TMR

Usage

This is a timer variable that was predefined in the system.
This is used to read or specify the current count value.

Syntax

TMR(<Number>) = <Value>
<Variable> = TMR(<Number>)

Description

Unlike the local variables, this can be used without declaration in a program.
The value is allocated at the time when the integer value is entered and increases by 1 in every 10 ms interval.
This is initialized when meeting the value that was set in the itme, 1.HW_CONF(2/3) → 3.TMR.

1) The <Number> is used by entering a number between “(” and “)”.
2) The input value must be the integer variables or constants.

Return value

<Variable> Data type: <Integer Type>
The range of <Integer Type> is 0 ~ 60,000.

Arguments

<Number> Data type: <Integer Type>
The range of <Number> is 0 ~ 1.

Examples

```
INT AA, BB              // Declare an integer variables
AA = -100               // Assign the initial value to AA variable
TMR(0) = -50            // Input the count value -50 in TMR 0
TMR(1) = AA             // Input AA value in TMR 1
DLAY 100                // 0.1 second Time Delay
BB = TMR(0)             // Store the TMR 0 count value in BB variable
```

2.12 MVR

Usage

This is a variable that was predefined in the system and returns the moving distance ratio of the total moving distance.

Syntax

[<Variable> =] MVR[(<Robot Number>)]
MVR[(<Robot Number>)] = <Value>

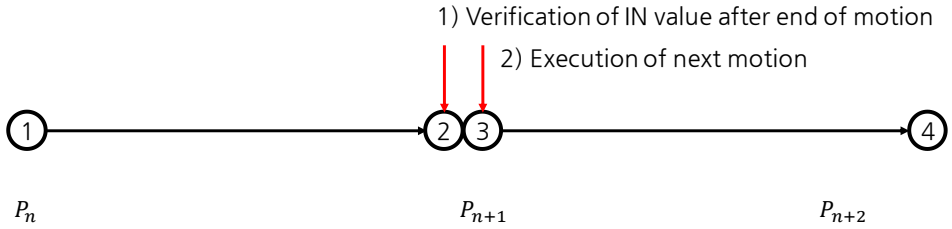
Description

Unlike the local variables, this can be used without declaration in a program.
This returns the percentage value of the total moving distance.

- MVR = (Current Moving Distance / Total Moving Distance) * 100

This is frequently used in both conditional divergence and I/O parallel processing among robot motions. In this case, it is used together with the WITH … ENDWT statement.
This can have an advantage of reducing the entire program execution speed by processing the necessary conditions in advance required when moving to next position (P_{n+1}, P_{n+2}) from the position (P_{n+1}, P_{n+2}) during the movement as shown below.

1) Example of common use



2) Example of using MVR

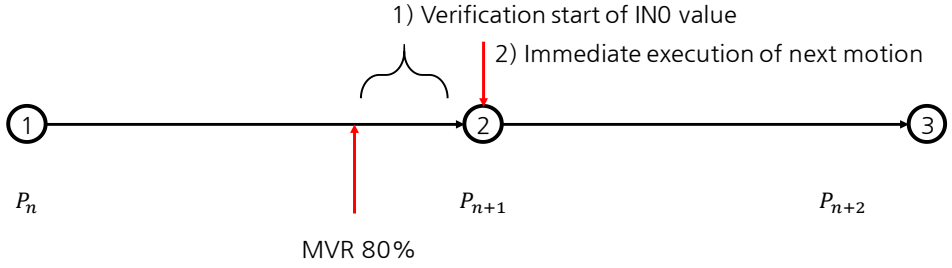


Figure 4-2 Example of Comparison between Common Use and MVR Use

The figure above illustrates examples of reducing the entire program execution speed by checking IN0 value in advance that is the moving execution condition of a next position when moving from a position (P_n, P_{n+1}).

1) By entering the <Robot Number>, it is possible to bring in the moving distance ratio of another robot other than the robot that was selected when storing the JOB program.

Return value

<Variable> Data type: <Integer Type>

Arguments

<Robot Number> Data type: <Integer Type>
The range of <Robot Number> is 1 ~ 3.

Examples

Example 1

```
REAL VALUE                  // Declare an integer variable
VALUE = MVR                 // Assign MVR value
VALUE = MVR(2)              // Assign MVR value of Robot 2
```

Example 2

```
SVON                        // Servo on
MVR = 0                     // Initialize MVR value to be 0
WITH                        // WITH statement Start
    JMOV P0                 // Move JMOV from current position to P0
    WHILE MVR <100          // Iterate the execution if MVR is smaller than 100
        IF (MVR>80)&&(IN0 == 1)  // In case that MVR is over 80% and IN0 is on(1)
        THEN
            OUT0 = 1        // Alter the signal of OUT0 to on(1)
            GOTO BB         // Move to the branch point BB
        ENDIF               // IF statement End
    ENDWL                   // Endless Loop End
ENDWT                       // WITH statement End
LABL BB                     // Declare a branch point BB
    JMOV P1                 // Move JMOV from current position to P1
```

Related terms

HERE, HERE_REF, WITH ··· ENDWT, WHILE ··· ENDWL, IF ···(ELSE IF ···ELSE···) ENDIF

1) It may occur that the conditions cannot be checked because MVR value alters rapidly when operating with high speed in short moving distance. Caution is required in setting MVR value. In addition, it is necessary to lower the speed of robot in such a case.

2.13 HERE

Usage

This is a predefined variable in the system and alters the current position with the joint angle value.

Syntax

〈Variable〉 = HERE
〈Variable〉 = HERE(〈Robot Number〉)

Description

This returns current position of robot with joint angle value. When trying to store the value in other position variables, an alarm occurs.

1) It is possible to bring the position value of another robot that was not stored in the JOB program by inputting 〈Robot Number〉.

Return value

〈Variable〉	Data type: 〈Position Type of Joint Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.

Examples

Example 1	POS CUR_POS CUR_POS = HERE	// Declare a position variable of joint coordinate system // Assign a current joint angle value
Example 2	POS R2_POS R2_POS = HERE(2)	// Declare a position variable of joint coordinate system // Assign a current joint angle value of the 2nd robot

Related terms

MVR, POS, P, GP, HERE_REF

2.14 HERE_REF

Usage

This is a predefined variable in the system and returns the current reference position with the joint angle value.

Syntax

〈Variable〉 = HERE_REF
〈Variable〉 = HERE_REF (〈Robot Number〉)

Description

This returns the current reference position with the joint angle value. When trying to store the value in other position variables, an alarm occurs.

1) It is possible to bring the position value of another robot that was not stored in the JOB program by inputting 〈Robot Number〉.

Return value

〈Variable〉	Data type: 〈Position Type of Joint Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.

Examples

Example 1	POS CUR_POS CUR_POS = HERE_REF	// Declare a position variable of joint coordinate system // Assign a current reference joint angle value
Example 2	POS R2_POS R2_POS = HERE_REF(2)	// Declare a position variable of joint coordinate system // Assign a current reference joint angle value of the 2nd robot

Related terms

MVR, POS, P, GP, HERE

3. Preprocessor & Comments

3.1 INCLUDE

Usage

This allows to use the specific program file as one program by attaching it current position.

Syntax

INCLUDE “<JOB File Name>”

Description

INCLUDE command must be used before creating a Main Function.
It is possible to use the internal function of attached JOB file and a global variable.
When trying to execute a job main program of attached file, “_MAIN()” must be added in the JOB file behind.

1) Write down only the file name at <JOB File Name> except the file end-mark “.” and the extension “JOB”.

Examples

Example 1

```
INCLUDE “INC”                    // Attaches a file INC.JOB

MAIN                             // Declare Main Function Start
    ...
EOP                              // Main Function End
```

Example 2

```
INCLUDE “INC”                    // This attaches a file INC.JOB

MAIN                             // Declare Main Function Start
    INC_MAIN()                   // Execute Main Function of a file INC.JOB
EOP                              // Main Function End
```

Related terms

JCALL

3.2 Comment

Usage

This do process so as not to have any effect on the execution.

Syntax

// [<Comment>]
/* [<Comment>] */

Description

Comments are typically used for creating technical descriptions, references or descriptive statements. They don't effect on the program execution.

1) When using the "//" mark, the entire line from the right side of the symbol is treated as a comment.
2) When using the "/*" and "*/" symbols, all string between two symbols are treated as a comment. This is used when using the multiple line as a comment.

Arguments

<Comment>	All strings

Examples

Example 1

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    JMOV P0             // Move JMOV from current position to P0
    //DLAY 1000         // Comment with a single line
    JMOV P1             // Move JMOV from current position to P1
    DLAY 1000           // 1 second Time Delay
EOP                     // Main Function End
```

Example 2

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    /*                  // Comment with multiple lines Start
    JMOV P0
    DLAY 1000
    */                  // Comment with multiple lines End
    JMOV P1             // Move JMOV from current position to P1
    DLAY 1000           // 1 second Time Delay
EOP                     // Main Function End
```

4. Operator

4.1 Assignment Operator

Usage	
	This is used for assigning a new value to a variable. This can be used in everything including constants and expressions.
Syntax	
	〈Variable〉 = 〈All constants, variables or expressions〉
Description	
	This alters the data to fit the variable type and stores. Alarm occurs if type conversion is not available.
Return value	
〈Variable〉	Data type: 〈All Type〉
Arguments	
	Data type: A value or result of expression is identical to the return value

Examples

Example 1	INT AA, BB, CC
	AA = 1
	BB = AA
	CC = (AA+BB)*5
		// Declare an integer variable
		// Assign a constant to a variable
		// Assign a variable to a variable
		// Assign an expression to a variable

Example 2	DEFSTR STR1, STR2
	STR1 = “Constant”
	STR2 = STR1+” ASSIGN”
		// Declare a string variable
		// Assign a constant to a variable
		// Assign an expression to a variable

Example 3	POS AP, BP, CP, DP
	AP = P0
	BP = GP0
	CP = 〈0,0,0,0,0,0〉
	DP = AP+BP+CP
		// Declare a position variable
		// Assign a program variable to a variable
		// Assign a global variable to a variable
		// Assign a constant to a variable
		// Assign an expression to a variable

4.2 Arithmetic Operator

Usage

This is an operator that performs an arithmetic operation.

Syntax

〈Variable〉 = 〈Term A〉〈Arithmetic Operator〉〈Term B〉

Description

This is used for defining the arithmetic operation type.
It follows the rules shown in the table below.

Symbol	Meaning	Applicable Data Type	
		Term A	Term B
+	Addition	All data types	
-	Subtraction	All data types except string	
*	Multiplication	All data types except string	Integer or real number
/	Division	All data types except string	Integer or real number
%	Remainder	Integer (Even if the real number is entered, it is altered to a integer)	

Refer to chapter2, section 9.3 Arithmetic Operator for the decision of sign when calculating the remainder (%) of division.

Return value

〈Variable〉 Data type: 〈All Type〉

Arguments

Data type: A value or result of expression is identical to the return value

Examples

Example 1
```
I2 = I0 + I1          // Add integer variables
GP2 = P0 + P1         // Add position variables
STR = “ADD” + “STRING” // Add string variables
```

Example 2
```
F2 = F0 - F1          // Subtract real numbers
GP2 = GP0 - GP1       // Subtract position variables
GP3 = GP0 - 1         // Subtract 1 from all positional individual data
```

Example 3
```
I2 = I0 * I1          // Multiply integer variables
GP2 = P0 * 1.5        // Multiply 1.5 to all positional individual data
```

Example 4
```
F2 = F0 / F1          // Divide real numbers
GP2 = 〈100,100,100,100,100,100〉 / 2  // Divide all positional individual data by 2
```

Example 5
```
I2 = I0 % I1          // Calculate the integer remainder
```

4.3 Relational Operator

Usage

This is an operator that compares the given operands to check the effectiveness of specified relationship. This is mainly used in the conditional expressions.

Syntax

[<Variable> =]<Term A><Relational Operator><Term B>

Description

This is mainly used together with the integer and real number data and applied in the expressions of IF or WHILE statements. Table below explains the meaning of each symbol.

Symbol	Meaning
<	Less
>	Bigger
<=	Less or Equal
>=	Bigger or Equal
==	Equal
!=	Not Equal

Return value

<Variable> Data type: <Integer Type>
The logic value (true: 1, false: 0) is returned as a result of relational operation.

Arguments

Data type: <Integer or Real Number Type>

Examples

Example 1

```
IF I0> I1 THEN          // If the value of I0 is bigger than that of I1
    OUT0 = 1            // This outputs 1 as the signal of OUT0
ENDIF                   // IF statement End
```

Example 2

```
INT VALUE               // Declare an integer variable
VALUE = F0> F1          // This compares the values of F0 and F1, then returns 1.
                        // if F0 is bigger, returns 0 if F1 is bigger
```

4.4 Logical Operator

Usage

This is an operator that is used to check the logical relationship of one or two operands.

Syntax

[<Variable> =]<Term A><Logic Operator><Term B>
[<Variable> =]<Logic Operator><Term A>

Description

Logical operators do not perform the general arithmetic conversion. Instead, they evaluate each operand as a concept corresponding to logical values (true: 1, false: 0).

Symbol	Meaning
&&	Logic AND operator returns 1 if both two operands are not 0. If any one of operands is 0, then the result is 0.
||	Logic OR operator returns 0 if both two operands are 0. If any one of operands is not 0, then the result is 1.
!	Logic NOT operator creates 0 if the corresponding operand is true (values other than 0), and 1 for false (0).

Return value

<Variable> Data type: <Integer Type>
The logic value (true: 1, false: 0) is returned as a result of logical operation.

Arguments

Data type: <Integer or Real Number Type>

Examples

Example 1

```
IF I0 <I1 && I2> I3 THEN          // If I0 value is smaller than I1 value and I2 value is bigger
                                  // than I3 value
    OUT(0) = 1                    // 1 is output as a signal of OUT0
ENDIF                             // IF Statement End
```

Example 2

```
SVON                              // Servo on
IF IN0 == 1 || IN1 == 1 THEN      // If IN0 value is 1 or IN1 value is 1,
    JMOV P0                       // Move JMOV from current position to P0
ENDIF                             // IF Statement End
```

Example 3

```
INT VALUE                         // Declare an integer variable
VALUE = 0                         // Initialize the variable
IF !VALUE THEN                    // If the value of VALUE variable is not 1(true),
    OUT(0) = 1                    // 1 is output as a signal of OUT0
ENDIF                             // IF Statement End
```

4.5 Bit Operator

Usage

This is an operator that is applied in one or two operands by the unit of a bit.

Syntax

[〈Variable〉 =]〈Term A〉〈Bit Operator〉〈Term B〉
[〈Variable〉 =]〈Bit Operator〉〈Term A〉

Description

Bit Operator is similar to Logical Operator but a big difference is to perform an operation by the unit of a bit. In addition, this can be used when moving the entire bits to the left or right.

Symbol	Meaning
&	AND operator in a bit unit sets the result to be 1 if both bits are all 1, otherwise sets to be 0.
^	XOR operator in a bit unit sets the result to be 1 if one bit is 0 and another bit is 1, otherwise sets to be 0.
|	OR operator in a bit unit sets the result to be 1 if any one bit is 1, otherwise sets to be 0.
<<	This moves the first operand to the left by the number that was specified by the second operand.
>>	This moves the first operand to the right by the number that was specified by the second operand.
~	Bit NOT operator makes a bit complement of operand. If a bit is 1, it reverts to 0, if 0 then, reverts to 1.

Return value

〈Variable〉 Data type: 〈Integer〉
The logic value (true: 1, false: 0) is returned as a result of relational operation.

Arguments

Data type: 〈Integer〉
Operand must be an integer value.

Examples

Example 1

```
IF I0 & I1 THEN
    OUT0 = 1
ENDIF
```
// If Bit AND operation of I0 and I1 is true,
// 1 is output as a signal of OUT0
// IF statement End

Example 2

```
INT VALUE
VALUE = I0 ^ I1
```
// Declare an integer variable
// Store bit XOR operation of I0 and I1

Example 3

```
SVON
IF I0 | I1 THEN
    JMOV P0
ENDIF
```
// Servo on
// If bit OR operation of I0 and I1 is true,
// Move JMOV from current position to P0
// IF statement End

Example 4

```
INT A, B
A = 0B1011
B = A<<2
```
// Declare integer variables
// Initialize the variable
// Shift by 2 toward the left

Example 5

```
INT A, B
A = 0B1011
B = A>>1
```
// Declare integer variables
// Initialize the variable
// Shift by 1 toward the right

Example 6

```
INT A, B
A = 0B1011
B = ~ A
```
// Declare integer variables
// Initialize the variable
// Store the value of A variable after reversal

4.6 Parenthesis

Usage

This is an operator that gives an expression or a sub-expression a higher priority than the other expressions by using a pair of parenthesis surrounding the expression so that it can be evaluated in advance by ignoring the priority of general operators.

Syntax

(<Expression>)

Description

This groups the expressions within the parentheses.

Examples

```
INT A, B, C, RES        // Declare integer variables
A = 1                   // Initialize the variables
B = 2
C = 3

RES = (A+B)*C           // This enables to execute the addition (+) operation prior to
                        // the multiplication (*) which has a higher priority.
```

5. Program Control

5.1 MAIN … EOP (Program Start/End)

Usage

This indicates a start and end of a main program.

Syntax

```
MAIN
    …
EOP
```

Description

Every program can create only one Main Function and must have one Main Function in order to be executed.

1) MAIN and EOP must be used together.
2) EOP (End of Program) indicates the end of block in a main program. JOB program is ended when meeting this during the JOB program execution. In case of JOB program that was called by a JCALL, it ends an execution of the called job and returns to the calling job when meeting an EOP command.

Examples

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    JMOV P1             // Move JMOV to P1 from current position
    JMOV P2             // Move JMOV to P2 from current position
EOP                     // Main Function End
```

Related terms

FUNC … END

5.2 FUNC … END (Program Start/End)

Usage

This indicates a start and end of a function.

Syntax

```
FUNC <Return Type> <Function Name> ([<Parameter>])
    …
    [RET <Variable>]
END
```

Description

In case that the function is used repeatedly or has a large program code, it is possible to create a different function by separating the function.
Multiple functions can be declared anywhere within a program.
It is possible to execute the corresponding function by entering the function name and parameter in the main program and a recursive call is also available.

※ Call Depth cannot exceed 100.

1) FUNC and END must be used together.
2) END indicates the end of block in a function. An execution of function stops when meeting this command during the execution of a function block.
3) All data types (INT, REAL, DEFSTR, POS, XPOS, UPOS) can be used in <Return Pattern> including VOID (without return). However, the global variables (P, GP, I, F) that were declared in advance and the system variables (TMR, etc.) cannot be used.
4) The <Function Name> can be composed of English characters and numbers. It cannot be identical to name of the global variables declared in advance.
5) The <Parameter> can be skipped, however, can use various variables to be used in the internal operation of function if required. All data types (INT, REAL, DEFSTR, POS, XPOS, UPOS) can be used as <Parameter>. However, the global variables (P, GP, I, F) that were declared in advance and the system variables (TMR, etc.) cannot be used.

Arguments

<Parameter>	Data type: <All data types, skipped in case on data>

Examples

Example 1

- **Type without return (VOID) and parameter**

```
FUNC VOID TEST()          // Declare a function without return value and parameter
    SVON                  // Servo on
    JMOV P0               // Move JMOV to P0
    JMOV P1               // Move JMOV to P1
END                       // Function End

MAIN                      // Declare Main Function Start
    TEST()                // Call a function TEST
EOP                       // Main Function End
```

Example 2

- **Type without return (VOID) and with parameter (Joint Coordinate System Position)**

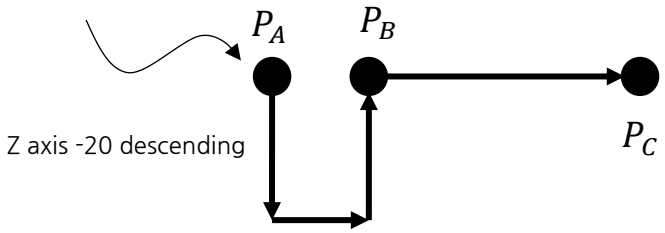


Figure 4-3 Example of Function Motion

```
MAIN                      // Declare Main Function Start
    MOVE(GP[0], GP[1])    // A, B positional variable as function call
    LMOV GP[2]            // LMOV to GP [2] (C position)
EOP                       // Main Function End

FUNC VOID MOVE(POS A, POS B)  // Declare a function without a return value and with position
                              // parameters
    XPOS TEMP             // Declare a position variable of base coordinate system
    SVON                  // Servo on
    JMOV A                // Move JMOV to Position A
    TEMP = JTOB(A)        // Convert the position of A into the position value of base
                          // coordinate system and store
    TEMP.3 = TEMP.3 - 20  // Store the position value where descends 20 to Z axis
                          // direction from the position A
    LMOV TEMP             // Move LMOV to Position TEMP
    TEMP = JTOB(B)        // Convert the Position B into the position value of base
                          // coordinate system and store
    TEMP.3 = TEMP.3 - 20  // Store the position value where descends 20 to Z axis
                          // direction from the position B
    LMOV TEMP             // Move LMOV to Position TEMP
    LMOV B                // Move LMOV to Position B
END                       // Function End
```

Example 3

- **Type with return (Real Number) and without parameter**

```
MAIN                                  // Declare Main Function Start
    REAL VALUE                        // Declare a real number variable
    VALUE = TEST()                    // Call a function TEST & Store return value in variable
    PRINT(0, VALUE)                   // Store in index 0 of monitoring log
EOP                                   // Main Function End

FUNC REAL TEST()                      // Declare a function with a return value(real number
                                      // variable) and without parameter
    REAL RES                          // Declare a real number variable
    RES = F[0]+F[1]                   // Add real number variables
    RET RES                           // Return a real number variable RES
END                                   // Function End
```

Example 4

- **Type with return (String) & with parameter (Integer, String)**

```
MAIN                                  // Declare Main Function Start
    INT R_VAL                         // Declare an integer variable
    DEFSTR STR                        // Declare a string variable
    WHILE 1                           // Iterative statement Start
        STR = TEST("IN(0)", IN(0))    // Call a function TEST & Store return value in variable
        PRINT(0, STR)                 // Store in index 0 of monitoring log
        STROUT(0,STR)                 // Send string to RS-232 port
        DLAY 100                      // 0.1 second Time Delay
    ENDWL                             // Iterative statement End
EOP                                   // Main Function End

FUNC DEFSTR TEST(DEFSTR A, INT B)     // Declare a function with a return value(string) and
                                      // parameters(string and integer)
    DEFSTR DATA, RES                  // Declare a string variable
    DATA = HTOS(B)                    // Convert an integer value to hexadecimal string
    RES = A + ":" + DATA              // Store the value to variable after adding each string
                                      // Return a string
    RET RES                           // Function End
END
```

Example 5 • **Type with return & with parameter (example of position)**

```
MAIN                                  // Declare Main Function Start
    XPOS A_P, B_P, X_T                // Declare a position variable of base coordinate system
    A_P = JTOB(GP[0])                 // Acquire position value of base coordinate system
    B_P = JTOB(GP[1])
    X_T = MD_X(A_P, B_P)              // Call a function MD_X & Store return value in variable
    SVON                              // Servo on
    LMOV X_T                          // Move LMOV to X_T
EOP                                   // Main Function End

FUNC XPOS MD_X(XPOS A, XPOS B)        // Declare a function with parameters & return value of Base
                                      // coordinate system

    XPOS RES                          // Declare a position variable of base coordinate system
    RES = (A+B)/2                     // Store the value that two values are added and divided by 2
    RET RES                           // Return a position variable of base coordinate system
END                                   // End of function
```

Example 6 • **Example of Recursive Function**

```
INT $COUNT                            // Declare a global variable
MAIN                                  // Declare Main Function Start
    $COUNT = 0                        // Initialize the variable
    RECALL()                          // Call a function RECALL
EOP                                   // Main Function End

FUNC VOID RECALL()                    // Declare a function RECALL
    $COUNT = $COUNT+1                 // Add 1 to the global variable $COUNT value
    PRINT(0,"COUNT",$COUNT)           // Leave monitoring log
    IF $COUNT <50 THEN                // If global variable $COUNT value is below 50
        RECALL()                      // Call a recursive function
    ENDIF                             // IF statement End
END                                   // Function End
```

Related terms

MAIN ··· EOP, RET

5.3 RET

Usage

This returns the value after a function execution and goes back to the point where the function was called.

Syntax

RET <Return Value>

Description

This cannot be used in the Main Function.

The <Return Value> is a return value of a function and determined according to the declaration of a function.
In case of VOID in <Return Type> at declaration of a function, RET command cannot be used.
All data types (INT, REAL, DEFSTR, POS, XPOS, UPOS) can be used as <Return Value>.
However, the global variables (P, GP, I, F) that were declared in advance and the system variables (TMR, etc.) cannot be used.

Arguments

Data type: This is determined according to a declaration of a function.
It is skipped in VOID.

Examples

```
FUNC INT TEST(INT A, INT B)     // Declare TEST function with a return value and parameters
                                // Declare an integer variable
    INT VALUE                   // Execute calculation
    VALUE = A*B                 // Return a calculation value
    RET VALUE                   // Function End
END
```

Related terms

FUNC ··· END

5.4 FOR … TO … [BY] … NEXT (Iterative Execution Statement)

Usage

This performs the block repeatedly until the condition becomes a False (0).

Syntax

FOR <Initialization Statement> TO <End Condition> [BY <Increment>]
…
NEXT

Description

This typically creates a loop that must be executed as the specified counts. After setting the initial value, it performs the FOR block repeatedly until the end condition is satisfied.

1) It is required to use FOR … TO … NEXT together.
2) The <Initialization Statement> is used in initializing the loop index and it is executed only once and may include the expressions or variables.
3) The <End Condition> is used in checking the criteria of loop end.
4) It is possible to specify the <Increment>. When skipped, the loop index increases by 1 at a time.

Examples

Example 1

```
INT AA, SUM                 // Declare integer variables
SUM = 0                     // Initialize the variable
SVON                        // Servo on
FOR AA = 1 TO 10 BY 2       // AA is Increased by 2 from 1 to 10
    SUM = SUM+AA            // Store the result value in SUM variable after adding SUM variable
                            // and AA variable
    JMOV P[SUM]             // Move JMOV to the P variable corresponding to SUM value
NEXT                        // FOR statement End
PRINT(0, SUM)               // Store in index 0 of monitoring log
```

Example 2

```
REAL AA, SUM                // Declare real number variables
SUM = 0.0                   // Initialize the variable
FOR AA = 0.0 TO 1 BY 0.1    // AA is Increased by 0.1 from 0 to 1
    SUM = SUM+AA            // Store the result value in SUM variable after adding SUM variable
                            // and AA variable
NEXT                        // FOR statement End
PRINT(0,SUM)                // Store in index 0 of monitoring log
```

Related terms

WHILE… ENDWL

5.5 WHILE ··· ENDWL (Conditional Iterative Execution Statement)

Usage

This performs a block repeatedly while the conditional expression is True (1).

Syntax

WHILE 〈Conditional Expression〉
  ···
ENDWL

Description

The WHILE statement performs the block repeatedly while the condition is satisfied (the result of conditional expression is true, that is, the result value is other than 0). When the result of a conditional expression is always true, it performs the endless loop.

1) Both WHILE and ENDWL must be used together.
2) When an expression is inputted into the 〈Conditional Expression〉, the result must be a logical value (true: 1, false: 0).
3) Be careful on the priority of operators when inputting into 〈Conditional Expression〉.

Examples

Example 1	WHILE 1 ENDWL	// Execute an endless loop for the block of WHILE ~ ENDWL.
Example 2	WHILE IN0 == 1 ENDWL	// Execute an endless loop while 1 is input in IN0.
Example 3	WHILE IN1 == 1 && IN0 == 1 ENDWL	// Execute an endless loop while 1 is input in IN0 and IN1.
Example 4	WHILE MVR〉 10 ENDWL	// Execute an endless loop while MVR value is larger than 10.

Related terms

FOR ··· TO ··· [BY] ··· NEXT

5.6 IF …THEN … ENDIF (Conditional Branch Command)

Usage

This executes a block according to the operation result of conditional expression.

Syntax

```
IF <Conditional Expression> THEN
    …
[ELSE IF <Conditional Expression> THEN]
    …
[ELSE]
    …
ENDIF
```

Description

It is possible to determine the execution of block according to the operation result of conditional expression.

1) The IF, THEN and ENDIF must be used together.
2) ELSEIF and ELSE can be selectively used by a user.
3) If the operation result of conditional expression is true (values other than 0), the statements after THEN are executed, and if it is false (0), the statements after ELSE are executed.
4) It is possible to use an IF statement again within the IF statement.
   In this case, make sure that IF and ENDIF are to be paired logically.

Examples

Example 1

```
SVON                    // Servo on
IF IN0 == 1 THEN        // If IN0 is 1
    JMOV P0             // Move JMOV from current position to P0
ENDIF                   // IF statement End
```

Example 2

```
SVON                    // Servo on
INT BB                  // Declare an integer variable
IF I0 == 1 THEN         // If I0 value is 1
    STOP                // Robot Stop
ELSEIF I0 == 2 THEN     // If I0 value is 2
    EXIT                // Program End
ELSE                    // If no conditional expression is satisfied
    BB=IN0              // IN0 value is stored in an integer variable BB
    IF BB == 1 THEN     // If BB variable is 1
        DLAY 10         // 0.01 second Time Delay
        JMOV P0         // Move JMOV from current position to P0
    ENDIF               // IF statement End
ENDIF                   // IF statement End
```

5.7 LABL ··· GOTO (Branch Command)

Usage

This is a command that specifies a branch position and moves to the corresponding position.

Syntax

```
LABL <Label Name>
    ...
GOTO <Label Name>
```

Description

GOTO command performs to branch off to corresponding position and is only available within the same block.
For example, a branch off cannot be available from MAIN block to FUNC block and vice versa.
When using IF, FOR or WHILE block, it is possible to branch off from inside to outside, but not possible in the opposite case.

1) <Label Name> can be composed of English characters and numbers. It cannot begin with any number in declaration. In addition, it cannot be used together with the global variables predefined and the same label name cannot be declared twice in one file.

Examples

Example 1

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    VEL 500             // Alter velocity
LABL REPEAT             // Declare a branch REPEAT
    JMOV P0             // Move JMOV from current position to P0
    JMOV P1             // Move JMOV from current position to P1
    GOTO REPEAT         // Move to a branch REPEAT
EOP                     // Main Function End
```

Exampl2

```
FUNC VOID TEST()        // Declare a function TEST
    SVON                // Servo on
    OUT0 = 1            // 1 is output at OUT0
LABL REPEAT             // Declare a branch REPEAT
    JMOV P0             // Move JMOV from current position to P0
    JMOV P1             // Move JMOV from current position to P1
    OUT0 = 0            // 0 is output at OUT0
    GOTO REPEAT         // Move to a branch REPEAT
END                     // Function End
```

5.8 CONTINUE

Usage

This is used when performing a branch off to the processing statement after an iterative statement or the conditional expression without executing the next statement within iterative statement loop.

Syntax

```
WHILE <Conditional Expression>
    IF <Conditional Expression> THEN
        CONTINUE
    ENDIF
ENDWL
```

Description

Meeting a CONTINUE statement moves to the very first line of iterative statement.

Examples

```
WHILE 1                     // Endless loop Start
    SVON                    // Servo on
    JMOV P1                 // Move JMOV from current position to P1
    IF IN(1) == 1 THEN      // If IN1 value is 1
        CONTINUE            // Move to the first line (→ WHILE 1)
    ENDIF                   // IF statement End
    JMOV P0                 // Move JMOV from current position to P0
ENDWL                       // Endless Loop End
```

Related terms

FOR … TO … [BY] … NEXT, WHILE … ENDWL, BREAK

5.9 BREAK

Usage

This is used to escape from the iterative statement when special conditions occur within an endless loop.

Syntax

```
WHILE <Conditional Expression>
    IF <Conditional Expression> THEN
        BREAK
    ENDIF
ENDWL
```

Description

This escapes from the iterative statement when it meets BREAK statement.

Examples

```
WHILE 1                          // Endless loop Start
    SVON                         // Servo on
    JMOV P1                      // Move JMOV from current position to P1
    IF IN(1) == 1 THEN           // If IN1 value is 1
        BREAK                    // Escape from endless loop
    ENDIF                        // IF statement End
    JMOV P0                      // Move JMOV from current position to P0
ENDWL                            // Endless Loop End
```

Related terms

FOR … TO … [BY] … NEXT, WHILE … ENDWL, CONTINUE

5.10 JCALL (Program Call)

Usage

This calls a JOB file and immediately executes.

Syntax

JCALL <JOB File Name>

Description

If the called JOB program meets EOP command, the execution line returns to the next to JCALL command of the called program.
Like INCLUDE command, it is possible to use the global variable and function of the called file inside.
A program must be created so that JCALL is not to be a recursive call.

1) The only file name except a file end-mark "." and file extension "JOB" can be entered in <JOB File Name>.

Examples

```
MAIN                    // Declare Main Function Start
    JCALL TEST          // Call a TEST.JOB program & immediately execute
EOP                     // Main Function End
```

Related terms

INCLUDE

5.11 STOP (Robot Stop)

Usage

This stops the motion of robot.

Syntax

STOP

Description

This stops the moving robot and processes the command of next STEP. In other words, It continuously executes next STEP after stopping the motion currently in execution.

Examples

```
MAIN                        // Declare Main Function Start
    SVON                    // Servo on
    WITH                    // Execute WITH statement
        JMOV P0             // Move JMOV from current position to P0
        WHILE MVR <90       // Endless Loop Start while MVR value is below 90
            IF IN0 == 1 THEN    // If IN0 is 1
                GOTO AA         // Move to the branch point AA
            ENDIF               // IF statement End
        ENDWL               // Endless Loop End
LABL AA                     // Declare a branch point AA
        STOP                // Robot Stop
    ENDWT                   // WITH statement End
EOP                         // Main Function End
```

Related terms

EXIT

5.12 EXIT (JOB Program Stop)

Usage

This stops a JOB program execution.

Syntax

EXIT

Description

EXIT command stops a JOB program execution. In this case, the controller occurs "**EXIT Instruction**" alarm.

Examples

```
MAIN                        // Declare Main Function Start
    SVON                    // Servo on
    WITH                    // Execute WITH statement
        JMOV P0             // Move JMOV from current position to P0
        WHILE MVR <90       // Endless Loop Start while MVR value is below 90
            IF IN0 == 1 THEN    // If IN0 is 1
                GOTO AA         // Move to the branch point AA
            ENDIF           // IF statement End
        ENDWL               // Endless Loop End
LABL AA                     // Declare a branch point AA
        EXIT                // Program Stop & EXIT alarm occurs
    ENDWT                   // WITH statement End
EOP                         // Main Function End
```

Related terms

STOP

6. Robot Motion

6.1 SVON (Servo on)

Usage

This turns on the power of servo motor.

Syntax

SVON[(<Robot Number>)]

Description

This turns on the servo motor powers of entire robot axes allocated to the JOB program.
If using the movement commands in servo motor power off, an alarm occurs.

1) If <Robot Number> is inputted and stored to the JOB program, the servo motor power of another robot can be turned on.

Arguments

<Robot Number>	Data type: <Integer Type> <Robot Number> range is 1 ~ 3.

Examples

Example 1

```
MAIN                    // Main Function Start
    SVON                // Servo on
    JMOV P0             // Move JMOV from current position to P0
    SVOF                // Servo off
EOP                     // Main Function End
```

Example 2

```
MAIN                    // Main Function Start
    INT STATE           // Declare an integer variable
    STATE = RSTATE(2,6) // Store of servo state of robot 2
    IF STATE == 0 THEN  // If servo off
        SVON(2)         // Servo on of robot 2
    ENDIF               // IF statement End
EOP                     // Main Function End
```

Related terms

SVOF

6.2 SVOF (Servo off)

Usage

This turns off the power of servo motor.

Syntax

SVOF[(<Robot Number>)]

Description

This turns off the servo motor powers of entire robot axes allocated to the JOB program.

1) If <Robot Number> is inputted and stored to the JOB program, the servo motor power of another robot can be turned off <Robot Number>.

Arguments

<Robot Number>	Data type: <Integer Type> <Robot Number> range is 1 ~ 3.

Examples

Example 1

```
MAIN                    // Main Function Start
    SVON                // Servo on
    JMOV P0             // Move JMOV from current position to P0
    SVOF                // Servo off
EOP                     // Main Function End
```

Example 2

```
MAIN                    // Main Function Start
    INT STATE           // Declare an integer variable
    STATE = RSTATE(2,6) // Store of servo state of robot 2
    IF STATE == 1 THEN  // If servo on
        SVOF(2)         // Servo off of robot 2
    ENDIF               // IF statement End
EOP                     // Main Function End
```

Related terms

SVON

6.3 JMOV (PTP Movement)

Usage

This moves to the target point from current position.

Syntax

JMOV <Target Position> [, FOS = <Value>, VEL = <Value>, ACC = <Value>, DEC = <Value>, T = <Value>, U = <Value>]

Description

This enables PTP (Point to Point) movement that moves from a certain point to another.
Moving path depends on the posture of a robot and is not limited to the linear motion.

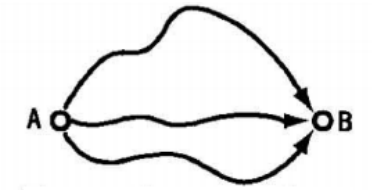


Figure 4-4 PTP Movement Example

In case of using commands, additional items of FOS, VEL, ACC, DEC, T, U can be additionally entered.
If keywords like FOS and VEL are specified and the values are assigned to them, motion command may be altered.
Each keyword can be used regardless of sequence and can be used partially.

1) <Target Position> can be used in all position variables. In case of using JMOV command, it is moved by converting the input position variable to the position variable of joint coordinate system.
2) FOS (Factor of smooth path) implies to move continuously to the next target position without stopping at the waypoint that doesn't need to stop by. The trajectory is altered by referring to distance ratio (%) to the next target point. The distance ratio is a percentage (%) of the entire moving distance.
3) VEL implies a moving velocity of axis and is expressed as permiliage.
   A moving velocity is calculated as the expression shown below.
   Rated velocity and initial velocity are set in MOTION parameters.
   - Velocity = Initial Velocity X Rated Velocity X Permiliage Value X 0.001
4) ACC implies an acceleration time of axis and is expressed as percentage (%). The acceleration time is calculated as the expression shown below. Rated acceleration time is set in MOTION parameter.
   - Acceleration Time = Rated Acceleration Time X Percentage Value X 0.01
5) DEC implies a deceleration time of axis and is expressed as percentage (%). The deceleration time is calculated as the expression shown below.
   Rated deceleration time is set in MOTION parameter.
   - Deceleration Time = Rated Deceleration Time X Percentage Value X 0.01
6) T refers to tool number and U refers to user number of coordinate system. These

are used to select the tools when executing a motion or the coordinate systems.

Arguments

Target Position	Data type: <All Position Types>
FOS	Factor of smooth path Data type: <Integer Type> Range of FOS is 0 ~ 50%.
VEL	Velocity Data type: <Integer Type> Range of VEL is 1 ~ 1,000%.
ACC	Acceleration Data type: <Integer Type> Range of ACC is 1 ~ 400%.
DEC	Deceleration Data type: <Integer Type> Range of DEC is 1 ~ 400%.
T	Tool number Data type: <Integer Type> Range of T is 0 ~ 16.
U	User number Data type: <Integer Type> Range of U is 0 ~ 16.

Examples

Example 1

```
MAIN                                    // Declare Main Function Start
    POS AP                              // Declare a position variable
    AP = <400.0, 50.0, 10.4, 10.0, 0, 0>  // Initialize the variable
    SVON                                // Servo on
    JMOV AP                             // Move JMOV to AP
EOP                                     // Main Function End
```

Example 2

```
MAIN                                    // Declare Main Function Start
    XPOS XP                             // Declare a position variable
    XP = <250,250,350,-30,30,60,ARM = 0,T = 2>  // Initialize the variable
    SVON                                // Servo on
    JMOV XP                             // Move JMOV to XP
EOP                                     // Main Function End
```

Example 3

```
MAIN                                    // Declare Main Function Start
    SVON                                // Servo on
    JMOV P0, FOS = 30, VEL = 100, ACC = 200,  // Move JMOV to P0 and P1 by
    DEC = 100                           // referring to each conditions
    JMOV P1, DEC = 150, VEL = 200, ACC = 150
EOP                                     // Main Function End
```

Related terms

LMOV, FOS, VEL, ACC, DEC, TOOL, USER

6.4 LMOV (Linear Interpolation Movement)

Usage

This performs a linear interpolation movement from current position to the target point.

Syntax

LMOV 〈Target Position〉 [, FOS = 〈Value〉, VEL = 〈Value〉, ACC = 〈Value〉, DEC = 〈Value〉, T = 〈Value〉, U = 〈Value〉]

Description

This interpolates the path from the starting position to the target position to be a linear.



Figure 4-5 Linear Interpolation Movement Example

In case of using commands, additional items of FOS, VEL, ACC, DEC, T, U can be additionally entered.
If keywords like FOS and VEL are specified and the values are assigned to them, motion command may be altered. Each keyword can be used regardless of sequence and can be used partially.

1) 〈Target Position〉 can be used in all position variables. In case of using LMOV command, it is moved by converting the input position variable to the position variable of base coordinate system.
2) FOS (Factor of smooth path) implies to move continuously to the next target position without stopping at the waypoint that doesn't need to stop by. The trajectory is altered by referring to distance ratio (%) to the next target point. The distance ratio is a percentage (%) of the entire moving distance.
3) VEL implies a moving velocity of axis and is expressed as permiliage.
   A moving velocity is calculated as the expression shown below.
   Rated velocity and initial velocity are set in MOTION parameters.
   Velocity = Initial Velocity X Rated Velocity X Permiliage Value X 0.001
4) ACC implies an Acceleration Time of Axis and is expressed as percentage (%).
   The acceleration time is calculated as the expression shown below.
   Rated acceleration time is set in MOTION parameter.
   Acceleration Time = Rated Acceleration Time X Percentage Value X 0.01
5) DEC implies a deceleration time of axis and is expressed as percentage (%).
   The deceleration time is calculated as the expression shown below.
   Rated deceleration time is set in MOTION parameter.
   Deceleration Time = Rated Deceleration Time X Percentage Value X 0.01
6) T refers to tool number and U refers to user coordinate system number. These are used to select the tools when executing a motion or the coordinate systems.

Arguments

〈Target Position〉	Data type: 〈All Position Types〉
FOS	Factor of smooth path Data type: 〈Integer Type〉 Range of FOS is 0 ~ 50%.
VEL	Velocity Data type: 〈Integer Type〉 Range of VEL is 1 ~ 1,000%.
ACC	Acceleration Data type: 〈Integer Type〉 Range of ACC is 1 ~ 400%.
DEC	Deceleration Data type: 〈Integer Type〉 Range of DEC is 1 ~ 400%.
T	Tool number Data type: 〈Integer Type〉 Range of T is 0 ~ 16. This plays a role of TOOL command.
U	User number Data type: 〈Integer Type〉 Range of U is 0 ~ 16.

Examples

Example 1

```
MAIN                                    // Declare Main Function Start
    XPOS XP                             // Declare a position variable
    XP = 〈250,250,350,-30,30,60,ARM = 0,T = 2〉  // Initialize the variable
    SVON                                // Servo on
    LMOV XP                             // Move LMOV to XP
EOP                                     // Main Function End
```

Example 2

```
MAIN                                    // Declare Main Function Start
    POS AP                              // Declare a position variable
    AP = 〈30.0, -60.0, 10.4, 10.0, 0, 0〉  // Initialize the variable
    SVON                                // Servo on
    LMOV AP, T = 1                      // Move LMOV to AP using T1
EOP                                     // Main Function End
```

Example 3

```
MAIN                                    // Declare Main Function Start
    SVON                                // Servo on
    LMOV P0, FOS = 30, VEL = 100, ACC = 200, DEC  // Move JMOV to P0 and P1 by
    = 100, U = 1                        // referring to each conditions
    LMOV P1, DEC = 150, VEL = 200, ACC = 150, T
    = 1
EOP                                     // Main Function End
```

Related terms

JMOV, FOS, VEL, ACC, DEC, TOOL, USER

1) There exists a case that is unable to execute as the posture and position of robot. In this case, the alarms such as “[1219] Range Over error”, or “[1415] ik position error” may occur. Caution is required when using the conditional command of robot motions.
2) Be careful with point teaching.

6.5 CMOV (Circular Interpolation Movement)

Usage

This moves in a circle passing through the waypoint 1 & 2 from current position.

Syntax

CMOV 〈Waypoint 1〉 〈Waypoint 2〉 [, VEL = 〈Value〉, ACC = 〈Value〉, DEC = 〈Value〉, T = 〈Value〉, U = 〈Value〉]

Description

This interpolates the path to get to the target position of a motion by drawing a circle. For circular interpolation movement, total 3 positional coordinates are required (Start, Waypoint 1, and Waypoint 2).

When using CMOV, it is recommended to use CMOV after moved using JMOV as shown in Figure 4-6

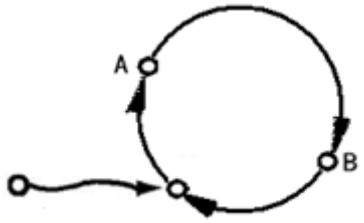


Figure 4-6 Circular Interpolation Example

In case of using commands, additional items of FOS, VEL, ACC, DEC, T, U can be additionally entered.

If a keyword like VEL is specified and its value is assigned, motion commands may be altered. Each keyword can be used regardless of sequence and can be used partially.

1) The 〈Waypoint 1〉 and 〈Waypoint 2〉 can be used in all position variables. In case of using LMOV command, it is moved by converting the input position variable to the position variable of base coordinate system.
2) VEL implies a moving velocity of axis and is expressed as permiliage.
   A moving velocity is calculated as the expression shown below.
   Rated velocity and initial velocity are set in MOTION parameters.
   - Velocity = Initial Velocity X Rated Velocity X Permiliage Value X 0.001
3) ACC implies an acceleration time of axis and is expressed as percentage (%). The acceleration time is calculated as the expression shown below.
   Rated acceleration time is set in MOTION parameter.
   - Acceleration Time = Rated Acceleration Time X Percentage Value X 0.01
4) DEC implies a deceleration time of axis and is expressed as percentage (%). The deceleration time is calculated as the expression shown below.
   Rated deceleration time is set in MOTION parameter.
   - Deceleration Time = Rated Deceleration Time X Percentage Value X 0.01
5) T refers to tool number and U refers to user coordinate system number. These are

used to select the tools when executing a motion or the coordinate systems.

Arguments

〈Waypoint 1〉 〈Waypoint 2〉	Data type: 〈All Position Types〉
VEL	Velocity Data type: 〈Integer Type〉 Range of VEL is 1 ~ 1,000%.
ACC	Acceleration Data type: 〈Integer Type〉 Range of ACC is 1 ~ 400%.
DEC	Deceleration Data type: 〈Integer Type〉 Range of DEC is 1 ~ 400%.
T	Tool number Data type: 〈Integer Type〉 Range of T is 0 ~ 16. This plays a role of TOOL command.
U	User number Data type: 〈Integer Type〉 Range of U is 0 ~ 16.

Examples

Example 1

```
MAIN                        // Declare Main Function Start
    SVON                    // Servo on
    JMOV P1                 // Move JMOV to P1
    CMOV P2 P3              // Circular interpolation passing through P2, P3
EOP                         // Main Function End
```

Example 2

```
MAIN                        // Declare Main Function Start
    SVON                    // Servo on
    JMOV P0                 // Move JMOV to P0
    CMOV P1 P2, DEC = 50, VEL = 100,   // Circular Interpolation passing through P1, P2
    ACC = 50, T = 1
EOP                         // Main Function End
```

Related terms

AMOV, VEL, ACC, DEC, TOOL, USER



1) There exists a case that is unable to execute as the posture and position of robot. In this case, the alarms such as “[1219] Range Over error”, or “[1415] ik position error” may occur. Caution is required when using the conditional command of robot motions.
2) Be careful with task point teaching.

6.6 AMOV (Arc Interpolation Movement)

Usage

This moves in an arc passing through waypoint 1 & 2 from current position.

Syntax

AMOV 〈Waypoint 1〉 〈Waypoint 2〉 [, VEL = 〈Value〉, ACC = 〈Value〉, DEC = 〈Value〉, T = 〈Value〉, U = 〈Value〉]

Description

This interpolates the path to get to the target position of a motion by drawing an arc. For arc interpolation movement, total 3 positional coordinates are required (Start, Waypoint 1, and Waypoint 2).
When using AMOV, it is recommended to use AMOV after moved using JMOV as shown in Figure 4-7



Figure 4-7 Arc Interpolation Example

In case of using commands, additional items of FOS, VEL, ACC, DEC, T, U can be additionally entered.
If a keyword like VEL is specified and its value is assigned, motion commands may be altered. Each keyword can be used regardless of sequence and can be used partially.

1) The 〈Waypoint 1〉 and 〈Waypoint 2〉 can be used in all position variables. In case of using CMOV command, it is moved by converting the input position variable to the position variable of base coordinate system.
2) VEL implies a moving velocity of axis and is expressed as permiliage.
   A moving velocity is calculated as the expression shown below.
   Rated velocity and initial velocity are set in MOTION parameters.
   - Velocity = Initial Velocity X Rated Velocity X Permiliage Value X 0.001
3) ACC implies an acceleration time of axis and is expressed as percentage (%). The acceleration time is calculated as the expression shown below.
4) Rated acceleration time is set in MOTION parameter.
   - Acceleration Time = Rated Acceleration Time X Percentage Value X 0.01
5) DEC implies a deceleration time of axis and is expressed as percentage (%). The deceleration time is calculated as the expression shown below.
   Rated deceleration time is set in MOTION parameter.
   - Deceleration Time = Rated Deceleration Time X Percentage Value X 0.01

6) T refers to tool number and U refers to user coordinate system number. These are used to select the tools when executing a motion or the coordinate systems.

Arguments

〈Waypoint 1〉 〈Waypoint 2〉	Data type: 〈All Position Types〉
VEL	Velocity Data type: 〈Integer Type〉 Range of VEL is 1 ~ 1,000%.
ACC	Acceleration Data type: 〈Integer Type〉 Range of ACC is 1 ~ 400%.
DEC	Deceleration Data type: 〈Integer Type〉 Range of DEC is 1 ~ 400%.
T	Tool number Data type: 〈Integer Type〉 Range of T is 0 ~ 16. This plays a role of TOOL command.
U	User number Data type: 〈Integer Type〉 Range of U is 0 ~ 16.

Examples

Example 1

```
MAIN                                  // Declare Main Function Start
    SVON                              // Servo on
    JMOV P1                           // Move JMOV to P1
    AMOV P2 P3                        // Arc interpolation to P3 passing through P2
EOP                                   // Main Function End
```

Example 2

```
MAIN                                  // Declare Main Function Start
    SVON                              // Servo on
    JMOV P0                           // Move JMOV to P1
    AMOV P1 P2, DEC = 50, VEL = 150,  // Arc interpolation to P2 passing through P1
    ACC = 50, T = 1
EOP                                   // Main Function End
```

Related terms

CMOV, VEL, ACC, DEC, TOOL, USER

1) There exists a case that is unable to execute as the posture and position of robot. In this case, the alarms such as “[1219] Range Over error”, or “[1415] ik position error” may occur. Caution is required when using the conditional command of robot motions.
2) Be careful with task point teaching.

7. Robot Motion Condition

7.1 WITH … ENDWT (Simultaneous Processing Commands)

Usage

This processes the robot motions and execution statement within a block simultaneously.

Syntax

```
WITH
    …
ENDWT
```

Description

This processes the first movement command within WITH statement and the next command in parallel.
By using the iterative statement and conditional expression within WITH statement block, the required works can be proceeded in advance before completing the movement.

Examples

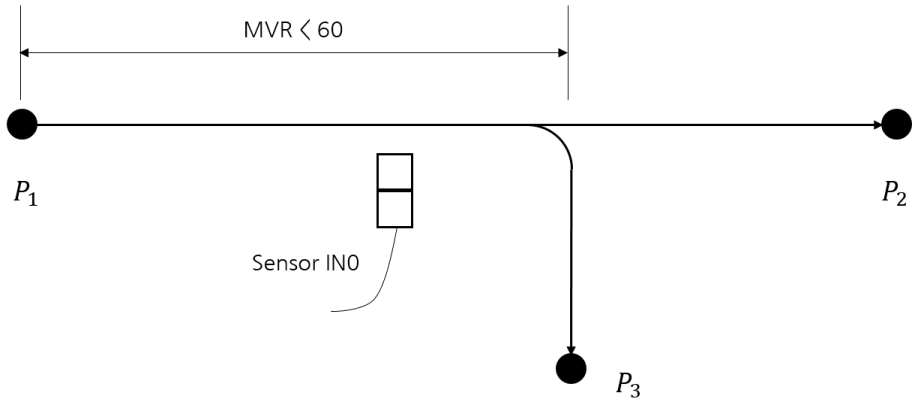


Figure 4-8 WITH Statement Use Example

```
MAIN                        // Declare Main Function Start
    SVON                    // Servo on
    JMOV P1                 // Move JMOV to P1
    WITH                    // Use of WITH statement, simultaneous processing
        JMOV P2             // Move JMOV to P2
        WHILE MVR <60       // If MVR is below 60, endless loop
            IF IN0 == 1 THEN    // If IN0 is 1
                STOP            // Robot Stop
                GOTO AA         // Move to the branch point AA
            ENDIF               // IF statement End
        ENDWL               // Endless Loop End
    ENDWT                   // WITH statement End
```

```
LABL AA                    // Declare a branch point AA
    JMOV P3                // Move JMOV to P3
EOP                        // Main Function End
```

Related terms

MVR



1) When using a branch command to escape from WITH ··· ENDWT statement while executing WITH statement, a robot must be stopped before the branch command.

7.2 VEL (Axis Moving Velocity Setting)

Usage

This sets a permiliage (%) of Axis Moving Velocity.

Syntax

VEL 〈Velocity Ratio〉

Description

This is a command on moving velocity of robot and alters the velocity of movement commands in entire program.
Rated velocity and initial velocity are set in MOTION parameter.

- Axis Velocity = Initial Velocity X Rated Velocity X Permiliage Value X 0.001

When altering the velocity in a movement command, this has precedence over the velocity value that was inserted in the command.

1) The 〈Velocity Ratio〉 implies the ratio of Axis Moving Velocity and can be set by permiliage (%).

Arguments

〈Velocity Ratio〉	Data type: 〈Integer Type〉 Range of 〈Velocity Value〉 is 1 ~ 1,000%.

Examples

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    VEL 1000            // Velocity: Initial Velocity X Rated Velocity X 0.001 X 1,000
    JMOV P0             // Move JMOV to P0
    VEL 200             // Velocity: Initial Velocity X Rated Velocity X 0.001 X 200
    JMOV P1             // Move JMOV to P1
EOP                     // Main Function End
```

Related terms

JMOV, LMOV, CMOV, AMOV, FOS, ACC, DEC



1) In case of using the machine in over max allowed RPM, there are risks of noises and damages. The machine must be used after checking the label attached.
2) This is applied in only JOB program inside and no parameter value is altered.

7.3 ACC (Axis Acceleration Time Setting)

Usage

This sets a percentage (%) of the acceleration time.

Syntax

ACC 〈Acceleration Time Ratio〉

Description

This is a command on moving velocity of robot and alters the acceleration time of movement commands in entire program.
Rated acceleration time is set in MOTION parameter and can be altered by adjusting the percentage value.

- Acceleration Time = Rated Acceleration Time X Percentage Value X 0.01

When altering the acceleration time in a movement command, this has precedence over the acceleration time that was inserted in the command.

1) The 〈Acceleration Time Ratio〉 implies the ratio of Axis Acceleration Time and can be set by percentage (%).

Arguments

〈Acceleration Time Ratio〉	Data type: 〈Integer Type〉 Range of 〈Acceleration Time Ratio〉 is 1 ~ 400%.

Examples

- An example of the case that rated acceleration time was set to 0.4 sec in the parameter.

```
MAIN          // Declare Main Function Start
    SVON      // Servo on
    ACC 200   // Acceleration Time: Rated Acceleration Time (0.4s) X 0.01 X 200 = 0.8s
    JMOV P0   // Move JMOV to P0
    ACC 50    // Acceleration Time: Rated Acceleration Time (0.4s) X 0.01 X 50 = 0.2s
    JMOV P1   // Move JMOV to P1
EOP           // Main Function End
```

Related terms

JMOV, LMOV, CMOV, AMOV, FOS, VEL, DEC



1) In case of setting the acceleration time ratio to be less than 100%, the noises and vibrations can occur in mechanical part. Thus, a caution is required in use.
2) This is applied in only JOB program inside and no parameter value is altered.

7.4 DEC (Axis Deceleration Time Setting)

Usage

This sets a percentage (%) of the deceleration time.

Syntax

DEC <Deceleration Time Ratio>

Description

This is a command on moving velocity of robot and alters the deceleration time of movement commands in entire program.
Rated deceleration time is set in MOTION parameter and can be altered by adjusting the percentage value.

Deceleration Time = Rated Deceleration Time X Percentage Value X 0.01

When altering the deceleration time in a movement command, this has precedence over the deceleration time that was inserted in the command.

1) The <Deceleration Time Ratio> implies the ratio of Axis Deceleration Time and can be set by percentage (%).

Arguments

<Deceleration Time Ratio>	Data type: <Integer Type> Range of <Deceleration Time Ratio> is 1 ~ 400%.

Examples

- An example of the case that rated deceleration time was set to 0.6 sec in the parameter

```
MAIN          // Declare Main Function Start
    SVON      // Servo on
    DEC 200   // Deceleration Time: Rated Deceleration Time (0.6s) X 0.01 X 200 = 1.2s
    JMOV P0   // Move JMOV to P0
    DEC 50    // Deceleration Time: Rated Deceleration Time (0.6s) X 0.01 X 50 = 0.3s
    JMOV P1   // Move JMOV to P1
EOP           // Main Function End
```

Related terms

JMOV, LMOV, CMOV, AMOV, FOS, VEL, ACC

1) In case of setting the deceleration time ratio to be less than 100%, the noises and vibrations can occur in mechanical part. Thus, a caution is required in use.
2) This is applied in only JOB program inside and no parameter value is altered.

7.5 FOS (Setting Factor of smooth path)

Usage

Changes trajectory to the next target point before arriving of tip of axis at a target point.

Syntax

FOS <Distance Ratio>: if single Command is used
LMOV P0, FOS = <Distance Ratio>: if used with Move Command
JMOV P1, FOS = <Distance Ratio>: if used with Move Command

Description

Continuously moves the Robot without stopping by starting Move Command to the next position before arriving at designated position when FOS Command is executed. Sets FOS Command and sets starting point for the next Move Command within Decel time of Move Motion currently in progress.
In case of FOS Command, JMOV and JMOV Move Command can continuously move and,
In case of Move Command among LMOV, AMOV and CMOV, continuously move is possible.
However, in case of JMOV and LMOV(AMOV, CMOV), since Moving Coordinate System is different, execute the next Move Command after arriving current Move Command without continuously executing Move Command.

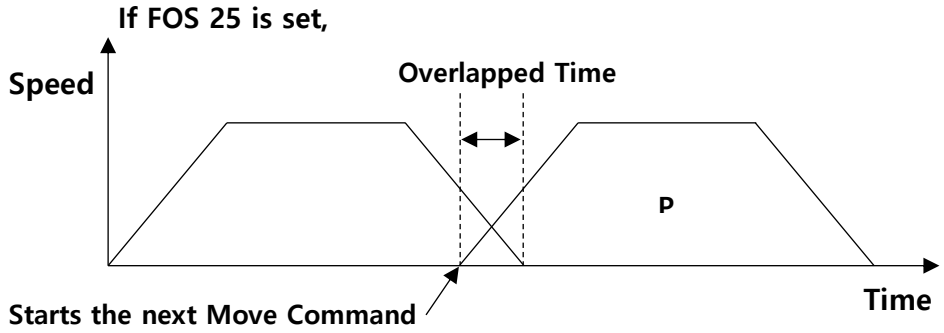
In case of using FOS Command, if moving velocity changes even for same moving distance, trajectory in connecting section changes. If velocity rises, connecting section becomes longer and movement trajectory according to velocity is needed to be checked.

Current \ Next	JMOV	LMOV	AMOV	CMOV
JMOV	Continuously moveable	Continuously immovable	Continuously immovable	Continuously immovable
LMOV	Continuously immovable	Continuously moveable	Continuously moveable	Continuously moveable
AMOV	Continuously immovable	Continuously moveable	Continuously moveable	Continuously moveable
CMOV	Continuously immovable	Continuously moveable	Continuously moveable	Continuously moveable

If FOS Command is used, no INPOS Signal outputs.

1) 〈Accel/decel time rate〉 sets Decel time rate (%) of the current Move Command.

Ex) If FOS 25 is set, the next Move Command starts at 50% point of Decel time.



Ex) If FOS 50 is set, starts the next Move Command at 0% point of Decel time.

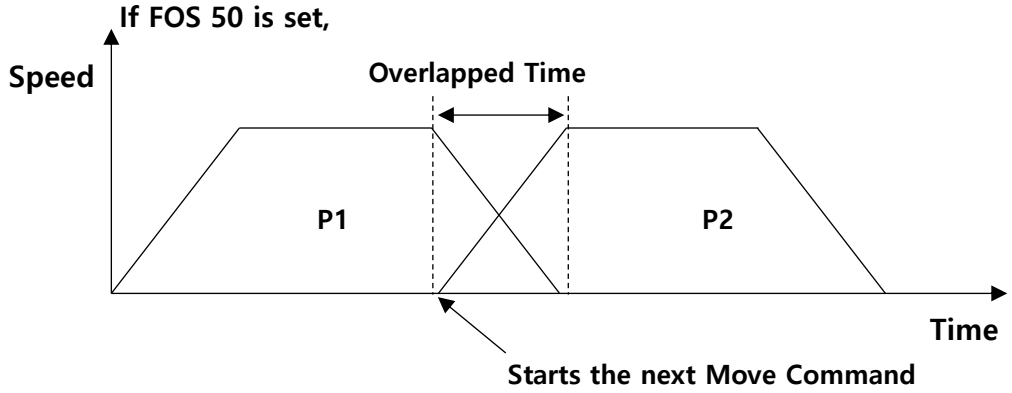


Figure 4-9 FOS Rate

Arguments

〈Distance Ratio〉	Data type: 〈Integer〉 〈Decel time rate〉 range is 0 ~ 50%. If 1 ~ 50 is set, FOS Command function is activated. If 0 is set, it is deactivated.

Examples

Example 1

- FOS 20% before arriving at P1

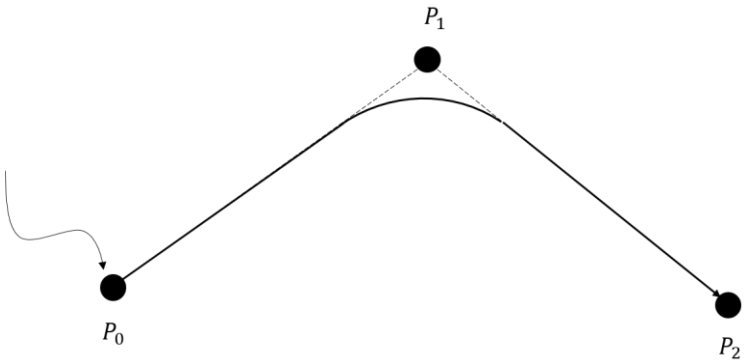


Figure 4-10 Basic Examples of using FOS

```
MAIN                    // Starts Main Function
    SVON                // Servo ON
    JMOV P0             // JMOV TO P0
    FOS 20              // Sets FOS 20%
    JMOV P1             // JMOV TO P1
    FOS 0               // Sets FOS 0%
    JMOV P2             // JMOV TO P2
EOP                     // Ends Main Function
```

Example 2

- Output of OUT0 Signal while moving from P1 to P2

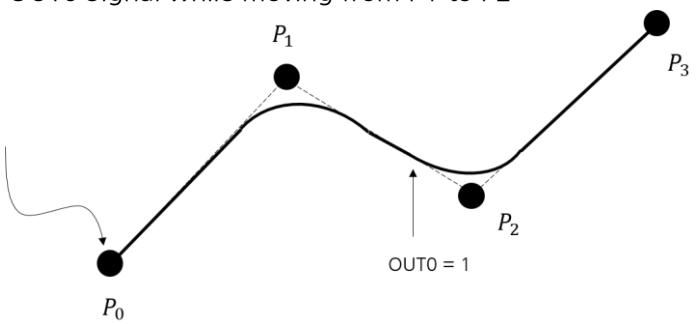


Figure 4-11 Example of Signal Output while Moving

```
MAIN                    // Starts Main Function
    SVON                // Servo ON
    JMOV P0             // JMOV TO P0
    FOS 10              // Sets FOS 10%
    LMOV P1             // LMOV to P1
    LMOV P2             // LMOV to P2
    OUT0 = 1            // Outputs 1Value at OUT0
    FOS 0               // Sets FOS 0%
    JMOV P3             // JMOV to P3
EOP                     // Ends Main Function
```

Example 3 • Selection of Path after checking IN0 Signal while moving from P0 to P1

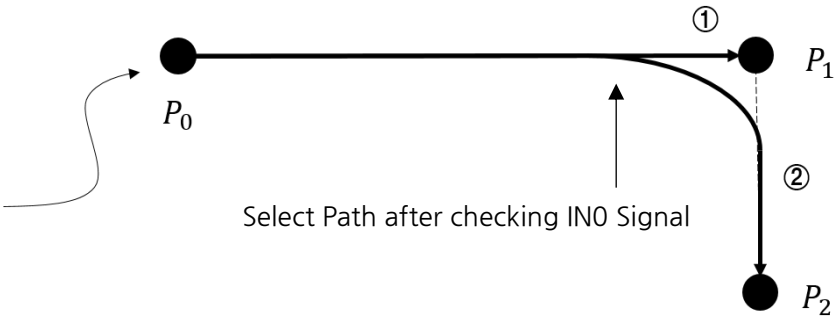


Figure 4-12 Example of checking Signal Input while moving

```
MAIN                        // Starts Main Function
    SVON                    // Servo ON
    JMOV P0                 // JMOV TO P0
    FOS 10                  // Sets FOS 10%
    JMOV P1                 // JMOV TO P1
    IF IN0 == 1 THEN        // If IN0 is 1,
        JMOV P2             // JMOV TO P2
    ENDIF                   // Ends IF STATEMENT
EOP                         // Ends Main Function
```

Related terms

JMOV, LMOV, CMOV, AMOV, VEL, ACC, DEC

1) If there are many Delay Commands (DLAY), calculation formulas and sequence operations besides Move Command between Move Commands, velocity can drop in connecting section of Move Command.
2) Accel/decel time of the current Move Command differs from it of the next Move Command, velocity can drop in connecting section.
3) If moving velocity won't change in same moving distance when FOS Command is used, movement trajectory in connecting section will change.

7.6 DLAY (Delay Time Setting)

Usage

This pauses the program processing for specified duration.

Syntax

DLAY <Delay Time>

Description

Program processing is stopped until the input <Delay Time> is passed.

1) The unit of <Delay Time> is 1ms.

Arguments

<Delay Time>	Data type: <Integer Type>

Examples

```
MAIN                    // Main Function Start
    SVON                // Servo on
    JMOV P0             // Move JMOV to P0
    DLAY 1000           // 1 second Time Delay
    JMOV P1             // Move JMOV to P1
    DLAY 2000           // 2 second Time Delay
EOP                     // Main Function End
```

7.7 OFFS (Offset Designation)

Usage

Move to the position that is offset by the specified value from the target point.

Syntax

OFFS 〈Deviation Value〉

Description

If a movement command is used after execution of OFFS command, all move to the position by considering the offset value.
As shown in Figure 4-13, in case that the original path is the dotted line from A to B, the OFFS command can alter the path to the solid line from A to B.

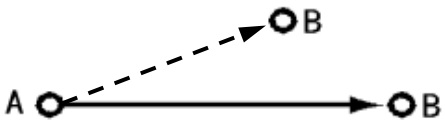


Figure 4-13 OFFS Command Example

OFFS command is only effective in case of executing the JOB program and is applied in the entire programs. Thus, it can be used in the program with JCALL and INCLUDE commands.

1) The 〈Deviation Value〉 can be used in all position variables except the position constants.
2) Final target position of the movement command is updated by referring to the coordinate system of position variables inputted as 〈Deviation Value〉.

Arguments

〈Deviation Value〉	Data type: 〈All Position Types〉

Examples

Example 1

```
POS PDATA                    // Declare a position variable
PDATA = 〈0,0,0,0,30,30〉      // Initialize the position variable
SVON                         // Servo on
OFFS PDATA                   // OFFSET setting with PDATA
JMOV P0                      // Move JMOV to P
```

Example 2

```
OFFS P0                      // OFFSET setting with P0
SVON                         // Servo on
JMOV GP0                     // Move JMOV to GP
OFFS GP0                     // OFFSET Re-Setting with GP0
JMOV P0                      // Move JMOV to P0
```

Example 3

```
XPOS AP                          // Declare a position variable of base coordinate system
UPOS BP                          // Declare a position variable of user coordinate system
AP = <10,10,0,10,10,10,ARM =     // Initialize the position variable of base coordinate system
0,T = 1>
BP = <5,10,5,0,0,0,ARM = 0,T =   // Initialize the position variable of user coordinate system
1,U = 1>
SVON                             // Servo on
OFFS AP                          // OFFSET setting with position variable
LMOV GP0                         // Move LMOV to GP0
OFFS BP                          // OFFSET setting with position variable
JMOV GP1                         // Move JMOV to GP1
```

Related terms

OFFSCLR

1) The “[1219] Range Over error” may occur in case of using OFFS command when executing a JOB program even if the teaching value is not off the S/W movement limitation range.
2) To clear OFFS, use OFFSCLR command.

7.8 OFFSCLR (Offset Clear)

Usage

This clears the offset value that was set.

Syntax

OFFSCLR

Description

User can clear the configured offset value using OFFS command and alter to referring to the original position value.
This applies to entire programs same as OFFS command.

Examples

```
OFFS P0                 // OFFSET setting with P0
SVON                    // Servo on
JMOV GP0                // Move JMOV to GP0
OFFSCLR                 // OFFSET Clear
JMOV P0                 // Move JMOV to P0
```

Related terms

OFFS

7.9 LIMT (Axis Moving Range Limitation)

Usage

This limits the moving range of each axis.

Syntax

LIMT 〈(-) Moving Range〉 〈(+) Moving Range〉

Description

This sets the moving limitation range of S/W within JOB programs.
LIMT command is only effective in executing the JOB programs and commonly applied in entire programs. Thus, it can be used the programs with JCALL and INCLUDE commands.

1) The (-) moving range indicates the min position that can move toward (-) direction.
2) The (+) moving range indicates the max position that can move toward (+) direction.
3) When inputting the (-) and (+) moving range, position variables of joint coordinate system must be used and position constants cannot be used.

Arguments

〈(-) Moving Range〉	Data type: 〈Position Type of Joint Coordinate System〉
〈(+) Moving Range〉	Data type: 〈Position Type of Joint Coordinate System〉

Examples

Example 1	LIMT P0 GP1	// Alteration of S/W moving limitation range
	SVON	// Servo on
	JMOV P1	// Move JMOV to P1
Example 2	POS AP, BP	// Declare position variables
	AP = 〈-100,-100,-100,-100,-100,-100〉	// Initialize the position variable
	BP = 〈100,100,100,100,100,100〉	
	LIMT AP BP	// Alteration of S/W moving limitation range
	SVON	// Servo on
	JMOV P0	// Move JMOV to P0



1) The “[1219] Range Over error” may occur in case of using LIMT command when executing a JOB program even if the teaching value is not off the S/W movement limitation range.
2) Be careful so that the (+) moving range is not smaller than the (-) moving range.

7.10 TRQ (Collision Detection Command)

Usage

This is a command that creates alarms in collision by setting a torque limit.

Syntax

TRQ <Robot Number> <Axis Number> <Torque Limit Value>

Description

This creates the "[1434] Over Trq error" alarm if the torque exceeding the limit configured appears longer than the parameter TOL time while a robot operates after setting the torque limit of each axis.

1) Robot is specified by entering <Robot Number>.
2) Axis is specified by entering <Axis Number> to monitor the torque.
3) Limit value is set by entering <Torque Limit Value> to create alarms.

Arguments

<Robot Number>	Data type: <Integer Type> Range of <Robot Number> is 1 ~ 3.
<Axis Number>	Data type: <Integer Type> Range of <Axis Number> is 1 ~ 8.
<Torque Limit Value>	Data type: <Integer Type> Range of <Torque Limit Value> is 1 ~ 300.

Examples

Example 1

```
SVON                // Servo on
TRQ 1 1 50          // Set Torque Limit of J1 Axis to be 50%
TRQ 1 2 100         // Set Torque Limit of J2 Axis to be 100%
LMOV P0             // Move LMOV to P0 from current position
LMOV P1             // Move LMOV to P1 from current position
TRQ 1 1 300         // Set Torque Limit of J1 Axis to be 300%
TRQ 1 2 300         // Set Torque Limit of J2 Axis to be 300%
```



1) In case of JOB START, TRQ value is initialized to 300.
2) Torque value is not monitored in servo off state.

7.11 GAIN (Converting Gain Group)

Usage

Converts Gain Group.

Syntax

GAIN <Group Number>

Description

Selects Gain Group to be used to execute Motion.
Gain Group includes Position Loop Gain, speed Loop Gain, integer for speed Loop, and integer for torque Command Filter.

1) Input <Group Number> to select Gain Group.

Arguments

<Group Number>	Data type: <Integer> <Group Number> range is 1 ~ 2.

Examples

Example 1

```
MAIN                    // Declares Main Function
    SVON                // Servo ON
    GAIN 1              // Applies Gain Group1
    JMOV P0             // JMOV to P0 when Gain Group1 is applied
    GAIN2               // Applies Gain Group2
    JMOV P1             // JMOV to P1 when Gain Group2 is applied
EOP                     // Ends Main Function
```



1) Wait Time and Conversion Time exist to convert Gain. Hence, be pay attention to it.

7.12 IPE (Setting In-position Allowable Time)

Usage

Sets Max Allowable Time value until reaches at target position (In-Position).

Syntax

IPE 〈Allowable Time〉

Description

Sets Max allowable time value from it approaches near target position value to arrive at target position.
It must arrive within target position error allowable range (IPA) within Allowable Time value.

1) 〈Allowable Time〉 unit is 1ms.

Arguments

〈Allowable Time〉	Data type: 〈Integer〉 〈Allowable Time〉 range is 0 ~ 5,000(ms).

Examples

Example 1

```
MAIN                    // Declares Main Function
    SVON                // Servo ON
    IPE 100             // Sets In-Position Max Allowable Time value
    JMOV P0             // JMOV to P0
    JMOV P1             // JMOV to P1
EOP                     // Ends Main Function
```

1) In case of not reaching to target position within Allowable Time, [1426] Inposition error alarm occurs. Hence, be cautious of setting values.
2) If IPE is set to “0”, In-Position is not checked during JOB Run.

8. Input/Output Control

8.1 IN (Digital Bit Input)

Usage

This reads the bit number value specified by unit of 1 bit.

Syntax

IN〈Bit Number〉 = 〈0/1〉 or IN(〈Bit Number〉 [, 〈Waiting Time〉]) = 〈0/1〉
〈Variable〉 = IN〈Bit Number〉 or 〈Variable〉 = IN(〈Bit Number〉)

Description

IN command presents two methods for use.
First method is to assign 0 or 1 to the specified bit. This waits until the input signal of corresponding bit becomes the same as the assigned value.
Second methods is to read the signal value of specified bit number and stores it to a variable.

1) The 〈Bit Number〉 can be used by either using the parenthesis or directly inputting the number. Any blank is not allowed between the character IN and 〈Number〉.
2) The range of 〈Bit Number〉 is 0 ~ 19 and it can be extended to max 83 when using the expansion board.
3) 〈Waiting Time〉 can be set only when parentheses are used, and the setting range is from 100 to 99,999 msec.

Return value

〈Variable〉	Data type: 〈Integer Type〉 The 0(off) or 1(on) value is returned as a result value.

Arguments

〈0/1〉	Data type: 〈Integer Type〉 Only 0(off) or 1(on) can be inputted.
〈Bit Number〉	Data type: 〈Integer〉 • General Input: IN0 ~ IN19 • Expansion 1 Input: IN20 ~ IN51 • Expansion 2 Input: IN52 ~ IN83
〈Waiting Time〉	Data type: 〈Integer〉 Range of 〈Wating Time〉 is 100 to 99,999 msec.

Examples

Example 1

```
SVON                    // Servo on
JMOV P0                 // Move JMOV to P0
IN0 = 1                 // Wait until Input signal of 0 becomes 1
JMOV P1                 // Move JMOV to P1
```

Example 2

```
SVON                          // Servo on
WHILE 1                       // Endless loop Start
    IN(0,1000) = 1            // Wait 1 second until IN0 input signal is 1
    IF IN0 == 0 THEN          // If IN0 signal value is 0
        JMOV GP0              // Move JMOV from current position to GP0
    ELSE                      // If no conditional expression is satisfied
        JMOV GP1              // Move JMOV from current position to GP1
    ENDIF                     // IF statement End
ENDWL                         // Endless Loop End
```

Example 3

```
INT STATE, NUM                // Declare integer variables
NUM = 0                       // Initialize the integer variable
STATE = IN0                   // IN0 signal value store
PRINT(0,STATE)                // Store in monitoring log 0 index
STATE = IN(0)                 // IN0 signal value store
PRINT(1,STATE)                // Store in index 1 of monitoring log
STATE = IN(NUM)               // IN signal store by referring to NUM variable value
PRINT(2,STATE)                // Store in index 2 of monitoring log
```

Related terms

BIN, PIN

1) Be careful in use when waiting for signal value.

8.2 OUT (Digital Bit Output)

Usage

This outputs the specified value to specified bit number.

Syntax

OUT〈Bit Number〉 = 〈0/1〉 or OUT(〈Bit Number〉) = 〈0/1〉
〈Variable〉 = OUT〈Bit Number〉 or 〈Variable〉 = OUT(〈Bit Number〉)
OUT〈Bit Number〉 = 〈0/1〉[〈Pulse Width〉]
OUT〈Bit Number〉 = 〈0/1〉[〈Pulse Width〉 〈Pulse Cycle〉]

Description

OUT command presents 4 methods of use.
First method is to output 0 or 1 into the specified bit number.
Second method is to read the value that was output into the specified bit number and store it to a variable.
Third method is to send an impulse signal to the specified bit number.
Last method is to send the pulse cycle wave to the specified bit number.

1) The 〈Bit Number〉 can be used by either using the parenthesis or directly inputting the number. Any blank is not allowed between the character OUT and 〈Number〉.
2) The range of 〈Bit Number〉 is 0 ~ 19 and it can be extended to max 83 when using the expansion board.
3) In case of additionally input 〈Pulse Width〉, it sends out the impulse signal for the same duration as input time and returns to the former value.
4) Once the 〈Pulse Cycle〉 is input, the pulse cycle waves, which have a cycle of the input time, are sent. The 〈Pulse Cycle〉 always must be inputted together with the 〈Pulse Width〉 and must be larger than the value of 〈Pulse Width〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉 The 0(off) or 1(on) is returned as a result value.

Arguments

〈0/1〉	Data type: 〈Integer Type〉 Only 0(off) or 1(on) can be input.
〈Bit Number〉	Data type: 〈Integer Type〉 • General Input: IN0 ~ IN19 • Expansion 1 Input: IN20 ~ IN51 • Expansion 2 Input: IN52 ~ IN83
〈Pulse Width〉	Data type: 〈Integer Type〉 The input unit is 1ms. However, values below 2ms are ignored, so please enter in multiples of 2.
〈Pulse Cycle〉	Data type: 〈Integer Type〉 The input unit is 1ms. However, values below 2ms are ignored, so please enter in multiples of 2. Pulse Cycle must be larger than Pulse Width.

Examples

Example 1

```
OUT(0) = 1          // 1 is output to the output signal 0
DLAY 1000           // 1 second Time Delay
OUT0 = 0            // 0 is output to the output signal 0
```

Example 2

```
INT STATE           // Declare a integer variable
STATE = OUT1        // OUT1 signal value store
PRINT(0,STATE)      // Store in index 0 of monitoring log
STATE = OUT(1)      // OUT1 signal value store
PRINT(1,STATE)      // Store in index 1 of monitoring log
STATE = OUT(I0)     // OUT signal store by referring to global variable I0 value
PRINT(2,STATE)      // Store in index 2 of monitoring log
```

Example 3

```
OUT0 = 0            // 0 is output to the output signal 0
OUT(0) = 1 100      // Impulse with a length of 0.1 sec is output to output signal 0
```

Example 4

```
OUT1 = 0            // 0 is output to the output signal 1
OUT1 = 1 1000 5000  // Waves with a pulse width of 1 sec and 5 sec cycle are output to
                    // output signal 1
```

Related terms

POUT

1) The input unit of 〈Pulse Width〉 and 〈Pulse Cycle〉 is 1ms when creating impulse signals or pulse cycle waves using OUT command. But, the input value must be a multiple of 2 since the values below 2ms are ignored.
   - OUT0 = 1 1 → No pulse output, Stays with the output value
   - OUT0 = 1 15 → Output of impulse with 14ms length
   - OUT0 = 1 10 15 → Output of waves with 14ms cycle and 10ms pulse width
   - OUT0 = 1 15 29 → Output of waves with 28ms cycle and 14ms pulse width

8.3 PIN (Digital Port Input)

Usage

This reads the value of port number that was specified by unit of a port.

Syntax

〈Variable〉 = PIN〈Port Number〉 or 〈Variable〉 = PIN(〈Port Number〉)

Description

This reads 16 bits data value of the port number at one time.

1) The 〈Port Number〉 can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character PIN and 〈Number〉.
2) The range of 〈Port Number〉 is 0 ~ 1 and it can be extended to max 5 when using the expansion board.

Return value

〈Variable〉 Data type: 〈Integer Type〉
The integer in 16 bits is returned as a result value.
Data values increase as moving from the right to the left.

Bit / Value	F E D C	B A 9 8	7 6 5 4	3 2 1 0
31242	0 1 1 1	1 0 1 0	0 0 0 0	1 0 1 0
-63	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 1
0HF419	1 1 1 1	0 1 0 0	0 0 0 1	1 0 0 1
0B11110011	0 0 0 0	0 0 0 0	1 1 1 1	0 0 1 1

Arguments

〈Port Number〉 Data type: 〈Integer Type〉

- General Input: PIN0 ~ 1
  (PIN0: IN0 ~ IN15, PIN1: IN16 ~ 19)
- Expansion 1 Input: PIN0 ~ 3
  (PIN0: IN0 ~ IN15, PIN1: IN16 ~ 31, PIN2: IN32 ~ 47, PIN3: IN48 ~ 51)
- Expansion 2 Input: PIN0 ~ 5
  (PIN0: IN0 ~ IN15, PIN1: IN16 ~ 31, PIN2: IN32 ~ 47, PIN3: IN48 ~ 63, PIN4: 64 ~ 79, PIN5: 80 ~ 83)

Examples

Example 1

```
INT STATE                   // Declare an integer variable
STATE = PIN0                // PIN0 signal value store
PRINT(0,STATE)              // Store in index 0 of monitoring log
STATE = PIN(0)              // PIN0 signal value store
PRINT(1,STATE)              // Store in index 1 of monitoring log
```

Example 2

- Bit Operator Application Example

```
INT AA, RES                 // Declare integer variables
DEFSTR STR
AA = PIN0                   // PIN0 signal value store
RES = AA & 0H00F0           // AND operation of 4 ~ 7th bit values
STR = BINS(RES)             // Converts into result data binary Character String
PRINT(0, STR)               // Store in index 0 of monitoring log as binary string
RES = PIN0 | 0H000F         // OR operation of 0 ~ 3th bit values
STR = BINS(RES)             // Converts into result data binary Character String
PRINT(1, STR)               // Store in index 1 of monitoring log as binary string
RES = 0B11110000 ^ PIN0     // XOR operation of 4 ~ 7th bit values
STR = BINS(RES)             // Converts into result data binary Character String
PRINT(2, STR)               // Store in index 2 of monitoring log as binary string
```

Related terms

IN

8.4 POUT (Digital Port Output)

Usage

This outputs the specified value to the specified port number.

Syntax

POUT<Port Number> = <Value> or POUT(<Port Number>) = <Value>
<Variable> = POUT<Port Number> or <Variable> = POUT(<Port Number>)

Description

POUT command presents 2 methods to use.
First method is to output the specified value. Second is to read the output signal value in the specified port number and to store it in a variable.

1) The <Port Number> can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character POUT and <Number>.
2) The range of <Port Number> is 0 ~ 1 and it can be extended to max 5 when using the expansion board.

Return value

<Variable> Data type: <Integer Type>
The integer in 16 bits is returned as a result value.
Data values increase as moving from the right to the left.

Bit / Value	F E D C	B A 9 8	7 6 5 4	3 2 1 0
31242	0 1 1 1	1 0 1 0	0 0 0 0	1 0 1 0
-63	1 1 1 1	1 1 1 1	1 1 0 0	0 0 0 1
0HF419	1 1 1 1	0 1 0 0	0 0 0 1	1 0 0 1
0B11110011	0 0 0 0	0 0 0 0	1 1 1 1	0 0 1 1

Arguments

〈Value〉	Data type: 〈Integer Type〉 The allowed input value range is -32,768 ~ 32,767 in 16 bits.
〈Port Number〉	Data type: 〈Integer Type〉 • General output: POUT0 ~ 1 (POUT0: OUT0 ~ OUT15, OUT1: OUT16 ~ OUT19) • Expansion 1 output: POUT0 ~ 3 (POUT0: OUT0 ~ OUT15, POUT1: OUT16 ~ OUT 31, POUT2: OUT32 ~ OUT 47, POUT3: OUT48 ~ OUT 51) • Expansion 2 output: POUT0 ~ 5 (POUT0: OUT0 ~ OUT15, POUT1: OUT16 ~ OUT31, POUT2: OUT32 ~ OUT 47, POUT3: OUT48 ~ OUT 63, POUT4: OUT 64 ~ OUT 79, POUT5: OUT 80 ~ OUT 83)

Examples

Example 1

```
POUT(0) = 6245          // Output the value to port 0
DLAY 1000               // 1 second Time Delay
POUT0 = 0               // 0 is output to port 0
```

Example 2

```
INT STATE               // Declare an integer variable
STATE = POUT1           // Output signal value POUT1 store
PRINT(0,STATE)          // Store in index 0 of monitoring log
STATE = POUT(1)         // Output signal value POUT1 store
PRINT(1,STATE)          // Store in index 1 of monitoring log
STATE = POUT(I0)        // POUT signal store by referring to global variable I0
PRINT(2,STATE)          // value
                        // Store in index 2 of monitoring log
```

Related terms

OUT

8.5 CIN (Field Bus Bit Input)

Usage

This reads the value of bit number specified in field bus card.

Syntax

CIN<Bit Number> = <0/1> or CIN(<Bit Number> [, <Waiting Time>]) = <0/1>
<Variable> = CIN<Bit Number> or <Variable> = CIN(<Bit Number>)

Description

CIN command presents 2 methods to use.
First method is to assign 0 or 1 to the specified bit and it waits until the input signal of corresponding bit becomes the same value as the assigned. Second method is to read the signal value of the specified bit number and stores it in a variable.

1) The <Bit Number> can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CIN and <Number>.
2) Caution is required because a range of <Bit Number> can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.
3) <Waiting Time> can be set only when parentheses are used, and the setting range is from 100 to 99,999 msec.

Return value

<Variable>	Data type: <Integer Type> The 0(off) or 1(on) is returned as a result value.

Arguments

<0/1>	Data type: <Integer Type> Only 0(off) or 1(on) can be input.
<Bit Number>	Data type: <Integer Type> • CC-Link Ver. 1.0: CIN0 ~ CIN127 (Remote Input/output Area)
<Waiting Time>	Data type: <Integer> Range of <Wating Time> is 100 to 99,999 msec.

Examples

Example 1

```
SVON                // Servo on
JMOV P0             // Move JMOV to P0
CIN0 = 1            // Standby until input signal of CIN0 becomes 1
JMOV P1             // Move JMOV to P1
```

Example 2

```
SVON                          // Servo on
WHILE 1                       // Endless loop Start
    CIN(0,1000) = 1           // Wait 1 second until CIN0 input signal is 1
    IF CIN0 == 0 THEN         // If CIN0 signal value is 0
        JMOV GP0              // Move JMOV from current position to GP0
    ELSE                      // If no conditional expression is satisfied
        JMOV GP1              // Move JMOV from current position to GP1
    ENDIF                     // IF statement End
ENDWL                         // Endless
```

Example 3

```
INT STATE, NUM                // Declare integer variables
NUM = 0                       // Initialize an integer variable
STATE = CIN0                  // CIN0 signal value store
PRINT(0, STATE)               // Store in index 0 of monitoring log
STATE = CIN(0)                // CIN0 signal value store
PRINT(1,STATE)                // Store in index 1 of monitoring log
STATE = CIN(NUM)              // CIN signal value store by referring to NUM variable
PRINT(2,STATE)                // Store in index 2 of monitoring log
```

Related terms

CBIN, CWIN, CFIN

1) Caution is required to use in case of signal value standby.

8.6 COUT (Field Bus Bit Output)

Usage

This outputs the specified value to the specified bit number of field bus card.

Syntax

COUT<Bit Number> = <0/1> or COUT(<Bit Number>) = <0/1>
<Variable> = COUT<Bit Number> or <Variable> = COUT(<Bit Number>)
COUT<Bit Number> = <0/1>[<Pulse Width>]
COUT<Bit Number> = <0/1>[<Pulse Width> <Pulse Cycle>]

Description

COUT command presents 4 methods to use.
First method is to output the specified value.
Second method is to read the output signal value in the specified bit number and to store it in a variable.
Third method is to send an impulse signal to the specified bit number.
Last method is to send the pulse cycle wave to the specified bit number.

1) The <Bit Number> can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character COUT and <Number>.
2) Caution is required because a range of <Bit Number> can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.
3) In case of additionally input <Pulse Width>, it sends out the impulse signal for the same duration as input time and returns to the former value.
4) Once the <Pulse Cycle> is input, the pulse cycle waves, which have a cycle of the input time, are sent. The <Pulse Cycle> always must be inputted together with the <Pulse Width> and must be larger than the value of <Pulse Width>.

Return value

<Variable>	Data type: <Integer Type> The 0(off) or 1(on) is returned as a result value.

Arguments

〈0/1〉	Data type: 〈Integer Type〉 Only 0(off) or 1(on) can be input.
〈Bit Number〉	Data type: 〈Integer Type〉 • CC-Link Ver. 1.0: COUT0 ~ COUT127 (Remote Input/output Area)
〈Pulse Width〉	Data type: 〈Integer Type〉 The input unit is 1ms. However, values below 2ms are ignored, so please enter in multiples of 2.
〈Pulse Cycle〉	Data type: 〈Integer Type〉 The input unit is 1ms. However, values below 2ms are ignored, so please enter in multiples of 2. Pulse cycle must be larger than pulse width.

Examples

Example 1	COUT(0) = 1 DLAY 1000 COUT0 = 0	// 1 is output to the output signal of COUT0 // 1 second Time Delay // 0 is output to the output signal of COUT0

Example 2	INT STATE
	STATE = COUT1
	PRINT(0,STATE)
	STATE = COUT(1)
	PRINT(1,STATE)
	STATE = COUT(I0)
	PRINT(2,STATE)
		// Declare an integer variable
		// COUT1 signal value store
		// Store in index 0 of monitoring log
		// COUT1 signal value store
		// Store in index 1 of monitoring log
		// COUT signal store by referring to global variable I0 value
		// Store in index 2 of monitoring log

Example 3	COUT0 = 0
	COUT(0) = 1 100
		// 0 is output to the output signal of COUT0
		// Impulse with a length of 0.1 sec is output to output signal of
		// COUT0

Example 4	COUT1 = 0
	COUT1 = 1 1000 5000
		// 0 is output to the output signal of COUT1
		// Waves with a pulse width of 1 sec and 5 sec cycle are output to
		// output signal of COUT1

Related terms

CBOUT, CWOUT, CFOUT

1) The input unit of 〈Pulse Width〉 and 〈Pulse Cycle〉 is 1ms when creating impulse signals or pulse cycle waves using COUT command. But, the input value must be a multiple of 2 since the values below 2ms are ignored.
   - COUT0 = 1 1 → No pulse output, Stays with the output value
   - COUT0 = 1 15 → Output of impulse with 10ms length
   - COUT0 = 1 10 15 → Output of waves with 14ms cycle and 10ms pulse width
   - COUT0 = 1 15 29 → Output of waves with 28ms cycle and 14ms pulse width

8.7 CBIN (Field Bus Byte Input)

Usage

This reads the value of byte number specified in field bus card.

Syntax

CBIN<Byte Number> = <Value> or CBIN(<Byte Number> [, <Waiting Time>]) = <Value>
<Variable> = CBIN< Byte Number> or <Variable> = CBIN(<Byte Number>)

Description

This reads the signal value of the specified byte number and stores it in a variable.

1) The <Byte Number> can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CBIN and <Number>.
2) Caution is required because a range of <Byte Number> can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.
3) <Waiting Time> can be set only when parentheses are used, and the setting range is from 100 to 99,999 msec.

Return value

<Variable> Data type: <Integer Type>
The 8 bits constant is returned as a result value. Data values increase as moving to right direction. Refer to PIN (Digital Port Input).

Arguments

<Value> Data type: <Integer Type>
Range of <Value> is 0 to 256

<Byte Number> Data type: <Integer Type>

- CC-Link Ver. 1.0: CBIN0 ~ CBIN15 (Remote Input/output Area)

<Waiting Time> Data type: <Integer>
Range of <Wating Time> is 100 to 99,999 msec.

Examples

Example 1

```
INT STATE                   // Declare an integer variable
STATE = CBIN0               // CBIN0 signal value store
PRINT(0,STATE)              // Store in index 0 of monitoring log
STATE = CBIN(0)             // CBIN0 signal value store
PRINT(1,STATE)              // Store in index 1 of monitoring log
```

Example 2

```
WHILE 1                          // Endless loop Start
    CBIN(0,1000) = 255           // Wait 1 second until CBIN0 input signal is 1
    IF CBIN0 == 255 THEN         // If CBIN0 signal value is 255
        OUT(0) = 0               // 0 is output to the output signal 0
    ELSE                         // If no conditional expression is satisfied
        OUT(0) = 1               // 1 is output to the output signal 0
    ENDIF                        // IF statement End
ENDWL                            // Endless Loop End
```

Example 3

- Bit Operator Application Example

```
INT AA, RES                      // Declare integer variables
DEFSTR STR
AA = CBIN0                       // CBIN0 signal value store
RES = AA | 0HF0                  // Bit OR operation for the 4th ~ 7th value
STR = BINS(RES)                  // Converts into result data binary Character String
PRINT(0, STR)                    // Store in index 0 of monitoring log as binary string
RES = 0B0110 & CBIN0             // Bit AND operation for the 1st & 2nd values
STR = BINS(RES)                  // Converts into result data binary Character String
PRINT(1, STR)                    // Store in index 1 of monitoring log as binary string
```

Related terms

CIN, CWIN, CFIN

8.8 CBOUT (Field Bus Byte Output)

Usage

This outputs the specified value to the byte number of specified field bus card.

Syntax

CBOUT<Byte Number> = <Value> or CBOUT(<Byte Number>) = <Value>
<Variable> = CBOUT<Byte Number> or <Variable> = CBOUT(<Byte Number>)

Description

CBOUT command presents 2 methods to use.
First method is to output the specified value. Second method is to read the output signal value in the specified byte number and to store it in a variable.

1) The <Byte Number> can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CBOUT and <Number>.
2) Caution is required because a range of <Byte Number> can vary as a type of Field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.

Return value

<Variable> Data type: <Integer Type>
The 8 bits constant is returned as a result value. Data values increase as moving to right direction. Refer to POUT (Digital Port Output).

Arguments

<Value> Data type: <Integer Type>
The allowed range of Input Value is -128 ~ 127 in 8 bits.

<Byte Number> Data type: <Integer Type>

- CC-Link Ver. 1.0: CBOUT0 ~ CBOUT15 (Remote Input/output Area)

Examples

Example 1

```
CBOUT(0) = 62          // Byte value is output to CBOUT0
CBOUT0 = 0             // 0 is output to CBOUT0
```

Example 2

```
INT STATE              // Declare an integer variable
STATE = CBOUT1         // Output CBOUT1 signal value store
PRINT(0,STATE)         // Store in index 0 of monitoring log
STATE = CBOUT(1)       // Output CBOUT1 signal value store
PRINT(1,STATE)         // Store in index 1 of monitoring log
STATE = CBOUT(I0)      // CBOUT signal store by referring to global variable I0 value
PRINT(2,STATE)         // Store in index 2 of monitoring log
```

Related terms

COUT, CWOUT, CFOUT

8.9 CDIN (Inputting Fieldbus Double Word)

Usage

Reads value of designated Double Word Number in an integer form from the Fieldbus Card.

Syntax

〈Variable〉 = CDIN〈Double Word Number〉
〈Variable〉 = CDIN(〈Double Word Number〉)

Description

Reads Signal value of the designated Double Word Number and saves it at Variable.

1) Use parenthesis to input 〈Double Word Number〉 or directly input number. Must be no blank between Character CDIN and 〈Number〉.
2) 〈Double Word Number〉 range may differ from Type of used Fieldbus. Fieldbus Type can be set in 1.HW_CONF(1/3) → 4.OPTCARD of Public Parameter.

Return value

〈Variable〉 Data type: 〈Integer〉
32bit integer value returns as result value.

Arguments

〈Double Word Number〉 Data type: 〈Integer〉
- CC-Link Ver. 1.0: CDIN0 ~ CDIN7 (Remote Input/output Area)

Examples

```
INT DATA                    // Declares Integer Variable
DATA = CDIN0                // Saves CDIN0 Signal value
PRINT(0, DATA)              // saves in integer value at Monitoring Log Index No 0
DATA = CDIN(0)              // Saves CDIN0 Signal value
PRINT(1, DATA)              // saves in integer value integer value at Monitoring Log Index
                            // No 1
```

Related terms

CIN, CBIN, CWIN

8.10 CDOUT (Outputting Fieldbus Double Word)

Usage

Outputs designated value in an integer form at designated Fieldbus Card Double Word Number.

Syntax

CDOUT〈Double Word Number〉 = 〈value〉
CDOUT(〈Double Word Number〉) = 〈value〉
〈Variable〉 = CDOUT〈Double Word Number〉
〈Variable〉 = CDOUT(〈Double Word Number〉)

Description

CDOUT command suggests 2 use methods.
The first method, output designated value in an integer form.
The second method, read Signal value from designated Double Word Number and save it at Variable.

1) Use parenthesis to input 〈Double Word Number〉 or directly input number. Must be no blank between Character CDOUT and 〈Number〉.
2) 〈Double Word Number〉 range may differ from Type of used Fieldbus. Fieldbus Type can be set in OPTCARD of Public Parameter.

Return value

〈Variable〉 Data type: 〈Integer〉
32bit integer value returns as result value.

Arguments

〈value〉 Data type: 〈Integer〉
Imputable value range is 32bit -2,147,483,648 ~ 2,147,483,647.

〈Double Word Number〉 Data type: 〈Integer〉

- CC-Link Ver. 1.0: CFIN0 ~ CFIN7 (Remote Input/output Area)

Examples

Example 1

```
CDOUT(0) = 145748        // Outputs integer value to CDOUT No 0
CDOUT0 = 0               // Outputs 0 value to CDOUT No 0
```

Example 2

```
INT STATE                // Declares Integer Variable
STATE = CDOUT1           // Saves output CDOUT1 Signal value
PRINT(0,STATE)           // Saves at Monitoring Log Index No 0
STATE = CDOUT(1)         // Saves output CDOUT1 Signal value
PRINT(1,STATE)           // Saves at Monitoring Log Index No 1
STATE = CDOUT(I0)        // Saves CDOUT Signal by referring to Global Variable No I0
PRINT(2,STATE)           // saves at Monitoring Log Index No 2
```

Related terms

COUT, CBOUT, CWOUT

8.11 CWIN (Field Bus Word Input)

Usage

This reads the value of word number specified in field bus card.

Syntax

〈Variable〉 = CWIN〈Word Number〉 or 〈Variable〉 = CWIN(〈Word Number〉)

Description

This reads the signal value of the specified word number and stores it in a variable.

1) The 〈Word Number〉 can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CWIN and 〈Number〉.
2) Caution is required because a range of 〈Word Number〉 can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.

Return value

〈Variable〉 Data type: 〈Integer Type〉
The 16 bits integer is returned as a result value. Data values increase as moving to right direction. Refer to PIN (Digital Port Input).

Arguments

〈Word Number〉 Data type: 〈Integer Type〉

- CC-Link Ver. 1.0: CWIN0 ~ CWIN15 (Remote Register Area)

Examples

Example 1

```
INT STATE                  // Declare an integer variable
STATE = CWIN0              // CWIN0 signal value store
PRINT(0,STATE)             // Store in index 0 of monitoring log
STATE = CWIN(0)            // CWIN0 signal value store
PRINT(1,STATE)             // Store in index 1 of monitoring log
```

Example 2

- Bit Operator Application Example

```
INT AA, RES                // Declare integer variables
DEFSTR STR                 // Converts into result data binary Character String
AA = CWIN0                 // CWIN0 signal value store
RES = AA | 0HFF00          // Bit OR operation for the 8 ~ 15th value
STR = BINS(RES)            // Converts into result data binary Character String
PRINT(0, STR)              // Store in index 0 of monitoring log as binary string
RES = 0B10000110 & CBIN0   // Bit AND operation for 1 ~ 2, 7th value
STR = BINS(RES)            // Converts into result data binary Character String
PRINT(1, STR)              // Store in index 1 of monitoring log as binary string
```

Related terms

CIN, CBIN, CFIN

8.12 CWOUT (Field Bus Word Output)

Usage

This outputs the specified value to the word number of specified field bus card.

Syntax

CWOUT〈Word Number〉 = 〈Value〉 or CWOUT(〈Word Number〉) = 〈Value〉
〈Variable〉 = CWOUT〈Word Number〉 or 〈Variable〉 = CWOUT(〈Word Number〉)

Description

CWOUT command presents 2 methods to use.
First method is to output the specified value. Second method is to read the output signal value in the specified word number and to store it in a variable

1) The 〈Word Number〉 can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CWOUT and 〈Number〉.
2) Caution is required because a range of 〈Word Number〉 can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.

Return value

〈Variable〉 Data type: 〈Integer Type〉
The 16 bits integer is returned as a result value. Data values increase as moving to right direction. Refer to PIN (Digital Port Output).

Arguments

〈Value〉 Data type: 〈Integer Type〉
The allowed input value range is -32,768 ~ 32,767 in 16 bits

〈Word Number〉 Data type: 〈Integer Type〉
- CC-Link Ver. 1.0: CWOUT0 ~ CWOUT15 (Remote Register Area)

Examples

Example 1

```
CWOUT(0) = -2456        // 16 bits constant value is output to CWOUT 0
CWOUT0 = 0              // 0 is output to CWOUT 0
```

Example 2

```
INT STATE               // Declare an integer variable
STATE = CWOUT1          // Output CWOUT1 signal value store
PRINT(0,STATE)          // Store in index 0 of monitoring log
STATE = CWOUT(1)        // Output CWOUT1 signal value store
PRINT(1,STATE)          // Store in index 1 of monitoring log
STATE = CWOUT(I0)       // CWOUT signal store by referring to global variable I0
PRINT(2,STATE)          // Store in index 2 of monitoring log
```

Related terms

COUT, CBOUT, CFOUT

8.13 CFIN (Field Bus Double Word Input)

Usage

This reads the value of double word number specified in field bus card.

Syntax

〈Variable〉 = CFIN〈Double Word Number〉 or 〈Variable〉 = CFIN(〈Double Word Number〉)

Description

This reads the signal value of the specified double word number and stores it in a variable.

1) The 〈Double Word Number〉 can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CFIN and 〈Number〉.
2) Caution is required because a range of 〈Double Word Number〉 can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3) → 4.OPTCARD among the public parameters.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The 32 bits of real number is returned as a result value.

Arguments

〈Double Word Number〉	Data type: 〈Integer Type〉 • CC-Link Ver. 1.0: CFIN0 ~ CFIN7 (Remote Register Area)

Examples

```
REAL DATA                  // Declare an real number variable
DATA = CFIN0               // CFIN0 signal value store
PRINT(0, DATA)             // Store in index 0 of monitoring log as integer value
DATA = CFIN(0)             // CFIN0 signal value store
PRINT(1, DATA)             // Store in index 1 of monitoring log as real number value
```

Related terms

CIN, CBIN, CWIN

8.14 CFOUT (Field Bus Double Word Output)

Usage

This outputs the specified value to the double word number of specified field bus card.

Syntax

CFOUT〈Double Word Number〉 = 〈Value〉 or CFOUT(〈Double Word Number〉) = 〈Value〉
〈Variable〉 = CFOUT〈Double Word Number〉or〈Variable〉 = CFOUT(〈Double Word Number〉)

Description

CFOUT command presents 2 methods to use.
First method is to output the specified value. Second method is to read the output signal value in the specified double word number and to store it in a variable

1) The 〈Double Word Number〉 can be used by either using the parenthesis or directly inputting the number. No blank is allowed between the character CFOUT and 〈Number〉.
2) Caution is required because a range of 〈Double Word Number〉 can vary as a type of field bus used. Field bus type can be set in 1.HW_CONF(1/3)→4. OPTCARD among the public parameters.

Return value

〈Variable〉 Data type: 〈Real Number Type〉
The 32 bits of real number is returned as a result value.

Arguments

〈Value〉 Data type: 〈Real Number Type〉
The allowed input value range is 3.402,823E-38 ~ 3.402,823E+38 in 32 bits.

〈Double Word Number〉 Data type: 〈Integer〉

- CC-Link Ver. 1.0: CFIN0 ~ CFIN7 (Remote Register Area)

Examples

Example 1

```
CFOUT(0) = 14.5748      // Real number value is output to CFOUT 0
CFOUT0 = 0.0            // 0 is output to CFOUT 0
```

Example 2

```
REAL STATE              // Declare an real number variable
STATE = CFOUT1          // Output CFOUT1 signal value store
PRINT(0,STATE)          // Store in index 0 of monitoring log
STATE = CFOUT(1)        // Output CFOUT1 signal value store
PRINT(1,STATE)          // Store in index 1 of monitoring log
STATE = CFOUT(I0)       // CFOUT signal store by referring to global variable I0 value
PRINT(2,STATE)          // Store in index 2 of monitoring log
```

Related terms

COUT, CBOUT, CWOUT

8.15 XIN (Inputting CNET bit)

Usage

Reads value from CNET Receiving Buffer by referring to the designated Device Area and bit Number.

Syntax

〈Integer Variable〉 = XIN(〈Device Area〉, 〈bit Number〉)

Description

Reads bit input from CNET Receiving Buffer by referring to the designated Device Area and bit Number.

1) 〈Device Area〉 inputs Character String "M" or "D".
2) 〈Bit Number〉 range differs from Device Area.

Return value

〈Variable〉	Data type: 〈Integer〉 0(OFF) or 1(ON) value returns as result value.

Arguments

〈Device Area〉	Data type: 〈Character String〉 Only "M" or "D" Character String can be input.
〈bit Number〉	Data type: 〈Integer〉 • Device M Area: 0 ~ 383 (Total 384 ea) • Device D Area: 0 ~ 2047 (Total 2048 ea)

Examples

Example 1

```
INT DATA                    // Declares Integer Variable
DATA = XIN("M", 0)          // reads Device M Area No 0 bit value
```

Example 2

```
SVON                        // Servo on
IF XIN("D", 100) == 1 THEN  // If Device D Area No 100 bit value is 1,
    JMOV P1                 // JMOV to P1 position
ENDIF                       // If statement end
```

Related terms

XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.16 XOUT (CNET bit output)

Usage

Outputs value to CNET Device Sending Buffer by referring to the designated Device and bit Number.

Syntax

XOUT(<Device Area>, <bit Number>) = <0/1>
<Integer Variable> = XOUT(<Device Area>, <bit Number>)

Description

XOUT command suggests 2 use methods.
The first method, output designated value to CENT Sending Buffer.
The second method, read Signal value output in designated bit Number and save it at Variable.

1) <Device Area> inputs Character String "M" or "D".
2) <Bit Number> range differs from Device Area.

Return value

<Variable>	Data type: <Integer> 0(OFF) or 1(ON) value returns as result value.

Arguments

<Device Area>	Data type: <Character String> Only "M" or "D" Character String can be input.
<bit Number>	Data type: <Integer> • Device M Area: 0 ~ 383 (Total 384 ea) • Device D Area: 0 ~ 2,047 (Total 2048 ea)

Examples

Example 1	XOUT("M", 0) = 1	// Outputs 1 value to Device M Area No 0 bit

Example 2	INT DATA
	XOUT("D", 100) = 1
	DATA = XOUT("D", 100)
		// Declares Integer Variable
		// Outputs 1 value to Device D Area No 100 bit
		// Reads Device D Area No 100 bit Output Value

Related terms

XIN, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.17 XBIN (Inputting CNET Byte)

Usage

Reads value from CNET Receiving Buffer by referring to designated Device and Byte Number.

Syntax

〈Integer Variable〉 = XBIN(〈Device Area〉, 〈Byte Number〉)

Description

Reads Byte input from CNET Receiving Buffer by referring to the designated Device Area and Byte Number.

1) 〈Device Area〉 inputs Character String "M" or "D".
2) 〈Byte Number〉 range differs from Device Area.

Return value

〈Variable〉 Data type: 〈Integer〉
8bit integer value returns as result value and data value increases from right to left.
Refer to description of PIN (Digital Port Input).

Arguments

〈Device Area〉 Data type: 〈Character String〉
Only "M" or "D" Character String can be input.

〈Byte Number〉 Data type: 〈Type〉

- Device M Area: 0 ~ 47 (Total 48 ea)
- Device D Area: 0 ~ 255 (Total 256 ea)

Examples

Example 1

```
INT DATA                    // Declares Integer Variable
DATA = XBIN("M", 0)         // Reads Device M Area No 0 Byte data
```

Example 2

- bit Operator Application Example

```
INT DATA                    // Declares Integer Variable
DEFSTR STR                  // Declares Character String Variable
DATA = XBIN("D",0) & 0H000F // PERFORMS AND OPERATION 0 ~ 3rd value bit of
                            // Device D Area No 0 Byte
STR = BINS(DATA)            // Converts into result data binary Character String
PRINT(0,STR)                // Outputs Character String to Monitoring Log Index No 0
DATA = 0B11110000 ^ XBIN("D",0) // PERFORMS XOR OPERATION 4 ~ 7th value bit of Device
                            // D Area No 0 Byte
STR = BINS(DATA)            // Converts into result data binary Character String
PRINT(1,STR)                // Outputs Character String to Monitoring Log Index No 2
```

Related terms

XIN, XOUT, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.18 XBOUT (Outputting CNET Byte)

Usage

Outputs value to CNET Device Sending Buffer by referring to the designated Device and Byte Number.

Syntax

XBOUT(<Device Area>, <Byte Number>) = <Output Value>
<Integer Variable> = XBOUT(<Device Area>, <Byte Number>)

Description

XBOUT command suggests 2 use methods.
The first method, output designated value to CENT Sending Buffer.
The second method, read Signal value output in designated Byte Number and save it at Variable.

1) <Device Area> inputs Character String "M" or "D".
2) <Byte Number> range differs from Device Area.

Return value

<Variable>	Data type: <Integer> 8bit integer value returns as result value and data value increases from right to left. Refer to description of PIN (Digital Port Input).

Arguments

<Device Area>	Data type: <Character String> Only "M" or "D" Character String can be input.
<Byte Number>	Data type: <Type> • Device M Area: 0 ~ 47 (Total 48 ea) • Device D Area: 0 ~ 255 (Total 256 ea)

Examples

Example 1	XBOUT("M", 0) = 0HFF	// Writes value at Device M Area No 0 Byte
Example 2	INT DATA DATA = XBOUT("D", 0)	// Declares Integer Variable // Reads Device D Area No 0 Byte Output Value

Related terms

XIN, XOUT, XBIN, XWIN, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.19 XWIN (Inputting CNET Word)

Usage

Reads value from CNET Receiving Buffer by referring to the designated Device and Word Number.

Syntax

〈Integer Variable〉 = XWIN(〈Device Area〉, 〈Word Number〉)

Description

Reads Word Input from CNET Receiving Buffer by referring to the designated Device Area and Word Number.

1) 〈Device Area〉 inputs Character String "M" or "D".
2) 〈Word Number〉 range differs from Device Area.

Return value

〈Variable〉 Data type: 〈Integer〉
16bit integer value returns as result value and data value increases from right to left. Refer to description of PIN (Digital Port Input).

Arguments

〈Device Area〉 Data type: 〈Character String〉
Only "M" or "D" Character String can be input.

〈Word Number〉 Data type: 〈Type〉

- Device M Area: 0 ~ 23 (Total 24 ea)
- Device D Area: 0 ~ 127 (Total 128 ea)

Examples

Example 1

```
INT DATA                    // Declares Integer Variable
DATA = XWIN("M", 0)         // Reads Device M Area No 0 Word value
```

Example 2

- bit Operator Application Example

```
INT RES                     // Declares Integer Variable
DEFSTR STR                  // Declares Character String Variable
RES = XWIN("D", 0) | 0HFF00 // PERFORMS OR OPERATION 8 ~ 15th value bit of
                            // Device D Area No 0 Word
STR = BINS(RES)             // Saves in binary Character String
PRINT(0,STR)                // Outputs Character String to Monitoring Log Index No 0
RES = 0B10000110 & XWIN("D", 0) // PERFORMS AND OPERATION 1 ~ 2, 7th value bit of
                            // Device D Area No 0 Word
STR = BINS(RES)             // Saves in binary Character String
PRINT(1,STR)                // Outputs Character String at Monitoring Log Index No 1
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWOUT, XDIN, XDOUT, XFIN, XFOUT

8.20 XWOUT (Outputting CNET Word)

Usage

Outputs value to CNET Device Sending Buffer by referring to the designated Device and Word Number.

Syntax

XWOUT(<Device Area>, <Word Number>) = <Output Value>
<Integer Variable> = XWOUT(<Device Area>, <Word Number>)

Description

XWOUT command suggests 2 use methods.
The first method, output designated value to CENT Sending Buffer.
The second method, read Signal value output by designated Word Number and save it at Variable.

1) <Device Area> inputs Character String "M" or "D".
2) <Word Number> range differs from Device Area.

Return value

<Variable>	Data type: <Integer> 16bit integer value returns as result value and data value increases from right to left. Refer to description of PIN (Digital Port Input).

Arguments

<Device Area>	Data type: <Character String> Only "M" or "D" Character String can be input.
<Word Number>	Data type: <Type> • Device M Area: 0 ~ 23 (Total 24 ea) • Device D Area: 0 ~ 127 (Total 128 ea)

Examples

Example 1	XWOUT("M", 0) = 0HFFFF	// Writes value at Device M Area No 0 Word
Example 2	INT DATA DATA = XWOUT("D", 0)	// Declares Integer Variable // Reads Device D Area No 0 Word Output Value

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XDIN, XDOUT, XFIN, XFOUT

8.21 XDIN (Inputting CNET Double Word)

Usage

Reads value from CNET Receiving Buffer by referring to the designated Device and Double Word Number.

Syntax

〈Integer Variable〉 = XDIN(〈Device Area〉, 〈Double Word Number〉)

Description

Reads Double Word Input from CNET Receiving Buffer by referring to the designated Device Area and Double Word Number.

1) 〈Device Area〉 inputs Character String "M" or "D".
2) 〈Double Word Number〉 range differs from Device Area.

Return value

〈Variable〉	Data type: 〈Integer〉 32bit integer value returns as result value.

Arguments

〈Device Area〉	Data type: 〈Character String〉 Only "M" or "D" Character String can be input.
〈Double Word Number〉	Data type: 〈Type〉 • Device M Area: 0 ~ 11 (Total 12 ea) • Device D Area: 0 ~ 63 (Total 64 ea)

Examples

Example 1

```
INT DATA                    // Declares Integer Variable
DATA = XDIN("M", 0)         // Reads Device M Area No 0 Double Word value
```

Example 2

• bit Operator Application Example

```
INT RES                     // Declares Integer Variable
DEFSTR STR                  // Declares Character String Variable
RES = XDIN("D", 0) | 0HFFFF0000  // PERFORMS OR OPERATION 16 ~ 32nd value bit of
                            // Device D Area No 0 Double Word
STR = BINS(RES)             // Saves in binary Character String
PRINT(0,STR)                // Outputs Character String to Monitoring Log Index No 0
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDOUT, XFIN, XFOUT

8.22 XDOUT (Outputting CNET Double Word)

Usage

Outputs value to CNET Device Sending Buffer by referring to the designated Device and Double Word Number.

Syntax

XDOUT(<Device Area>, <Double Word Number>) = <Output Value>
<Integer Variable> = XDOUT(<Device Area>, <Double Word Number>)

Description

XDOUT command suggests 2 use methods.
The first method, output designated value to CENT Sending Buffer.
The second method, read Signal value from designated Double Word Number and save it at Variable.

1) <Device Area> inputs Character String "M" or "D".
2) <Double Word Number> range differs from Device Area.

Return value

<Variable>	Data type: <Integer> 32bit integer value returns as result value.

Arguments

<Device Area>	Data type: <Character String> Only "M" or "D" Character String can be input.
<Double Word Number>	Data type: <Type> • Device M Area: 0 ~ 11 (Total 12 ea) • Device D Area: 0 ~ 63 (Total 64 ea)

Examples

Example 1	XDOUT("M", 0) = 0HFFFFFFFF	// Writes value at Double Word in Device M Area No 0
Example 2	INT DATA DATA = XDOUT("D", 0)	// Declares Integer Variable // Reads Device D Area No 0 Double Word Output value

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XFIN, XFOUT

8.23 XFIN (Inputting CNET real number)

Usage

Reads value from CNET Receiving Buffer in real number form by referring to the designated Device and Double Word Number.

Syntax

〈Real Type Variable〉 = XFIN(〈Device Area〉, 〈Double Word Number〉)

Description

Reads real number input from CNET Receiving Buffer by referring to the designated Device Area and Double Word Number.

1) 〈Device Area〉 inputs Character String "M" or "D".
2) 〈Double Word Number〉 range differs from Device Area.

Return value

〈Variable〉	Data type: 〈real number type〉 32bit real number value is returned as result value.

Arguments

〈Device Area〉	Data type: 〈Character String〉 Only "M" or "D" Character String can be input.
〈Double Word Number〉	Data type: 〈Type〉 • Device M Area: 0 ~ 11 (Total 12 ea) • Device D Area: 0 ~ 63 (Total 64 ea)

Examples

```
Example
REAL DATA1, DATA2          // Declares Real Type Variable
DATA1 = XFIN("M", 0)       // Reads Real number value of Device M Area No 0 Double
                           // Word Area
DATA2 = XFIN("D", 63)      // Reads real number value of Device D Area No 63 Double
                           // Word Area
```

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFOUT

8.24 XFOUT (Outputting CNET real number)

Usage

Outputs real number value to CNET Device Sending Buffer by referring to the designated Device and Double Word Number.

Syntax

XFOUT(<Device Area>, <Double Word Number>) = <real number value>
<Real Type Variable> = XFOUT(<Device Area>, <Double Word Number>)

Description

XFOUT command suggests 2 use methods.
The first method, output designated value to CENT Sending Buffer.
The second method, read real number value output in designated Double Word Number and save it at Variable.

1) <Device Area> inputs Character String "M" or "D".
2) <Double Word Number> range differs from Device Area.

Return value

<Variable>	Data type: <real number type> 32bit real number value is returned as result value.

Arguments

<Device Area>	Data type: <Character String> Only "M" or "D" Character String can be input.
<Double Word Number>	Data type: <Type> • Device M Area: 0 ~ 11 (Total 12 ea) • Device D Area: 0 ~ 63 (Total 64 ea)

Examples

Example 1	XFOUT("M", 0) = 12.45	// Writes real number value at Double Word in Device M // Area No 0

Example 2	REAL DATA
	DATA = XFOUT("D", 0)
		// Declares Real Type Variable
		// Reads Real number value of Double Word Area in
		// Device D Area No 0

Related terms

XIN, XOUT, XBIN, XBOUT, XWIN, XWOUT, XDIN, XDOUT, XFIN

9. Multithreading Control

9.1 CTHREAD (Thread Creation)

Usage

This executes a function or program simultaneously.

Syntax

[<Variable> =] CTHREAD(<Function Name>, <Thread Number>, [<Thread Alarm Stop>])

Description

This allocates the functions or programs to thread and executes them simultaneously. In case of allocating a function, the return type must be VOID and no parameter is included. In allocating a program, the file must be attached in advance using INCLUDE command.

1) The name of function or program can be input to <Function Name>.
2) The thread's number is input to <Thread Number>. In case that the selected thread number has already been used by another thread, No thread is created.
3) The <Thread Alarm Stop> sets the stop of thread when alarm occurs.

Return value

<Variable>	Data type: <Integer Type> The (-) values (Thread Creation Failure) or 0(Thread Creation Success) are returned as a result value.

Arguments

<Function Name>	Data type: <Function or Program Name> In case of inputting JOB program name, the "_MAIN" must be added at the behind of the Program Name. • Example: TEST.JOB -> CTHREAD(TEST_MAIN, 2)
<Thread Number>	Data type: <Integer Type> The range of thread number is 2 ~ 4(TH2 ~ TH4).
<Thread Alarm Stop>	Data type: <Integer Type> Either Thread Stop (1) or Thread Not Stop (0) is input in case of Alarm.

Examples

Example 1

- Program Call Example

```
INCLUDE “TEST”                      // Include TEST.JOB File
MAIN                                // Declare Main Function Start
    INT T_RET                       // Declare an integer variable
    T_RET = CTHREAD(TEST_MAIN,2)    // Allocate Main Function of TEST.JOB to thread 2
    IF T_RET == 0 THEN              // If creation result of thread is failure
        EXIT                        // JOB program End
    ENDIF                           // IF statement End

    ...
    WHILE 1                         // Endless Loop Start
    ENDWL                           // Endless Loop End
EOP                                 // Main Function End
```

Example 2

- Function Call Example

```
MAIN                                // Declare Main Function Start
    CTHREAD(MOVE, 2, 0)             // Allocate MOVE function to thread 2
                                    // In case of alarm occurring, thread not ended
    WHILE 1                         // Endless Loop Start
    ENDWL                           // Endless Loop End
EOP                                 // Main Function End

FUNC VOID MOVE()                    //Declare MOVE function(VOID type, no parameters)
    SVON                            // Servo on
    WHILE 1                         // Endless Loop Start
        JMOV P0                     // Move JMOV to P0
        DLAY 1000                   // 1 second Time Delay
        JMOV P1                     // Move JMOV to P1
        DLAY 1000                   // 1 second Time Delay
    ENDWL                           // Endless Loop End
END                                 // Function End
```

Related terms

ETHREAD, TH_STATE, TH_SUSPEND, TH_RESUME

9.2 ETHREAD (Thread End)

Usage

This forcedly ends a thread.

Syntax

[<Variable> =] ETHREAD(<Thread Number>)

Description

This completely ends the thread in execution.
If ETHREAD is used to end the thread, it is not possible to re-start using TH_RESUME command.

1) Thread number to be ended is input to <Thread Number>.
CTHREAD command with the inputted thread number is used to end the thread.

Return value

<Variable>	Data type: <Integer Type> Either 0(Thread End Success) or -1(Thread End Failure) is returned as a result value.

Arguments

<Thread Number>	Data type: <Integer Type> The range of thread number is 2 ~ 4(TH2 ~ TH4).

Examples

```
INCLUDE "TEST"                  // Include TEST.JOB file
MAIN                            // Declare Main Function Start
    INT T_RET                   // Declare an integer variable
    CTHREAD(TEST_MAIN,2,1)      // Allocate Main Function of TEST.JOB to thread 2
                                // Thread not ended in case of alarm occurring
    IN0 = 1                     // Standby until input signal in 0 becomes 1
    T_RET = ETHREAD(2)          // Thread forcedly end

    IF T_RET <0 THEN            // In case that execution result of thread is failure
        EXIT                    // JOB program End
    ENDIF                       // IF statement End
EOP                             // Main Function End
```

Related terms

CTHREAD, TH_STATE, TH_SUSPEND, TH_RESUME

9.3 TH_STATE (Thread Status Reading)

Usage

This checks current status of thread.

Syntax

〈Variable〉 = TH_STATE(〈Execution Mode〉, 〈Thread Number〉)

Description

This returns current status of each thread.

1) It is possible to check the status of thread in Robot Mode and System Mode by specifying 〈Execution Mode〉.
2) This can read corresponding thread by specifying 〈Thread Number〉.

Return value

〈Variable〉 Data type: 〈Integer Type〉
Current status of thread is returned.
- 0 (TH_ERROR): Thread Number Error
- 1 (TH_IDLE): No Thread Allocation, Newly Usable Status
- 2 (TH_STOP): Thread Stop
- 3 (TH_RUN): Thread in Operation

Arguments

〈Execution Mode〉 Data type: 〈Integer Type〉
Either Auto Mode (1) or Robot Mode (0) can be input.

〈Thread Number〉 Data type: 〈Integer Type〉
The range of Thread Number is 1 ~ 4(TH1 ~ TH4).

Examples

```
INCLUDE “MOVE”                          // Include MOVE.JOB file
MAIN                                    // Declare Main Function Start
    INT ST                              // Declare an integer variable
    INT P_ID, T_ID                      // Declare integer variables

    P_ID = 0                            // Specifying Robot mode
    T_ID = 2                            // Specifying thread 2

    CTHREAD(MOVE_MAIN,T_ID,1)           // Allocate Main Function of TEST.JOB to thread
                                        // Thread ended in case of alarm occurring
    WHILE 1                             // Endless Loop Start
        ST = TH_STATE(P_ID, T_ID)       // Status check of thread 2 in Robot mode

        IF (ST == 2) THEN               // If current status of thread 2 is TH_STOP state
            TH_RESUME(T_ID)             // Thread Re-start
        ELSEIF (ST == 3 && IN0 == 1) THEN // If status is TH_RUN state & input signal 0 is 1,
            TH_SUSPEND(T_ID)            // Thread Stop
        ENDIF                           // IF statement End

        DLAY 1000                       // 1 second Time Delay

        IF (ST == 1 && ST == 0) THEN    // In case of number thread allocated or error
            ETHREAD(T_ID)               // Thread End
        ENDIF                           // IF statement End

    ENDWL                               // Endless Loop End
EOP                                     // Main Function End
```

Related terms

CTHREAD, ETHREAD, TH_SUSPEND, TH_RESUME

9.4 TH_SUSPEND (Thread Stop)

Usage

This stops thread in execution.

Syntax

[<Variable> =] TH_SUSPEND(<Thread Number>)

Description

This stops thread in execution.
Thread in stop state can be re-executed by TH_RESUME command.

1) Thread number to be stopped is input to <Thread Number>.

Return value

<Variable>	Data type: <Integer Type> Either 0(Thread Stop Success) or -1(Thread Stop Failure) is returned as a result value.

Arguments

<Thread Number>	Data type: <Integer Type> The range of thread number is 2 ~ 4(TH2 ~ TH4).

Examples

```
INCLUDE “MOVE”                          // Include MOVE.JOB file
MAIN                                    // Declare Main Function Start
    INT ST                              // Declare an integer variable
    INT P_ID, T_ID                      // Declare integer variables

    P_ID = 0                            // Specifying Robot mode
    T_ID = 2                            // Specifying thread 2

    CTHREAD(MOVE_MAIN,T_ID,1)           // Allocate Main Function of TEST.JOB to thread
                                        // Thread ended in case of alarm occurring
    WHILE 1                             // Endless Loop Start
        ST = TH_STATE(P_ID, T_ID)       // Status check of thread 2 in Robot mode

        IF (ST == 3 && IN0 == 1) THEN   // If TH_RUN state, input signal in 0 is 1
            TH_SUSPEND(T_ID)            // Thread Stop
        ENDIF                           // IF statement End
    ENDWL                               // Endless Loop End
EOP                                     // Main Function End
```

Related terms

CTHREAD, ETHREAD, TH_STATE, TH_RESUME

9.5 TH_RESUME (Thread Re-Start)

Usage

This re-start the stopped thread.

Syntax

[<Variable> =] TH_RESUME(<Thread Number>)

Description

This re-starts the stopped thread.
Only the stopped thread by TH_SUSPEND command can be re-start.

1) Thread number is input to <Thread Number>.

Return value

<Variable>	Data type: <Integer Type> Either 0(Thread Stop Success) or -1(Thread Stop Failure) is returned as a result value.

Arguments

<Thread Number>	Data type: <Integer Type> The range of thread number is 2 ~ 4(TH2 ~ TH4).

Examples

```
INCLUDE “MOVE”                          // Include MOVE.JOB file
MAIN                                    // Declare Main Function Start
    INT ST                              // Declare an integer variable
    INT T_ID

    T_ID = 2                            // Specify thread 2

    CTHREAD(MOVE_MAIN,T_ID,1)           // Allocate Main Function of TEST.JOB to thread
                                        // Thread ended in case of alarm occurring
    WHILE 1                             // Start of endless loop
        ST = TH_STATE(0, T_ID)          // Status check of thread 2 in Robot mode

        IF (ST == 2) THEN               // If TH_STOP state
            TH_RESUME(T_ID)             // Thread Re-start
        ELSEIF(ST == 3 && IN0 == 1) THEN // If TH_RUN state, input signal in 0 is 1
            TH_SUSPEND(T_ID)            // Thread Stop
        ENDIF                           // IF statement End
    ENDWL                               // Endless Loop End
EOP                                     // Main Function End
```

Related terms

CTHREAD, ETHREAD, TH_STATE, TH_SUSPEND

10. Function

10.1 TOOL (Tool Coordinate System Selection)

Usage

This selects a using tool and corresponding coordinate system that were attached to robot.

Syntax

TOOL <Number>

Description

Tool number to be used in work is specified.
Position and direction of end-effector of tool are calculated by referring to the specified corresponding tool parameter values in case of coordinate conversion.
The value altered by this command is effective until the end of program.
In case of altering the tool value in the movement commands, the number inserted in the command has a priority.

1) A Tool and Coordinate System can be selected by inputting <Number>.
   Tool coordinate information can be altered in COORD-TOOL parameters.

Arguments

<Number>	Data type: <Integer Type> Tool Number can be 0 ~ 15.

Examples

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    JMOV P0             // Move JMOV to P0

    VEL 200             // Velocity value alteration
    TOOL 1              // Specify TOOL 1

    LMOV P1             // Move LMOV to P1
    LMOV P2             // Move LMOV to P2
EOP                     // Main Function End
```

Related terms

USER

1) TOOL 0 indicates the end-effector of robot and a state of no tool attached.
   It is not possible to alter corresponding parameters.
2) In case that TOOL command is not used in a JOB program, the value of TOOL 0 is applied.

10.2 USER (User Coordinate System Selection)

Usage

This selects user coordinate system.

Syntax

USER 〈Number〉

Description

User number to be used in work is specified.
Position and direction of end-effector of robot are calculated by referring to the specified corresponding user parameter values in case of coordinate conversion.
The value altered by this command is effective until the end of program.
In case of altering the user value in the movement commands, the number inserted in the command has a priority.

1) User coordinate system can be selected by inputting 〈Number〉.
   User coordinate information can be altered in COORD-USER parameters.

Arguments

〈Number〉	Data type: 〈Integer Type〉 Available user coordinate system number can be 0 ~ 15.

Examples

```
MAIN                    // Declare Main Function Start
    SVON                // Servo on
    JMOV P0             // Move JMOV to P0

    VEL 200             // Velocity value alteration
    USER 1              // Specify USER 1

    LMOV P1             // Move LMOV to P1
    LMOV P2             // Move LMOV to P2
EOP                     // Main Function End
```

Related terms

TOOL



1) In case that USER command is not used in a JOB program, the value of USER 0 is applied. Unlike TOOL, caution is required to use since USER 0 cannot be edited.

10.3 JTOB (Coordinate Data Calculation, JOINT → BASE)

Usage

This calculates the position value of base coordinate system from the position value of joint coordinate system.

Syntax

〈Variable〉 = JTOB([〈Robot Number〉,] 〈Input Variable〉)

Description

This calculates the position value of base coordinate system from the position value of joint coordinate system by referring to tool information of input position variable.

1) It is possible to select different robot other than the one specified when storing in JOB program by inputting 〈Robot Number〉.
2) A variable format must be entered when inputting the value to 〈Input Variable〉. If a constant is entered, alarm occurs.

Return value

〈Variable〉	Data type: 〈Position Type of Base Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Input Variable〉	Data type: 〈Position Type of Joint Coordinate System〉

Examples

```
MAIN                              // Declare Main Function Start
  POS PDATA                       // Declare a position variable of joint coordinate system
  XPOS XRDATA                     // Declare a position variable of base coordinate system
  PDATA = 〈30,15,-15,45,-45,60〉  // Assign the joint position value
  PDATA.T = 1                     // Assign the tool information
  XRDATA = JTOB(PDATA)            // Acquire a position variable of base coordinate system
  PRINT(0,XRDATA)                 // Store coordinate value in index 0 of monitoring log
  XRDATA = JTOB(2,PDATA)          // Acquire a position variable of base coordinate system
                                  // of robot 2
  PRINT(1,XRDATA)                 // Store coordinate value in index 1 of monitoring log
EOP                               // Main Function End
```

Related terms

BTOJ, JTOU, UTOJ, BTOU, UTOB, TOOL, USER

10.4 BTOJ (Coordinate Data Calculation, BASE → JOINT)

Usage

This calculates the position value of joint coordinate system from the position value of base coordinate system.

Syntax

〈Variable〉 = BTOJ([〈Robot Number〉,] 〈Input Variable〉)

Description

This calculates the position value of joint coordinate system from the position value of base coordinate system by referring to tool information of input position variable and robot posture.
If robot posture was not specified, position value of joint coordinate system is calculated based on current robot posture.

1) It is possible to select different robot other than the one specified when storing in JOB program by inputting 〈Robot Number〉.
2) A variable format must be entered when inputting the value to 〈Input Variable〉. If a constant is entered, alarm occurs.

Return value

〈Variable〉	Data type: 〈Position Type of Joint Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Input Variable〉	Data type: 〈Position Type of Base Coordinate System〉

Examples

```
MAIN                                  // Declare Main Function Start
    POS RDATA                         // Declare a position variable of joint coordinate
                                      // system
    XPOS XRDATA                       // Declare a position variable of base coordinate system
    XRDATA = 〈100,100,100,30,45,-     // Assign the base position value
    60,ARM = 0,T = 1〉
    RDATA = BTOJ(XRDATA)              // Acquire position variable of joint coordinate
                                      // system
    PRINT(0, RDATA)                   // Store coordinate value in index 0 of monitoring log
    RDATA = BTOJ(2,XRDATA)            // Acquire position variable of joint coordinate
                                      // system of robot 2
    PRINT(1, RDATA)                   // Store coordinate value in index 1 of monitoring log
EOP                                   // Main Function End
```

Related terms

JTOB, JTOU, UTOJ, BTOU, UTOB, TOOL, USER

10.5 JTOU (Coordinate Data Calculation, JOINT → USER)

Usage

This calculates the position value of user coordinate system from the position value of joint coordinate system.

Syntax

〈Variable〉 = JTOU([〈Robot Number〉,] 〈Input Variable〉)

Description

This calculates the position value of user coordinate system from the position value of joint coordinate system by referring to tool information of input position variable and user coordinate system information.

1) It is possible to select different Robot other than the one specified when storing in JOB program by inputting 〈Robot Number〉.
2) A variable format must be entered when inputting the value to 〈Input Variable〉. If a constant is entered, alarm occurs.

Return value

〈Variable〉	Data type: 〈Position Type of User Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Input Variable〉	Data type: 〈Position Type of Joint Coordinate System〉

Examples

```
MAIN                              // Declare Main Function Start
  POS PDATA                       // Declare a position variable of joint coordinate system
  UPOS URDATA                     // Declare a position variable of base coordinate system

  PDATA = 〈30,15,-15,45,-45,60〉   // Assign position variable of joint coordinate system
  PDATA.T = 2                     // Assign the tool information
  PDATA.U = 3                     // Assign the user coordinate system information

  URDATA = JTOU(PDATA)            // Acquire position variable of user coordinate system
  PRINT(0,URDATA)                 // Store coordinate value in index 0 of monitoring log
  URDATA = JTOU(2,PDATA)          // Acquire position variable of user coordinate system of
                                  // robot 2
  PRINT(1,URDATA)                 // Store coordinate value in index 1 of monitoring log
EOP                               // Main Function End
```

Related terms

JTOB, BTOJ, UTOJ, BTOU, UTOB, TOOL, USER

10.6 UTOJ (Coordinate Data Calculation, USER → JOINT)

Usage

This calculates the position value of joint coordinate system from the position value of user coordinate system.

Syntax

〈Variable〉 = UTOJ([〈Robot Number〉,] 〈Input Variable〉)

Description

This calculates the position value of joint coordinate system from the position value of user coordinate system by referring to tool information of input position variable and user coordinate system information.
If robot posture was not specified, the position value of joint coordinate system is calculated based on current robot posture.

1) It is possible to select different robot other than the one specified when storing in JOB program by inputting 〈Robot Number〉.
2) A variable format must be entered when inputting the value to 〈Input Variable〉. If a constant is entered, alarm occurs.

Return value

〈Variable〉	Data type: 〈Position Type of User Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Input Variable〉	Data type: 〈Position Type of Joint Coordinate System〉

Examples

```
MAIN                          // Declare Main Function Start
    POS RDATA                 // Declare a position variable of joint coordinate system
    UPOS URDATA               // Declare a position variable of base coordinate system

    URDATA  =  〈0,0,0,30,45,-  // Assign the user coordinate value
    60,ARM = 0,T = 1,U = 3〉

    RDATA = UTOJ(URDATA)      // Acquire position variable of joint coordinate system
    PRINT(0, RDATA)           // Store coordinate value in index 0 of monitoring log
    RDATA = UTOJ(2,URDATA)    // Acquire position variable of joint coordinate system of
                              // robot 2
    PRINT(1, RDATA)           // Store coordinate value in index 1 of monitoring log
EOP                           // Main Function End
```

Related terms

JTOB, BTOJ, JTOU, BTOU, UTOB, TOOL, USER

10.7 BTOU (Coordinate Data Calculation, BASE → USER)

Usage

This calculates the position value of user coordinate system from the position value of base coordinate system.

Syntax

<Variable> = BTOU([<Robot Number>,] <Input Variable>)

Description

This calculates the position value of user coordinate system from the position value of base coordinate system by referring to user coordinate system information of input position variable.

1) It is possible to select different robot other than the one specified when storing in JOB program by inputting <Robot Number>.
2) A variable format must be entered when inputting the value to <Input Variable>. If a constant is entered, Alarm occurs.

Return value

<Variable>	Data type: <Position Type of User Coordinate System>

Arguments

<Robot Number>	Data type: <Integer Type> The range of <Robot Number> is 1 ~ 3.
<Input Variable>	Data type: <Position Type of Base Coordinate System>

Examples

```
MAIN                          // Declare Main Function Start
    XPOS XRDATA               // Declare a position variable of base coordinate system
    UPOS URDATA               // Declare a position variable of user coordinate system

    XRDATA = <10,10,10,30,45,-  // Assign base coordinate value
    60,ARM = 0,T = 1>
    XRDATA.U = 3              // Assign user coordinate system information

    URDATA = BTOU(XRDATA)     // Acquire position variable of user coordinate system
    PRINT(0, URDATA)          // Store coordinate value in index 0 of monitoring log
    URDATA = BTOU(2,XRDATA)   // Acquire position variable of user coordinate system of
                              // robot 2
    PRINT(1, URDATA)          // Store coordinate value in index 1 of monitoring log
EOP                           // Main Function End
```

Related terms

JTOB, BTOJ, JTOU, UTOJ, UTOB, TOOL, USER

10.8 UTOB (Coordinate Data Calculation, USER → BASE)

Usage

This calculates the position value of base coordinate system from the position value of user coordinate system.

Syntax

〈Variable〉 = UTOB([〈Robot Number〉,] 〈Input Variable〉)

Description

This calculates the position value of base coordinate system from the position value of user coordinate system by referring to user coordinate system information of input position variable.

1) It is possible to select different robot other than the one specified when storing in JOB program by inputting 〈Robot Number〉.
2) A variable format must be entered when inputting the value to 〈Input Variable〉. If a constant is entered, Alarm occurs.

Return value

〈Variable〉	Data type: 〈Position Type of Base Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Input Variable〉	Data type: 〈Position Type of User Coordinate System〉

Examples

```
MAIN                          // Declare Main Function Start
    XPOS XDATA                // Declare a position variable of base coordinate system
    UPOS UDATA                // Declare a position variable of user coordinate system

    UDATA = 〈10,10,10,30,45,- // Assign user coordinate value
    60,ARM = 0,T = 1,U = 2〉

    XDATA = UTOB(UDATA)       // Acquire position variable of base coordinate system
    PRINT(0, XDATA)           // Store coordinate value in index 0 of monitoring log
    XDATA = UTOB(2, UDATA)    // Acquire position variable of base coordinate system of
                              // robot 2
    PRINT(1, XDATA)           // Store coordinate value in index 1 of monitoring log
EOP                           // Main Function End
```

Related terms

JTOB, BTOJ, JTOU, UTOJ, BTOU, TOOL, USER

10.9 RELBASE (Relative move based on Base Coordinate System)

Usage

Calculates Relatively moved position value based on Base Coordinate System information as much as Displacement substituted at Reference Position.

Syntax

〈Variable〉 = RELBASE([〈Robot Number〉,] 〈Reference Position〉, 〈Displacement〉)

Description

Calculates relatively moved result value from Base Coordinate System Reference Position to current Base Coordinate System direction to return it to Base Coordinate System value.

1) Input 〈Robot Number〉 to select another Robot that is not the Robot selected when the JOB Program is saved.
2) To input value in 〈Reference Position〉, be sure to input value in Base Coordinate System Positional Variable type. If Constant or other Variables is entered, an alarm occurs.
3) To input value in 〈Displacement〉, be sure to input value in Positional Variable type. If Constant value is input, an alarm occurs.

Return value

〈Variable〉	Data type: 〈Base Coordinate System Positional Variable〉

Arguments

〈Robot Number〉	Data type: 〈Integer〉 〈Robot Number〉 range is 1 ~ 3.
〈Reference Position〉	Data type: 〈Base Coordinate System Positional Variable〉
〈Displacement〉	Data type: 〈All Coordinate System Positional Variable〉

Examples

```
MAIN                                  // Declares Main Function
    POS ACUR, DATA                    // Declares Joint Coordinate System Positional Variable
    XPOS XDATA1, XDATA2               // Declares Base Coordinate System Positional Variable
    XPOS RDATA1, RDATA2
    DATA = 〈100, 0, 100, 0, 0, -50〉  // institutes displacement

    ACUR = HERE                       // Saves current position value
    XDATA1 = JTOB(ACUR)               // Converts into Base Coordinate System value
    ACUR = HERE(2)                    // Saves current position value of Robot 2
    XDATA2 = JTOB(2,ACUR)             // Converts into Base Coordinate System value of
                                      // robot 2
    RDATA1 = RELBASE(XDATA1,          // Calculates moved result as much as Displacement
    DATA)                             // based on Base Coordinate System
```

```
    PRINT(0, RDATA1)                  // Saves result coordinate value at Monitoring Log 0
    RDATA2 = RELBASE(2,XDATA2,        // Calculates moved result as much as Displacement
    DATA)                             // based on Base Coordinate System of Robot 2
    PRINT(1, RDATA2)                  // Saves result coordinate value at Monitoring Log 1
EOP                                   // Ends Main Function
```

Related terms

RELTOOL, RELUSER

10.10 RELTOOL (TOOL Coordinate System Based Relative Movement)

Usage

This is a function that calculates the relative movement value from the reference position based on the tool coordinate system.

Syntax

〈Variable〉 = RELTOOL([〈Robot Number〉,] 〈Reference Position〉, 〈Movement〉)

Description

This is a function that calculates the relative movement result value toward current tool coordinate system from the reference position of base coordinate system and returns the result with base coordinate system value.
If the tool coordinate system is not declared separately, it performs a relative movement based on tool 0.

1) It is possible to select different robot other than the one specified when storing in JOB program by inputting 〈Robot Number〉.
2) A variable format of base coordinate system must be entered when inputting a value to 〈Reference Position〉. If a constant or other variable are entered, alarm occurs.
3) A position variable format must be entered when inputting a value to 〈Movement〉. If a constant is entered, alarm occurs.

Return value

〈Variable〉	Data type: 〈Position Type of Base Coordinate System〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Reference Position〉	Data type: 〈Position Type of Base Coordinate System〉
〈Movement〉	Data type: 〈All Position Type〉

Examples

```
MAIN                                    // Declare Main Function Start
    POS ACUR, DATA1, DATA2              // Declare position variables of joint coordinate
                                        // system
    XPOS XDATA1, XDATA2                 // Declare position variables of base coordinate
    XPOS RDATA1, RDATA2                 // system

    DATA1 = <100, 0, 100, 0, 0, -50>    // Assign movement
    DATA2 = <-100, 0, -10, 0, 0, -50>
    TOOL 1                              // Select TOOL 1 coordinate system

    ACUR = HERE                         // Store current position value
    XDATA1 = JTOB(ACUR)                 // Convert to base coordinate system value
    ACUR = HERE(2)                      // Store current position value of robot 2
    XDATA2 = JTOB(2,ACUR)               // Convert to current position value of robot 2

    RDATA1 = RELTOOL(XDATA1,            // Calculate moving result by movement based on
    DATA1)                              // tool coordinate system

    PRINT(0, RDATA1)                    // Store coordinate value in index 0 of monitoring log
    RDATA2 = RELTOOL(2, XDATA2,         // Calculate moving result by movement based on
    DATA2)                              // tool coordinate system of robot 2

    PRINT(1,RDATA2)                     // Store coordinate value in index 1 of monitoring log

    IF (RDATA1.3 <RDATA2.3) THEN        // If calculation result is less than Z axis value of
                                        // robot 2
        SVON                            // Servo on
        LMOV RDATA1                     // Move LMOV to RDATA1 from current position
    ENDIF                               // IF statement End
EOP                                     // Main Function End
```

10.11 RELUSER (Relative move based on User Coordinate System)

Usage

Calculates relatively moved position value based on User Coordinate System information as much as Displacement substituted at Reference Position.

Syntax

〈Variable〉 = RELUSER([〈Robot Number〉,] 〈Reference Position〉, 〈Displacement〉)

Description

Calculates relatively moved result value from User Coordinate System Reference Position to designated User Coordinate System direction to return it to User Coordinate System value.
Performs relative move of User Coordinate System by referring to User Coordinate System No of Reference Position.

1) Input 〈Robot Number〉 to select another Robot that is not the Robot selected when the JOB Program is saved.
2) To input value in 〈Reference Position〉, be sure to input value in User Coordinate System Positional Variable type. If Constant or other Variables is entered, an alarm occurs.
3) To input value in 〈Displacement〉, be sure to input value in Positional Variable type. If Constant value is input, an alarm occurs.

Return value

〈Variable〉	Data type: 〈User Coordinate System Positional Variable〉

Arguments

〈Robot Number〉	Data type: 〈Integer〉 〈Robot Number〉 range is 1 ~ 3.
〈Reference Position〉	Data type: 〈User Coordinate System Positional Variable〉
〈Displacement〉	Data type: 〈All Coordinate System Positional Variable〉

Examples

```
MAIN                                    // Declares Main Function
    POS ACUR, DATA1, DATA2              // Declares Joint Coordinate System Positional
                                        // Variable
    UPOS UDATA1,UDATA2                  // Declares User Coordinate System Positional
    UPOS RDATA1, RDATA2                 // Variable

    DATA1 = 〈100, 0, 100, 0, 0, -50〉    // Institutes displacement
    DATA2 = 〈-100, 0, -100, 0, 0, -50〉

    ACUR = HERE                         // Saves current position value
```

```
        ACUR.T = 0                          // Designates Tool Coordinate System No 0
        ACUR.U = 1                          // Designates User Coordinate System No 1
        UDATA1 = JTOU(ACUR)                 // Converts into User Coordinate System value
        ACUR = HERE(2)                      // Saves current position value of Robot 2
        ACUR.T = 1                          // Designates Tool Coordinate System No 1
        ACUR.U = 2                          // Designates User Coordinate System No 2
        UDATA2 = JTOU(2,ACUR)               // Converts into User Coordinate System value
                                            // of robot 2
        RDATA1 = RELUSER(UDATA1, DATA1)     // Calculates moved result as much as
                                            // displacement based on User Coordinate
                                            // System
        PRINT(0, RDATA1)                    // Saves result coordinate value at Monitoring
                                            // Log 0
        RDATA2 = RELUSER(2, UDATA2,         // Calculates moved result as much as
        DATA2)                              // displacement based User Coordinate System
                                            // of robot 2
        PRINT(1,RDATA2)                     // Saves result coordinate value at Monitoring
                                            // Log 1
    EOP                                     // Ends Main Function
```

Related terms

RELBASE, RELTOOL

10.12 GBASEDISP (Displacement detection based on Base Coordinate System)

Usage

Calculates displacement based on Base Coordinate System between entered 2 Coordinates.

Syntax

〈Variable〉 = GBASEDISP([〈Robot Number〉,] 〈Position 1〉, 〈Position 2〉)

Description

Returns displacement between entered 〈Position 1〉 and 〈Position 2〉 to Base Coordinate System reference value.
〈Position 2〉 must be Positional Variable with same type in 〈Position 1〉. In addition, in case of substituting Base Coordinate System Positional Variable, 〈Position 1〉 must be same with Tool Coordinate System Number. In case of substituting User Coordinate System Positional Variable, 〈Position 1〉 must be same with Tool Coordinate System Number and User Coordinate System Number.

1) Input 〈Robot Number〉 to select another Robot that is not the Robot selected when the JOB Program is saved.
2) 〈Position 1〉 can use All Positional Variable Types. If Constant or other Variables is entered, an alarm occurs.
3) Variable Type or Coordinate System Number of 〈Position 1〉 different from those of 〈Position 2〉, an alarm occurs. Hence, be cautious of it.

Return value

〈Variable〉	Data type: 〈Base Coordinate System Positional Variable〉

Arguments

〈Robot Number〉	Data type: 〈Integer〉 〈Robot Number〉 range is 1 ~ 3.
〈Position 1〉	Data type: 〈All Coordinate System Positional Variable〉
〈Position 2〉	Data type: 〈All Coordinate System Positional Variable〉

Examples

Example 1

```
MAIN                                  // Declares Main Function
    POS AP1, AP2                      // Declares Joint Coordinate System Positional
                                      // Variable
    XPOS DELTA                        // Declares Base Coordinate System Positional
                                      // Variable
    AP1 = GP0                         // Substitutes GP0 for AP1
    AP2 = GP1                         // Substitutes GP0 for AP2
    DELTA = GBASEDISP(AP1, AP2)       // Calculates Base Coordinate System between AP1
                                      // and AP2 displacement
```

```
        PRINT(0, DELTA)                 // Saves result value at Monitoring Log 0
    EOP                                 // Ends Main Function
```

Example 2

```
    MAIN                                // Declares Main Function
        XPOS XP1, XP2, DELTA            // Declares Base Coordinate System Positional
                                        // Variable
        GP0.T = 1                       // Designates Tool Coordinate System No 1 of GP0
        XP1 = JTOB(GP0)                 // Converts GP0 value into Base Coordinate System
                                        // value
        GP1.T = 1                       // Designates Tool Coordinate System No 1 of GP1
        XP2 = JTOB(GP1)                 // Converts GP1 value into Base Coordinate System
                                        // value
        DELTA = GBASEDISP(XP1, XP2)     // Calculates Base Coordinate System between XP1
                                        // and XP2 displacement
        PRINT(1,DELTA)                  // Saves result value at Monitoring Log 1
        DELTA = GBASEDISP(1,XP1, XP2)   // Calculates Base Coordinate System between XP1
                                        // and XP2 displacement of Robot 1
        PRINT(2,DELTA)                  // Saves result value at Monitoring Log 2
    EOP                                 // Ends Main Function
```

Example 3

```
    MAIN                                // Declares Main Function
        XPOS DELTA                      // Declares Base Coordinate System Positional
                                        // Variable
        UPOS UP1, UP2                   // Declares User Coordinate System Positional
                                        // Variable
        GP0.T = 1                       // Designates Tool Coordinate System No 1 of GP0
        GP0.U = 1                       // Designates User Coordinate System No 1 of GP0
        UP1 = JTOU(GP0)                 // Converts GP0 value into User Coordinate System
                                        // value
        GP1.T = 1                       // Designates Tool Coordinate System No 1 of GP1
        GP1.U = 1                       // Designates User Coordinate System No 1 of GP1
        UP2 = JTOU(GP1)                 // Converts GP1 value into User Coordinate System
                                        // value
        DELTA = GBASEDISP(UP1,UP2)      // Calculates Base Coordinate System between UP1
                                        // and UP2 displacement
        PRINT(3,DELTA)                  // Saves result value at Monitoring Log 3
        DELTA = GBASEDISP(1,UP1,UP2)    // Calculates Base Coordinate System between UP1
                                        // and UP2 displacement of Robot 1
        PRINT(4,DELTA)                  // Saves result value at Monitoring Log 4
    EOP                                 // Ends Main Function
```

Related terms

GTOOLDISP, GUSERDISP

10.13 GTOOLDISP (Displacement detection based on Tool Coordinate System)

Usage

Calculates displacement based on Tool Coordinate System between 2 Coordinates.

Syntax

〈Variable〉 = GTOOLDISP([〈Robot Number〉,] 〈Position 1〉, 〈Position 2〉)

Description

Returns displacement between entered 〈Position 1〉 and 〈Position 2〉 to Tool Coordinate System reference value.
〈Position 2〉 must be Positional Variable with same type in 〈Position 1〉. In addition, in case of substituting Base Coordinate System Positional Variable, 〈Position 1〉 must be same with Tool Coordinate System Number. In case of substituting User Coordinate System Positional Variable, 〈Position 1〉 must be same with Tool Coordinate System Number and User Coordinate System Number.

1) Input 〈Robot Number〉 to select another Robot that is not the Robot selected when the JOB Program is saved.
2) 〈Position 1〉 can use All Positional Variable Types. If Constant or other Variables is entered, an alarm occurs.
3) Variable Type or Coordinate System Number of 〈Position 1〉 different from those of 〈Position 2〉, an alarm occurs. Hence, be cautious of it.

Return value

〈Variable〉	Data type: 〈Base Coordinate System Positional Variable〉

Arguments

〈Robot Number〉	Data type: 〈Integer〉 〈Robot Number〉 range is 1 ~ 3.
〈Position 1〉	Data type: 〈All Coordinate System Positional Variable〉
〈Position 2〉	Data type: 〈All Coordinate System Positional Variable〉

Examples

Example 1

```
MAIN                              // Declares Main Function
    POS AP1, AP2                  // Declares Joint Coordinate System Positional Variable
    XPOS DELTA                    // Declares Base Coordinate System Positional Variable
    AP1 = GP0                     // Substitutes GP0 for AP1
    AP2 = GP1                     // Substitutes GP0 for AP2
    DELTA = GTOOLDISP(AP1, AP2)   // Calculates Tool Coordinate System between AP1
                                  // and AP2 displacement
    PRINT(0, DELTA)               // Saves result value at Monitoring Log 0
EOP                               // Ends Main Function
```

Example 2

```
MAIN                                  // Declares Main Function
    XPOS XP1, XP2, DELTA              // Declares Base Coordinate System Positional Variable
    GP0.T = 1                         // Designates Tool Coordinate System No 1 of GP0
    XP1 = JTOB(GP0)                   // Converts GP0 value into Base Coordinate System
                                      // value
    GP1.T = 1                         // Designates Tool Coordinate System No 1 of GP1
    XP2 = JTOB(GP1)                   // Converts GP1 value into Base Coordinate System
                                      // value
    DELTA = GTOOLDISP(XP1, XP2)       // Calculates Tool Coordinate System between XP1
                                      // and XP2 displacement
    PRINT(1,DELTA)                    // Saves result value at Monitoring Log 1
    DELTA = GTOOLDISP(1,XP1,          // Calculates Tool Coordinate System between XP1
    XP2)                              // and XP2 displacement of Robot 1

    PRINT(2,DELTA)                    // Saves result value at Monitoring Log 2
EOP                                   // Ends Main Function
```

Example 3

```
MAIN                                  // Declares Main Function
    XPOS DELTA                        // Declares Base Coordinate System Positional Variable
    UPOS UP1, UP2                     // Declares User Coordinate System Positional Variable
    GP0.T = 1                         // Designates Tool Coordinate System No 1 of GP0
    GP0.U = 1                         // Designates User Coordinate System No 1 of GP0
    UP1 = JTOU(GP0)                   // Converts GP0 value into User Coordinate System
                                      // value
    GP1.T = 1                         // Designates Tool Coordinate System No 1 of GP1
    GP1.U = 1                         // Designates User Coordinate System No 1 of GP1
    UP2 = JTOU(GP1)                   // Converts GP1 value into User Coordinate System
                                      // value
    DELTA = GTOOLDISP(UP1,UP2)        // Calculates Tool Coordinate System between UP1
                                      // and UP2 displacement
    PRINT(3,DELTA)                    // Saves result value at Monitoring Log 3
    DELTA = GTOOLDISP(1,UP1,UP2)      // Calculates Tool Coordinate System between UP1
                                      // and UP2 of Robot 1 displacement
    PRINT(4,DELTA)                    // Saves result value at Monitoring Log 4
EOP                                   // Ends Main Function
```

Related terms

GBASEDISP, GUSERDISP

10.14 GUSERDISP (Displacement detection based on User Coordinate System)

Usage

Calculates displacement based on entered User Coordinate System between 2 Coordinates.

Syntax

〈Variable〉 = GUSERDISP([〈Robot Number〉,] 〈Position 1〉, 〈Position 2〉)

Description

Returns displacement between entered 〈Position 1〉 and 〈Position 2〉 to User Coordinate System reference value.
〈Position 2〉 must be Positional Variable with same type in 〈Position 1〉. In addition, in case of substituting Base Coordinate System Positional Variable, 〈Position 1〉 must be same with Tool Coordinate System Number. In case of substituting User Coordinate System Positional Variable, 〈Position 1〉 must be same with Tool Coordinate System Number and User Coordinate System Number.

1) Input 〈Robot Number〉 to select another Robot that is not the Robot selected when the JOB Program is saved.
2) 〈Position 1〉 can use All Positional Variable Types. If Constant or other Variables is entered, an alarm occurs.
3) Variable Type or Coordinate System Number of 〈Position 1〉 different from those of 〈Position 2〉, an alarm occurs. Hence, be cautious of it.

Return value

〈Variable〉	Data type: 〈User Coordinate System Positional Variable〉

Arguments

〈Robot Number〉	Data type: 〈Integer〉 〈Robot Number〉 range is 1 ~ 3.
〈Position 1〉	Data type: 〈All Coordinate System Positional Variable〉
〈Position 2〉	Data type: 〈All Coordinate System Positional Variable〉

Examples

Example 1

```
MAIN                                // Declares Main Function
    POS AP1, AP2                    // Declares Joint Coordinate System Positional Variable
    UPOS DELTA                      // Declares User Coordinate System Positional Variable
    AP1 = GP0                       // Substitutes GP0 for AP1
    AP2 = GP1                       // Substitutes GP0 for AP2
    DELTA = GUSERDISP(AP1, AP2)     // Calculates User Coordinate System displacement
                                    // between GP0 and GP1
    PRINT(0, DELTA)                 // Saves result value at Monitoring Log 0
EOP                                 // Ends Main Function
```

Example 2

```
MAIN                                  // Declares Main Function
    XPOS XP1, XP2                     // Declares Base Coordinate System Positional Variable
    UPOS DELTA                        // Declares User Coordinate System Positional Variable
    GP0.T = 1                         // Designates Tool Coordinate System No 1 of GP0
    XP1 = JTOB(GP0)                   // Converts GP0 value into Base Coordinate System
                                      // value
    GP1.T = 1                         // Designates Tool Coordinate System No 1 of GP1
    XP2 = JTOB(GP1)                   // Converts GP1 value into Base Coordinate System
                                      // value
    DELTA = GUSERDISP(XP1, XP2)       // Calculates User Coordinate System between XP1
                                      // and XP2 displacement
    PRINT(1,DELTA)                    // Saves result value at Monitoring Log 1
EOP                                   // Ends Main Function
```

Example 3

```
MAIN                                  // Declares Main Function
    UPOS UP1, UP2, DELTA              // Declares User Coordinate System Positional Variable
    GP0.T = 1                         // Designates Tool Coordinate System No 1 of GP0
    GP0.U = 1                         // Designates User Coordinate System No 1 of GP0
    UP1 = JTOU(GP0)                   // Converts GP0 value into User Coordinate System
                                      // value
    GP1.T = 1                         // Designates Tool Coordinate System No 1 of GP1
    GP1.U = 1                         // Designates User Coordinate System No 1 of GP1
    UP2 = JTOU(GP1)                   // Converts GP1 value into User Coordinate System
                                      // value
    DELTA = GUSERDISP(UP1,UP2)        // Calculates User Coordinate System between UP1
                                      // and UP2 displacement
    PRINT(2,DELTA)                    // Saves result value at Monitoring Log 2
EOP                                   // Ends Main Function
```

Related terms

GBASEDISP, GTOOLDISP

10.15 ABS (Absolute Value)

Usage

This is a function that returns the absolute value of input value.

Syntax

〈Variable〉 = ABS(〈Input Value〉)

Description

This returns a size of displacement from the origin to the input value. That is, if an integer format is entered, it returns a positive (+) Integer value and if a real number format is entered, it returns a positive (+) real number value.

1) Constants or variables in either an integer or a real number format can be entered in the 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Integer or Real Number Type〉 This returns a size of displacement.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA,BB	// Declare integer variables
	AA = -5	// Initialize the integer variable AA to -5
	I0 = ABS(-5)	// Store 5 to global integer variable I0
	BB = ABS(AA)	// Store 5 to integer variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA =-1.5	// Initialize the integer variable AA to -1.5
	F0 = ABS(-1.5)	// Store 1.5 to global real number variable F0
	BB = ABS(AA)	// Store 1.5 to real number variable BB

10.16 DEG (Conversion of Radian to Angle)

Usage

This is a function that converts Radian angle (Circular Method) to Degree angle (Hexadecimal Method).

Syntax

〈Variable〉 = DEG(〈Input Value〉)

Description

This is a function that converts Radian angle to Degree angle.
Below is the angular conversion formula.

$$degree = (radian) \times \left(\frac{180^{\circ}}{\pi}\right)$$

1) Radian angle (Circular Method) is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is angle (°).

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA                  // Declare an integer variable
REAL BB                 // Declare a real number variable
AA = 1                  // Assign Radian angle(1) to integer variable AA
F0 = DEG(1)             // Convert to Degree angle(57.29,578) using constant(1) and store it
BB = DEG(AA)            // Convert to Degree angle(57.29,578) using variable AA and store it
```

Example 2

```
REAL AA, BB             // Declare real number variables
AA = 3.1416             // Assign Radian angle(3.1416) to real number variable AA
F0 = DEG(3.1416)        // Convert to Degree angle(180) using constant(3.1416) and store it
BB = DEG(AA)            // Convert to Degree angle(180) using variable AA and store it
```

Related terms

RAD

※ **Circular Method and Hexadecimal Method of Major Angles**

Hexadecimal Method	0°	30°	45°	60°	90°	180°	270°	360°
Circular Method	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	π	$\frac{3}{2}\pi$	2π

10.17 RAD (Conversion of Angle to Radian)

Usage

This is a function that converts Degree angle (Hexadecimal Method) to Radian angle (Circular Method).

Syntax

〈Variable〉 = RAD(〈Input Value〉)

Description

This is a function that converts Radian angle to Degree angle.
Below is the Radian conversion formula.

$$radian = (degree) \times \left(\frac{\pi}{180^{\circ}}\right)$$

1) Degree angle (Hexadecimal value) can be entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is Radian (rad).

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA              // Declare an integer variable
REAL BB             // Declare a real number variable
AA = 180            // Assign Angle(180) to integer variable AA
F0 = RAD(180)       // Convert to Radian angle(3.14,159) using constant(180) and store it
BB = RAD(AA)        // Convert to Radian angle(3.14,159) using variable AA and store it
```

Example 2

```
REAL AA, BB         // Declare real number variables
AA = 180.0          // Assign Angle(180.0) to real number variable AA
F0 = RAD(180.0)     // Convert to Radian angle(3.14,159) using constant(180.0) and store it
BB = RAD(AA)        // Convert to Radian angle(3.14,159) using variable AA and store it
```

Related terms

DEG

※ **Circular Method and Hexadecimal Method of Major Angles**

Hexadecimal Method	0°	30°	45°	60°	90°	180°	270°	360°
Circular Method	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	π	$\frac{3}{2}\pi$	2π

10.18 POW (Exponential Function)

Usage

This is a function to obtain the power value.

Syntax

〈Variable〉 = POW(〈Value of Base Number〉, 〈Exponential Value〉)

Description

This is a function that calculates the power (x^y) value.
For example, this returns the $2^5 = 32$ value when calculating the 5^{th} power of 2.

1) The base number of the power is entered in 〈Value of Base Number〉.
2) The exponential value is entered in 〈Exponential Value〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result of power is returned. If the base number is below 0 and the exponential value is not the integer numbers, it returns Nan (Not-A-Number). If the base number is 0 and the exponential value is below 0, it returns infinity (Inf).

Arguments

〈Base Number Value〉	Data type: 〈Integer or Real Number Type〉
〈Exponential Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA, BB              // Declare integer variables
REAL CC                 // Declare a real number variable
AA = 2                  // Assign 2 for the base number
BB = 5                  // Assign 5 for exponential value
F0 = POW(2, 5)          // Store integer constant (2^5) to F0
CC = POW(AA, BB)        // Store integer variable (AA^BB) to CC
```

Example 2

```
REAL AA, BB, CC         // Declare real number variables
AA = 2.0                // Assign 2.0 for the base number
BB = 5.0                // Assign 5.0 for exponential value
F0 = POW(2.0, 5.0)      // Store real number constant (2^5) to F0
CC = POW(AA, BB)        // Store real number variable (AA^BB) to CC
```

Related terms

SQRT

10.19 RND (Round-off)

Usage

This rounds off the fixed point part and returns with the integer value.

Syntax

〈Variable〉 = RND(〈Input Value〉)

Description

This rounds off the fixed point part and returns with the integer format.
For instance, if 12.8 is entered, then it returns 13.
The below is a conversion formula.

$$result = (input\ value) + 0.5$$

1) The target to be rounded off is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉 The result value of round-off is returned.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA, BB	// Declare integer variables
	AA = 12	// Assign 12 to integer variable
	I0 = RND(12)	// If integer constant(12) is entered, 12 is returned
	BB = RND(AA)	// If integer variable AA is entered, 12 is returned
Example 2	REAL AA	// Declare a real number variable
	INT BB	// Declare an integer variable
	AA = 12.5	// Assign 12.5 to real number variable
	I0 = RND(12.5)	// If real number constant(12.5) is entered, 13 is returned
	BB = RND(AA)	// If real number variable AA is entered, 13 is returned
Example 3	REAL AA	// Declare a real number variable
	AA = RND(49.7/0.125)	// The round-off value of result of equation is returned

10.20 EXP (Exponential e^x)

Usage

This returns the power of irrational number e^x. This is an inverse function of Logarithm Function.

Syntax

〈Variable〉 = EXP(〈Input Value〉)

Description

This is a function to obtain Exponential value.
Irrational number $e = 2.718,281,828,459,045,235,360,287,471,3$ ….
For instance, in case of $\mathrm{EXP}(1) = 2.718,282$,
$\mathrm{EXP}(2) = 2.718,282 * 2.718,282 = 7.389,056$ and
$\mathrm{EXP}(3) = 2.718,282 * 2.718,282 * 2.718,282 = 20.08,554$.

1) The exponential x of irrational number e^x is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 This returns the result value of irrational number e^x.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 2	// Assign 2 to the integer variable
	F0 = EXP(2)	// Store the result of e^2 to global variable F0
	BB = EXP(AA)	// Store the result of $e^{(AA=2)}$ to BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 2.0	// Assign 2 to the real number variable
	F0 = EXP(2.0)	// Store the result of $e^{2.0}$ to F0
	BB = EXP(AA)	// Store the result of $e^{(AA=2.0)}$ to BB

Related terms

POW, LN, LOG

10.21 LN (Natural Logarithm)

Usage

This returns the result value of Natural Logarithm $\log_e x$.

Syntax

〈Variable〉 = LN(〈Input Value〉)

Description

This is a function to obtain Natural Logarithm value.
For instance,$\mathrm{LN}(5) = 1.704{,}748$.
The below is a conversion formula.

$$y = \ln x = \log_e x$$

1) Input x of Natural Logarithm is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 This returns the result value y.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 15	// Assign 15 to integer variable
	F0 = LN(15)	// Store the value of $\log_e x$ (2.708) to global variable F0
	BB = LN(AA)	// Store the value of $\log_e x$ (2.708) to real number variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 15.0	// Assign 15.0 to real number variable
	F0 = LN(15.0)	// Store the value of $\log_e x$ (2.708) to global variable F0
	BB = LN(AA)	// Store the value of $\log_e x$ (2.708) to real number variable BB

Related terms

POW, EXP, LOG

10.22 LOG (Common Logarithm)

Usage

This returns the result value of the Common Logarithm $\log_{10} x$.

Syntax

〈Variable〉 = LOG(〈Input Value〉)

Description

This is a function to obtain Common Logarithm value.
For instance, $\mathrm{LOG}(10) = 1.0$, $\mathrm{LOG}(100) = 2.0$, $\mathrm{LOG}(1000) = 3.0$.
The below is a conversion formula.

$$y = \log_{10} x$$

1) Input x of Common Logarithm is entered in 〈Input Value〉

Return value

〈Variable〉	Data type: 〈Real Number Type〉 This returns the result value y.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA                  // Declare an integer variable
REAL BB                 // Declare a real number variable
AA = 100                // Assign 100 to integer variable
F0 = LOG(100)           // Store the result value of log10 x (2.0) to variable F0
BB = LOG(AA)            // Store the result value of log10 x (2.0) to variable BB
```

Example 2

```
REAL AA, BB             // Declare real number variables
AA = 100.0              // Assign 100.0 to real number variable
F0 = LOG(100.0)         // Store the result value of log10 x (2.0) to variable F0
BB = LOG(AA)            // Store the result value of log10 x (2.0) to variable BB
```

Related terms

POW, EXP, LN

10.23 SQRT (Square Root)

Usage

This returns the Square Root.

Syntax

〈Variable〉 = SQRT(〈Input Value〉)

Description

This returns the square root (y) of input value(x).
For instance,
$\mathrm{SQRT}(1) = 1.00{,}000,\ \mathrm{SQRT}(2) = 1.41.421,\ \mathrm{SQRT}(3) = 1.73{,}205,\ \mathrm{SQRT}(4) = 2.00{,}000.$
The below is a conversion formula.

$y = \sqrt{x}$

1) The input value x of the square root formula is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 This returns the result value y. If the input value is a negative (-), it returns Nan (Not-A-Number).

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 4	// Assign 4 to integer variable
	F0 = SQRT(4)	// Store result value of $\sqrt{4}$ 2 to global variable F0
	BB = SQRT(AA)	// Store result value of $\sqrt{(AA = 4)}$ to variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 4.0	// Assign 4.0 to real number variable
	F0 = SQRT(4.0)	// Store result value of $\sqrt{4}$ 2 to global variable F0
	BB = SQRT(AA)	// Store result value of $\sqrt{(AA = 4)}$ to variable BB

Related terms

POW

10.24 SIN (Sine Function)

Usage

This returns the result value of the Sine Function.

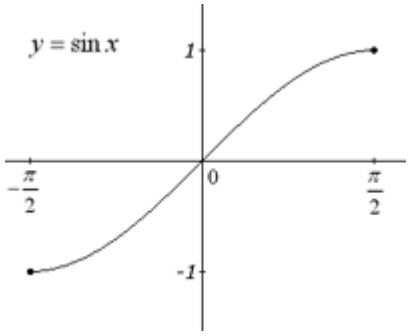
Syntax

〈Variable〉 = SIN(〈Input Value〉)

Description

The result value y of the $\sin(x)$ is returned.

$y = \sin(x)$



For instance, in special angles of Trigonometric Function;
$\mathrm{SIN}(\mathrm{RAD}(0)) = 0.0$,
$\mathrm{SIN}(\mathrm{RAD}(30)) = 0.5$,
$\mathrm{SIN}(\mathrm{RAD}(45)) = 0.70{,}711$,
$\mathrm{SIN}(\mathrm{RAD}(60)) = 0.86{,}603$,
$\mathrm{SIN}(\mathrm{RAD}(90)) = 1$.

1) The x value of Sine Function is entered in 〈Input Value〉. (x unit: $radian$)

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result value y is returned. The range of result value is $-1 \le y \le 1$.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 30	// Assign Degree angle(30) to the integer variable
	F0 = SIN(RAD(30))	// Store result value of $\sin(30^\circ) = 0.5$ to F0
	BB = SIN(RAD(AA))	// Store result value to real number variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 30.0	// Assign Degree angle(30) to the real number variable
	F0 = SIN(RAD(30.0))	// Store result value of $\sin(30^\circ) = 0.5$ to F0
	BB = SIN(RAD(AA))	// Store result value to real number variable BB

Related terms

ASIN, COS, ACOS, TAN, ATAN, ATAN2

10.25 ASIN (Arc Sine Function)

Usage

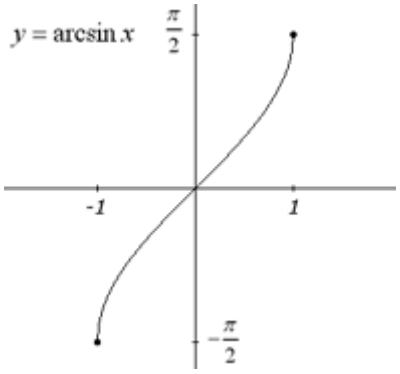
This returns the result value of Arc Sine Function.

Syntax

〈Variable〉 = ASIN(〈Input Value〉)

Description

The $\text{asin}(x)$ function is an inverse function of $\sin(x)$ function and it returns the asin value that is within $-\frac{\pi}{2} \sim \frac{\pi}{2}$ radian.
If x is smaller than -1 or larger than 1, the asin returns the infinity value.



For instance,
$\text{ASIN}(-1) = -1.5708$,
$\text{ASIN}(0) = 0.0$,
$\text{ASIN}(1) = 1.5708$.

1) The x value of Arc Sine is entered in 〈Input Value〉. $(-1 \le x \le 1)$

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result value y is returned. The range of result value is $-\frac{\pi}{2} \le y \le \frac{\pi}{2}$.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉 The Input Value range is $-1 \le x \le 1$.

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 1	// Assign 1 to integer variable
	F0 = ASIN(1)	// Store result value of $\text{asin}(1) = 1.5708$ to F0
	BB = ASIN(AA)	// Store result value to real number variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 1.0	// Assign 1.0 to real number variable
	F0 = ASIN(1.0)	// Store result value of $\text{asin}(1) = 1.5708$ to F0
	BB = ASIN(AA)	// Store result value to real number variable BB

Related terms

SIN, COS, ACOS, TAN, ATAN, ATAN2

10.26 COS (CoSine Function)

Usage

This returns the result value of Cosine Function.

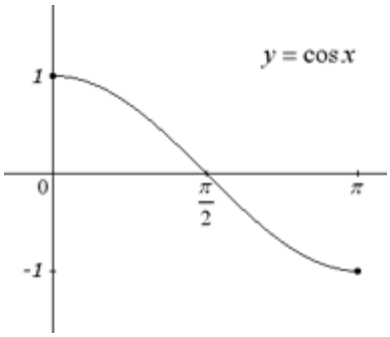
Syntax

〈Variable〉 = COS(〈Input Value〉)

Description

The result value of $\cos(x)$ function is returned.

$y = \cos(x)$



For instance, in special angles of Trigonometric Function;
$\mathrm{COS}(\mathrm{RAD}(0)) = 1.0$,
$\mathrm{COS}(\mathrm{RAD}(30)) = 0.86{,}603$,
$\mathrm{COS}(\mathrm{RAD}(45)) = 0.70{,}711$,
$\mathrm{COS}(\mathrm{RAD}(60)) = 0.5$.
$\mathrm{COS}(\mathrm{RAD}(90)) = 0.0$.

1) The x value of Cosine Function is entered in 〈Input Value〉. (x $unit$: $radian$)

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result value y is returned. The range of result value is $-1 \le y \le 1$.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA
	REAL BB
	AA = 60
	F0 = COS(RAD(60))
	BB = COS(RAD(AA))
		// Declare an integer variable
		// Declare a real number variable
		// Assign Degree angle(60) to integer variable
		// Store result value of $\cos(60^\circ) = 0.5$ to F0
		// Store result value to real number variable BB

Example 2	REAL AA, BB
	AA = 60.0
	F0 = COS(RAD(60.0))
	BB = COS(RAD(AA))
		// Declare real number variables
		// Assign Degree angle(60) to real number variable
		// Store result value of $\cos(60^\circ) = 0.5$ to F0
		// Store result value to real number variable BB

Related terms

SIN, ASIN, ACOS, TAN, ATAN, ATAN2

10.27 ACOS (Arc CoSine Function)

Usage

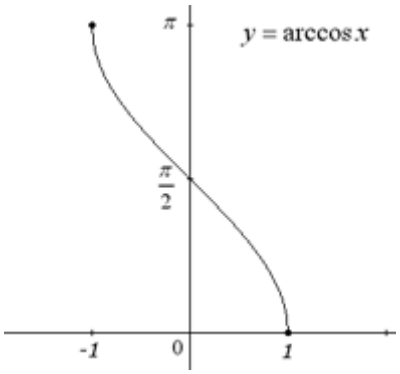
This returns the result value of Arc Cosine Function.

Syntax

〈Variable〉 = ACOS(〈Input Value〉)

Description

The $\text{acos}(x)$ is the inverse function of $\cos(x)$ function and it returns the acos value that is within $0 \sim \pi$ radian.
If x is smaller than -1 or larger than 1, the acos returns the infinity value.



For instance,
$\text{ACOS}(-1) = 3.14{,}159.$
$\text{ACOS}(0) = 1.5708.$
$\text{ACOS}(1) = 0.0.$

1) The x value of Arc Cosine Function is entered in 〈Input Value〉. $(-1 \le x \le 1)$

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result value y is returned. The range of result value is $0 \le y \le \pi$.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉 The range of Input Value is $-1 \le x \le 1$.

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 0	// Assign 0 to integer variable
	F0 = ACOS(0)	// Store result value of $\text{acos}(0) = 1.5708$ to F0
	BB = ACOS(AA)	// Store result value to real number variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 0.0	// Assign 0.0 to real number variable
	F0 = ACOS(0.0)	// Store result value of $\text{acos}(0) = 1.5708$ to F0
	BB = ACOS(AA)	// Store result value to real number variable BB

Related terms

SIN, ASIN, COS, TAN, ATAN, ATAN2

10.28 TAN (Tangent Function)

Usage

This returns the result value of Tangent Function.

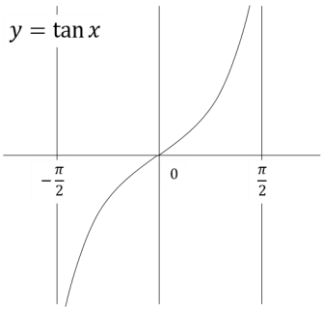
Syntax

〈Variable〉 = TAN(〈Input Value〉)

Description

The result value of $\tan(x)$ function is returned.

$y = \tan(x)$



For instance, in special angles of Trigonometric Function;
$\mathrm{TAN}(\mathrm{RAD}(0)) = 0.0$, $\mathrm{TAN}(\mathrm{RAD}(30)) = 0.57{,}735$,
$\mathrm{TAN}(\mathrm{RAD}(45)) = 1.0$, $\mathrm{TAN}(\mathrm{RAD}(60)) = 1.73{,}205$,
$\mathrm{TAN}(\mathrm{RAD}(90)) = *$ (Exceed of Expression Range)

1) The x value of Tangent Function is entered in 〈Input Value〉. (x unit: $radian$)

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result value y is returned.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 45	// Assign Degree angle(45) to integer variable
	F0 = TAN(RAD(45))	// Store result value of $\tan(45^\circ) = 1.0$ to F0
	BB = TAN(RAD(AA))	// Store result value to real number variable BB
Example 2	REAL AA, BB	// Declare real number variables
	AA = 45.0	// Assign Degree angle(45) to real number variable
	F0 = TAN(RAD(45.0))	// Store result value of $\tan(45^\circ) = 1.0$ to F0
	BB = TAN(RAD(AA))	// Store result value to real number variable BB

Related terms

SIN, ASIN, COS, ACOS, ATAN, ATAN2

1) In case of diverging to the infinity value like the $\tan(90^\circ)$, it is difficult to quantitatively express the data value on the controller because data value is too big. Thus, pay attention that it is expressed "*" at teaching pendant.

10.29 ATAN (Arc Tangent Function)

Usage

This returns the result value of Arc Tangent Function.

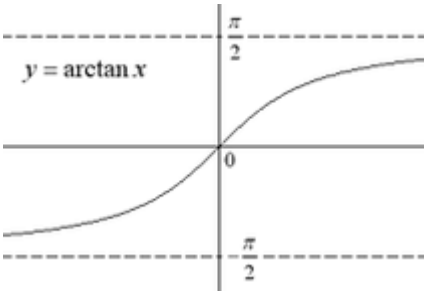
Syntax

〈Variable〉 = ATAN(〈Input Value〉)

Description

The $\mathrm{atan}(x)$ is the inverse function of $\tan(x)$ function and it returns the atan value.

For instance,
$\mathrm{ATAN}(-1) = -0.7854$,
$\mathrm{ATAN}(0) = 0.0$,
$\mathrm{ATAN}(1) = 0.7854$.



1) The x value of Arc Tangent Function is entered in 〈Input Value〉. (x unit: $radian$)

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The result value y is returned. The range of result value is $-\frac{\pi}{2} \le y \le \frac{\pi}{2}$.

Arguments

〈Input Value〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1	INT AA	// Declare an integer variable
	REAL BB	// Declare a real number variable
	AA = 1	// Assign 1 to integer variable
	F0 = ATAN(1)	// Store result value of $\mathrm{atan}(1) = 0.7854$ to F0
	BB = ATAN(AA)	// Store result value to real number variable BB
Example 2	REAL AA, BB	// Declare a real number variable
	AA = 1.0	// Assign 1.0 to real number variable
	F0 = ATAN(1.0)	// Store result value of $\mathrm{atan}(1) = 0.7854$ to F0
	BB = ATAN(AA)	// Store result value to real number variable BB

Related terms

SIN, ASIN, COS, ACOS, TAN, ATAN2

10.30 ATAN2 (The 2nd Arc Tangent Function)

Usage

This returns the result value of the 2nd Arc Tangent Function.

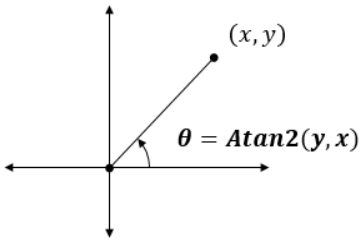
Syntax

〈Return Value〉 = ATAN2(〈Y Ratio〉, 〈X Ratio〉)

Description

The result value of $\text{atan2}(x)$ function is returned.

$\text{y} = \text{atan2}(x)$



For instance,
$\text{ATAN2}(1.0, 1.0) = 0.7854$,
$\text{ATAN2}(2.0, 1.0) = 1.10{,}715$,
$\text{ATAN2}(1.0, 2.0) = 0.46{,}365$.

1) The height(y axis) of a right triangle is entered in 〈Y Ratio〉.
2) The width (x axis) of a right triangle is entered in 〈X Ratio〉.

Return value

〈Variable〉 Data type: 〈Real Number Type〉
The result value y is returned.
The range of result value is $-\pi \leq y \leq \pi$.

Arguments

〈Y Ratio〉 Data type: 〈Integer or Real Number Type〉

〈X Ratio〉 Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA, BB                  // Declare integer variables
REAL CC                     // Declare a real number variable
AA = 1                      // Assign 1 to integer variable
BB = 1                      // Assign 1 to integer variable
F0 = ATAN2(1,1)             // Store result value of atan2(1,1) = 0.7854 to F0
CC = ATAN2(AA,BB)           // Store result value to real number variable CC
```

Example 2

```
REAL AA, BB,CC              // Declare real number variables
AA = 1.0                    // Assign 1.0 to real number variable
BB = 1.0                    // Assign 1.0 to real number variable
F0 = ATAN2(1.0,1.0)         // Store result value of atan2(1,1) = 0.7854 to F0
CC = ATAN2(AA,BB)           // Store result value to real number variable CC
```

Related terms

SIN, ASIN, COS, ACOS, TAN, ATAN

10.31 MIN (Minimum Value)

Usage

This returns the minimum value after comparing two input values.

Syntax

〈Variable〉 = MIN(〈Input Value 1〉, 〈Input Value 2〉)

Description

This compares 〈input value 1〉 and 〈input value 2〉, then returns the minimum value with real number format.

1) The value 1 for comparing the minimum value is entered in 〈Input Value 1〉.
2) The value 2 for comparing the minimum value is entered in 〈Input Value 2〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The minimum value is returned as a result after comparing two input values.

Arguments

〈Input Value1〉	Data type: 〈Integer or Real Number Type〉
〈Input Value2〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA, BB                  // Declare integer variables
REAL CC                     // Declare a real number variable
AA = 2                      // Assign 2 to Integer Comparison Variable 1
BB = 5                      // Assign 5 to Integer Comparison Variable 2
F0 = MIN(2,5)               // Store returned minimum value 2.0 to F0
CC = MIN(AA,BB)             // Store returned minimum value 2.0 to variable CC
```

Example 2

```
REAL AA, BB, CC             // Declare real number variables
AA = 2.0                    // Assign 2.0 to Real Number Comparison Variable 1
BB = 2.5                    // Assign 5.0 to Real Number Comparison Variable 2
F0 = MIN(2.0,2.5)           // Store returned minimum Value 2.0 to F0
CC = MIN(AA,BB)             // Store returned minimum Value 2.0 to variable CC
```

Related terms

MAX

10.32 MAX (Maximum Value)

Usage

This returns the maximum value after comparing two input values.

Syntax

〈Variable〉 = MAX(〈Input Value 1〉, 〈Input Value 2〉)

Description

This compares 〈input value 1〉 and 〈input value 2〉, then returns the maximum value with real number format.

1) The value 1 for comparing the maximum value is entered in 〈Input Value 1〉.
2) The value 2 for comparing the maximum value is entered in 〈Input Value 2〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The maximum value is returned as a result after comparing two input values.

Arguments

〈Input Value1〉	Data type: 〈Integer or Real Number Type〉
〈Input Value2〉	Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
INT AA, BB                  // Declare integer variables
REAL CC                     // Declare a real number variable
AA = 2                      // Assign 2 to Integer Comparison Variable 1
BB = 5                      // Assign 5 to Integer Comparison Variable 2
F0 = MAX(2,5)               // Store returned maximum 5.0 to F0
CC = MAX(AA,BB)             // Store returned maximum 5.0 to variable CC
```

Example 2

```
REAL AA, BB, CC             // Declare real number variables
AA = 2.0                    // Assign 2.0 to Real Number Comparison Variable 1
BB = 2.5                    // Assign 2.5 to Real Number Comparison Variable 2
F0 = MAX(2.0,2.5)           // Store returned Maximum Value 2.5 to F0
CC = MAX(AA,BB)             // Store returned Maximum Value 2.5 to variable CC
```

Related terms

MIN

10.33 RAND (Random Number Creation)

Usage

This creates the random number values.

Syntax

〈Variable〉 = RAND()

Description

The random number function creates the random number from 0 to RAND_MAX. The RAND_MAX value is 2,147,483,647 in 32 bits Linux system. That is, it is the max value of positive integer variable.

- Random Number Creation Method from 1 to 10: RAND()%10+1
- Random Number Creation Method from 1 to 100: RAND()%100+1
- Random Number Creation Method from 1 to 1,000: RAND()%1000+1

Return value

〈Variable〉	Data type: 〈Integer Type〉 The range of result value is 0 ~ RAND_MAX.

Examples

Example 1

```
REAL AA, BB                 // Declare real number variables
WHILE 1                     // Endless Loop Start
    AA = RAND()             // Store Random Value(0 ~ 7FFFFFFF) to AA
    BB = AA%100+1           // Create Random Value between 1 ~ 100 and store it
    DLAY 100                // 0.1 second Time Delay
ENDWL                       // Endless Loop End
```

Example 2

```
INT AA                      // Declare an integer variable
AA = ((RAND())%100)+1       // Create Random Value between 1 ~ 100 and store it
```

10.34 ASC (Returning of First Character Code in String)

Usage

This returns the integer value of character code corresponding to the first character in string.

Syntax

〈Variable〉 = ASC(〈String〉)

Description

This returns the first character of input string.

Character code refers to a number (integer) corresponding to a character.
Ex) Character “A” is an integer (65) and character “B” is an integer (66).
Refer to ASCII code table.

1) The string to detect the first character is entered in 〈String〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉 The range of result value is 0 ~ 127.

Arguments

〈String〉	Data type: 〈String Type〉

Examples

Example 1

```
MAIN                                // Declare Main Function Start
    I0 = ASC(”STATEMENT”)           // Convert first character “S” into ASCII Code(83)
EOP                                 // Main Function End
```

Example 2

```
MAIN                                // Declare Main Function Start
    DEFSTR STR                      // Declare a string variable
    INT VAL                         // Declare an integer variable
    STR = ”STATEMENT”               // Assign “STATEMENT” to string STR
    VAL = ASC(STR)                  // Convert first character “S” into ASCII Code(83)
EOP                                 // Main Function End
```

Related terms

CHR

10.35 BINS (Conversion of Binary Number to String)

Usage

This is a function that converts the input integer value into binary string.

Syntax

〈Variable〉 = BINS(〈Input Value〉)

Description

This converts the input integer value into binary and returns with 32 bytes string.

1) The integer value to be altered into binary string is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈String Type〉 Binary string for input value is returned.

Arguments

〈Input Value〉	Data type: 〈Integer Type〉 The available number for input is -2^{31}(-2,147,483,648) ~ 2^{31}(2,147,483,647).

Examples

Example 1

```
MAIN                    // Declare Main Function Start
    DEFSTR STR1,STR2    // Declare string variables
    INT AA              // Declare an integer variable
    AA = 16             // Initialize the integer variable
    STR1 = BINS(16)     // Store string  “00000000000000000000000000010000”
    STR2 = BINS(AA)     // that is binary value of 16 to STR variable
EOP                     // Main Function End
```

Example 2

```
MAIN                    // Declare Main Function Start
    DEFSTR STR1,STR2    // Declare string variables
    REAL AA             // Declare a real number variable
    AA = 32.0           // Initialize the real number variable
    STR1 = BINS(32.0)   // Store string  “00000000000000000000000000010000”
    STR2 = BINS(AA)     // that is binary value of 16 to STR Variable
EOP                     // Main Function End
```

Related terms

FTOS, HTOS, SVAL

10.36 CHR (Conversion of Integer into ASCII Text)

Usage

This returns ASCII code character corresponding to input integer number.

Syntax

〈Variable〉 = CHR(〈Input Value〉)

Description

This returns ASCII code character corresponding to input integer number.

1) The integer number (0 ~ 127) within ASCII code is entered in 〈Input Value〉.

Return value

〈Variable〉	Data type: 〈String Type〉

Arguments

〈Input Value〉	Data type: 〈Integer Type〉

Examples

Example 1

```
MAIN                    // Declare Main Function Start
    DEFSTR STRA,STRB    // Declare string variables
    INT AA              // Declare an integer variable
    AA = 66             // Initialize the integer variable
    STRA = CHR(65)      // Assign “A” Corresponding to ASCII Code(65) to STRA
    STRB = CHR(AA)      // Assign “B” Corresponding to ASCII Code(66) to STRA
EOP                     // Main Function End
```

Example 2

```
MAIN                    // Declare Main Function Start
    DEFSTR STRA,STRB    // Declare string variables
    REAL AA             // Declare a real number variable
    AA = 66.0           // Initialize the real number variable
    STRA = CHR(65.0)    // Assign “A” Corresponding to ASCII Code(65) to STRA
    STRB = CHR(AA)      // Assign “B” Corresponding to ASCII Code(66) to STRA
EOP                     // Main Function End
```

Related terms

ASC

10.37 FLUSH (Clearing Communication Input/output Buffer)

Usage

Clears Communication Input, Output Buffer.

Syntax

FLUSH <Port Number> <Buffer Index>

Description

Clears Input/output Buffer used to send/receive Communication.

1) <Port Number> range is 0 ~ 65,535 and The meaning is as below.
   0 : RS-232 Serial Communication Port
   10 ~ 12 : TCP I/P Server Port
   (10 : Port Number[38201], 11 : Port Number[38202], 12 : Port Number[38203])
   ~ 65,535: TCP I/P Client Port
2) <Buffer Index> range is 1 ~ 2 and The meaning is as below.
   1: Input Buffer Clear
   2: Output Buffer Clear
   3: Input, Output Buffer Clear

Arguments

<Port Number>	Data type: <Integer> <Buffer Index> range is 0 ~ 65,535.
<Buffer Index>	Data type: <Integer> <Buffer Index> range is 1 ~ 3.

Examples

Example 1

```
MAIN                    // Starts Main Function
    DEFSTR STR          // Declares Character String Variable
    STR = STRIN(0,1000) // RS-232 Serial Communication Character String
                        // Receive(Receiving wait 1 sec)
    FLUSH 0 1           // Clears Input Buffer
EOP                     // Ends Main Function
```

Example 2

```
MAIN                    // Starts Main Function
    DEFSTR STR          // Declares Character String Variable
    INT AA              // Declares Integer Variable
    AA = 10             // Initializes Integer Variable
    STR = STRIN(10,1000) // TCP I/P Server Communication Character String
                        // Receive(Receiving wait 1 sec)
    FLUSH AA 1          // Clears Input Buffer
EOP                     // Ends Main Function
```

10.38 FTOS (Conversion of Integer/Real Number Value into String)

Usage

This converts integer or real number into string.

Syntax

〈Variable〉 = FTOS(〈Input Value〉)

Description

This converts integer or real number data into string.

Ex) FTOS(1234) → “1234”
Ex) FTOS(-123.45) → “-123.45”

Return value

〈Variable〉 Data type: 〈String Type〉

Arguments

〈Input Value〉 Data type: 〈Integer or Real Number Type〉

Examples

Example 1

```
MAIN                        // Declare Main Function Start
    DEFSTR STR1,STR2        // Declare string variables
    INT AA                  // Declare an integer variable
    AA = 12345              // Initialize the integer variable
    STR1 = FTOS(1234)       // Store after converting 1234 into string “1234”
    STR2 = FTOS(AA)         // Store after converting 12345 into string “12345”
EOP                         // Main Function End
```

Example 2

```
MAIN                        // Declare Main Function Start
    DEFSTR STR1,STR2        // Declare string variables
    REAL AA                 // Declare a real number variable
    AA = -123.45            // Initialize the real number variable
    STR1 = FTOS(-12.345)    // Store after converting -12.345 into string “-12.345”
    STR2 = FTOS(AA)         // Store after converting -12.345 into String “-12.345”
EOP                         // Main Function End
```

Related terms

BINS, HTOS, SVAL

10.39 HTOS (Conversion of Integer into Hexadecimal String)

Usage

This converts integer into hexadecimal string.

Syntax

〈Variable〉 = HTOS(〈Input Value〉)

Description

This converts the input integer into hexadecimal string.

Ex)HTOS(10) → “A”
Ex)HTOS(11) → “B”
Ex)HTOS(12) → “C”
Ex)HTOS(13) → “D”
Ex)HTOS(14) → “E”
Ex)HTOS(15) → “F”

Return value

〈Variable〉	Data type: 〈String Type〉.

Arguments

〈Input Value〉	Data type: 〈Integer Type〉

Examples

Example 1

```
MAIN                    // Declare Main Function Start
    DEFSTR STR1,STR2    // Declare string variables
    INT AA              // Declare an integer variable
    AA = 11             // Initialize the integer variable
    STR1 = HTOS(10)     // Convert integer 10 into hexadecimal string(“A”)
    STR2 = HTOS(AA)     // Convert integer 11 into hexadecimal string(“B”)
EOP                     // Main Function End
```

Example 2

```
MAIN                    // Declare Main Function Start
    DEFSTR STR1,STR2    // Declare string variables
    REAL AA             // Declare a real number variable
    AA = 13.0           // Initialize the real number variable
    STR1 = HTOS(12.0)   // Convert integer 12.0 into hexadecimal string(“C”)
    STR2 = HTOS(AA)     // Convert integer 13.0 into hexadecimal string(“D”)
EOP                     // Main Function End
```

Related terms

BINS, FTOS, SVAL

10.40 SLEFT (Left String Extraction)

Usage

This extracts the left string from the string that was input.

Syntax

〈Variable〉 = SLEFT(〈String〉, 〈Extraction Count〉)

Description

This extracts the string as much as the input integer from the left side of the string that was input.

1) The 〈String〉 indicates the target for extraction.
2) The 〈Extraction Count〉 indicates the number of strings to be extracted from the left side of the string.

Return value

〈Variable〉	Data type: 〈String Type〉 This returns strings as much as the selected string.

Arguments

〈String〉	Data type: 〈String Type〉
〈Extraction Count〉	Data type: 〈Integer Type〉

Examples

Example 1

```
MAIN                          // Declare Main Function Start
    DEFSTR STR0,STR1,STR2     // Declare string variables
    INT AA                    // Declare an integer variable
    STR0 = “TEST”             // Assign string variable “TEST” to STR0
    AA = 2                    // Initialize the integer variable
    STR1 = SLEFT(“TEST”, 1)   // Store “T” in string variable STR1
    STR2 = SLEFT(STR0, AA)    // Store “TE” in string variable STR2
EOP                           // Main Function End
```

Example 2

```
MAIN                          // Declare Main Function Start
    DEFSTR STR0,STR1,STR2     // Declare string variables
    REAL AA                   // Declare a real number variable
    STR0 = “TEST”             // Assign string variable “TEST” to STR0
    AA = 4.0                  // Initialize the real number variable
    STR1 = SLEFT(“TEST”, 3.0) // Store “TES” in string variable STR1
    STR2 = SLEFT(STR0, AA)    // Store “TEST” in string variable STR2
EOP                           // Main Function End
```

Related terms

SLEN, SMID, SPOS, SRIGHT

10.41 SLEN (Returning of String Length)

Usage

This returns the length of string inputted.

Syntax

〈Variable〉 = SLEN(〈String〉)

Description

This returns the length of string inputted.

1) The string that measures the length is entered in 〈String〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉 This returns the length of string inputted.

Arguments

〈String〉	Data type: 〈String Type〉

Examples

Example 1

```
MAIN                                  // Declare Main Function Start
    DEFSTR STR1                       // Declare a string variable
    INT AA                            // Declare an integer variable
    STR1 = “STATEMENT”                // Assign string variable “STATEMENT”
    I0 = SLEN(“STATEMENT”)            // Store string count in global variable I0
    AA = SLEN(STR1)                   // Store string count in variable AA
EOP                                   // Main Function End
```

Related terms

SLEFT, SMID, SPOS, SRIGHT

10.42 SMID (Extraction of String in Specified Position)

Usage

This extracts the string as much as inputted value at the specified position of string.

Syntax

〈Variable〉 = SMID(〈String〉, 〈Starting Position〉, 〈Extraction Count〉)

Description

This extracts the characters as much as inputted integer from the specified position.
The first position of string increments by 1 at a time from 0.

1) The 〈String〉 is a target string to be extracted.
2) The 〈Starting Position〉 specifies the starting position of string extraction.
   The first position of string is 0.
3) The number of characters to read from the specified position is entered in 〈Extraction Count〉.

Return value

〈Variable〉	Data type: 〈String〉 This returns the string as much as the set count at the specified position.

Arguments

〈String〉	Data type: 〈String〉
〈Starting Position〉	Data type: 〈Integer〉
〈Extraction Count〉	Data type: 〈Integer〉

Examples

```
MAIN                                // Declare Main Function Start
    DEFSTR STR0,STR1,STR2           // Declare string variables
    INT AA, BB                      // Declare integer variables
    AA = 2                          // Initialize the integer variable (Starting Position)
    BB = 5                          // Initialize the integer variable (Extraction Count)
    STR0 = “STATEMENT”              // Assign string variable “STATEMENT”
    STR1 = SMID(“STATEMENT”, 2, 3)  // Store “ATE” in variable STR1
    STR2 = SMID(STR0,AA,BB)         // Store “ATEME” in variable STR2
EOP                                 // Main Function End
```

Related terms

SLEFT, SLEN, SPOS, SRIGHT

10.43 SPOS (Returning of Start Position in String Matching)

Usage

This returns the position where both string 1 and string 2 start to match.

Syntax

〈Variable〉 = SPOS(〈String 1〉, 〈String 2〉)

Description

This returns the position where both string 1 and string 2 start to match.

1) String to be compared is entered in 〈String 1〉.
2) String to search among 〈String 1〉 is entered in 〈String 2〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉 This compares string 1 with string 2 and returns the matching start position. If there is not matching string, then the -1 is returned.

Arguments

〈String 1〉	Data type: 〈String Type〉
〈String 2〉	Data type: 〈String Type〉

Examples

```
MAIN                                // Declare Main Function Start
    DEFSTR STR1,STR2                // Declare string variables
    INT AA                          // Declare an integer variable
    STR1 = “TEST LINE”              // Initialize string variable (Entire Strings)
    STR2 = “LINE”                   // Initialize string variable (Search Strings)
    I0 = SPOS(“TEST LINE”, “LINE”)  // Store position of matching string in global variable
    AA = SPOS(STR1,STR2)            // Store position of matching string in local variable
EOP                                 // Main Function End
```

Related terms

SLEFT, SLEN, SMID, SRIGHT

10.44 SRIGHT (Extraction of Right Side String)

Usage

This extracts the right side strings from the strings that were input.

Syntax

〈Variable〉 = SRIGHT(〈String〉, 〈Extraction Count〉)

Description

This extracts the strings as much as the input Integer from the right side of the strings that were input.

1) The 〈String〉 is a target of string extraction.
2) The 〈Extraction Count〉 indicates the number of strings to be extracted from the right side of the string.

Return value

〈Variable〉	Data type: 〈String Type〉 This returns strings as much as the number of selected strings.

Arguments

〈String〉	Data type: 〈String Type〉
〈Extraction Count〉	Data type: 〈Integer Type〉

Examples

Example 1

```
MAIN                          // Declare Main Function Start
    DEFSTR STR0,STR1,STR2     // Declare string variables
    INT AA                    // Declare an integer variable
    STR0 = “TEST”             // Assign string variable “TEST”
    AA = 2                    // Initialize the integer variable
    STR1 = SRIGHT(“TEST”, 1)  // Store “T” in string variable STR1
    STR2 = SRIGHT(STR0, AA)   // Store “ST” in string variable STR2
EOP                           // Main Function End
```

Example 2

```
MAIN                          // Declare Main Function Start
    DEFSTR STR0,STR1,STR2     // Declare string variables
    REAL AA                   // Declare a real number variable
    STR0 = “TEST”             // Assign string variable “TEST”
    AA = 4.0                  // Initialize the real number variable
    STR1 = SRIGHT(“TEST”, 3.0) // Store “EST” in string variable STR1
    STR2 = SRIGHT(STR0, AA)   // Store “TEST” in string variable STR2
EOP                           // Main Function End
```

Related terms

SLEFT, SLEN, SMID, SPOS

10.45 STRIN (Receiving of Communication String)

Usage

This reads string inputted via serial and Ethernet communications.

Syntax

〈Variable〉 = STRIN(〈Port Number〉, 〈Standby Time〉)

Description

This waits communication input for standby time (unit: ms) inputted. If no data is received for specified standby time, the standby state is checked in the system area and the next step command is executed.

CR+LF (Default Setting) or CR, LF are used to determine the end of string in case of data received through a STRIN function and these referred to as new line characters. New line character must be positioned at the end of a string and the end of input cannot be determined without a new line character.

Setting for serial and Ethernet communications is available in communication setting of public parameter.

1) 〈Port Number〉 designates a physical port and can be entered from 10 to 12.
   10: Port number 38201
   11: Port number 38202
   12: Port number 38203
2) The 〈Standby Time〉 sets the receiving standby time.

Return value

〈Variable〉 Data type: 〈String Type〉
This returns the strings received during the receiving standby time.

Arguments

〈Port Number〉 Data type: 〈Integer Type〉
Input is available only through RS-232(0) or Ethernet (10, 11, 12).

〈Standby Time〉 Data type: 〈Integer Type〉

Examples

```
MAIN                          // Declare Main Function Start
    DEFSTR RECVPK             // Declare a string variable
    INT LEN                   // Declare an integer variable
    RECVPK = STRIN(0,1000)    // Wait 1 second for receiving string via RS-232 Port
    LEN = SLEN(RECVPK)        // Store Received string length
EOP                           // Main Function End
```

Related terms

STROUT



1) TCP port number is additionally required besides an IP address to send or receive controller and strings with Ethernet communication from outside. This is different from the <Port Number> that is used together with STRIN or STROUT command. It is currently fixed value of 38201, 38202, 38203

10.46 STROUT (Sending of Communication Strings)

Usage

This sends out string via serial and Ethernet communications

Syntax

〈Variable〉 = STROUT(〈Port Number〉, 〈String〉)

Description

The strings inputted can be sent out through serial communication and returns the number of characters that were not sent. For instance, when sending out the characters of “ABCDEFGH”, the return value in success is 0 and 8 is returned in case of transmission delay failure to send out the all characters. Thus, a negative value is returned in failure.
STROUT function automatically adds CR+LF(Default Setting) or CR, LF characters when sending out the data in order to identify the end of string functions and this is called new line characters.

Settings for serial and Ethernet communication can be available in communication setting of public parameter.

1) 〈Port Number〉 designates a physical port and can be entered from 10 to 12.
   10: Port number 38201
   11: Port number 38202
   12: Port number 38203
2) String to be sent is entered in 〈String〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉 Successful Transmission : Return 0 During Transmission : Return the number of characters that were not sent Failed Transmission : Return a negative (-) value

Arguments

〈Port Number〉	Data type: 〈Integer Type〉 Input is available only through RS-232(0) or Ethernet (10, 11, 12).
〈String〉	Data type: 〈String Type〉

Examples

```
MAIN                              // Declare Main Function Start
    DEFSTR SENDPK                 // Declare a string variable
    INT LEN                       // Declare an integer variable
    SENDPK = “SUCCESS”            // Initialize the sending string
    I0 = STROUT(10,“TEST”)        // Send out string “TEST” via Ethernet Port
    LEN = STROUT(10,SENDPK)       // Send out string “SUCCESS” via Ethernet Port
EOP                               // Main Function End
```

Related terms

STRIN



1) TCP port number is additionally required besides an IP address to send or receive controller and strings with Ethernet communication from outside. This is different from the <Port Number> that is used together with STRIN or STROUT command. It is currently fixed value of 38201, 38202, 38203

10.47 SVAL (Conversion of String into Number)

Usage

This converts string into number.

Syntax

〈Variable〉 = SVAL(〈String〉)

Description

This converts the string into integer or real number data.

1) The string to be converted is entered in 〈String〉.

Return value

〈Variable〉	Data type: 〈Integer Type〉

Arguments

〈String〉	Data type: 〈String Type〉

Examples

```
MAIN                        // Declare Main Function Start
    DEFSTR STR1,STR2        // Declare string variables
    INT IVAL1               // Declare an integer variable
    REAL FVAL               // Declare a real number variable
    STR1 = “1234”           // Assign “1234” for string to convert to integer
    STR2 = “12.34”          // Assign “12.34” for string to convert to real number
    I0 = SVAL(“1234”)       // Store after converting “1234” into integer constant 1234
    IVAL1 = SVAL(STR1)
    F0 = SVAL(“12.34”)      // Store after converting “12.34” into real number constant
    FVAL = SVAL(STR1)       // 12.34
EOP                         // Main Function End
```

Related terms

BINS, FTOS, HTOS

11. System

11.1 SETERR (SYSTEM EMG Alarm Occurrence)

Usage

This is a user defined alarm and invokes system emergency.

Syntax

SETERR(<User Alarm Number>[, <User Alarm Message>])

Description

This is a command that creates “[2103] system emergency” during JOB program execution by an intention of a user. User Alarm Number and Message are displayed on the alarm message window. This command is mainly used in main thread of auto mode.

1) A user is required to set <User Alarm Number> within the integer range.
2) The <User Alarm Message> is to help a user identify easily on the alarms that a user specified.

Arguments

<User Alarm Number>	Data type: <Integer Type> Alarm number that a user has arbitrarily set.
<User Alarm Message>	Data type: <String Type> Alarm message that a user has arbitrarily output.

Examples

Example 1

```
IF (IN0 == 1) THEN          // If Input Signal 0 is 1
    SETERR(1)               // User Alarm Number 1, causing system emergency
ENDIF                       // IF statement End
```

Example 2

```
IF (IN0 == 1) THEN          // If Input Signal 0 is 1
    SETERR(1, “USER ERR 1”) // System emergency occurs with message
ENDIF                       // IF statement End
```

Related terms

RESET

11.2 RESET (Alarm Clear)

Usage

This clears all alarms occurred.

Syntax

RESET[(<Robot Number>)]

Description

This clears all alarms currently occurred.
This command is mainly used in main thread of System mode.

1) The robot to clear all alarms is selected by entering the number in <Robot Number>. In case of no input, alarms of all robots are cleared.

Arguments

<Robot Number>	Data type: <Integer Type> The range of <Robot Number> is 1 ~ 3.

Examples

```
WHILE 1                          // Endless Loop Start
    IF(IN0 == 1) THEN            // If Input Signal 0 is 1
        RESET                    // Clear All Alarms
    ELSEIF (IN1 == 1) THEN       // If Input Signal 1 is 1
        RESET(1)                 // Clear Alarms of Robot 1
    ELSEIF (IN2 == 1) THEN       // If Input Signal 2 is 1
        RESET(2)                 // Clear Alarms of Robot 2
    ELSEIF (IN3 == 1) THEN       // If Input Signal 3 is 1
        RESET(3)                 // Clear Alarms of Robot 3
    ENDIF                        // IF statement End
ENDWL                            // Endless Loop End
```

Related terms

SETERR

11.3 RERROR (Returning of Last Alarm Code)

Usage

This returns the alarm code that was occurred last (most recently).

Syntax

〈Variable〉 = RERROR(〈Robot Number〉)

Description

This returns the Last (most recently) Alarm Code among alarms that were occurred. This command is mainly used in main thread of System mode.

1) The 〈Robot Number〉 selects the robot. If entering 0, it returns the last (most recently) alarm code among entire alarms of all robots.

Return value

〈Variable〉	Data type: 〈Integer Type〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3. (0: All, 1 ~ 3: Robot Selection)

Examples

```
INT A_ST, ECODE                 // Declare integer variables

WHILE 1                         // Endless Loop Start
    A_ST=RSTATE(1,1)            // Check Alarm Status
    IF(A_ST == 1) THEN          // If status is alarm state
        ECODE = RERROR(0)       // Return Last Alarm Code
        PRINT(0,ECODE)          // Store Last Alarm Code in index 0 of monitoring log
    ENDIF                       // IF statement End
ENDWL                           // Endless Loop End
```

Related terms

RERRCNT, RERRCODE, RERRTEXT, READ_DERR

11.4 RERRCNT (Returning of Alarm Count Occurred)

Usage

This returns a total number of alarms occurred.

Syntax

〈Variable〉 = RERRCNT(〈Robot Number〉)

Description

This returns a total number of alarms occurred. This command is mainly used in main thread of System mode.

1) Robot is selected by entering the number in 〈Robot Number〉.
If entering 0, it returns the total number of alarms of all robots.

Return value

〈Variable〉	Data type: 〈Integer Type〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3. (0: All, 1 ~ 3: Robot Selection)

Examples

```
INT A_ST, ECNT                          // Declare integer variables

WHILE 1                                 // Endless Loop Start
    A_ST=RSTATE(1,1)                    // Check Alarm Status
    IF(A_ST == 1) THEN                  // If status is alarm state
        ECNT = RERRCNT(0)               // Return total number of alarms of all robots
        PRINT(0,ECNT)                   // Store total number of alarms in index 0 of log
    ENDIF                               // IF statement End
ENDWL                                   // Endless Loop End
```

Related terms

RERROR, RERRCODE, RERRTEXT, READ_DERR

11.5 RERRCODE (Returning of Alarm Coe of Index Number)

Usage

This returns alarm code for index number inputted.

Syntax

〈Variable〉 = RERRCODE (〈Robot Number〉. 〈Index Number〉)

Description

This returns the alarm code of index number that were inputted among various alarms occurred. This command is mainly used in main thread of System mode.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) Entering 0 means all robots.
3) The 〈Index Number〉 means alarm occurred sequence. Thus, alarm code is returned as the specified sequence among all alarms of selected robot.

Return value

〈Variable〉	Data type: 〈Integer Type〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3. (0: All, 1 ~ 3: Robot Selection)
〈Index Number〉	Data type: 〈Integer Type〉 Index number should not exceed the total number of alarms.

Examples

```
INT A_ST, AA                          // Declare integer variables
INT ECNT, ECODE

WHILE 1                               // Endless Loop Start
    A_ST = RSTATE(1,1)                // Check Alarm Status
    IF(A_ST == 1) THEN                // If status is alarm state
        ECNT = RERRCNT(0)             // Return total number of alarms of all robots
        PRINT(0,ECNT)                 // Store total number of alarms in index 0 of log

        FOR AA = 1 TO ECNT            // Check alarm codes with the number of alarms
            ECODE = RERRCODE(0,AA)    // Return alarm code of specified index
            PRINT(1+AA, ECODE)        // Store alarm code in monitoring log
        NEXT                          // FOR statement End
    ENDIF                             // IF statement End
ENDWL                                 // Endless Loop End
```

Related terms

RERROR, RERRCNT, RERRTEXT, READ_DERR

11.6 RERRTEXT (Returning of Alarm Content)

Usage

This returns alarm content corresponding to index number inputted.

Syntax

〈Variable〉 = RERRTEXT(〈Robot Number〉, 〈Index Number〉)

Description

This returns alarm content corresponding to index number inputted among various alarms occurred. This command is mainly used in main thread of System mode.

1) Robot is selected by entering the number in 〈Robot Number〉.
   Entering 0 means all robots.
2) The 〈Index Number〉 means alarm occurred sequence. Thus, alarm code is returned as the specified sequence among all alarms of selected robot.

Return value

〈Variable〉	Data type: 〈String Type〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3. (0: All, 1 ~ 3: Robot Selection)
〈Index Number〉	Data type: 〈Integer Type〉 Index number should not exceed the total number of alarms.

Examples

```
INT A_ST, AA                                // Declare integer variables
INT ECNT, ECODE
DEFSTR E_STR                                // Declare a string variable
WHILE 1                                     // Endless Loop Start
    A_ST = RSTATE(1,1)                      // Check Alarm Status
    IF(A_ST == 1) THEN                      // If status is alarm state
        ECNT=RERRCNT(0)                     // Return total number of alarms of all robots
        PRINT(0,ECNT)                       // Store total number of alarms in index 0 of log
        FOR AA = 1 TO ECNT                  // Check alarm codes with the number of alarms
            ECODE = RERRCODE(0,AA)          // Return alarm code value of specified index
            E_STR = RERRTEXT(0,AA)          // Return alarm code content of specified index
            PRINT(1+AA, ECODE, E_STR)       // Store alarm code in monitoring log
        NEXT                                // FOR statement End
    ENDIF                                   // IF statement End
ENDWL                                       // Endless Loop End
```

Related terms

RERROR, RERRCNT, RERRCODE, READ_DERR

11.7 READ_DERR (Returning of Driver Alarm Code)

Usage

This returns alarm code of driver that was occurred at specified axis.

Syntax

〈Variable〉 = READ_DERR(〈Robot Number〉, 〈Axis Number〉)

Description

This returns alarm code of driver of the specified axis.
This command is mainly used in main thread of System mode.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) The 〈Axis Number〉 indicates a connection sequence of robot joints.

Return value

〈Variable〉	Data type: 〈String Type〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 Axis number should not exceed the max number of axis or should not lack the min number of axis.

Examples

```
INT ECODE, AA                   // Declare integer variables

FOR AA = 1 TO 6                 // Execution of Iterative statement for Axis 1 ~ 6
    ECODE = READ_DERR(1,AA)     // Return driver alarm by each axis of robot 1
    PRINT(AA, ECODE)            // Store in monitoring log
NEXT                            // FOR statement End
```

Related terms

RERROR, RERRCNT, RERRCODE, RERRTEXT

11.8 RSTATE (Acquiring of Robot Status Information)

Usage

This is a command that returns robot status.

Syntax

〈Variable〉 = RSTATE(〈Robot Number〉, 〈Status Index〉)

Description

This reads the index information of specified robot.

1) The 〈Robot Number〉 indicates the robot number of the robot to read.
2) The 〈Status Index〉 indicates the predefined specific state information of robot.

Return value

〈Variable〉 Data type: 〈Integer Type〉

Arguments

〈Robot Number〉 Data type: 〈Integer Type〉
The range of 〈Robot Number〉 is 1 ~ 3.

〈Status Index〉 Data type: 〈Integer Type〉
The range of 〈Status Index〉 is 0 ~ 16.
※ The index number that was not described in the table below is not currently used.

<table>
<tr><th>INDEX</th><th>STATE NAME</th><th>Content</th></tr>
<tr><td>0</td><td>ALL</td><td>Returning entire status information</td></tr>
<tr><td>1</td><td>ALARM</td><td>Returning alarm status</td></tr>
<tr><td>2</td><td>MC STATUS (READY)</td><td>Returning MC status (0: off, 1: on)</td></tr>
<tr><td>3</td><td>ORIGIN</td><td>Returning origin completion status
(If an encoder is ABS type, it is always on)</td></tr>
<tr><td>4</td><td>INPOSITION</td><td>Target position reached</td></tr>
<tr><td>5</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>6</td><td>SERVO STATE</td><td>Returning SERVO MOTOR on(1)/off(0) state</td></tr>
<tr><td>7</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>8</td><td>MODE MISMATCH</td><td>Return T/P mode switch information and status of upper equipment (INDEX) mode
<table>
<tr><th>Value</th><th>T/P MODE</th><th>INDEX MODE</th></tr>
<tr><td rowspan="2">0</td><td>SYSTEM</td><td>SYSTEM</td></tr>
<tr><td>ROBOT</td><td>ROBOT</td></tr>
<tr><td>1</td><td>SYSTEM</td><td>ROBOT</td></tr>
</table></td></tr>
</table>

		ROBOT / SYSTEM ※ Supported from IF F/W V1.2B or higher.
9	SYSTEM MODE	Status return of SYSTEM(1)/ROBOT(0)
10	Reserved	Reserved
11	EMG	On(1) in case of emergency stop
12 ~ 14	Reserved	Reserved
15	ROBOT MOVING	Notice of motion execution state of corresponding robot (0: No Movement, 1: In Moving)
16	FAN STATE	Notice of fan alarm state (0: No Alarm, 1: Alarm occurred)

Examples

Example 1

- Individually output to I/O after checking the entire status

```
INT TEMP                    // Declare an integer variable
TEMP = RSTATE(1,0)          // Read entire status information of robot1
OUT10 = (TEMP>>6)&0H01      // Output Servo Status
OUT9 = (TEMP>>9)& 0H01      // Output System Status
OUT8 = TEMP&0H01            // Output Alarm Status
```

Example 2

- Output to I/O after checking Individual status

```
OUT10 = RSTATE(2,6)         // Output Servo Status of Robot 2
OUT9 = RSTATE(2,9)          // Output System Status of Robot 2
OUT8 = RSTATE(2,1)          // Output Alarm Status of Robot 2
```

11.9 SWLIMIT (Reading of S/W LIMIT Parameter Value)

Usage

This reads the parameter values that limit the movement range by each axis of robot.

Syntax

〈Variable〉 = SWLIMIT(〈Robot Number〉, 〈Axis Number〉, 〈Max/Min〉)

Description

This reads the S/W Limit specified in BODY-RANG among robot parameters.

1) The robot to read for a parameter, is selected by entering the number in 〈Robot Number〉.
2) The axis, which the value will be read among various axes, is selected by entering the number in 〈Axis Number〉.
3) The 〈Max/Min〉 selects the value to be read among Min value or Max value.

Return value

〈Variable〉	Data type: 〈Real Number Type〉

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 The range of 〈Axis Number〉 is 1 ~ DOF.
〈Max/Min〉	Data type: 〈Integer Type〉 In case of reading Min value, 0 is selected and 1 is selected if the Max value is read.

Examples

```
REAL LIMIT_MAX, LIMIT_MIN       // Declare integer variables
LIMIT_MIN = SWLIMIT(1,1,0)      // Read Min Movement Range Value of Axis1 of Robot1
LIMIT_MAX = SWLIMIT(1,1,1)      // Read Max Movement Range Value of Axis1 of Robot1
```

11.10 READ_RPM (Reading of RPM Value)

Usage

This reads current RPM value by each axis of robot.

Syntax

〈Variable〉 = READ_RPM(〈Robot Number〉, 〈Axis Number〉)

Description

This reads current RPM value corresponding the specified robot and axis number.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) Axis to read for current RPM is selected by entering the number in 〈Axis Number〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is RPM.

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 The range of 〈Axis Number〉 is 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // Declare integer variables
REAL RPM[6]                            // Declare a real number variable
DEFSTR STR                             // Declare a string variable
WHILE 1                                // Endless Loop Start
    STR = “RPM:”                       // Initialize the string variable
    FOR AA = 0 TO 5                    // AA is increased by 1 from 0 to 5
        RPM[AA] = READ_RPM(1,AA+1)     // Read RPM of each axis and store them to array
        STR = STR + FTOS(RPM[AA])      // Store in string variable
    NEXT                               // FOR statement End
    R_VAL = STROUT(10, STR)            // Send String Data to Ethernet Port
    DLAY 100                           // 0.1 second Time Delay
ENDWL                                  // Endless Loop End
```

Related terms

READ_LOAD, READ_TRQMAX, READ_VEL, READ_TRQ, READ_MTEMP, READ_CTEMP

11.11 READ_LOAD (Load Rate Reading)

Usage

This reads current load rate of each axis of robot.

Syntax

〈Variable〉 = READ_LOAD(〈Robot Number〉, 〈Axis Number〉)

Description

This reads current load rate of the specified robot and axis.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) Axis to read for current load rate is selected by entering the number in 〈Axis Number〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result is %.

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 The range of 〈Axis Number〉 is 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // Declare integer variables
REAL LOAD[6]                           // Declare a real number variable
DEFSTR STR                             // Declare a string variable
WHILE 1                                // Endless Loop Start
    STR = “LOAD:”                      // Initialize the string variable
    FOR AA = 0 TO 5                    // AA is increased by 1 from 0 to 5
   LOAD [AA] = READ_LOAD(1,AA+1)       // Read Load rate of each axis and store them to array
   STR = STR + FTOS(LOAD [AA])         // Store in string variable
    NEXT                               // FOR statement End
    R_VAL = STROUT(10, STR)            // Send string data to Ethernet Port
    DLAY 100                           // 0.1 second Time Delay
ENDWL                                  // Endless Loop End
```

Related terms

READ_RPM, READ_TRQMAX, READ_VEL, READ_TRQ, READ_MTEMP, EAD_CTEMP

11.12 READ_TRQMAX (Reading of Max Torque Value)

Usage

This reads the max torque value of each axis of robot.

Syntax

<Variable> = READ_TRQMAX(<Robot Number>, <Axis Number>)

Description

This reads the max torque value of the specified robot and axis.

1) Robot is selected by entering the number in <Robot Number>.
2) Axis to read for the max torque value is selected by entering the number in <Axis Number>.

Return value

<Variable>	Data type: <Real Number Type> The unit of result is %.

Arguments

<Robot Number>	Data type: <Integer Type> The range of <Robot Number> is 1 ~ 3.
<Axis Number>	Data type: <Integer Type> The range of <Axis Number> is 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // Declare integer variables
REAL T_MAX[6]                          // Declare a real number variable
DEFSTR STR                             // Declare a string variable
WHILE 1                                // Endless Loop Start
    STR = "TRQ_MAX:"                   // Initialize the string variable
    FOR AA = 0 TO 5                    // AA is increased by 1 from 0 to 5
        T_MAX[AA] = READ_TRQMAX(1,AA+1) // Read Max torque of each Axis and
                                       // Store them to Array
        STR = STR + FTOS(T_MAX [AA])   // Store in String Variable
    NEXT                               // FOR statement End
    R_VAL = STROUT(10, STR)            // Send string to Ethernet Port
    DLAY 100                           // 0.1 second Time Delay
ENDWL                                  // Endless Loop End
```

Related terms

READ_RPM, READ_LOAD, READ_VEL, READ_TRQ, READ_MTEMP, READ_CTEMP

11.13 READ_VEL (Reading of Velocity)

Usage

This reads the current velocity value of each axis of robot.

Syntax

〈Variable〉 = READ_VEL(〈Robot Number〉, 〈Axis Number〉)

Description

This reads the current velocity value of the specified robot and axis.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) Axis to read for the current velocity value is selected by entering the number in 〈Axis Number〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is degree/s for the rotational axis and mm/s for the linear movement axis.

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 The range of 〈Axis Number〉 is 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // Declare integer variables
REAL V_VAL[6]                          // Declare a real number variable
DEFSTR STR                             // Declare a string variable
WHILE 1                                // Endless Loop Start
    STR = “VEL:”                       // Initialize the string variable
    FOR AA = 0 TO 5                    // AA is increased by 1 from 0 to 5
        V_VAL[AA] = READ_VEL(1,AA+1)   // Read velocity of each axis and store them to array
        STR = STR + FTOS(V_VAL[AA])    // Store in string variable
    NEXT                               // FOR statement End
    R_VAL = STROUT(10, STR)            // Send string to Ethernet Port
    DLAY 100                           // 0.1 second Time Delay
ENDWL                                  // Endless Loop End
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_TRQ, READ_MTEMP, READ_CTEMP

11.14 READ_TRQ (Reading of Torque Value)

Usage

This reads the current torque value of each axis of robot.

Syntax

〈Variable〉 = READ_TRQ(〈Robot Number〉, 〈Axis Number〉)

Description

This reads the current torque value of the specified robot and axis.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) Axis to read for the current torque value is selected by entering the number in 〈Axis Number〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is %.

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 The range of 〈Axis Number〉 is 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // Declare integer variables
REAL TRQ_VAL[6]                        // Declare a real number variable
DEFSTR STR                             // Declare a string variable
WHILE 1                                // Endless Loop Start
    STR = "TRQ:"                       // Initialize the string variable
    FOR AA = 0 TO 5                    // AA is increased by 1 from 0 to 5
        TRQ_VAL[AA] = READ_TRQ(1,AA+1) // Read current torque of each axis and
                                       // store them to array
        STR = STR + FTOS(TRQ_VAL[AA])  // Store in string variable
    NEXT                               // FOR statement End
    R_VAL = STROUT(10, STR)            // Send string to Ethernet Port
    DLAY 100                           // 0.1 second Time Delay
ENDWL                                  // Endless Loop End
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_VEL, READ_MTEMP, READ_CTEMP

11.15 READ_MTEMP (Reading of Motor Encoder temperature Value)

Usage

This reads the current motor encoder temperature of each axis of robot.

Syntax

〈Variable〉 = READ_MTEMP(〈Robot Number〉, 〈Axis Number〉)

Description

This reads the current motor encoder temperature of the specified robot and axis.

1) Robot is selected by entering the number in 〈Robot Number〉.
2) Axis to read for the current motor encoder temperature value is selected by entering the number in 〈Axis Number〉.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is (℃).

Arguments

〈Robot Number〉	Data type: 〈Integer Type〉 The range of 〈Robot Number〉 is 1 ~ 3.
〈Axis Number〉	Data type: 〈Integer Type〉 The range of 〈Axis Number〉 is 1 ~ DOF.

Examples

```
INT AA, R_VAL                          // Declare integer variables
REAL TEMP[6]                           // Declare a real number variable
DEFSTR STR                             // Declare a string variable
WHILE 1                                // Endless Loop Start
    STR = " ENCODER TEMP:"             // Initialize the string variable
    FOR AA = 0 TO 5                    // AA is increased by 1 from 0 to 5
        TEMP[AA] = READ_MTEMP(1,AA+1)  // Read current Motor Encoder
                                       // Temperature of each Axis and Store
                                       // them to Array
        STR = STR + FTOS(TEMP[AA])     // Store in string variable
    NEXT                               // FOR statement End
    R_VAL = STROUT(10, STR)            // Send string to Ethernet Port
    DLAY 100                           // 0.1 second Time Delay
ENDWL                                  // Endless Loop End
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_VEL, READ_TRQ, READ_CTEMP

11.16 READ_CTEMP (Reading of controller Internal Temperature Value)

Usage

This reads the internal temperature value of controller.

Syntax

〈Variable〉 = READ_CTEMP()

Description

This reads the internal temperature value of controller.

Return value

〈Variable〉	Data type: 〈Real Number Type〉 The unit of result value is (℃).

Examples

```
INT R_VAL                          // Declare an integer variable
REAL C_TEMP                        // Declare a real number variable
DEFSTR STR                         // Declare a string variable
WHILE 1                            // Endless Loop Start
    STR = " CONTROLLER TEMP:"      // Initialize the string variable
    C_TEMP = READ_CTEMP ()         // Read current controller temperature of each axis
                                   // and store them to array
    STR = FTOS(C_TEMP)             // Store in string variable
    R_VAL = STROUT(10, STR)        // Send string to Ethernet Port
    DLAY 100                       // 0.1 second Time Delay
ENDWL                              // Endless Loop End
```

Related terms

READ_RPM, READ_LOAD, READ_TRQMAX, READ_VEL, READ_TRQ, READ_MTEMP

11.17 PRINT (Monitoring Log)

Usage

This records monitoring log.

Syntax

PRINT(〈Index〉, 〈Log Content1〉 [{, 〈Log Content2〉, … 〈Log Content〉}])

Description

This is used for recording the logs that can be checked while executing a JOB program.

It is possible to assign all variables and constants to the log contents during log recording. But, it is not currently available to assign a function to 〈Log Content〉 and to store its return value.

- Possible Example: PRINT(0, 12, “STRING DATA”, F[12], I0, AA) ← (○)
- Impossible Example: PRINT(0,FTOS(R_VAL)) ← (×)

The record by PRINT function can be checked only in DISP window 9.PRINT of RUN execution screen of T/P. It is possible to check the data by moving to the index number specified by a user and the logs inputted are separated by a mark “,”. In the case of positional data, the symbols of “<” and “>” are separately used. The position value of each axis is displayed within a symbol.

- DISP Screen 9.PRINT Display Example

```
000:[0:0]<-11.860,8.636,18.761,57.015,12.979,-100.037>
001:[0:0]1,DATA,12.4000000,45
002:
003:
```

In case of using PRINT function, a user can additionally mark the mode and thread number that a JOB program is executed before the log contents.

- Display example: [〈Execution Mode〉:〈Thread Number〉] 〈Log Content1〉, ⋯, 〈Log Content〉

Refer to the table below for execution mode and the meaning of thread number.

Classification	Description
〈Execution Mode〉	System (1), Robot (0)
〈Thread Number〉	0 ~ 3

1) Place to be stored is selected by specifying 〈Index〉.
2) Data can be recorded in 〈Log Content〉 as a user desires.

Arguments

〈Index〉	Data type: 〈Integer Type〉 The range of 〈Index〉 is 0 ~ 999.
〈Log Content〉	Data type: 〈All variables & Constant〉s

Examples

Example 1

```
MAIN                                // Declare Main Function Start
    POS ACUR                        // Declare a position variable
    ACUR = HERE                     // Store current robot position in variable
    PRINT(0,ACUR)                   // Store position value in index 0 of monitoring log
EOP                                 // Main Function End
```

- T/P Output Example

```
000:[0:0]<-11.860,8.636,18.761,57.015,12.979,-100.037>
```

Example 2

```
MAIN                                // Declare Main Function Start
    INT I_VAL                       // Declare an integer variable
    REAL R_VAL                      // Declare a real number variable

    I_VAL = 1                       // Initialize the integer variable
    R_VAL = 12.4                    // Initialize the real number variable
    PRINT(1,I_VAL, “DATA”, R_VAL,45)  // Store various values in index 1 of monitoring log
EOP                                 // Main Function End
```

- T/P Output Example

```
001:[0:0]1,DATA,12.4000000,45
```

Related terms

WLOG

11.18 WLOG (User Log)

Usage

This stores the random user log with time information as a file.

Syntax

WLOG(〈Log Content1〉, [{, 〈Log Content2〉, ··· 〈Log Content〉}])

Description

This is used when log is stored during JOB program execution. The logs including time information are stored as a file and can be downloaded using Uni-Host program or USB backup features. Refer to Uni-Host Manual (N2-HM-E□□) and Operation Manual (N2-OM-E□□) for further detail descriptions on the downloading features.

All variables and constants can be assigned to the log contents. But, it is not currently available to assign a function to 〈Log Content〉 and to store its return value.

- Possible Example: WLOG(12, "STRING DATA", F[12], I0, AA) ← (○)
- Impossible Example: WLOG(FTOS(R_VAL)) ← (×)

It is possible to check the record by using WLOG function as following procedure; Robot Mode 9.INFO → 3.LOG → 3.ULOG Screen. But, when using small T/P, only most recent 100 lines are available. Logs inputted are separated by a symbol ",". In case of positional data, the symbols of "〈" and "〉" are separately used. Position value of each axis is displayed within a symbol.

- ULOG Screen Output Example

```
00:[180901 15:00:06.085]
01:[0:0]WLOG DATA1,12.4000000,45
02:[180901 15:15:12.072]
03:[0:0]WLOG DATA2,9.548000,0
```

When using WLOG function, the time information that the log was recorded is first recorded. Then, the mode and thread number that the JOB program is executed are additionally recorded before the user input log contents.

- Screen Output Example: [〈Record Time Information〉]
  [〈Execution Mode〉:〈Thread Number〉] 〈Log Content1〉, ···, 〈Log Content〉

Refer to the table below for the meaning of execution mode and thread number.

Classification	Description
〈Execution Mode〉	System (1), Robot (0)
〈Thread Number〉	0 ~ 3

1) Data can be recorded in 〈Log Content〉 as a user desires.

Arguments

〈Log Content〉	Data type: 〈All variables & Constants〉

Examples

Example 1

```
MAIN                                    // Declare Main Function Start
    POS ACUR                            // Declare a position variable
    ACUR = HERE                         // Store current robot position to variable
    WLOG(“CUR POS”, ACUR)               // Store position value in User Log
EOP                                     // Main Function End
```

- Log Example

```
01:[180901 15:30:05.085]
02:[0:0]CUR POS,〈-11.860,8.636,18.761,57.015,12.979,-100.037〉
```

Example 2

```
MAIN                                    // Declare Main Function Start
    INT I_VAL                           // Declare an integer variable
    REAL R_VAL                          // Declare a real number variable

    I_VAL = 100                         // Initialize the integer variable
    R_VAL = 12.4                        // Initialize the real number variable
    WLOG(“WLOG DATA”, R_VAL, I_VAL)     // Store various values in User Log
EOP                                     // Main Function End
```

- Log Example

```
01:[180901 15:30:06.072]
02:[0:0]WLOG DATA,12.4000000,100
```

Related terms

PRINT

11.19 CLOG (Communication Log) ※ Not currently supported.

11.20 WGPNT (Global Point Variable File Store)

Usage

This stores the global point variable as a file.

Syntax

[<Variable> =] WGPNT(<Synchronization>)

Description

In case of modifying the global point variable during executing the JOB program, WGPNT function is used to store in a disk.

1) Inputting a value in <Synchronization> enables to select whether or not to wait until being synchronized.

Return value

<Variable>	Data type: <Integer Type> Either 0 (Write Normal) or -1(Write Error) is returned as a result value.

Arguments

<Synchronization>	Data type: <Integer Type> Only 0 (No Standby for Disk Synchronization) or 1(Standby for Disk Synchronization) can be entered.

Examples

```
MAIN                                  // Declare Main Function Start
    POS ACUR                          // Declare a position variable
    ACUR = <100,100,100,100,100,100>  // Initialize the position variable
    GP0 = ACUR                        // Assign position variable to GP0
    WGPNT(1)                          // Store global point variable as a file
EOP                                   // Main Function End
```

Related terms

WGINT, WGFLT

1) If no store command is issued after altering a global variable during program execution, altered data may not maintained if power cut-off or re-booting.
2) Data can be lost if the power is cut abruptly during a file saving.

11.21 WGINT (Global Variable File Store)

Usage

This stores the global integer variable as a file.

Syntax

[<Variable> =] WGINT(<Synchronization>)

Description

In case of modifying the global integer variable during executing the JOB program, WGINT function is used to store in a disk.

1) Inputting a value in <Synchronization> enables to select whether or not to wait until being synchronized.

Return value

<Variable>	Data type: <Integer Type> Either 0 (Write Normal) or -1(Write Error) is returned as a result value.

Arguments

<Synchronization>	Data type: <Integer Type> Only 0 (No Standby for Disk Synchronization) or 1(Standby for Disk Synchronization) can be entered.

Examples

```
MAIN                        // Declare Main Function Start
    INT I_VAL               // Declare an integer variable
    I_VAL = 50 +25          // Initialize the integer variable
    I100 = 100              // Assign value to I100
    I101 = I_VAL            // Assign value to I101
    WGINT(1)                // Store global integer variable as a file
EOP                         // Main Function End
```

Related terms

WGPNT, WGFLT



1) If no store command is issued after altering a global variable during program execution, altered data may not maintained if power cut-off or re-booting.
2) Data can be lost if the power is cut abruptly during a file saving.

11.22 WGFLT (Global Real Number Variable File Store)

Usage

This stores the global point variable as a file.

Syntax

[<Variable> =] WGFLT(<Synchronization>)

Description

In case of modifying the global real number variable during executing the JOB program, WGFLT function is used to store in a disk.

1) Inputting a value in <Synchronization> enables to select whether or not to wait until being synchronized.

Return value

<Variable> Data type: <Integer Type>
Either 0 (Write Normal) or -1(Write Error) is returned as a result value.

Arguments

<Synchronization> Data type: <Integer Type>
Only 0 (No Standby for Disk Synchronization) or 1(Standby for Disk Synchronization) can be entered.

Examples

```
MAIN                    // Declare Main Function Start
    REAL R_VAL          // Declare a real number variable
    R_VAL = RAD(90)     // Initialize the real number variable
    F100 = 3.14         // Assign value to F100
    F101 = R_VAL        // Assign value to F101
    WGFLT(1)            // Store global real number variable as a file
EOP                     // Main Function End
```

Related terms

WGPNT, WGINT

1) If No store command is issued after altering a global variable during program execution, altered data may not maintained if power cut-off or re-booting.
2) Data can be lost if the power is cut abruptly during a file saving.

11.23 TACT_START (Setting Time measurement start position)

Usage

Sets Start position for Tact Time measurement.

Syntax

TACT_START()

Description

To measure Move Command or Tact Time while running JOB Program, use TACT_START Function to set start position.

Examples

```
MAIN                        // Starts Main Function
    SVON                    // Servo on
    INT VAL                 // Declares Integer Variable
    TACT_START()            // Sets Time measurement start position
    JMOV P0                 // JMOV to P0
    LMOV P1                 // LMOV to P1
    VAL = TACT_STOP()       // Sets Time measurement end position
   PRINT(0,VAL)             // Saves measurement time at Monitoring
EOP                         // Log Index No 0
                            // Ends Main Function
```

Related terms

TACT_START, TACT_STOP

1) If Time measurement end position is not set using TACT_STOP Function, Time measurement function can't be used.

11.24 TACT_STOP (Setting Time measurement end position)

Usage

Sets End position for Tact Time measurement.

Syntax

[<Variable> =] TACT_STOP()

Description

To measure Move Command or Tact Time while running JOB Program and to set measurement end position, use TACT_STOP Function.

Return value

<Variable>	Data type: <Integer> Measurement time returns as result value. result value unit is ms.

Examples

```
MAIN                    // Starts Main Function
    SVON                // Servo on
    INT VAL             // Declares Integer Variable
    TACT_START()        // Sets Time measurement start position
    JMOV P0             // JMOV to P0
    LMOV P1             // LMOV to P1
    VAL = TACT_STOP()   // Sets Time measurement end position
   PRINT(0,VAL)         // Saves measurement time at Monitoring Log Index No 0
EOP                     // Ends Main Function
```

Related terms

TACT_START, TACT_STOP

1) If Time measurement start position is not set using TACT_START Function, Time measurement function can't be used.

12. System Mode Dedicated Commands

12.1 PSEL (Program Selection)

Usage

This matches JOB file that was input to the thread number of specified robot.

Syntax

[<Variable> =] PSEL(<Thread Number>, "<JOB File Name>")

Description

This reads JOB file to check if there is error in the file and matches JOB file with thread of specified robot.
This command is only available in System mode of controller not used in Robot mode.

1) Entering the number in <Thread Number> selects JOB program to execute.
2) The <JOB File Name> can have only the file name except the file end-mark "." and extension "JOB".

Return value

<Variable> Data type: <Integer Type>
Either 0 (Selection Success) or negative (-) values (Selection Failure) are returned as a result value.

- Cause of Selection Failure

Value	Cause
-1	In case of not SYSTEM mode of small T/P, or in case that PSEL command was used in Robot mode
-2	Input Data Error
-3	When JOB file that was input doesn't exist
-4	When the name of JOB file that was input is "BGTD" or "BGTF
-5	When the selected thread is in execution
-6	When JOB file load was failed

Arguments

<Thread Number> Data type: <Integer Type>
The range of thread number is 1 ~ 4(TH1 ~ TH4).

Examples

```
INT R_VAL, STATE                          // Declare integer variables

R_VAL = PSEL(1,"MOVE1")                   // Select MOVE1.JOB in thread of robot 1
IF (R_VAL<0) THEN                         // If return value is negative
    GOTO END_PROGRAM                      // Failure in program selection,
                                          // Move to End Branch
ENDIF                                     // End of IF statement

WHILE 1                                   // Endless Loop Start
    STATE = PSTATE(1)                     // Read Program Execution Status
    IF ((STATE == 2)&&(IN0 == 1))THEN     // If State is 2(TH_STOP), Input 0 is 1
        R_VAL = PSEL(1,"MOVE2")           // Select MOVE2.JOB in thread of robot 1
        IF (R_VAL<0) THEN                 // If return value is negative
            GOTO END_PROGRAM              // Failure in Program Selection,
                                          // Move to End Branch
        ENDIF                             // End of IF statement
    ELSEIF ((STATE == 2)&&(IN1 == 1))THEN // If State is 2(TH_STOP), Input for 1 is 1
        R_VAL = PSEL(1,"MOVE1")           // Select MOVE1.JOB in Thread of Robot 1
        IF (R_VAL<0) THEN                 // If Negative Return Value Confirmed
            GOTO END_PROGRAM              // Failure in Program Selection,
                                          // Move to End Branch
        ENDIF                             // End of IF statement
    ELSEIF ((STATE == 2)&&(IN2 == 1)) THEN // If State is 2(TH_STOP), Input 2 is 1
        PSTART(1)                         // Start of Program Execution
    ELSEIF ((STATE == 3)&&(IN3 == 1)) THEN // If State is 3(TH_RUN), Input 3 is 1
        PSTOP(1)                          // Program Stop
    ELSEIF ((STATE == 2)&&(IN4 == 1)) THEN // If State is 2(TH_STOP), Input 4 is 1
        PRESTART(1)                       // Program Re-start
    ELSEIF ((STATE == 2)&&(IN5 == 1)) THEN // If State is 2(TH_STOP), Input 5 is 1
        PEXIT(1)                          // Program Clear
    ELSEIF STATE == 0 THEN                // If State is 0(TH_ERROR)
        GOTO END_PROGRAM                  // Move to End Branch
    ENDIF                                 // IF statement End
ENDWL                                     // Endless Loop End
LABL END_PROGRAM                          // Declare Program End as a Branch Point
```

Related terms

PSTART, PSTOP, PSTATE, PRESTART, PEXIT

1) This command is only available in System mode of controller and it creates “Cannot use PSEL at foretask” alarm when used in Robot mode.
2) Only TH1 (1) can be selected as thread number at the moment.

12.2 PSTART (Program Start)

Usage

This executes a JOB program selected in thread number of specified robot.

Syntax

[〈Variable〉 =] PSTART(〈Thread Number〉)

Description

This executes the program loaded on thread number of specified robot starting from the first line.

1) Entering the number in 〈Thread Number〉 selects a JOB program to execute.

Return value

〈Variable〉 Data type: 〈Integer Type〉
Either 0 (Selection Success) or negative (-) values (Selection Failure) are returned as a result value.

- Cause of Selection Failure

Value	Cause
-1	In case of not SYSTEM Mode of small T/P, or in case that PSEL command was used in Robot Mode
-2	Input Data Error
-3	Unable to execute thread or already in execution

Arguments

〈Thread Number〉 Data type: 〈Integer Type〉
The range of thread number is 1 ~ 4(TH1 ~ TH4).

Examples

```
INT R_VAL                          // Declare an integer variable
PSEL(1, “MOVE”)                    // Select MOVE.JOB in thread of robot 1
R_VAL = PSTART(1)                  // Start to Program Execution matched thread 1
PRINT(0, “RETURN VALUE: ”,R_VAL)   // Store return value in index 0 of monitoring log
WHILE 1                            // Endless Loop Start
ENDWL                              // Endless Loop End
```

Related terms

PSEL, PSTOP, PSTATE, PRESTART, PEXIT

1) This command is only available in System Mode of controller and it creates “Cannot use PSEL at foretask” alarm when used in Robot Mode.
2) Only TH1 (1) can be selected as thread number at the moment.

12.3 PSTOP (Program Execution Stop)

Usage

This stops a JOB program execution corresponding number of robot to input.

Syntax

[〈Variable〉 =] PSTOP(〈Thread Number〉)

Description

This stops a JOB program that is executed in thread of specified Robot.

1) Entering the number in 〈Thread Number〉 stops the JOB program in execution.

Return value

〈Variable〉 Data type: 〈Integer Type〉
Either 0 (Stop Success) or negative (-) value (Stop Failure) is returned.

- Cause of Stop Failure

Value	Cause
-1	In case of not SYSTEM Mode of small T/P, or in case that PSEL command was used in Robot Mode
-2	Input Data Error

Arguments

〈Thread Number〉 Data type: 〈Integer Type〉
The range of thread number is 1 ~ 4(TH1 ~ TH4).

Examples

```
PSEL(1, “MOVE”)          // Select MOVE.JOB in thread of robot 1
PSTART(1)                // Start to Program Execution matched thread 1
WHILE 1                  // Endless Loop Start
    IF IN0 == 1 THEN     // If Input 0 is 1
        PSTOP(1)         // Stop Program matched to thread of robot 1
    ENDIF                // IF statement End
ENDWL                    // Endless Loop End
```

Related terms

PSEL, PSTART, PSTATE, PRESTART, PEXIT

1) This command is only available in System Mode of controller and it creates “Cannot use PSEL at foretask” alarm when used in Robot Mode.
2) Only TH1 (1) can be selected as thread number at the moment.

12.4 PSTATE (Program State Check)

Usage

This reads the status of thread number of specified robot.

Syntax

〈Variable〉 = PSTATE(〈Thread Number〉)

Description

This reads the current status of thread number of specified robot.

1) Entering the number in 〈Thread Number〉 reads the status.

Return value

〈Variable〉

Data type: 〈Integer Type〉
This returns current status information of robot thread.

- 0 (TH_ERROR): Thread Number Error
- 1 (TH_IDLE): No Thread Allocation, New Use Available
- 2 (TH_STOP): Thread Stop
- 3 (TH_RUN): Thread in Operation

A negative (-) value is returned when failure in status check.

- Cause of Failure

Value	Cause
-1	In case of not SYSTEM Mode of small T/P, or in case that PSEL command was used in Robot Mode
-2	Input Data Error

Arguments

〈Thread Number〉

Data type: 〈Integer Type〉
The range of thread number is 1 ~ 4(TH1 ~ TH4).

Examples

```
INT STATE                          // Declare an integer variable
PSEL(1, “MOVE”)                    // Select MOVE.JOB in thread of robot 1
PSTART(1)                          // Start to Program Execution matched thread 1
WHILE 1                            // Endless Loop Start
    STATE = PSTATE(1)              // Status check of thread of robot 1
    IF STATE <0 THEN               // If Failure in Status Check
        PRINT(0,”ERROR:”, STATE)   // Store failure cause in index 0 of monitoring log
        BREAK                      // Forcedly terminate iterative statement
    ENDIF                          // IF statement End
ENDWL                              // Endless Loop End
```

Related terms

PSEL, PSTART, PSTOP, PRESTART, PEXIT

1) This command is only available in System Mode of controller and it creates "Cannot use PSEL at foretask" alarm when used in Robot Mode.
2) Only TH1 (1) can be selected as thread number at the moment.

12.5 PRESTART (Program Re-start)

Usage

This re-starts a JOB program stopped in thread number of specified robot.

Syntax

[<Variable> =] PRESTART(<Thread Number>)

Description

This re-starts the program loaded on thread number of specified robot from the stopped line.

1) Entering the number in <Thread Number> re-starts JOB program.

Return value

<Variable> Data type: <Integer Type>

Either 0 (Execution Success) or negative (-) value (Execution Failure) is returned.

- Cause of Execution Failure

Value	Cause
-1	In case of not SYSTEM Mode of small T/P, or in case that PSEL command was used in Robot Mode
-2	Input Data Error

Arguments

<Thread Number> Data type: <Integer Type>

The range of thread number is 1 ~ 4(TH1 ~ TH4).

Examples

```
INT STATE                                  // Declare an integer variable
PSEL(1, "MOVE")                            // Select MOVE.JOB in thread of robot 1
PSTART(1)                                  // Start to Program Execution matched thread 1
WHILE 1                                    // Endless Loop Start
    STATE = PSTATE(1)                      // Status Check of Thread of Robot 1
    IF ((STATE == 3)&&(IN0 == 1)) THEN     // If State Value is 3(TH_RUN), Input 0 is 1
        PSTOP(1)                           // Program Stop
    ELSEIF ((STATE == 2)&&(IN1 == 1)) THEN // If State Value is 2(TH_STOP), Input 1 is 1
        PRESTART(1)                        // Program Re-start
    ELSEIF ((STATE == 2)&&(IN2 == 1)) THEN // If State Value is 2(TH_STOP), Input 2 is 1
        PEXIT (1)                          // Program Clear
    ENDIF                                  // IF statement End
ENDWL                                      // Endless Loop End
```

Related terms

PSEL, PSTART, PSTOP, PSTATE, PEXIT



1) This command is only available in System Mode of controller and it creates "Cannot use PSEL at foretask" alarm when used in Robot Mode.
2) Only TH1 (1) can be selected as thread number at the moment.

12.6 PEXIT (Program Clear)

Usage

This clears the JOB program in thread number of specified robot.

Syntax

[<Variable> =] PEXIT(<Thread Number>)

Description

This clears the JOB program loaded on thread number of specified robot.

1) Entering the number in <Thread Number> clears the JOB program.

Return value

<Variable> Data type: <Integer Type>
Either 0 (Execution Success) or negative (-) value (Execution Failure) is returned.

- Cause of Execution Failure

Value	Cause
-1	In case of not SYSTEM Mode of small T/P, or in case that PSEL command was used in Robot Mode
-2	Input Data Error

Arguments

<Thread Number> Data type: <Integer Type>
The range of thread number is 1 ~ 4(TH1 ~ TH4).

Examples

```
INT STATE                              // Declare an integer variable
PSEL(1, "MOVE")                        // Select MOVE.JOB in thread of robot 1
PSTART(1)                              // Start to Program Execution matched thread 1
WHILE 1                                // Endless Loop Start
    STATE = PSTATE(1)                  // Status check of thread of robot 1
    IF ((STATE == 2)&&(IN1 == 1)) THEN // If State Value is 2(TH_STOP), Input 1 is 1
        PSTOP(1)                       // Program End
        PEXIT(1)                       // Program Clear
    ENDIF                              // IF statement End
ENDWL                                  // Endless Loop End
```

Related terms

PSEL, PSTART, PSTOP, PSTATE, PEXIT

1) This command is only available in System Mode of controller and it creates "Cannot use PSEL at foretask" alarm when used in Robot Mode.
2) Only TH1 (1) can be selected as thread number at the moment.

13. External Communications

13.1 NET_OPEN (Creating TCP I/P Client Socket and requesting for connection)

Usage

Creates TCP I/P Client Socket and requests for connection to the Server.

Syntax

<Integer Variable> = NET_OPEN(<IP Number>, <Port Number>, <Wait Time>)

Description

Creates TCP I/P Client Socket and requests for connection to the TCP I/P Server relevant to entered IP, Port Number.
Connection up to 3 Ports at the same time is possible, but additional creation is not possible.
Wait as much as Wait Time entered to request for connection to the Server. If connected, returns File descriptor of the relevant Socket. If not connected, returns -1.

1) Input IP Number of the Server in <IP Number>.
2) Input TCP Port Number in <Port Number>.
3) Input <Wait Time> by msec unit. After connecting to the Server, wait response for entered Wait Time.

Return value

<Variable>	Data type: <Integer> If connected, returns File Descriptor (File descriptor) Number of the Socket. If not connected, returns -1.

Arguments

<IP Number>	Data type: <Character String >
<Port Number>	Data type: <Integer> Port Number range is 1 ~ 65,535.
<Wait Time>	Data type: <Integer> Wait Time unit is msec and range is 1 ~ 10,000.

Examples

```
MAIN                                    // Declares Main Function
    INT FD_NUM                          // Declares Return value Integer Variable
    DEFSTR IP                           // Declares IP Number Character String Variable
                                        // Declares Port Number Integer Variable
    INT PORT                            // Declares Wait Time Integer Variable
    INT WAIT_T
    IP = “192.168.1.196”                // Defines IP Number
    PORT = 10000                        // Defines Port Number
```

```
    WAIT_T = 1000                          // Defines Wait Time (1 sec)

    FD_NUM = NET_OPEN(IP,PORT, WAIT_T)     // Requests TCP I/P connection
EOP                                        // Ends Main Function
```

Related terms

NET_CLOSE, NET_STATUS, NET_KEEPALIVE, NET_DELIMITER, NET_FD

13.2 NET_CLOSE (Termination of TCP I/P Communication)

Usage

Ends TCP I/P Communication of the Controller and removes the Client Socket.

Syntax

[<Integer Variable> =] NET_CLOSE(<File Descriptor Number>)

Description

Ends connected TCP I/P Communication of the Controller.

1) Input File Descriptor (File descriptor) Number returned by NET_FD or NET_OPEN command in <File Descriptor Number>.

Return value

<Variable>	Data type: <Integer> 0(end successful) or -1(end failed) value returns as result value.

Arguments

<File Descriptor Number>	Data type: <Integer> File Descriptor Number range is -1 ~ 1,024.

Examples

```
MAIN                                  // Declares Main Function
    INT FD_NUM                        // Declares File Descriptor Number Integer Variable
    DEFSTR IP                         // Declares IP Number Character String Variable
    INT PORT                          // Declares Port Number Integer Variable
    INT WAIT_T                        // Declares Wait Time Integer Variable

    IP = “192.168.1.196”              // Defines IP Number
    PORT = 10000                      // Defines Port Number
    WAIT_T = 1000                     // Defines Wait Time (1 sec)

    FD_NUM = NET_OPEN(IP,PORT, WAIT_T)    // Requests TCP I/P connection

    NET_CLOSE(FD_NUM)                 // Requests TCP I/P end request
EOP                                   // Ends Main Function
```

Related terms

NET_OPEN, NET_STATUS, NET_KEEPALIVE, NET_DELIMITER, NET_FD



1) File Descriptor Number is unique number of the Socket allotted by the system.
2) In case of TCP I/P Server, returned using NET_FD command. In case of Client, returned using NET_OPEN command.

13.3 NET_STATUS (Checking TCP I/P Communication status)

Usage

Checks TCP I/P Communication status of the Controller.

Syntax

<Integer Variable> = NET_STATUS(<File Descriptor Number>)

Description

Returns by checking TCP I/P Communication status of the Controller.

1) Input File Descriptor (File descriptor) Number returned by NET_FD or NET_OPEN command in <File Descriptor Number>.

Return value

<Variable>	Data type: <Integer> 0(end successful) or -1(Error or connection failed) value returns as result value.

Arguments

<File Descriptor Number>	Data type: <Integer> File Descriptor Number range is -1 ~ 1,024.

Examples

```
MAIN                                  // Declares Main Function
    INT FD_NUM                        // Declares File Descriptor Number Integer Variable
    DEFSTR IP                         // Declares IP Number Character String Variable
    INT PORT                          // Declares Port Number Integer Variable
    INT WAIT_T                        // Declares Wait Time Integer Variable
    IP = "192.168.1.196"              // Defines IP Number
    PORT = 10000                      // Defines Port Number
    WAIT_T = 1000                     // Defines Wait Time (1 sec)

    FD_NUM = NET_OPEN(IP,PORT, WAIT_T)  // Requests TCP I/P connection
    RET_VEL = NET_STATUS(FD_NUM)        // Requests TCP I/P status information
EOP                                   // Ends Main Function
```

Related terms

NET_OPEN, NET_CLOSE, NET_KEEPALIVE, NET_DELIMITER, NET_FD



1) File Descriptor Number is unique number of the Socket allotted by the system.
2) In case of TCP I/P Server, returned using NET_FD command. In case of Client, returned using NET_OPEN command.

13.4 NET_KEEPALIVE (Option setting to check the TCP I/P Client Session)

Usage

Periodically check Communication status during TCP I/P Communication to detect disconnection of the LAN Cable or abnormal session end.

Syntax

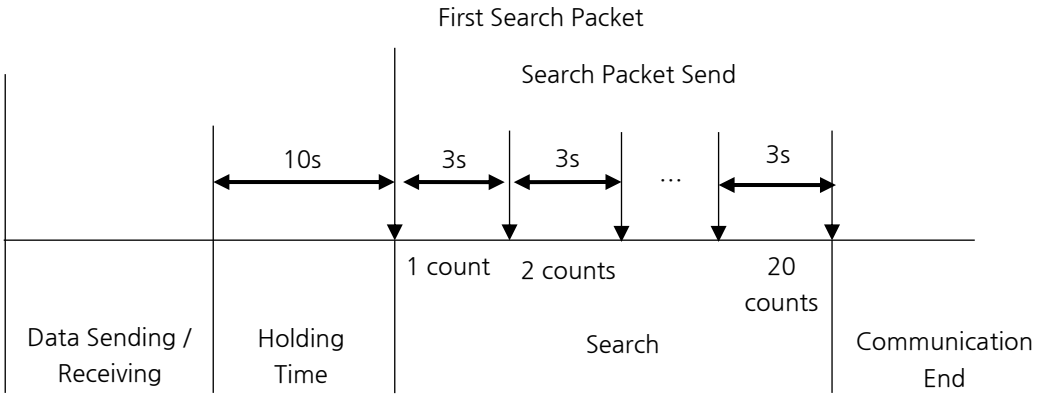
〈Integer Variable〉 = NET_KEEPALIVE(〈File Descriptor Number〉, 〈Holding Time〉, 〈Max search count〉, 〈Search period〉)

Description

If 〈Holding Time〉 exceeded after receiving the last Packet, send Search Packet and wait for 〈Search period〉. In case of no response, send Search Packet at each 〈Search period〉. In case of no response for 〈Max search count〉, ends Communication.
10 sec, 20 counts, 3 sec values are set as default values. Change setting values using NET_KEEPALIVE command.

1) Input File Descriptor (File descriptor) Number returned by NET_FD or NET_OPEN command in 〈File Descriptor Number〉.
2) 〈Holding Time〉 is from the time when the last Packet is transmitted to first point when Search Packet is started to be sent.
3) 〈Max search count〉 is max count to repeatedly send Search Packet in case of no 는 Response of the Search Packet.
4) 〈Search period〉 is interval send Search Packet.

EX) If 〈Holding Time: 10 sec〉, 〈Max search count: 20〉 and 〈Search period: 3 sec〉 are entered, If no data is received for 10 sec after receiving the last Packet, sends Search Packet. In case of no response, sends Search Packet at 3 sec period. If Search Packet is sent for 20 counts and Max search count is exceeded, ends Communication.



Return value

〈Variable〉 Data type: 〈Integer〉
0(Execution successful) or -1(Execution failed) returns as result value.

Arguments

〈File Descriptor Number〉	Data type: 〈Integer〉 File Descriptor Number range is -1 ~ 1,024.
〈Holding Time〉	Data type: 〈Integer〉 Holding Time unit is sec and range is 1 ~ 10,000.
〈Max search count〉	Data type: 〈Integer〉 Repeat count unit is integer and range is 1 ~ 10,000.
〈Search period〉	Data type: 〈Integer〉 Repeat period unit is sec and range is 1 ~ 10,000.

Examples

```
MAIN                                    // Declares Main Function
    INT RET_V                           // Declares Return value Integer Variable
    INT FD                              // Declares File Descriptor Number Integer Variable
    DEFSTR IP                           // Declares IP Number Character String Variable
    INT PORT, WAIT_T                    // Declares Integer Variable
    INT T_IDLE, T_CNT, T_INTV

    IP = "192.168.1.196"                // Defines IP Number
    PORT = 10000                        // Defines Port Number
    WAIT_T = 1000                       // Defines Wait Time (1 sec)

    FD = NET_OPEN(IP,PORT, WAIT_T)      // Requests TCP I/P connection

    T_IDLE = 10                         // Defines Holding Time to 10 sec
    T_CNT = 20                          // Defines Max search count to 20 counts
    T_INTV = 3                          // Defines Search period to 3 sec

    RET_V                           =   // Executes Function
NET_KEEPALIVE(FD,T_IDLE,T_CNT,T_INTV)
EOP                                     // Ends Main Function
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_DELIMITER, NET_FD

1) File Descriptor Number is unique number of the Socket allotted by the system.
2) In case of TCP I/P Server, returned using NET_FD command. In case of Client, returned using NET_OPEN command.

13.5 NET_FD (TCP I/P Server FD Return)

Usage

Returns File Descriptor (File descriptor) Number of Client Socket connected to the Controller TCP I/P Server.

Syntax

〈Integer Variable〉 = NET_FD(〈Index〉)

Description

Returns File Descriptor Number of the Client Socket connected to the TCP I/P Server Port relevant to 〈Index〉.
If connected, returns the relevant File Descriptor Number of the Client Socket. If not connected, returns -1.

1) 〈Index〉 range is 10 ~ 12 and each meaning is as below.
   10: Port Number 38201
   11: Port Number 38202
   12: Port Number 38203

Arguments

〈Index〉	Data type: 〈Constant, Integer Variable〉 〈Index〉 range is 10 ~ 12.

Examples

```
MAIN                                // Declares Main Function
    INT FD_NUM                      // Declares Return value Integer Variable
    INT INDEX                       // Declares Index value Integer Variable

    INDEX = 10                      // Sets Index value

    FD_NUM = NET_FD(FD_NUM)         // Executes Function
EOP                                 // Ends Main Function
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_KEEPALIVE, NET_DELIMITER

13.6 NET_STRIN (Receiving Communication String)

Usage

Reads Character String input through Ethernet Communication.

Syntax

<Character String Variable> = NET_STRIN(<File Descriptor Number>, <Wait Time>)

Description

Waits for Communication input as much as entered Wait Time (unit: ms). If no data is received for setting Wait Time, check wait status in system area and execute Command of the next Step.

Data input using NET_STRIN Function uses CR+LF (default setup) or CR, LF Character to decide End of Character String. This is called as newline. Newline must be at the end of Character String. In case of no newline, end of input can't be decided.

1) Input File Descriptor (File descriptor) Number returned by NET_FD or NET_OPEN command in <File Descriptor Number>.
2) <Wait Time> sets Receiving Wait Time.

Return value

<Variable>	Data type: <Character String> Returns Character String received for Receiving Wait Time.

Arguments

<File Descriptor Number>	Data type: <Integer> File Descriptor Number range is -1 ~ 1,024.
<Wait Time>	Data type: <Integer>

Examples

```
MAIN                                  // Starts Main Function
    DEFSTR RECVPK                     // Declares Character String Variable
    INT FD_NUM                        // Declares Integer Variable
    RECVPK = NET_STRIN(FD_NUM,1000)   // Waits Character String Receive for 1 sec
EOP                                   // Ends Main Function
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_STROUT, NET_KEEPALIVE, NET_FD

13.7 NET_STROUT (Sending Communication String)

Usage

Sends Character String through Serial and Ethernet Communication.

Syntax

〈Variable〉 = NET_STROUT(〈File Descriptor Number〉, 〈Character String〉)

Description

Outputs entered Character String through Serial Communication and returns number of unsent Character. For an example, supposing “ABCDEFGH” Characters are to be sent, return value is 0 if sending is successful. If failed to send all of them, return value will be 8. In addition, returns negative number if transmission fails
Data output by using NET_STROUT Function System adds and sends CR+LF(default setup) or CR, LF Character to decide End of Character String. This is called newline.

1) Input File Descriptor (File descriptor) Number returned by NET_FD or NET_OPEN command in 〈File Descriptor Number〉.
2) Input Character String to be sent in 〈Character String〉.

Return value

〈Variable〉	Data type: 〈Integer〉 If transmission is successfully completed : Returns 0 If transmission is not completed: Returns number of unsent Character If transmission error occurred: Returns negative number

Arguments

〈File Descriptor Number〉	Data type: 〈Integer〉 File Descriptor Number range is -1 ~ 1,024.
〈Character String〉	Data type: 〈Character String〉

Examples

```
MAIN                                    // Starts Main Function
    INT FD_NUM                          // Declares Integer Variable
    DEFSTR SENDPK                       // Declares Character String Variable
    INT LEN                             // Declares Integer Variable
    SENDPK = “SUCCESS”                  // Initializes Send Character String
    I0 = NET_STROUT(FD_NUM,“TEST”)      // Sends “TEST” Character String
    LEN = NET_STROUT(FD_NUM,SENDPK)     // Sends “SUCCESS”Character String
EOP                                     // Ends Main Function
```

Related terms

NET_OPEN, NET_CLOSE, NET_STATUS, NET_STRIN, NET_KEEPALIVE, NET_FD

Chapter 5 Revision

Revision	Date	Revised Detail
1	2019.01.21	Initial Distribution
2	2019.06.19	Add new command - External communication (commands dedicated to TCP I/P Client)
3	2020.01.21	Correct command syntax - Added timeout function for IN, CIN, CBIN instructions - Add RSTATE command status index definition
4	2020.05.07	Correct command syntax - Add RSTATE command status index definition - OUT and COUT instruction specification changes

A. Document Reference

All the literature, which are required for performing services, repair or installation of all robot system that uses this product, are specified in this chapter.
In the ID of all literature, the very first word indicates the controller name and the second word means the abbreviation of corresponding literature. The last indicates language and its version.

Language is marked according to the rules below.

- Korean: K
- English: E
- Chinese: C
- Vietnam: V

Document ID	Description
N2-IM-E□□	**Installation and handling manual** This explains the Controller structure and installation as well as the methods to interface with external devices.
N2-OM-E□□	**Operation manual** This explains the method to use the Controller and Teach Pendant, parameter setting, JOB program editing and additional functions.
N2-PM-E□□	**Programming manual** This explains the method to create RRL (Robostar robot language) that is the Robostar Robot program and describes the commands.
N2-HM-E□□	**Unihost manual** This explains about Unihost that is Robostar on-line PC program.
N2-AM-E□□	**Alarm and maintenance manual** This explains the information on problems occurred in the Controller-based Robot system as well as solutions and procedure for the problems.

B. Hazard Stages & Signs

Sign	Designation	Meaning
	DANGER	This warns that deadly and serious injuries or serious product damages can be caused from the accidents unless complying the guidelines.
	WARNING	This warns that accidents may occur unless complying the guidelines and it leads to deadly and serious injuries or serious product damages can be caused from the accidents.
	Caution	This warns that accidents may occur unless complying the guidelines and it leads to serious product damages can be caused from the accidents.
	ELECTRICAL SHOCK	This sign indicates a hazard on electric shock which may lead to serious or deadly injuries.
	NOTE	This sign notices important facts and conditions.
	Prohibition	This sign notices the prohibitions for normal operation.

C. Command List (Alphabetical Order)

Num.	Command	Description	Ref.
1	ABS	This is a function that returns the absolute value of Input.	10.15
2	ACC	This sets the percentage (%) of acceleration time.	7.3
3	ACOS	This returns the result value of Arc CoSine Function.	10.27
4	AMOV	This moves in arc passing through the Via point 1, 2 from current position.	6.6
5	ASC	This returns the Integer value of character code corresponding to the first character of String.	10.34
6	ASIN	This returns the result value of Arc Sine Function.	10.25
7	ATAN	This returns the result value of Arc Tangent Function.	10.29
8	ATAN2	This returns the result value of the 2nd Arc Tangent Function.	10.30
9	BINS	This is a function that converts the Integer number to Binary String.	10.35
10	BREAK	This is used to escape from a loop of Iterative statement.	5.9
11	BTOJ	This calculates Joint Coordinate System Position value from Base Coordinate System Position value.	10.4
12	BTOU	This calculates User Coordinate System Position value from Base Coordinate System Position value.	10.7
13	CBIN	This reads the value of specified Byte Number in Field Bus Card.	8.7
14	CBOUT	This outputs the specified value to Byte Number of Field Bus Card.	8.8
15	CDIN	Reads value of designated Double Word Number in an integer form from the Fieldbus Card.	8.9
16	CDOUT	Outputs designated value in an integer form at the designated Fieldbus Card Double Word Number.	8.10
17	CFIN	This reads the value of specified Double Word Number in Field Bus Card.	8.13
18	CFOUT	This outputs the specified value to Double Word Number of Field Bus Card.	8.14
19	CHR	This returns ASCII Code character for Integer value inputted.	10.36
20	CIN	This reads the value of specified Bit Number in Field Bus Card.	8.5
21	CLOG	※ **Not currently supported.**	11.19
22	CMOV	This moves in a circle passing through the Via point 1, 2 from current position.	6.5
23	Comment	This do process so as not to have any effect on the execution.	3.2
24	CONTINUE	This is used when branching off to the Processing statement and IF statement after processing the Iterative statement without executing the next statement within the Iterative Loop.	5.8
25	COS	Thus returns the result value of Cosine Function.	10.26

Num.	Command	Description	Ref.
26	COUT	This outputs the specified value to Bit Number of Field Bus Card.	8.6
27	CTHREAD	This executes Function and Program simultaneously.	9.1
28	CWIN	This reads the value of specified Word Number in Field Bus Card.	8.11
29	CWOUT	This outputs the specified value to Word Number of Field Bus Card.	8.12
30	DEC	This sets the percentage (%) of deceleration time.	7.4
31	DEFSTR	This declares String Variable that can be used in the system inside.	2.3
32	DEG	This is a function that converts Radian (circular method) to Angle (hexadecimal method).	10.16
33	DLAY	This temporarily pauses the program execution for specified duration.	7.6
34	ETHREAD	This ends Thread forcedly.	9.2
35	EXIT	This stops the execution of JOB program.	5.12
36	EXP	This returns the result value of the Exponential Function. This is the Inverse Function of Logarithm Function.	10.20
37	F	This is a Real Number Global Variable that was declared in advance in the system and can be commonly used in the entire program.	2.9
38	FLUSH	This clears the Input and Output Buffers of Serial Communication.	10.37
39	FOR … TO … [BY] … NEXT	This executes the block repeatedly until the condition becomes False.	5.4
40	FOS	This alters a trajectory to the next target before axis end reaches the target point.	7.5
41	FTOS	This converts Integer or Real Number into String.	10.38
42	FUNC … END	This represents the start and end of Function.	5.2
43	GAIN	Converts Gain Group.	7.11
44	GBASEDISP	Displacement based on Base Coordinate System between 2 Coordinates.	10.12
45	GP	This is a Joint Coordinate System Position Variable that was declared in advance in the system and can be commonly used in the entire program.	2.10
46	GTOOLDISP	Calculates displacement based on Tool Coordinate System between 2 Coordinates.	10.13
47	GUSERDISP	Calculates displacement based on User Coordinate System between 2 Coordinates.	10.14
48	HERE	This is a Variable that was declared in advance in the system and converts current position into Joint Angle value.	2.13

Num.	Command	Description	Ref.
49	HERE_REF	This is a Variable that was declared in advance in the system and converts current reference position into Joint Angle value.	2.14
50	HTOS	This converts Integer into hexadecimal Binary String.	10.39
51	I	This is an Integer Global Variable that was declared in advance in the system and can be commonly used in the entire program.	2.8
52	IF …THEN … ENDIF	This executes the block as the operation result of conditional expression.	5.6
53	IN	This reads the value of specified Bit Number in 1 bit unit.	8.1
54	INCLUDE	This enables to use the specific program file as one file by adding it to current position.	3.1
55	INT	This declares an Integer Variable that can be used in the program inside.	2.1
56	IPE	Sets Max Allowable Time value until reaching target position.	7.12
57	JCALL	This calls a JOB file and immediately execute it.	5.10
58	JMOV	This moves to target position from current position.	6.3
59	JTOB	This calculates Base Coordinate System Position value from Joint Coordinate System Position value.	10.3
60	JTOU	This calculates User Coordinate System Position value from Joint Coordinate System Position value.	10.5
61	LABL … GOTO	This is a command that specifies the branch-off position and moves to corresponding position.	5.7
62	LIMT	This limits the movement range of each axis.	7.9
63	LMOV	This performs a straight interpolation movement to target position from current position.	6.4
64	LN	This returns the result value of the Natural Logarithm.	10.21
65	LOG	This returns the result value of the Common Logarithm.	10.22
66	MAIN … EOP	This shows a start and end of main program.	5.1
67	MAX	This returns the max value after comparing two input values.	10.32
68	MIN	This returns the min value after comparing two input values.	10.31
69	MVR	This is a Variable that was declared in advance in the system and returns the moving distance ration of entire movement distance.	2.12
70	NET_CLOSE	Ends TCP I/P Communication connected to the Controller.	13.2
71	NET_FD	Returns FD of Client Socket connected to the Controller TCP I/P Server.	13.5
72	NET_KEEPALIVE	Periodically check session during TCP I/P Communication to detect disconnection or abnormal session end of the LAN Cable.	13.4
73	NET_OPEN	Creates TCP I/P Client Socket and requests for connection to the Server.	13.1
74	NET_STATUS	Checks TCP I/P Communication status of the Controller.	13.3

Num.	Command	Description	Ref.
75	NET_STRIN	Reads Character String input through Ethernet Communication.	13.6
76	NET_STROUT	Sends Character String through Ethernet Communication.	13.7
77	OFFS	This moves the target position by considering the specified value.	7.7
78	OFFSCLR	This clears the specified offset value.	7.8
79	Operators	These are symbols that instructs the program to perform mathematical or logical operations.	4
80	OUT	This output the specified value to the specified Bit Number.	8.2
81	P	This is a Joint Coordinate System Position Variable that was declared in advance in the system. It is stored in the program file and is used as a Global Variable in the corresponding program.	2.7
82	PEXIT	This clears JOB program of Thread Number in specified Robot.	12.6
83	PIN	This reads the value of Port Number specified by port.	8.3
84	POS	This declares a Joint Coordinate System Position Variable that can be used in the program inside.	2.4
85	POUT	This outputs the specified value to the specified Port Number.	8.4
86	POW	This is a function that is used to obtain the Square Root value.	10.18
87	PRESTART	This re-executes the JOB program of Thread Number in the specified Robot.	12.5
88	PRINT	This records a log for monitoring.	11.17
89	PSEL	This matches the JOB program inputted in the thread Number in the specified Robot.	12.1
90	PSTART	This executes the job of the Thread Number in the specified Robot.	12.2
91	PSTATE	This reads the state of Thread Number in the specified Robot.	12.4
92	PSTOP	This stops the job of Thread Number in the specified Robot.	12.3
93	RAD	This is a function that converts Angle (hexadecimal method) into Radian (circular method).	10.17
94	RAND	This creates the random number values.	10.33
95	READ_CTEMP	This reads Internal Temperature of controller.	11.16
96	READ_DERR	This returns alarm code of Driver corresponding the specified axis.	11.7
97	READ_LOAD	This reads current Load Rate of each axis in Robot.	11.11
98	READ_MTEMP	This reads current Motor Encoder Temperature value of each axis in Robot.	11.15
99	READ_RPM	This reads current RPM value of each axis in Robot.	11.10
100	READ_TRQ	This reads current Torque value of each axis in Robot.	11.14
101	READ_TRQMAX	This reads max Torque value of each axis in Robot.	11.12
102	READ_VEL	This reads current velocity of each axis in Robot.	11.13
103	REAL	This declares Real Number variable that can be used program inside.	2.2

Num.	Command	Description	Ref.
104	RELBASE	Calculates Relatively moved position value based on Base Coordinate System information as much as Displacement substituted at Reference Position.	10.9
105	RELTOOL	This is a function that calculates the relative movement position by assigned movement from the reference position based on the tool coordinate system information.	10.10
106	RELUSER	Calculates relatively moved position value based on User Coordinate System information as much as Displacement substituted at Reference Position.	10.11
107	RERRCNT	This returns total number of alarms occurred.	11.4
108	RERRCODE	This returns alarm code corresponding to specified Index Number.	11.5
109	RERROR	This returns the last occurred alarm code.	11.3
110	RERRTEXT	This returns alarm contents corresponding to specified Index Number.	11.6
111	RESET	This clears all alarms occurred.	11.2
112	RET	This returns the value after executing a function and back to the position where the function was called.	5.3
113	RND	This rounds off the decimal part and returns with the Integer value.	10.19
114	RSTATE	This is a command that returns the status of Robot.	11.8
115	SETERR	User creates System emergency alarms arbitrarily.	11.1
116	SIN	This returns the result value of Sine Function.	10.24
117	SLEFT	This extracts the Left side String of the inputted String.	10.40
118	SLEN	This returns the Length of String inputted.	10.41
119	SMID	This extracts String as much as inputted value from specified String.	10.42
120	SPOS	This returns the starting position where String 1 and String 2 match together.	10.43
121	SQRT	This returns the Square Root.	10.23
122	SRIGHT	This extracts the right side string of inputted Strings.	10.44
123	STOP	This stops Motion of Robot.	5.11
124	STRIN	This reads the String inputted through Serial and Ethernet Communication.	10.45
125	STROUT	This sends out String via Serial and Ethernet Communication.	10.46
126	SVAL	This converts String to number.	10.47
127	SVOF	This turns off the power of Servo Motor.	6.2
128	SVON	This turns on the power of Servo Motor.	6.1
129	SWLIMIT	This reads the parameter value that limits movement range of each axis in Robot.	11.9
130	TACT_START	Sets Start position for Tact Time measurement.	11.23

Num.	Command	Description	Ref.
131	TACT_STOP	Sets End position for Tact Time measurement.	11.24
132	TAN	This returns the result value of Tangent Function.	10.28
133	TH_RESUME	This re-executes the stopped Thread.	9.5
134	TH_STATE	This checks current status of Thread.	9.3
135	TH_SUSPEND	This stops the Thread currently in execution.	9.4
136	TMR	This is a timer that was declared in the system in advance. It is a variable for using to specify the count value or to read current count value.	2.11
137	TOOL	This selects the use tool attached to Robot and corresponding coordinate system.	10.1
138	TRQ	This creates alarm and stops if current Torque value is exceeded the limited value.	7.10
139	UPOS	This declares Use Coordinate System Position Variable that can be used in the system inside.	2.6
140	USER	This selects User Coordinate System.	10.2
141	UTOB	This calculates Base Coordinate System Position value from User Coordinate System Position value.	10.8
142	UTOJ	This calculates Joint Coordinate System Position value from User Coordinate System Position value.	10.6
143	VEL	This sets the permiliage (%) of the moving velocity of Axis.	7.2
144	WGFLT	This stores Global Float Variable as a file.	11.22
145	WGINT	This stores Global Integer Variable as a file.	11.21
146	WGPNT	This stores Global Point Variable as a file.	11.20
147	WHILE ··· ENDWL	This executes repeatedly a block until the conditional expression is True.	5.5
148	WITH ··· ENDWT	This treats both Robot Motion and Execution statement in a block simultaneously.	7.1
149	WLOG	This stores a random user log including timing information as a file.	11.18
150	XBIN	Reads value from CNET Receiving Buffer by referring to designated Device and Byte Number.	8.17
151	XBOUT	Outputs value to CNET Device Sending Buffer by referring to the designated Device and Byte Number.	8.18
152	XDIN	Reads value from CNET Receiving Buffer by referring to the designated Device and Double Word Number.	8.21
153	XDOUT	Outputs value to CNET Device Sending Buffer by referring to designated Device and Double Word Number.	8.22
154	XFIN	Reads value in a real number form from the CNET Receiving Buffer by referring to designated Device and Double Word Number.	8.23
155	XFOUT	Outputs real number value to CNET Device Sending Buffer by referring to designated Device and Double Word Number.	8.24
156	XIN	Reads value from CNET Receiving Buffer by referring to designated Device and bit Number.	8.15

Num.	Command	Description	Ref.
157	XOUT	Outputs value from CNET Device Sending Buffer by referring to designated Device and bit Number	8.16
158	XPOS	This declares the Base Coordinate System Position variable that can be used in the program inside.	2.5
159	XWIN	Reads value from CNET Receiving Buffer by referring to designated Device and Word Number.	8.19
160	XWOUT	Outputs value to CNET Device Sending Buffer by referring to the designated Device and Word Number.	8.20

N2 Series controller

Programming Manual
Fourth edition, May 07, 2020

ROBOSTAR CO., LTD.
ROBOT R&D CENTER