ROBOSTAR ROBOT

RcT-Series

PROGRAMMING MANUAL

- ☐ CONTROLLER MANUAL
- ☐ OPERATION MANUAL
- ☑ PROGRAMMING MANUAL
- ☐ UNI-HOST MANUAL
- ☐ ALARM CODE MANUAL

Robostar Co., Ltd

Robostar

www.robostar.co.kr

Contents

CH.1 OVERVIEW 1-1

1.1 OVERVIEW 1-1

CH.2 COMMAND LIST 2-2

2.1 COMMAND LIST 2-2

2.1.1 Program Control Commands 2-2

2.1.2 Robot Operation Commands 2-4

2.1.3 Input/Output Commands 2-5

2.1.4 Commands Related to Robot Operating Conditions 2-8

2.1.5 Variable Declaration 2-9

2.1.6 System Variables 2-9

2.1.7 Constant 2-9

2.1.8 Operator 2-10

2.1.9 String 2-11

2.1.10 THREAD Commands 2-13

2.1.11 LOG Commands 2-13

2.1.12 LATCH Commands 2-14

2.1.13 ERROR Commands 2-15

2.1.14 MAPPING Commands (RS422 Communication Type only) 2-15

CH.3 COMMAND INTERPRETATION 3-16

3.1 FUNC, END (FUNCTION DECLARATION/END) 3-16

3.1.1 Examples of Program Usage 3-17

3.2 INCLUDE 3-18

3.3 FOR, END (REPEATED EXECUTABLE STATEMENT) 3-19

3.3.1 Examples of Program Usage 3-19

3.4 WHILE, END (STATEMENT FOR CONDITION REPEAT RUN) 3-21

3.4.1 Examples of Program Usage 3-21

3.4.2 Examples of Conditional Expression Usage 3-22

3.5 CONTINUE, BREAK(REPEAT STATEMENT CONTROL) 3-23

3.5.1 Examples of Program Usage 3-23

3.6 IF, END (CONDITION BRANCH COMMANDS) 3-24

3.6.1 Examples of Program Usage 3-25

3.6.2 IF(Condition) THEN *3-26*
3.7 LABL, GOTO (BRANCH COMMANDS) 3-27
3.7.1 Examples of Program Usage *3-27*
3.8 STOP, EXIT (ROBOT AND JOB STOP COMMANDS) 3-29
3.8.1 Examples of Program Usage *3-29*
3.9 CTHREAD (THREAD CREATING COMMAND) 3-30
3.9.1 Examples of Program Usage *3-30*
3.10 ETHREAD (THREAD END COMMAND) 3-31
3.10.1 Examples of Program Usage *3-31*
3.11 TH_STATE(THREAD STATUS COMMAND) 3-32
3.11.1 Examples of Program Usage *3-32*
3.12 TH_SUSPEND(THREAD SUSPEND COMMAND) 3-34
3.12.1 Examples of Program Usage *3-34*
3.13 TH_RESUME(THREAD RESUME COMMAND) 3-34
3.13.1 Examples of Program Usage *3-35*
3.14 TAKE (ROBOT SELECT COMMAND) 3-36
3.14.1 Examples of Program Usage *3-36*
3.15 RELEASE(ROBOT RELEASE COMMAND) 3-37
3.15.1 Examples of Program Usage *3-37*
3.16 JMOV (PTP MOVEMENT COMMAND) 3-38
3.16.1 Examples of Program Usage *3-38*
3.17 WITH, ENDWT (CONCURRENT PROCESSING COMMAND) 3-40
3.17.1 Examples of Program Usage (1) *3-40*
3.18 OUT, POUT (EXTERNAL OUTPUT COMMAND) 3-42
3.18.1 Examples of Program Usage *3-45*
3.19 IN, PIN (EXTERNAL INPUT COMMAND) 3-46
3.19.1 Examples of Program Usage *3-48*
3.20 BIN (EXTERNAL INPUT COMMAND) 3-49
3.21 BOUT (EXTERNAL OUTPUT COMMAND) 3-50
3.22 WIN(EXTERNAL INPUT COMMAND) 3-51
3.23 WOUT (EXTERNAL OUTPUT COMMAND) 3-52
3.24 DIN(EXTERNAL INPUT COMMAND) 3-53
3.25 DOUT (EXTERNAL OUTPUT COMMAND) 3-54
3.26 OUTPUT COMMAND FOR FIELD BUS 3-55
(COUT, CBOUT, CWOUT, CDOUT, CFOUT) 3-55
3.27 INPUT COMMAND FOR FIELD BUS (CIN, CBIN, CWIN, CDIN, CFIN) 3-56
3.28 VEL(AXIS VELOCITY SETUP COMMANDS) 3-57

3.28.1 Examples of Program Usage *3-57*
3.29 ACC, DEC (ACCELERATION/DECELERATION SETUP COMMAND) 3-58
3.29.1 Examples of Program Usage *3-59*
3.30 FOS (CONTINUOUS TRAJECTORY GENERATION COMMAND) 3-60
3.30.1 Examples of Program Usage *3-61*
3.31 SVON, SVOF (SERVO ON/OFF COMMAND) 3-63
3.31.1 Examples of Program Usage *3-63*
3.32 DLAY (TIME DELAY COMMAND) 3-65
3.32.1 Examples of Program Usage *3-65*
3.33 RSTATE (ROBOT STATUS CHECK COMMAND) 3-66
3.33.1 Examples of Program Usage *3-66*
3.34 DIST(COMMAND FOR DIFFERENCE OF TWO POSITION VARIABLES) 3-67
3.34.1 Examples of Program Usage *3-67*
3.35 SWLIMIT(COMMAND FOR READING SWLIMIT VALUE SET TO PARAMETER) 3-68
3.35.1 Examples of Program Usage *3-68*
3.36 GETLP(COMMAND FOR READING LOCAL POINT) 3-69
3.36.1 Examples of Program Usage *3-69*
3.37 VARIABLES 3-70
3.37.1 Types of Variables *3-70*
3.37.2 How to Use *3-70*
3.38 INTEGER TYPE(INT, GI), REAL TYPE(DOUBLE, GF), STRING VARIABLES 3-71
3.38.1 LOCAL Variable *3-71*
3.38.2 Examples of Program Usage *3-71*
3.38.3 GLOBAL Variable *3-73*
3.38.4 Examples of Program Usage *3-73*
3.38.5 Integer-type GLOBAL Variable GI *3-74*
3.38.6 Examples of Program Usage *3-74*
3.38.7 Real-type GLOBAL Variable GF *3-76*
3.38.8 Examples of Program Usage *3-76*
3.39 POSITION VARIABLE 3-77
3.39.1 POS Variable *3-77*
3.39.2 POINT Variable *3-78*
3.39.3 Examples of Program Usage *3-78*
3.40 SYSTEM VARIABLES (CNT, TMR, MVR, HERE) 3-79
3.40.1 CNT, TMR Variables *3-79*
3.40.2 Examples of Program Usage *3-79*
3.40.3 MVR Variable *3-80*

3.40.4 Examples of Program Usage *3-81*
3.40.5 HERE Variable *3-82*
3.40.6 Examples of Program Usage *3-82*
3.41 CONSTANT 3-83
3.41.1 Examples of Program Usage *3-83*
3.42 OPERATOR 3-84
3.42.1 Assignment Operator *3-84*
3.42.2 Arithmetic Operator *3-84*
3.42.3 Relational Operator *3-85*
3.42.4 Logical Operator *3-86*
3.42.5 Bit Operator *3-86*
3.43 BUILT-IN FUNCTION 3-87
3.44 STRING 3-88
3.44.1 ASC *3-88*
3.44.2 BINS *3-89*
3.44.3 CHR *3-90*
3.44.4 FLUSH *3-91*
3.44.5 FTOS *3-92*
3.44.6 HTOS *3-93*
3.44.7 SLEFT *3-94*
3.44.8 SLEN *3-95*
3.44.9 SMID *3-96*
3.44.10 SPOS *3-97*
3.44.11 SRIGHT *3-98*
3.44.12 STRIN *3-99*
3.44.13 STROUT *3-100*
3.44.14 SVAL *3-101*
3.45 LOG COMMAND 3-102
3.45.1 PRINT *3-102*
3.45.2 WLOG *3-103*
3.46 LATCH COMMAND (ETHERCAT COMMUNICATION TYPE) 3-104
3.46.1 LATCH_INIT *3-104*
3.46.2 LATCH_CLEAR *3-106*
3.46.3 LATCH_START *3-107*
3.46.4 LATCH_STOP *3-108*
3.46.5 LATCH_POS *3-109*
3.46.6 LATCH_STATE *3-110*

3.47 LATCH COMMAND (RS422 COMMUNICATION TYPE) 3-111
3.47.1 LATCH_INIT 3-111
3.47.2 LATCH_CLEAR 3-113
3.47.3 LATCH_START 3-115
3.47.4 LATCH_STOP 3-116
3.47.5 LATCH_POS 3-117
3.47.6 LATCH_STATE 3-119
3.48 ERROR COMMAND 3-121
3.48.1 SETERR (SYSTEM EMG ALARM COMMAND) 3-121
3.48.2 RESET(ALARM CLEAR COMMAND) 3-122
3.48.3 RERROR(ERROR CODE RETURN COMMAND) 3-123
3.48.4 RERRCNT(Error Occurrence Count Return Command) 3-124
3.48.5 RERRCODE(Error Code Return Command) 3-125
3.48.6 RERRTEXT(Error Content Return Command) 3-126
3.49 MAPPING COMMAND (RS422 COMMUNICATION TYPE ONLY) 3-127
3.49.1 MAP_INIT 3-127
3.49.2 MAP_START 3-128
3.49.3 MAP_STOP 3-129
3.49.4 MAP_READ 3-130

CH.1 Overview

1.1 Overview

i. A robot language is a dedicated language created by Robostar and refers to a command used in writing a robot job program within the system.

ii. A robot language is largely divided into 4 groups

- FLOW Group

  Consists of commands that configure program conditions or unconditional branch, repeat count and sub-program call.

- MOVE Group

  Consists of commands that configure robot operating conditions

- I/O Group

  Consists of commands in control (16bit, 1bit) of external input/outputs.

- COND(=Condition) Group

  Consists of commands that configure speed and delay time.

Use an appropriate command suited for work purposes to write a desired job program.

CH.2 Command List

2.1 Command List

2.1.1 Program Control Commands

Command	Function	Format	Usage Example
INCLUDE	JOB Include	include 〈job name〉	Include "aa"
FUNC	Function Declare	func void main() : end	func void main() vel 20 jmov lp[1] jmov lp[2] end
VOID	Function Return Value Void		
END	Function End		
FOR	Step Repeat Run	for 〈variable〉=〈default value〉 to 〈end value〉 [step〈incremental amount〉] : end	: for a=1 to 5 step 2 vel 20 jmov lp[1] end :
TO			
STEP			
END			
WHILE	Condition Repeat Run	while〈conditional expression〉 : end	: while in(0)==1 jmov lp[1] jmov lp[2] end :
END			
IF	Condition Judgment Run	if〈conditional expression〉 : (elif 〈 conditional expression 〉) : (else) : end	: if in(3)==1 goto A0 elif in(4) == 1 goto B0 else jmov lp[1] end :
ELIF			
ELSE			
END			
LABL	Branch Position Appointment	labl 〈label name〉 : goto 〈label name〉	: labl A1 jmov lp[1] jmov lp[2] goto A1 :
GOTO	Statement Branch		
CONTINUE	Returns to Default Statement	while 〈 conditional expression〉 : continue end	while 1 if in(1) == 1 continue end jmov lp[0] end

Command	Function	Format	Usage Example
BREAK	Repeat Statement End	while 〈 conditional expression 〉 : break end	while 1 if in(1) == 1 **break** end jmov lp[0] end
RETURN	Function End	func int main() : return 0 end	func int add(int a, int b) int c c = a + b **return c** end
STOP	Robot Motion End	stop	
EXIT	JOB Run Stop	exit	

2.1.2 Robot Operation Commands

Command	Function	Format	Usage Example
JMOV	Moves PTP from current position to target point	jmov ⟨point variable⟩, ⟨FOS⟩, ⟨VEL⟩, ⟨ACC⟩, ⟨DEC⟩ ▶ FOS, VEL, ACC, DEC, optional	func void main(0 vel 20 jmov lp[1] jmov lp[2] end
SVON	Servo ON	svon(robot) → for all axes svon(robot, assigned axes) → 1(X), 2(Y), 3(Z), 4(W)	svon(1) svof(2, 2) svon(2, 2)
SVOF	Servo OFF	svof(robot) → for all axes svof(robot, assigned axes) → 1(X), 2(Y), 3(Z), 4(W)	
WITH	Concurrent processing of the following statement column while robot is running	with : end	: with jmov lp[1] mvr=0 while mvr⟨60 if in(1)==1 out(0)=1 end end end :
END			

2.1.3 Input/Output Commands

Command	Function	Format	Usage Example
IN	Refers to 1 bit unit, reading ON(=1), OFF(=0) values in the assigned bit number.	〈variable〉=in(input bit number) - Save the assigned bit input status value to the variable	int aa aa=in(0)
PIN	Refer to the port unit, reading the value of assigned input ports.	〈variable〉=pin(input port number) 〈Input port number〉 : Select among 0 ~ 2 0 → IN0 ~ IN15 1 → IN16 ~ IN31 2 → IN32 ~ IN47	int aa aa=pin(0)
OUT	Refers to 1 bit unit, turning the output of assigned bit number ON(=1), OFF(=0)	Out(output bit number)=〈0/1〉, [pulse valid time], [→] [pulse valid time] unit : 10ms Without [pulse valid time] : Output signal remains valid Surpassing [pulse valid time] : Returns to the previous status [→] : Outputs cycle waveform with Pulse valid time as cycle	out(0)=1 out(0)=1, 100 out(0)=1, 100, →
POUT	Refers to the port unit, issuing the assigned value to the assigned port.	pout(Output port number)=〈 Assigned output value 〉 〈 Output port number 〉: Select among 0 ~ 2 0 → OUT0 ~ OUT15 1 → OUT16 ~ OUT31 2 → OUT32 ~ OUT47	pout(0)=0 pout(0)=20 pout(1)=0H000F
BIN	Refers to the byte unit, reading the value of assigned byte.	〈Variable〉 = bin(Input byte number)	int aa aa = bin(1)
BOUT	Refers to the byte unit, outputting the assigned value with the assigned byte	bout(Output byte number) = 〈Assigned output value〉	bout(1) = 10
WIN	Refers to the word unit, reading the value of the assigned word.	〈variable〉 = win(Input word number〉	int aa aa = win(1)
WOUT	Refers to the word unit, outputting the assigned value with the assigned word.	wout(output word number) = 〈Assigned output value〉	wout(1) = 0hFFFF
DIN	Refers to the double-word unit, reading the value of the assigned double-word.	〈Variable〉 = din(input double-word number〉	int aa aa = din(1)

DOUT	Refers to the double-word unit, outputting the assigned value with the assigned double-word,	dout(output double-word number) = 〈Assigned output value〉	dout(0) = 0hFFFFFFFF

Command	Function	Format	Usage Example
CIN	Reads cclink input by bit.	〈variable〉=cin(Input bit number) - Save the assigned bit input status value to the variable	int aa aa = cin(0)
COUT	Outputs in bit unit to cclink.	cout(output bit number)=〈0/1〉	cout(1) = 1
CBIN	Reads cclink input by byte.	〈variable〉=cbin(input port number)	int aa aa = cbin(1)
CBOUT	Outputs in byte unit to cclink.	cbout(output port number)=〈assigned output value〉	cbout(0) = 123
CWIN	Reads cclink input by word.	〈variable〉=cwin(input port number)	int aa aa = cwin(1)
CWOUT	Outputs in word unit to cclink	cwout(output port number)=〈assigned output value〉	cwout(0) = 1234
CDIN	Reads cclink input by double-word	〈variable〉=cdin(input port number)	int aa aa = cdin(1)
CDOUT	Outputs in double-word unit to cclink.	cdout(output port number)=〈assigned output value〉	cdout(0) = 1234
CFIN	Reads cclink input by float	〈variable〉=cfin(input port number)	double da da = cfin(1)
CFOUT	Outputs in float unit to cclink	cfout(output port number)=〈assigned output value〉	cfout(0) = 123.56

2.1.4 Commands Related to Robot Operating Conditions

Command	Function	Format	Usage Example
VEL	Sets permillage in axis movement speed	vel 〈 Permillage value 〉 Axis movement speed =rated speed X 0.01 X Level X 10 **Rated speed is set by MOTION parameter.**	func void main() vel(1, 200) acc 70 dec 70 jmov lp[1] vel 100 jmov lp[2] acc 100 dec 100 end
ACC	Sets the permillage of acceleration time	acc 〈precentage(%)〉 = Acceleration time = Rated acceleration time X Percentage X 0.01 **Rated acceleration time is set by MOTION parameter.**	
DEC	Sets the permillage of deceleration time	dec 〈precentage(%)〉 Deceleration time = Rated deceleration time X percentage X 0.01 **Rated deceleration time is set by MOTION parameter**	
FOS	Changes the trajectory to the next target point before axis end arrives at the target point.	fos 〈Distance rate (%)〉 Distance rate is the percentage (%) of the entire movement distance Applies to JMOV.	fos 5 jmov lp[1] fos 0(Disable)
DLAY	Set delay time	dlay 〈Delay time〉 Unit of Delay time is 10ms (When Delay time is 500, it means 5 second delay)	jmov lp[1] dlay 20 jmov lp[2]

Command	Function	Format	Usage Example
RSTATE	Reads the robot state.	〈Variable〉 = rstate(robot id, state) ▶ state 1 : ALARM state 6 : SERVO ON/OFF state 9 : AUTO/MANUAL mode state 11 : EMG state	int r_st r_st = rstate(1, 6)
SWLIMIT	Reads swlimit value set to parameter.	〈variable〉 = swlimit(robot id, axis id, limit type) ▶ limit type 0 : low 1 : high	double limit_low limit_low = swlimit(1, 1, 0)
GETLP	Reads local point data.	〈Variable〉 = getlp(robot id, local point index)	pos t_lp t_lp = getlp(1, 0)

2.1.5 Variable Declaration

Command	Function	Format	Usage Example
INT	Declares integer-type variables	int 〈variable name〉, double 〈 variable name 〉, pos 〈 variable name 〉, string 〈 variable name 〉 A variable name consists of uppercase alphabet letters and numbers. A variable name may not start with a number.	func void main() int n double a,b pos mm string sa
DOUBLE	Declares real-type variables		
POS	Declares position-type variables		
STRING	Declares string variables		

2.1.6 System Variables

Command	Function	Format	Usage Example
CNT(n) TMR(0) TMR(1)	System definition variable (Counter variable, timer variable)	cnt(Pulse Input port number)=〈default value〉 TMR(0)=〈default value〉 ▶A counter variable begins to allocate its values from the moment the pulse Input port number is entered and then counts each pulse input. ▶A timer variable begins to allocate its values from the moment an integer is entered and increases by 1 at the defined time interval in the system parameter.	CNT(0)=2 TMR(0)=0 TMR(1)=-50
MVR	MOVE RATE	mvr 〈 Percentage of robot movement zone 〉	if mvr 〈 50
HERE	Current axis angle value variable	here	ap=here

2.1.7 Constant

	Function	Format	Usage Example
	Displays an integer, a real number, a binary integer, and a hexadecimal integer	[0H/0B]〈number〉 When 0H or 0B are not positioned before the number, it is a decimal number 0H : Hexadecimal number, 0B : Binary number	
String Constant	Indicates strings.	A string constant is indicated using “”. The maximum length of a string is less than 100 letters. Ex) “string constant”	

2.1.8 Operator

Command	Function	Format	Usage Example
+, −, *, /, %	Infix operator (Addition, subtraction, multiplication, division, and modulus)		A = B*C
( )	Priority operator		A = B/(A+C)
=	Substitution operator		A = B
&, I , ~, ^, ⟨⟨, ⟩⟩	Bitwise operator (BAND, BOR, Complement, BXOR, left shift, right shift)		
&&, ‖, ^^, !	Logical operator (AND, OR, XOR, Negation)		
⟩, ⟨, ⟩=, ⟨=, !=, ==	Comparision operator(Above, under, over, under, different, same)		
ABS	Takes an absolute value	abs(−10.5) is 10.5.	
DEG	Converts a radian value to an angle value	deg(3.1416) is 180.0.	
RAD	Converts an angle value to a radian value	rad(180.0) is 3.1416.	
POW	Expotential function	pow(2,4) is 16.	
RND	Convets a real−number value to an integer value by rounding off the fixed−point part	rnd(14.8) is 15.	
EXP	Exponent e^x	EXP(3) is 20.085.	
LN	Natural log, $\log_e X$	ln(15) is 2.708.	
LOG	Common log, $\log_{10} X$	log(100) is 2.	
SQRT	Calculates square roots	sqrt(16) is 4.	
SIN	Sine function	sin(rad(30)) is 0.5.	
ASIN	Asin function	asin(0.5) is 0.4794.	
COS	Cosine function	cos(0) is 1.0.	
ACOS	Arcos function	acos(0.5) is 1.0472.	
TAN	Tangent function	tan(rad(45)) is 1.0.	
ATAN	Arctangent function	atan(−1.0) is −0.7854.	
ATAN2	2^{nd} arctangent function	Atan2(Y,X) Atan2(1,−1) is 2.3562.	
MIN	Minimum function	min(1, 2) value is 2.	int mv mv = min(1,2)
MAX	Maximum function	max(1.1, 2.2) value is 2.2.	double db db = max(1.1, 2.2)
DIST	Calculate distance difference in position data		pos tmp tmp = dist(lp[0], lp[1])

2.1.9 String

Command	Function	Format	Usage Example
ASC	Returns the 1st character of a string to a character code	String variable = asc(string)	string aa aa = asc("ABC")
BINS	Converts an integer to a binary string	string variable = bins(integer)	string aa aa = bins(13)
CHR	Converts an integer to a character	string variable=chr(integer) ▶ Ex) aa = chr(65) A character, an ASCII Code corresponding to 65, saved to AA variable.	string aa aa = chr(65)
FLUSH	Clears input, output buffers	flush 〈 Clear buffer select 〉 ▶ Buffer selection range : 1~3 1: Input buffer clear 2: Output buffer clear 3: Input, output buffer clear	
FTOS	Converts an integer or a real number to a string	string variable = ftos(Integer or real number) ▶Converts integers 1234 to string "1234".	string aa aa = ftos(1234)
HTOS	Converts an integer to a hexadecimal string	string variable = htos(Integer) ▶Converts integer 10 to hexadecimal string "A"	string aa aa = htos(10)
SLEFT	Extracts the left part of string	string variable = sleft(String, number) ▶Extracts as many characters as the number from the left side of a string to save to the string variable	string aa aa = sleft("ABCDE",3)
SLEN	Converts a string length	Integer-type variable = slen(string)	int len len = slen("ABCDEF")
SMID	Extracts as many strings as a digit from the assigned position of string	string variable = smid(String, assigned position, extracted number of characters) ▶String position starts from 0. Ex) In "ABCDEFG", A is 0th, B is 1st and, C is 2nd.	string aa //"CDE" string to be saved to AA. aa = smid("ABCDEFG",2,3)
SPOS	Returns a position where string 2 is placed within string 1	Integer-type variable = spos(String1, string2)	int pp //"B" position 1 to be returned. pp = spos("ABCDEF","B")

SRIGHT	Extracts strings in right side	string variable = sright(String, number)
		▶Extracts as many characters as the number from the right side of a string to save to the string variable
			string aa
			"CDE" saved to //AA
			aa = sright("ABCDE",3)

STRIN	Enters a string up to the identifier	string variable = strin(Time-out period) ▶Time-out period takes ms unit.	string aa aa = strin(1000)
STROUT	String output	Integer-type variable = strout(String) ▶Saves characters unable to be transferred to integer-type variables	int rt rt = strout("ABCDEF")

SVAL	Converts a string to a number	Variable = sval(string)	int va double vb va = sval("1234") vb = sval("0.123")

2.1.10 THREAD Commands

Command	Function	Format	Usage Example
CTHREAD	Creates a thread	cthread(Function, thread number) ▶ thread can be used from 1 to 3 only.	int t_ret t_ret = cthread(aa, 1) cthread(bb, 2)
ETHREAD	Ends a thread	ethread(thread number)	ethread(1) ethread(2)
TH_STATE	Reads thread status information	th_state(thread number) Return Value. ▶TH_ERROR : thread number error ▶TH_IDLE : thread available for use ▶TH_STOP : thread at stoppage ▶TH_RUN : thread running	int t_state t_state = th_state(1)
TH_SUSPEND	Stops the running thread	th_suspend(thread number)	th_suspend(1)
TH_RESUME	Runs stopped thread	th_resume(thread number)	th_resume(1)
TAKE	Selects a robot to run from thread	take 〈robot id〉	take 1
RELEASE	Releases the selected robot with take	release	release

2.1.11 LOG Commands

Command	Function	Format	Usage Example
PRINT	Saves a log to memory	print(index, log content) ▶ index can be used from 0 to 999	int t_ret t_ret = cthread(aa, 1) print(1, "1st thread create : ", t_ret)
WLOG	Saves a log in file	wlog(log content)	int t_ret t_ret = cthread(aa, 1) wlog("1st thread create : ", t_ret)

2.1.12 LATCH Commands

1) EtherCAT Type

Command	Function	Format	Usage Example
LATCH_INIT	latch configuration	latch_init(axis, latch method, sensor count) ▶ axis : 1 ~ maximum axes ▶ latch method : 1 : rising edge 2 : falling edge ▶ sensor count : number of sensors to be used in latch (1 ~ 2)	int ret ret = latch_init(3, 1, 2)
LATCH_CLEAR	latch configuration rest	latch_clear(axis)	int ret ret = latch_clear(3)
LATCH_START	Starts latch	latch_start()	int ret ret = latch_start()
LATCH_STOP	Stops latch	Latch_stop()	int ret ret = latch_stop()
LATCH_POS	latch postion	latch_pos(axis, sensor number)	float l_pos[2] l_pos[0] = latch_pos(3, 1) l_pos[1] = latch_pos(3, 2)
LATCH_STATE	latch status	latch_state(axis, sensor number)	int ret ret = latch_state(3, 1) ret = latch_state(3, 2)

2) RS422 Type

Command	Function	Format	Usage Example
LATCH_INIT	latch configuration	int latch_init(int axis, int sensor start number, int sensor count) ▶ axis : 1 ~ maximum axis ▶ sensor start number – Range: 1~4 Range Value / Sensors used: 1 / No. 1, 2; 2 / No. 2, 3; 3 / No. 3, 4; 4 / No. 4 ▶ Sensor count – Range: 1~2 ▶ Return Value - 0: FAIL, 1: SUCCESS	int ret ret = latch_init(3, 1, 2)
LATCH_CLEAR	latch configuration reset	int latch_clear(int axis) ▶ Return Value - 0: FAIL, 1: SUCCESS	int ret ret = latch_clear(3)
LATCH_START	Starts latch	int latch_start() ▶ Return Value - 0: FAIL, 1: SUCCESS	int ret ret = latch_start()
LATCH_STOP	Stops latch	int latch_stop() ▶ Return Value - 0: FAIL, 1: SUCCESS	int ret ret = latch_stop()
LATCH_POS	latch position	double latch_pos(int axis, int sensor Index) ▶ sensor Index – Range: 1~2 ▶ Return Value - double type, Position value	float l_pos[2] l_pos[0] = latch_pos(3, 1) l_pos[1] = latch_pos(3, 2)

LATCH_STATE	latch status	int latch_state(int axis, int sensor Index) ▶ sensor Index - Range: 1~2 ▶ Return Value - 0: FAIL, 1: Unable to detect a signal, 2: signal detected	int ret ret = latch_state(3, 1) ret = latch_state(3, 2)

2.1.13 ERROR Commands

Command	Function	Format	Usage Example
SETERR	Sets an error	seterr(user definition error number)	if in(0) == 1 **seterr(1)** end
RESET	Clears an alarm	reset	reset
RERROR	Returns the last error code	integer-type variable = rerror()	int ecode ecode = rerror()
RERRCNT	Returns the number of currently-generated errors	integer-type variable = rerrcnt()	int ecount ecount = rerrcnt()

RERRCODE	Returns an error code coming under INDEX	integer-type variable = rerrcode(index)
		▶ index : Order of alarm occurrence
			Int ecode
			Ecode = rerrcode(1) //firss error
			//code return

RERRTEXT	Returns error contents coming under INDEX	Letter-type variable = rerrtext(index)
		▶ index : Order of alarm occurrence
			string etext
			etext = rerrtext(1) //first error
			//content return

2.1.14 MAPPING Commands (RS422 Communication Type only)

Command	Function	Format	Usage Example
MAP_INIT	Initializes Mapping-related internal variables	int map_init()	int ret ret = map_init()
MAP_START	Starts Mapping	int map_start(int axis, int sensor index, int start local point index) ▶ axis : 1 ~ maximum axis ▶ Sensor index : Sensor's index value (Range: 1~2) ▶ Start local point: Start point index to save mapping data, Range: 0~19999	int ret ret = map_start(2, 3,100)
MAP_STOP	Stops Mapping	int map_stop()	int ret ret = map_stop()
MAP_READ	Reads Mapping data	int map_read(int time-out period) ▶ Time-out period (ms)	int ret ret = map_read(1000)

CH.3 COMMAND INTERPRETATION

3.1 FUNC, END (Function Declaration/end)

Function Indicates function start and end.

Format
```
func function return type function name (parameter)
...
end
```

Description
1) **FUNC-END must be used together** to form a function block. A program begins in the function coming up the first.
2) Means the end of an END function block and when faced with this command in performing a function block, a function ends its implementation.
   But, the function performed by the function call ends its JOB implementation and returns to the **called function.**
3) **Function blocks (FUNC ~ END) needed in the following line of END can be continuously written**
4) When the return type is not VOID, be sure to implement RETURN commands prior to ending the function.
5) The first function of the job that runs the first time is the main program whose format must be a function void function name ().

3.1.1 Examples of Program Usage

func void main()	………	**Start Main Program**
take 1	………	Assign robot
vel(100)	………	Axis movement speed
while 1	………	Start condition repeat run
jmov lP[0]	………	Move to P0
jmov lP[1]	………	Move to P1
biton()	………	Call sub-program
End	………	End condition repeat run
release	………	Release robot assignment
End	………	**End Main Program**
func int biton()	………	**Assign Function**
int a	………	
out(0)=1, 200	………	Output in the assigned bit by 1-bit unit
out(1)=1, 200	………	Output in the assigned bit by 1-bit unit
out(2)=1, 200		Output in the assigned bit by 1-bit unit
a = in(0)		
return (a)	………	**Return Function**
end	………	**End Function**

3.2 INCLUDE

Function Includes other JOB.

Format include “job name”

Description 1) Should come the first in JOB and call the name of other JOB.

3.3 FOR, END (Repeated Executable Statement)

Function: Executes blocks repeatedly until variable values are met.

Format:
for ⟨Variable⟩=⟨Default Value⟩ to ⟨End Value⟩ (step ⟨Incremental Value⟩)
…
end

Terms:
⟨Variable⟩ : Utilizes a variable name declared as an integer type or real-number type. Ex) int a,b,c,aa,…
⟨Default value⟩ : Refers to integer values/real-number values set to the variable shortly before performing FOR block repeatedly.
⟨End value⟩ : Integer value/real-number value for limiting the number of repeat runs of FOR blocks.
⟨Incremental value⟩ : Integer value/real-number value used to increase the value of a variable regularly and steadily.

Description:
1) **FOR - END must be used together**.
2) After the default value is set to the integer-type variable or real-type variable, **FOR block is repeatedly performed until satisfying the end value**.
3) Whenever repeatedly performing FOR blocks, the value of an integer-type variable/real-type value is increased by the incremental value, and when the incremental value is skipped, **an increase is automatically made by 1 each time**
4) The end value should be **always greater** than the default value. (Default value≤ end value)

3.3.1 Examples of Program Usage

1) Using the FOR block, calculate the sum of odd numbers from 1 to 10 to save to Integer-type variable SUM.

```
func void main()
int aa, sum ……… Integer variable AA, SUM declaration
sum=0 ……… Integer variable initialization
for aa=1 to 10 step 2 ……… AA=1 integer 1 to 10, 5 times, by +2 increase each time
sum=sum+aa ……… Move to END after running 5 times
end
end
```

2) Using FOR block, reciprocating motion 20 times in points P1 ↔ P2

```
func void func()
int j                ……  Integer variable J declaration
j=0                  ……  Integer variable initialization
take 1
for j=1 to 20        ……  Move to END after making repeat operations
                         for FOR statement 20 times
jmov lp[1]           ……  JMOV P1
jmov lP[2]           ……  JMOV P2
end
release
End
```

3) Move after saving teaching point to Position variable

```
func void main()
int a                ……  Integer variable A declaration
pos ap[11]           ……  Position variable AP(11) declaration →
                         AP(0),AP(1) ~ AP(10)
for a=1 to 10        ……  Move to END after making repeat operations
                         for FOR statement 10 times
ap[a]=getlp(1, a)    ……  Save P(1) ~ P(10) to point variable AP(A)
end
end
```

4) Move after saving teaching point to Position variable

```
func void main()
int j,k              ……  Integer variables J,K declaration
pos tmp,ap[11]       ……  Position variable AP(11) declaration →
                         AP(0),AP(1) ~ AP(10)
take 1               ……
for k=1 to 10        ……  Move to END after making repeat operations
                         for FOR statement 10 times
ap[k]=lp[k]          ……  Save LP(1) ~ LP(10) to point variable AP(K)
end
if lp[100].3==1
for j=1 to 10            Move to END after making repeat operations
                         for FOR statement 10 times
tmp=ap[j]            ……  AP(J)=AP(I)=LP(1) ~ LP(10)
tmp.3=tmp.3+10           Add 10 to Z-axis value of position variable
                         TMP.
ap[j]=tmp
end
for k=1 to 10        ……  Move to END after making repeat operations
                         for FOR statement 10 times
jmov ap[k]               Move PTP to saved AP(1) ~ AP(10)
dlay 100
end
release
end
```

3.4 WHILE, END (Statement for Condition Repeat Run)

Function: Performs blocks repeatedly only while the conditional expression is satisfied.

Format:
```
while 〈Conditional expression〉
  …
end
```

Terms: 〈Conditional expression〉 : Refers to logical operation expression or comparison operation expression capable of judging if the expression is true.

Description:
1) WHILE statement performs WHILE block repeatedly while **conditions are met (Results of conditional expression are true or a value other than 0).**
2) **WHILE - ENDWL must be used together**.
3) **Unlimited repeat is performed when results of a conditional expression are always true**.

CAUTION

Precautions when entering a command
(O) WHILE_((IN(0)==1)&&(IN(1)==0))
(O) WHILE_IN(0)==1&&IN(1)==0
(X) WHILE_IN(0)==1
&&IN(1)==0

3.4.1 Examples of Program Usage

1) Repeat movement of points P0 ↔ P1

```
func void main()
take 1                ……          Selects robot 1
vel(100)              ……
while 1               ……          Perform blocks endlessly and repeatedly
jmov lp[0]            ……          Repeat movement of LP0 ↔ LP1
jmov lp[1]            ……
end
release
end
```

3.4.2 Examples of Conditional Expression Usage

WHILE(Conditional Expression)		Description
Constant	WHILE 1	Runs WHILE ~ END block endlessly and repeatedly.
Input	WHILE IN(0)==1	Repeat run of block while being entered by input signal IN0=1.
	WHILE ((IN(0)==1)&&(IN1==(0)))	Repeat run of block only while two conditions for input signal IN0=1, IN1=0 are simultaneously being met.
	WHILE ((IN(0)==1) || (IN(1)==0))	Repeat run of block only while either of two conditions for input signals IN0=1, IN1=0 is being met.
	WHILE PIN(0)==0H000F	Repeat run of block only while IN0, IN1, IN2, and IN3 in input PORT 0 all stay at "1".
	WHILE PIN(0)==32	Repeat run of block only while IN4 in the input signal status of input PORT 0 stays at "1".
Output	WHILE OUT(0)==1	Repeat run of block only while output OUT0 stays at "1".
	WHILE POUT(0)==0H000F	Repeat run of block only while OUT0, OUT1, OUT2, and OUT3 in output POUT 0 all stay at "1".
	WHILE POUT(0)==0B0000000011111111	Repeat run of block only while OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, and OUT7 in output POUT 0 all stay at "1".
Variable	WHILE TMP==1	Repeat run of block only while variable TMP stays at "1".
	WHILE MVR〉10	Repeat run of block only while the value of a MVR variable is greater than "10".
	WHILE TMR〉10	Repeat run of block only while the value of a TMR(TIMER) variable is greater than "10".
	WHILE CNT〉10	Repeat run of block only while the value of a variable CNT(COUNT) is greater than "10".
	WHILE AP[10].1==1	Repeat run of block only while the value of 1-axis for POSITION variable AP value is identical to "1".

3.5 CONTINUE, BREAK(Repeat Statement Control)

Function Cotrols a repeat statement with 'for' or 'while'.

Format
```
while 1
  if 〈Conditional expression〉
    continue
  else
    break
end
```

Terms 〈Conditional expression〉 : Refers to logical operation expression or comparison operation expression capable of judging if the expression is true.

Description
1) Moves to the beginning of a repeat statement when meeting a CONTINUE statement.
2) Exits the repeat statement when meeting a BRAKE statement.

3.5.1 Examples of Program Usage

func void run() take 1		
vel(100)		
while 1	……	Perform blocks endlessly and repeatedly
if in(0) == 1		
dlay 1000		1second standby
continue		Move to the beginning of a repeat statement
elif in(1) == 1		
break		End a repeat statement
end jmov lp[0] jmov lp[1] **end**	…… ……	LP0 ↔ LP1 repeat movement
release end		

3.6 IF, END (Condition Branch Commands)

Function Performs condition judgment on a conditional expression.

Format
if 〈Condition〉
…
(else)
…
end

Terms 〈Condition〉 : Refers to logical operation expression or comparison operation expression capable of judging if the expression is true.

Description
1) **IF - END must be used together**.
2) When the operation result of a conditional expression is true **(or value other than 0), sentences after THEN are implemented, whereas when false (or 0), sentences after ELSE are implemented.**
3) ELSE statement can be **selectively used** by the program writer.
4) Overlapping IF block using IF block again inside IF block can be used. Be careful writing should be made for IF and END to be logically in pair.

※ In case the number of IFs and ENDIFs are different, "Syntax Error" occurs.

Precautions when entering commands
(O) IF_((IN(0)==1)&&(IN(1)==0))
(O) IF_IN(0)==1&&IN(1)==0
(X) IF_IN(0)==1
&&IN(1)==0

3.6.1 Examples of Program Usage

1) Decide a point to move according to input IN0 status

```
func void main()
take 1
vel(1, 10)
if in(0)==1
jmov lp[1]          ……  When input IN0 is 1, move PTP to point LP1
else
jmov lp[2]          ……  When input IN0 is 0, move PTP to point LP2.
end
release
end
```

2) Input Port "1" (IN0 ~ 15), IN0 status check

```
func void main()
int aa, bb
take 1
aa=pin(1)           ……  Save Port1 (IN0~15) status to AA variable
if (aa==0HFFFF)         Judge AA value (Whether IN0~15 all are 1)
stop                ……  When "True", robot stops
else
bb=in(0)            ……  When "False", save IN0 status to BB
if (bb==1)              Judge if BB is 1
dlay10              ……
jmov lp[0]          ……  When "true(IN0=1)", move to point LP0
end
end
release
end
```

3) The program below is identical in content.

(But, identical description as long as the input applied simultaneously among IN0~IN3 is only one)

```
    :
labl A0
if in(0)==1
goto B0
end
if in(1)==1
goto C0
end
if in(2)==1
goto D0
end
if in(3)==1
goto A0
end
    :
```

=

```
    :
labl A0
if in(0)==1
goto B0
else
if in(1)==1
goto C0
else
if in(2)==1
goto D0
else
if in(3)==1
goto A0
end
end
end
end
    :
```

3.6.2 IF(Condition) THEN

IF(Condition)		Description
Input	IF_IN(0)==1	Is input IN0 "1"?
	IF_((IN(0)==1)&&(IN(1)==0))	Are two conditions for input signals IN0=1, IN1=0 met simultaneously?
	IF_((IN(0)==1) || (IN(1)==0))	Is at least one of two conditions for input signals IN0=1, IN1=0 met?
	IF_PIN(0)==0H000F	Are IN0, IN1, IN2, and IN3 in input PORT0 all "1"?
	IF_PIN(0)==32	Does the input signal status in input PORT0 have "1" in IN4?
Output	IF_OUT(0)==1	Is output OUT0 "1"?
	IF_POUT(0)==0H000F	Are OUT0, OUT1, OUT2, and OUT3 in output POUT0 all "1"?
	IF_POUT(0)==0B0000000011111111	Are OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, and OUT7 in output POUT0 all "1"?
Variable	IF_TMP==1	Is variable TMP "1"?
	IF_MVR〉10	Is the value of MVR variable greater than "10"?
	IF_TMR〉10	Is the value of TMR(TIMER) variable greater than "10"?
	IF_CNT〉10	Is the value of CNT(COUNT) variable greater than "10"?
	IF_AP[10].1==1	Is the value (1-axis value) of POSITION variable identical to "1"?

3.7 LABL, GOTO (Branch Commands)

Function: Commands for branch positioning and branch movement of executable statement.

Format:
labl ⟨Lable name⟩
goto ⟨Label name⟩

Terms: ⟨Label name⟩ : A character string combined with English letters and numbers.
But, "lp[0], lp[1], … " displaying a job point may not be used.
Ex) labl UNCLAMP1 **(O)**, GOTO CHECK2 **(O)**
labl p[1] **(X)**, goto p[100] **(X)**

Description:
1) GOTO command is **branched into the corresponding label unconditionally** and is enabled only in the identical block. For example, it is not possible to branch from the inside of MAIN block to the func block.
2) When overlapping IF block or FOR, or WHILE block before use, branching from the inside to the outside is possible, but **not possible the other way round.**

3.7.1 Examples of Program Usage

1) Make repeat movements of 2 points P1, P0.

```
func void main()
take 1
vel(100)
labl A0
jmov lp[1]
jmov lp[2]
goto A0
release
end
```

Unconditional jump to "LABL A0"

=

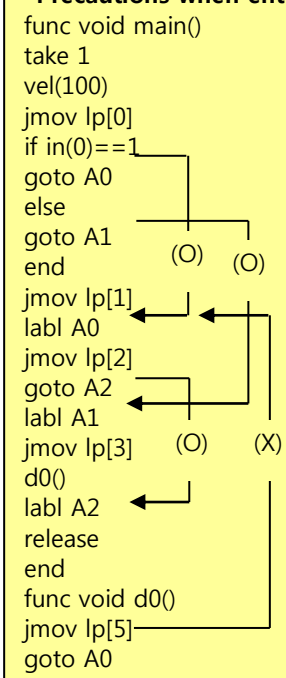
```
func void main()
take 1
vel(100)
while 1
jmov lp[1]
jmov lp[2]
end
release
end
```

2) Branch according to input IN0 status

func void main()		
take 1		
vel(100)		
jmov lp[0]		
if in(0)==1		
goto A0	……	When input IN0 is 1, unconditional jump is made to LABL A0
Else		
goto A1	……	Otherwise, unconditional jump to LABL A1
End		
jmov lp[1]		
labl A0	……	LABL A0 branch spot
jmov lp[2]		
goto A2	……	Unconditional jump to LABL A2
labl A1	……	LABL A1 branch spot
jmov lp[3]		
labl A2	……	LABL A2 branch spot
release		
end		

CAUTION

Precautions when entering commands



3.8 STOP, EXIT (Robot and JOB Stop Commands)

Function Robot motion STOP, JOB run stop (EXIT)

Format
stop
exit

Description
1) STOP command is used to **stop the robot in motion** in the thread where STOP command has been used and processes the command of the following STEP. That is, it **continues to run JOB after stop**.
2) Unable to stop the robot running in other thread.
3) EXIT command is used to **stop JOB run** of the thread where EXIT command is performed.

3.8.1 Examples of Program Usage

1) Following the point movement, turn ON/OFF output OUT1

```
func void main()
take 1
vel(500)
jmov lp[1]
mvr=0          ……                    Initialize "Travel zone percentage"
with           ……
jmov lp[2]     ……                    Move to point P2 (only JMOV available)
while mvr<60   ……                    While below 60 percent of travel zone
if in(0)==1    ……                    In case of IN0=1, jump to "LABL BB"
goto BB
End
End
out(0)=1
End

labl BB
stop           ……                    Stop robot motion and operation
end            ……                    Exit WITH statement (Must be inserted)
jmov lp[3]
out(1)=1
release
end
```

3.9 CTHREAD (Thread Creating Command)

Function Creates a thread.

Format cthread(Function, THREAD ID)

Terms Function : Refers to a function to run with a thread, in which a return type should be void type and no parameter should be present.
THREAD ID : thread ID to run
Possible input range is 1 ~ 3 only.
When the selected Thread ID is already in use by other thread, a thread is not created.

Return Value Result value
0 : thread creation fails
1 : thread creation succeeds

Description Unable to use a motion command.
A global variable is used when delivering data through a thread.
Thread can be created up to 3 units.

3.9.1 Examples of Program Usage

```
func void main()
int t_ret
t_ret=cthread(sum, 1)          ……….                Use a thread to execute a function calculating a
                               ….                  cumulative sum from 1 to 10
if t_ret == 0
print(0, “failed to
create sum thread”)
end
end

func void sum()                ……….                thread function (Required to take a Return Value as void,
                               ….                  with parameters not present.)
int a,b
for a = 1 to 10
    b = b + a
end
end
```

3.10 ETHREAD (Thread End Command)

Function	Ends a thread.
Format	ethread(thread ID)
Terms	thread ID : thread ID entered in cthread.
Return Value	No Return Value present.
Description	Ends the running thread.

3.10.1 Examples of Program Usage

```
func void main()
int t_id, t_ret
t_id = 1
t_ret=cthread(sum, t_id)  ……  Use a thread to execute a function calculating a cumulative sum from 1 to 10
dlay 1000
ethread(t_id)
end

func void sum()  ……  thread function
int a,b
for a = 1 to 10
    b = b + a
end
end
```

3.11 TH_STATE(Thread Status Command)

Function Checks the status of the running thread.

Format th_state(thread ID)

Terms thread ID : thread ID entered in cthread.

Return Value Returns Thread status.
0 : thread id is incorrectly entered (thread ID range is 0~3.)
1 : thread is not working.
2 : thread is stopped. (thread suspended by th_suspend command)
3 : thread is running.

Description Checks the status of the running thread.

3.11.1 Examples of Program Usage

```
func void main()
int t_id
int th_st
t_id = 1
cthread(sum, t_id)                ……                Use a thread to execute a function calculating a
                                  ……                cumulative sum from 1 to 10
while 1
dlay 1000
th_st = th_state(t_id)                              Checks thread status
if (th_st == 1)                                     thread not working
  cthread(sum, t_id)                                Creates a thread
elif (th_st == 2)                                   thread suspended
  th_resume(t_id)                                   Runs thread
elif (th_st == 3)                                   thread being operated
  th_suspend(t_id)                                  thread suspended
end
dlay 1000
th_st = th_state(t_id)
if ( th_st != 1 && th_st != 0)
  ethread(t_id)
end
end

func void sum()                   ……                thread function
                                  ……
int a,b
for a = 1 to 10
   b = b + a
```

```
end
end
```

3.12 TH_SUSPEND(Thread Suspend Command)

Function Suspends the running thread.

Format th_suspend(thread ID)

Terms thread ID : Thread ID entered in cthread.

Return Value No Return Value present.

Description Suspends the running thread.

3.12.1 Examples of Program Usage

```
func void main()
int t_id
int th_st
t_id = 1
cthread(sum, t_id)                  ……                  Use a thread to execute a function calculating a
                                    ……                  cumulative sum from 1 to 10
while 1
dlay 1000
th_st = th_state(t_id)                                  Checks thread status
if (th_st == 1)                                         Thread not working
  cthread(sum, t_id)                                    Creates thread
elif (th_st == 2)                                       Thread suspended
  th_resume(t_id)                                       Runs thread
elif (th_st == 3)                                       thread being operated
  th_suspend(t_id)                                      thread suspended
end
dlay 1000
th_st = th_state(t_id)
if ( th_st != 1 && th_st != 0)
  ethread(t_id)
end
end

func void sum()                     ……                  thread function
                                    ……
int a,b
for a = 1 to 10
   b = b + a
end
end
```

3.13 TH_RESUME(Thread Resume Command)

Function Resumes a thread suspended by Th_suspend command.

Formate th_ suspend (thread ID)

Terms thread ID : Thread ID entered in cthread.

Return Value No Return Value present.

Description Checks the status of the running thread.

3.13.1 Examples of Program Usage

```
func void main()
int t_id
int th_st
t_id = 1
cthread(sum, t_id)                  ……                Use a thread to execute a function calculating a
                                    ……                cumulative sum from 1 to 10
while 1
dlay 1000
th_st = th_state(t_id)                                Checks thread status
if (th_st == 1)                                       Thread not working
  cthread(sum, t_id)                                  Creates thread
elif (th_st == 2)                                     Thread suspended
  th_resume(t_id)                                     Runs thread
elif (th_st == 3)                                     Thread being operated
  th_suspend(t_id)                                    thread suspended
end
dlay 1000
th_st = th_state(t_id)
if ( th_st != 1 && th_st != 0)
  ethread(t_id)
end
end

func void sum()                     ……                thread function
                                    ……
int a,b
for a = 1 to 10
   b = b + a
end
end
```

3.14 TAKE (Robot Select Command)

Function Selects a robot.

Format take ⟨robot_id⟩

Terms robot_id : id of a robot to operate. (1~3)

Return Value No Return Value present.

Description Selects a robot to operate from the thread.
But, it is not allowed to select a robot ID selected from other thread.

3.14.1 Examples of Program Usage

```
func void main()
take 1                          Selects robot 1.
while 1
jmov lp[0]
dlay 100
jmov lp[1]
dlay 100
end
release                         Release the selected robot.
end
```

3.15 RELEASE(Robot Release Command)

Function Releases the robot selection

Format release

Terms

Thread ID entered in cthread	No Thread ID entered in cthread present.
Description	Releases the robot selected through 'take' command from the thread.

3.15.1 Examples of Program Usage

```
func void main()
take 1                          Selects robot 1.
while 1
jmov lp[0]
dlay 100
jmov lp[1]
dlay 100
end
release                         Releases the selected robot.
end
```

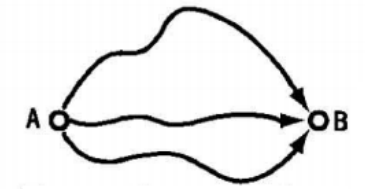
3.16 JMOV (PTP Movement Command)

Function: Does axis interpolation from the current position to the target point

Format:
jmov lp[Number], FOS, VEL, ACC, DEC
jmov position-type variable

Terms:
[Number] : Sets the taught position coordinate number.
lp - POINT used individually per robot.
Position-type variable : Means a variable name declared as POS type.
Following data is optional and returns to its original value when the current motion command ends.
FOS : fos quantity
VEL : speed of movement
ACC : acceleration in movement
DEC : deceleration in movement

Description: PTP(Point to Point)) refers to **movement from dot to dot**.
PTP travel path is dependent on robot posture and is **not limited by linear motions**.



travel path not limited to a straight line.

3.16.1 Examples of Program Usage

1) Move to the position coordinate where point teaching is done

```
func void main()
take 1
vel(100)
jmov lp[10]          ……          Move PTP to point LP10 (fos 0, speed 10%)
jmov lp[11], 10, 1000 ……         Move PTP to point LP11 (fos 10, speed 100%)
jmov lp[100]         ……          Move PTP to point LP100 (fos 0, speed 10%)
release
end
```

2) Movement using Position-type variables (1)

```
func void main()
pos a                          ……    Declare point-type variable A
a=〈400.0,50.0,10.4,10.0,0,0〉   ……    Assign A property value
take 1
vel(100)
jmov lp[0]                     ……    Move PTP to point LP10
jmov a                         ……    Move PTP to A position coordinate
jmov lp[100]                   ……    Move PTP to point LP100
release
end
```

3) Movement using Position-type variables (2)

```
func void main()
pos ap,ap1,ap2                 ……    Declare point-type variables AP,AP1,AP2
take 1
ap=lp[10] +〈10,10,10,10,0,0〉  ……    Save the value made by adding 10 to each axis
                                      values in point LP10
ap1=lp[11]-                    ……    Save the value made by subtracting 10 from
〈10,10,10,10,0,0〉                    each axis values in point LP11
ap2=lp[100]                    ……    Save value LP100 to point AP2
ap2.3=lp[100].3+10             ……    Save the value made by adding 10 to Z-axis
                                      value in point P100
jmov ap                        ……    Move PTP to AP position coordinate
jmov ap1                       ……    Move PTP to AP1 position coordinate
jmov ap2                       ……    Move PTP to AP2 position coordinate
release
end
```

i

3.17 WITH, ENDWT (Concurrent Processing Command)

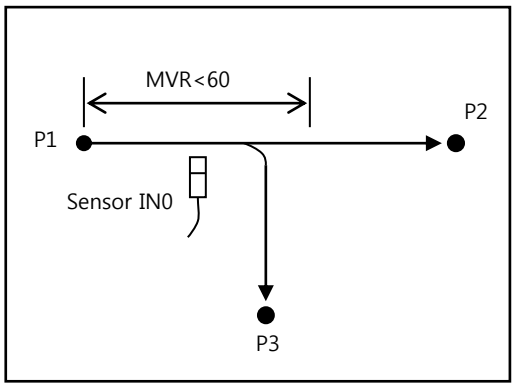
Function Processes the following command concurrently while the robot is in motion.

Format
```
with
  ...
end
```

Description
1) Performs the first **JMOV command** in a WITH statement while taking care of following commands through parallel processing.
2) Capable of processing the conditional function at the desired spot by dividing the robot movement distance into hundreds with the use of MVR variables.

3.17.1 Examples of Program Usage (1)

1) Job details →
When IN0 signal turns ON while moving from P1 to P2, movement is stopped and moves to P3. (But, the sensor input signal inspects only 60% of the travel distance (P1 to P2).



2) Robot USER I/O

IN		OUT	
0	Sensor detection	0	Arriving at P2 completed
1		1	Arriving at P3 completed

3) Write program

```
func void main()
take 1
vel(500)
jmov lp[1]
mvr=0 ……                    Initialize “Percentage of travel area”
with ……
jmov lp[2] ……                Move to point LP2 (Only JMOV available)
while mvr<60 ……              While in less than 60% of travel area
if in(0)==1 ……               In case of IN0=1, jump to “LABL BB”
goto BB
end
end
out(0)=1
end

labl BB
stop ……                     Robot stops motions
jmov lp[3]
out(1)=1
release
end
```

3.18 OUT, POUT (External Output Command)

Function Outputs designated values by 'bit' unit or 'port' unit.

Format OUT(Bit output port number)=〈0 or 1〉 [Pulse valid time] [→ or ~]
OUT(Integer-type variable))=〈0 or 1〉
POUT(output port number)=〈Data〉

Terms 〈Bit output port number〉 : Sets the bit output port number. (0 ≤Bit output number ≤ 95)

1) User Output : OUT(0) ~ OUT(15)
2) Extension 1 output (Option Output1) : OUT(16)~OUT(47) → Valid when installing one extension I/O Card
3) Extension 2 output (Option Output2) : OUT(48)~OUT(79) → Valid when installing two I/O Cards

〈Pulse valid time〉 : Valid time during which 0/1 value is displayed (Pulse Output) in the corresponding bit.

[→ or ~] :

-. Displays cycle waveforms that hold Pulse valid time as a cycle.

-. Pulse output, cycle waveform output can be used as many as maximum 32 points (One of these : User, Option1, Option2).

-. Port is not allowed to overlap in use. (Ex) User OUT 16 points + Option OUT 16 points)

1) "Pulse Output" and "Cycle Waveform Output" are implemented simultaneously with the progress of the stop. That is, when the next step is movement command (MOVE), output bit turns ON or OFF while moving.
2) For "Pulse Output" and "Cycle Waveform Output", the port to be used in the parameter should be set. For how to set, refer to "Parameter Manual".

〈output port number〉 : Sets output port numbers. (0 ≤ output port number ≤ 4)

-. 0 : Assign OUT0 ~ OUT15
-. 1 : Assign OUT16 ~ OUT31
-. 2 : Assign OUT32 ~ OUT47
-. 3 : Assign OUT48 ~ OUT63
-. 4 : Assign OUT64 ~ OUT79

〈Data〉 : Refers to a value (Possibly binary number, hexadecimal number) to be displayed in the corresponding output port.

Ex. 1) In POUT0=10, (10(Decimal number) =〉 0000 0000 0000 1010 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.

Ex. 2) In POUT0=0H0F0F, (0H0F0F (Hexadecimal number) =〉 0000 1111 0000 1111 (Binary number))

Turns ON(1) output ports 1,3, turning OFF(0) other output ports.

Description **OUT(Bit output port number)=〈0 or 1〉 [〈Pulse valid time〉] [→]**

1) Outputs the value 0 or 1 in the corresponding bit output port as much as the valid time of the designated pulse.
2) When pulse valid time does not exist, it keeps valid and it returns to the previous status after the pulse valid time is over.
3) The unit of pulse valid time is 10ms.

OUT((integer-type variable))=〈0 or 1〉

Integer-type variable cannot be used in “Bit output port number”.

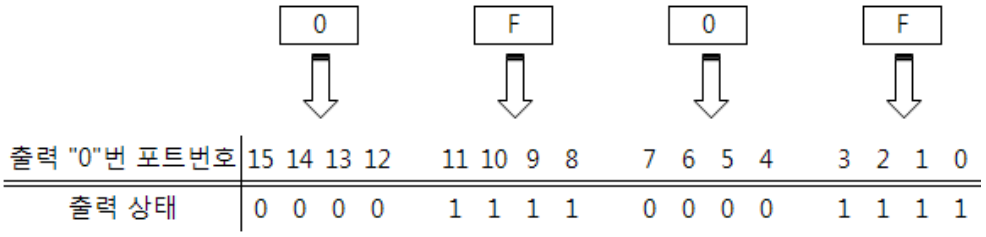
POUT(output port number)=〈Data〉

Outputs (16 bit) data value in the corresponding output port.

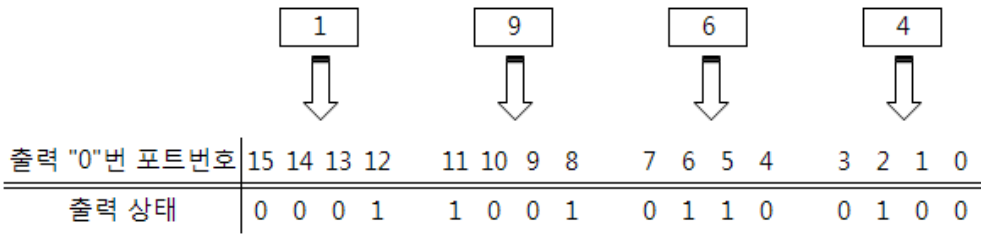
⚠CAUTION

- Data value of an output port number increases from right to left, in the order of 0,1,2 ~ 15.
- **Internal contact does not allow pulse output control.**
- Purchasing an extension I/O board is optional.

[POUT0=0H0F0F]



[POUT0=0H1964]

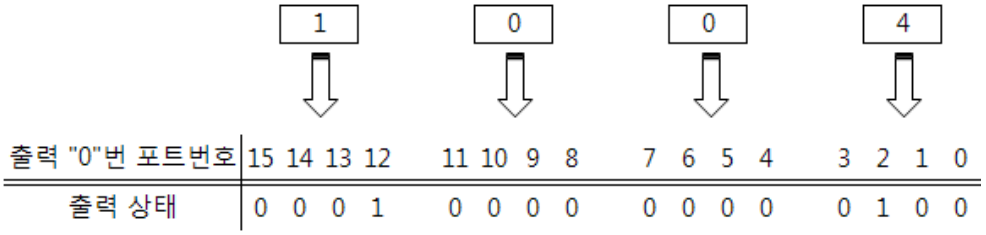


출력 “0”번 포트번호 Output “0” Port Number
출력상태 Output Status

3.18.1 Examples of Program Usage

1) Pulse Output, Data Output

```
func void main()
take 1
vel(100)
jmov lp[10]
out(11)=1, 100          ……….    Turn ON(1) the output port 11 during 100ms
jmov lp[100]
pout(0)=0H1004          ……….    Output in 0H1004(Hexadecimal number) in output
                                ports 0~15
release
end
```



출력 “0”번 포트번호 Output “0” Port Number
출력상태 Output Status

2) Use bit output port number as variables

```
func void main()
int a                   ……….    Declare integer-type variable A
for a=0 to 10           ……….    Perform repeat run from 0 to 10
out(a)=0                ……….    Turn OFF(0) all outputs (OUT0,OUT1 ~ OUT10)
end
end
```

3.19 IN, PIN (External Input Command)

Function — Receives values from inputs in assigned bit unit or port unit.

Format — 〈Variable〉=IN〈Input bit number〉
〈Variable〉=PIN〈Input bit number〉

Terms — (Bit input port number) : Sets a bit input port number. (0 ≤Bit input number≤ 31)

1) User Input : IN(0) ~ IN(31)
2) Extension1 Input(Option Input1) : IN(32) ~ IN(63) → Installing extension I/O Card

〈 Input port number〉 : Sets an input port number.
PIN, WIN : Refers to 16bit bundle (word).

-. 0 : Assign IN0 ~ IN15
-. 1 : Assign IN 16 ~ IN31
-. 2 : Assign IN 32 ~ IN 47
-. 3 : Assign IN 48 ~ IN 63

Description — **IN(Bit input port number)**

Saves ON/OFF status (1 or 0) of the assigned bit input port to IN<Bit input port number>variable.

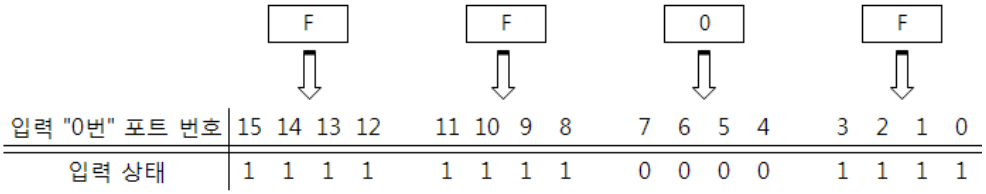
PIN(Input port number)

Reads the value (16 bit) of the assigned input port and saves it to the PIN<Input port number> variable.

⚠CAUTION

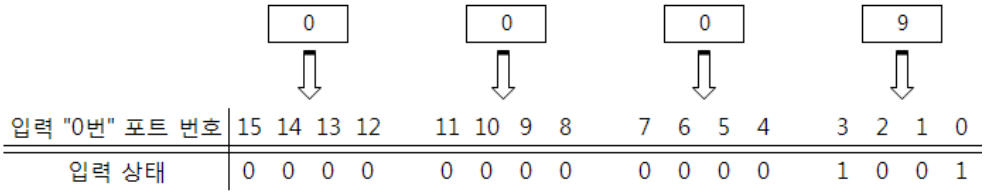
- ▶ Data value of an input port number increases from right to left, in the order of 0,1,2 ~ 15.
- ▶ Purchasing an extension I/O board is optional.

[PIN(0)=0HFF0F]



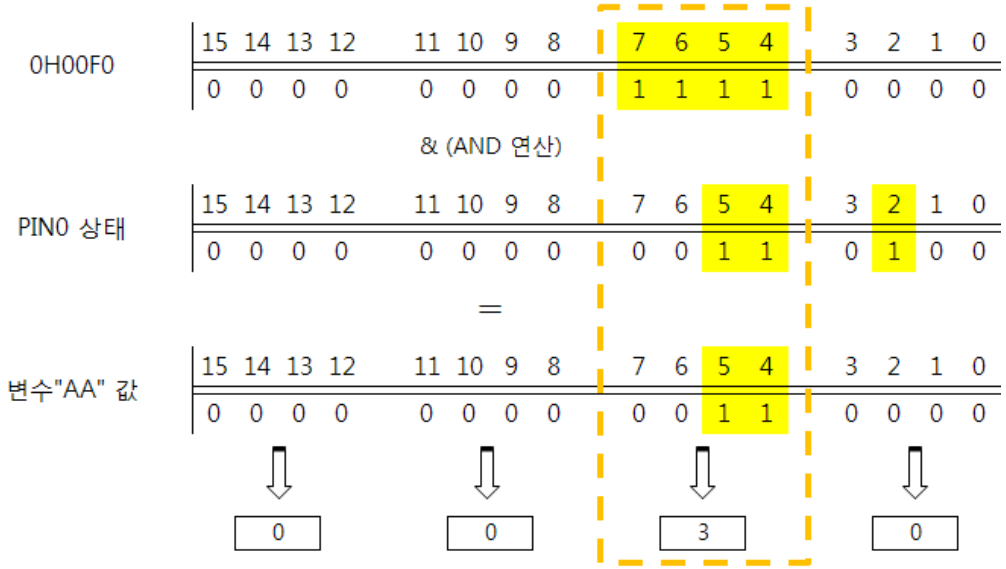
입력 “0”번 포트번호 Input “0” Port Number
입력상태 Input Status

[PIN(0)=0H0009]



To check out <u>signals only in "IN4~IN7"</u> among inputs in input "0" port?

[AA=0H00F0&PIN(0)]



AND 연산 AND Operation

3.19.1 Examples of Program Usage

1) Selects input wait, port input to be processed with an integer-type variable and moved.

```
func void main()
int d1 ……… Declare integer-type variable D1
take 1
vel(100)
while 1
jmov lp[0]
if in(0)==1 ……… When bit input port 0 turns ON(1)
d1=pin(0) ……… Save status of input ports 0~15 to D1
else
end
if d1==0HFF0F ……… Compare D1 status (That is, PIN0 status) with 0HFF0F
jmov lp[1]
else
jmov lp[2]
end
release
end
```

2) Compare MODEL using a BCD unit

- BCD input involves the use of inputs IN0 ~ IN7

```
func void main()
int m1,m2,md ……… Declare integer-type variable M1,M2,MD
take 1
while 1
m1=pin(0) & 0H000F ……… Compare PIN0 and 0H000F and save it to M1
m2=pin(0)&0H00F0 ……… Compare PIN0 and 0H000F0 and save it to M2
m2=(m2)>4)*10 ……… Shift M2 value to one's place and multiply by 10 Comment)
md=m1+m2 ……… Add calculated value of M1 and M2
if md==1
jmov lp[0]
else
if md==11
jmov lp[1]
end
end
release
end
```

CAUTION

▶ Port input performs hexadecimal operation. Therefore, when entering 10 with BCD unit, the controller recognizes it as 16. In this case, make a shift operation, as shown in the above example, and go through decimal conversion to conduct precise calculations.

3.20 BIN (External Input Command)

Function　Receives values from inputs in assigned bit unit.

Format　〈Variable〉=bin〈Input byte number〉

Terms　〈Input byte number〉 : Sets an input byte number.
Refers to a 8bit bundle (byte).
-. 0 : Assign IN0 ~ IN7
-. 1 : Assign IN 8 ~ IN15
-. 2 : Assign IN 16 ~ IN 23
-. 3 : Assign IN 24 ~ IN 31

Description　**bin(Input byte number)**
Reads the value of the assigned input byte and saves it to the variable.

⚠CAUTION

- ▶ Data value of an input port number increases from right to left, in the order of 0,1,2 ~ 15.
- ▶ Purchasing an extension I/O board is optional.

3.21 BOUT (External Output Command)

Function

Outputs designated values by 'byte' unit.

Format

bout(Output byte number)=〈Data〉
〈integer-type variable〉 = bout(Output byte number)

Terms

〈Output byte number〉 : Sets the output byte number. (0 ≤ output port number ≤ 3)
In use of extension I/O (0 ≤ output port number ≤ 7)
-. 0 : Assign OUT0 ~ OUT7
-. 1 : Assign OUT8 ~ OUT15
-. 2 : Assign OUT16 ~ OUT23
-. 3 : Assign OUT24 ~ OUT31

〈Data〉 : Refers to a value (Possibly binary number, hexadecimal number) to be displayed in the corresponding output byte.
Ex. 1) In bout(0)=10, (10(Decimal number) =〉 0000 1010 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.
Ex. 2) In bout(0)=0H0F0F, (0H0F (Hexadecimal number) =〉 0000 1111 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.

Description

bout(Output byte number)=〈Data〉
Outputs data in output byte number.
〈integer-type variable〉 = bout(Output byte number)
Reads the data of output byte number.

3.22 WIN(EXTERNAL INPUT COMMAND)

Function Receives values from inputs in assigned word unit.

Format 〈Variable〉=win〈Input word number〉

Terms 〈 Input word number 〉 : Sets an input word number.
Refers to a 16bit bundle (word).
-. 0 : Assign IN 0 ~ IN 15
-. 1 : Assign IN 16 ~ IN 31
-. 2 : Assign IN 32 ~ IN 47
-. 3 : Assign IN 48 ~ IN 63

Description **win(Input world number)**
Reads the value of an assigned input world and saves it to the variable.

3.23 WOUT (External Output Command)

Function — Outputs values designated by 'word' unit.

Format
wout(Output word number)=〈Data〉
〈integer-type variable〉 = wout(Output word number)

Terms
〈Output word number〉 : Sets an output word number. (0 ≤ output port number ≤ 1)
In use of extension I/O (0 ≤ output port number ≤ 3)
-. 0 : Assign OUT0 ~ OUT15
-. 1 : Assign OUT16 ~ OUT31
-. 2 : Assign OUT32 ~ OUT47
-. 3 : Assign OUT48 ~ OUT63

〈Data〉 : Refers to a value (Possibly binary number, hexadecimal number) to be displayed in the corresponding output byte.
Ex. 1) In wout(0)=10, (10(Decimal number) =〉 0000 0000 0000 1010 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.
Ex. 2) In wout(0)=0H0F0F, (0H0F0F (Decimal number) =〉 0000 1111 0000 1111 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.

Description
wout(Output word number)=〈Data〉
Outputs data in the output word number.
〈integer-type variable〉 = wout(Output word number)
Reads data in the output word number.

3.24 DIN(EXTERNAL INPUT COMMAND)

Function　Receives values from inputs in assigned double word unit.

Format　〈Variable〉=win〈Input double word number〉

Terms　〈 Input double-word number 〉 : Sets the input word number.
Refers to a 32bit bundle (double word).
-. 0 : Assign IN 0 ~ IN 31
-. 1 : Assign IN 32 ~ IN 63

Description　**din(Input double word number)**
Reads the assigned value of an input double word and saves it to the variable.

3.25 DOUT (EXTERNAL OUTPUT COMMAND)

Function — Outputs values designated by 'double word' unit.

Format
dout(Output double word number)=〈Data〉
〈integer-type variable〉 = dout(Output word number)

Terms
〈 Output word number 〉 : Sets an output word number. (0 ≤ output port number ≤ 1)
In use of extension I/O (0 ≤ output port number ≤ 2)
-. 0 : Assign OUT0 ~ OUT31
-. 1 : Assign OUT32 ~ OUT63

〈Data〉 : Refers to a value (Possibly binary number, hexadecimal number) to be displayed in the corresponding output byte.
Ex. 1) In dout(0)=10, (10(Decimal number) =〉 0000 0000 0000 1010 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.
Ex. 2) In dout(0)=0H0F0F, (0H0F0F (Hexadecimal number) =〉 0000 1111 0000 1111 (Binary number))
Turns ON(1) output ports 1,3, turning OFF(0) other output ports.

Description
dout(Output double word number)=〈Data〉
Outputs data in output double word number.
〈integer-type variable〉 = dout(Output double word number)
Reads data in output double word number.

3.26 Output Command for Field Bus (COUT, CBOUT, CWOUT, CDOUT, CFOUT)

Function: Outputs the assigned value to the field bus card by bit unit or port unit
→ CC-LINK Card, ProfiBus Card.

Format:
COUT〈Bit output port number〉=〈0 or 1〉
COUT〈(Integer-type variable)〉=〈0 or 1〉
CBOUT〈output port number〉=〈 Data 〉
CWOUT〈output port number〉=〈Data〉
CDOUT〈output port number〉=〈 Data 〉
CFOUT〈output port number〉=〈 Data 〉

Terms: 〈Bit output port number〉 : Sets bit output port number. (0 ≤ Bit output number≤ 799)
→ User Output : COUT0 ~ COUT799

〈output port number〉 : Sets output port number. (0 ≤ output port number ≤ 5)

1) CBOUT : Refers to a 8bit bundle (byte).
-. 0 : Assign COUT0 ~ COUT7
-. 1 : Assign COUT8 ~ COUT15
-. 2 : Assign COUT16 ~ COUT23
-. 3 : Assign COUT24 ~ COUT31
-. 4 : Assign COUT32 ~ COUT39
-. 5 : Assign COUT40 ~ COUT47

2) CWOUT : Refers to a 16bit bundle(word).
-. 0 : Assign COUT0 ~ COUT15
-. 1 : Assign COUT16 ~ COUT31
-. 2 : Assign COUT32 ~ COUT47
-. 3 : Assign COUT48 ~ COUT63

3) CDOUT : Refers to a 32bit bundle (double word).
-. 0 : Assign COUT0 ~ COUT31
-. 1 : Assign COUT32 ~ COUT63
-. 2 : Assign COUT64 ~ COUT95
-. 3 : Assign COUT96 ~ COUT127

4) CFOUT : Refers to storage space in real-number area. (32bit)

〈Data〉 : Refers to a value (Possibly binary number, hexadecimal number) to be displayed in the corresponding output port.

Description

CAUTION

▶ In case of CC-LINK where an Word area exists separately, CWOUT represents an index in the Word area instead of a 16-bit bundle in COUT area, therefore keep this in mind at time of use.

3.27 Input Command for Field Bus (CIN, CBIN, CWIN, CDIN, CFIN)

Function Receives input values in bit unit or port unit assigned by the field bus card.
→ CC-LINK Card, ProfiBus Card.

Format ⟨Variable⟩=CIN⟨Input bit number⟩
⟨Variable⟩=CBIN⟨Input port number⟩
⟨Variable⟩=CWIN⟨Input port number⟩
⟨Variable⟩=CDIN⟨Input port number⟩
⟨Variable⟩=CFIN⟨Input port number⟩

Terms ⟨Bit input port number⟩ : Sets bit input port number. (0 ≤ Bit input number ≤ 799)
→ User Input : CIN0 ~ CIN799

⟨Input port number⟩ : Sets input port number.

1) CBIN : Refers to an 8bit bundle (byte).
-. 0 : Assign CIN0 ~ CIN7
-. 1 : Assign CIN8 ~ CIN15
-. 2 : Assign CIN16 ~ CIN23
-. 3 : Assign CIN24 ~ CIN31
-. 4 : Assign CIN32 ~ CIN39
-. 5 : Assign CIN40 ~ CIN47

2) CWIN : Refers to a 16bit bundle (word).
-. 0 : Assign CIN0 ~ CIN15
-. 1 : Assign CIN 16 ~ CIN31
-. 2 : Assign CIN 32 ~ CIN 47
-. 3 : Assign CIN 48 ~ CIN 63

3) CDIN : Refers to a 32bit bundle (double word).
-. 0 : Assigns CIN0 ~ CIN31
-. 1 : Assign CIN32 ~ CIN63
-. 2 : Assign CIN64 ~ CIN95
-. 3 : Assign CIN96 ~ CIN127

4) CFIN : Refers to storage space in real-number area. (32bit)

Description

▶ In case of CC-LINK where an Word area exists separately, CWIN represents an index in the Word area instead of a 16-bit bundle in CIN area, therefore keep this in mind at time of use.

3.28 VEL(Axis Velocity Setup Commands)

Function: Sets the rate per thousand(%) for axis velocity

Format: vel(Velocity)

Terms: 〈Velocity〉: Sets a robot's velocity. (0 ≤ 〈Velocity〉 ≤ 1000)
Robot velocity = Mv(Maximum velocity on each axis) * 0.001 * velocity
(0 ≤ 〈Velocity〉 ≤ Maximum RPM).

Description
1) Refers to the speed when a robot moves, setting **maximum speed at 1000.**
2) In case velocity setup is not done in the program, **automatic setup** is done at the speed set in INIT_V parameter
3) The **variable (integer-type)** with a velocity value may be used.
4) **Operates only when the robot is selected through the command 'take'.**

CAUTION

▶ The velocity value is 1000.
When VEL1000 is entered, movement is made at 100% speed. (VEL 1000 => 100%)
▶ When used beyond maximum allowable RPM in the mechanism, a noise issue and risk of damage arise. Be sure to check out the label attached to the mechanism prior to use.

3.28.1 Examples of Program Usage

1) Speed change according to job positions

```
func void main()
take 1                                  Select robot
while 1
vel(1000)          ……                   Robot 1 velocity MV x 0.001 x 1000
jmov p[0]
jmov p[1]
vel(200)           ……                   Robot 1 velocity MV x 0.001 x 200
jmov p[2]
jmov p[3]
vel(1000)          ……                   Robot 1 velocity MV x 0.001 x 1000
jmov p[0]
end
release
end
```

3.29 ACC, DEC (Acceleration/Deceleration Setup Command)

Function Sets the permillage of acceleration/deceleration time

Format acc 〈Acceleration rate〉
dec 〈Deceleration rate〉

Terms 〈Acceleration rate / Deceleration rate〉 : (1 ≤ 〈Acceleration/deceleration rate〉 ≤ 2000)

-. Increases or decreases acceleration/deceleration time in the program.

-. Acceleration/deceleration time = At(Acceleration/deceleration time) * 0.001 * Acceleration/deceleration rate

(0.5* Maximum acceleration/deceleration time ≤ 〈 Acceleration/deceleration time 〉 ≤ 2* maximum acceleration/deceleration time)

Description

1) Sets robot's acceleration/deceleration time.
2) Set at 100% when the acceleration/deceleration rate is not set in the JOB program.
3) A variable (Integer-type) with an acceleration/deceleration ratio can be used.
4) Applies to all axes. (SCARA robot : Applies to A, B, Z, W axis)

Ex.) Parameter At : A-axis(0.3seconds), B-axis(0.3seconds), Z-axis(0.2seconds), W-axis (0.5seconds)
Run ACC(DEC) 50
At : A-axis(0.15seconds), B-axis(0.15seconds), Z-axis(0.1second), W-axis(0.25 seconds)
▶ Applies only within JOB program, not getting parameter value itself modified.

5) For returning to parameter acceleration/deceleration time (At), "ACC(DEC) 1000" is used and applied from the next step. (Refer to example program.)
6) **Operates only when the robot is selected through the command 'take'.**

CAUTION

▶ The acceleration/deceleration time set in the parameter is at optimal condition according to mechanism configuration.
▶ When the acceleration/deceleration time ratio in ACC/DEC commands is set at less than 100, the mechanism may have noise and vibration, so take caution when using.

3.29.1 Examples of Program Usage

1) Change acceleration/deceleration time according to job positions

- Example showing At value set at 0.3 seconds in the parameter

```
func void main()
take 1                                  Select robot
while 1
vel(500)
acc(500)                ……….  Set acceleration/deceleration to 0.15 sec
jmov lp[1]
acc(2000)               ……….  Set acceleration/deceleration to 0.6 sec
jmov lp[2]
jmov lp[3]
acc(1000)               ……….  Set acceleration/deceleration to 0.3 sec
jmov lp[0]
end
release
end
```

3.30 FOS (Continuous Trajectory Generation Command)

Function Changes a trajectory to the next target point before the axis end arrives at the target point.

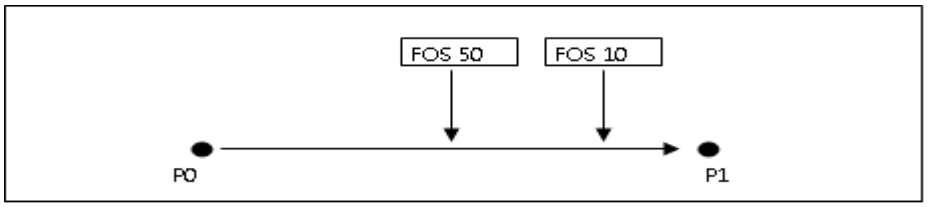
Format FOS 〈Distance rate〉

Terms 〈 Distance rate 〉 : (0 ≤ 〈Acceleration/deceleration ratio 〉〈 50%)

-. Sets the point at which the robot is moved to the next location or outputs before arriving at the assigned location

-. Distance rate is set at percentage for the movement distance between two points.

-. Using over 50% is prohibited in a motion other than JMOV.
(0 ≤ 〈 Distance rate 〉〈 50)



The above figure shows the point at which the robot, while moving from P0 to P1, moves to the next location or performs output according to each FOS value.

⚠CAUTION

- ▶ When teaching a point for continuous interpolation movement, the distance between two points should be at least over 5mm.
(But, when the velocity is 100mm/s)
- ▶ Alarm "Too much FOS" occurs when the FOS rate of a moving trajectory is shorter than the entire length of next position.

Description

1) When FOS is set, command MOVE **moves to the next location** before (Pre-set distance rate or distance value) arriving at the assigned location.
2) Output while moving is enabled in command OUT.
3) The job program makes use of **"FOS 0"** to return to the **status before FOS** is applied, commands followed by step 1 are applied.

```
func void main()
take 1
fos 10
jmov lp[0]
jmov lp[1]
fos 0
jmov lp[2]
jmov lp[3]
release
end
```

Command FOS 0 applied from jmov lp[2].

CAUTION

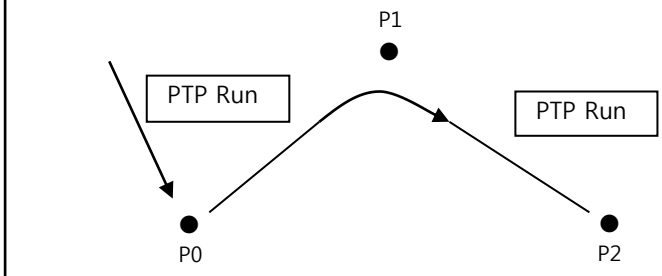
- Continuous interpolation movements are made by using FOS.
- "Unreachable Point","Inverse Error" may occur during Auto RUN due to a significant speed difference between two motions to create continuous trajectories.

3.30.1 Examples of Program Usage

1) PTP Operation (JMOV)

```
func void main()
take 1
vel(100)
jmov lp[0]
fos 5
jmov lP[1]
fos 0
jmov lP[2]
release
end
```

FOS 5 (5% before arriving at point P1)

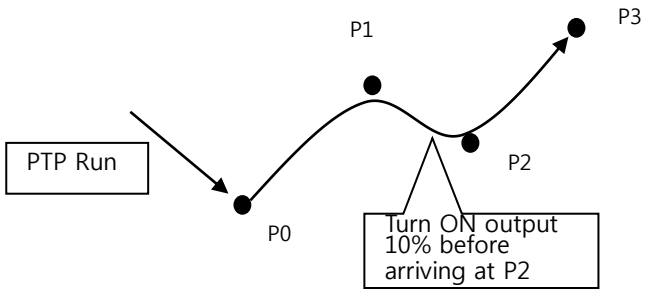


2) Robot in motion

3) Turn output signals (OUT, POUT) On, Off

```
func void main()
take 1
vel(100)
jmov lp[0]
fos 10
jmov lp[1]
jmov lp[2]
out[0]=1
fos 0
jmov lp[3]
release
end
```

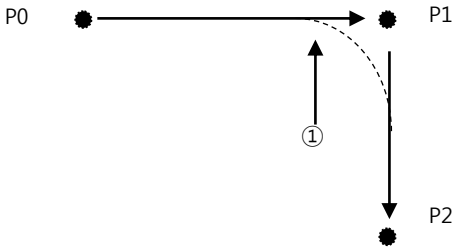
FOS 10



4) Process the next step command while the robot is moving

```
func void main()
take 1
while 1
vel(100)
jmov lp[0]
fos 10
if in(0)==1
jmov lp[1]
else
in(1)=1
jmov lp[2]
end
fos 0
end
release
end
```

FOS 10



Process IF command 10% before arriving at P1 in position ①. That is, when a signal gets into input "IN0", movement is made while drawing a trajectory like ②(Dotted line).

3.31 SVON, SVOF (Servo ON/OFF Command)

Function Servo ON, Servo OFF

Format
SVON(Robot) : All axes servo ON
SVON(Robot, assigned axis) : Assigned axes servo ON
SVOF(Robot) : All axes servo OFF
SVOF(Robot, assigned axis) : : Assigned axes servo OFF

Description
1) SVOF(Robot) :
-. Converts the robot body to free status (Robot enabled to move by hand) by **turning OFF the servo** while the robot is operating.
-. Turns OFF the servo on all currently-running axes.
2) SVON(Robot) :
-. Robot body put to servo ON, in relation to SVOF.
-. Without SVON following SVOF, robot operation (RUN) is in Dry RUN status
(Robot body does not move and JOB program performs 1 step each).
-. All axes being currently servo OFF are put into servo ON status.
3) SVON(SVOF)(Robot, assigned axis) : Individual axis put into servo ON/OFF. (Range :1~16)
SCARA robot → A-axis : 1, B-axis : 2, Z-axis : 3, W-axis : 4
Cartesian robot → X-axis : 1, Y-axis : 2, Z-axis : 3, W-axis : 4

3.31.1 Examples of Program Usage

1) PTP Operation (JMOV)

```
func void main()
take 1
vel(500)
jmov lp[0]
if in(1)==1
svof(1, 4)            ……  4th axis put into servo OFF
end
jmov lp[1]            ……  Move to point LP1 with the 4th axis in servo OFF status
if in(1)==0
svon(1, 4)            ……  4th axis put into servo ON
end
jmov lp[2]            ……  Move to point LP2 with all axes in servo OF status
end
release
end
```

2) PTP Operation (JMOV)

```
func void main()
take 1
vel(500)
jmov lp[0]
mvr=0
with
jmov lp[1]
while mvr<100
if in(0)==1          ……                  In IN0=1 while moving to LP1
stop
svof(2)                                  stop robot and turn servo OFF
goto L1
else
exit
end
end

labl L1
end
in(0)=1                                  After input IN0 becomes "1"
svon(2)              ……                  turn servo ON
jmov lp[2]
release
end
```

3.32 DLAY (Time Delay Command)

Function Sets delay time

Format dlay 〈Delay time〉

Terms 〈 Delay time〉 : Sets time to delay. (ms)
(1 ≤ 〈 Delay time 〉 ≤ 999)

Description
1) Set the set time in number.
2) Set time possibly ranges from 0.001 second to 9.99 seconds.
3) A variable with a time value (Integer-type variable) can be used.

3.32.1 Examples of Program Usage

1) PTP Operation (JMOV)

```
func void main()
take 1
vel(100)
jmov lp[10]
dlay 100                ……        Wait for 100ms(0.1second) in LP10 position
jmov lp[100]
release
end
```

3.33 RSTATE (Robot Status Check Command)

Function Checks robot status

Format integer-type variable = rstate(robot, state index)

Description

INDEX	STATE NAME	Description
1	ALARM	Returns alarm status
6	SERVO ON	Returns SERVO MOTOR ON/OFF status
9	AUTO MODE	Returns auto mode status

3.33.1 Examples of Program Usage

2) PTP Operation (JMOV)

```
func void main()
int tmp
take 1
vel(100)
tmp = rstate(1,6)                    ……        Check if robot 1 is servo on
if( tmp == 0 )                                 If not servo on,
  svon(1)                                      servon on
end
jmov lp[100]
release
end
```

3.34 DIST(Command for Difference of Two Position Variables)

Function Refers to a command calculating a difference between two position values

Format Position variable = dist(Position variable 1, 2)

Terms

Description 1) Returns a value in which position value 2 is subtracted from position value 1.
2) Data returned is a position data type.

3.34.1 Examples of Program Usage

```
func void main()
pos tmp
take 1
tmp = dist(lp[0], lp[1])          Save the value taken by subtracting lp[1] from lp[0]
                                  to position value tmp
release
end
```

3.35 SWLIMIT(Command for Reading Swlimit Value Set to Parameter)

Function Reads a SW limit value in a parameter

Format Real-type variable = swlimit(robot id, axis number, whether maximum or minimum)

Terms robot id : robot that will read sw limit ID(1~3)
axis number : axis number that will read sw limit (1~16)
whether maximum or minimum : select whether to read maximum value of sw limit or minimum.
(0 : minimum value, 1 : maximum value)

Description 1) Reads the sw limit value set at BODY-RANGE in the robot parameter.

3.35.1 Examples of Program Usage

```
func void main()
double tmp
tmp = swlimit(1, 1, 0)          Read the minimum value of sw limit on axis 1 of
                                robot 1.
end
```

3.36 GETLP(Command for Reading Local Point)

Function Reads Local point value

Format Position variable = getlp(robot id, local point index)

Terms robot id : robot that will read local point ID(1~3)
local point index : index(0~39999) of local point

Description 1) Reads the local point of the designated robot.
2) Position data read through lp[index] is required to select a robot through the command 'take' above all, whereas getlp command reads data on local point through the robot id input even if the robot is not selected.

3.36.1 Examples of Program Usage

```
func void main()
pos tmp
tmp = getlp(1, 120)          Reads position data in local point index 120 of robot 1.
end
```

3.37 Variables

3.37.1 Types of Variables

Variables may be divided into general variables, POSITON variables, and system variables.

		LOCAL	GLOBAL
General Variables	Integer-type	INT	GI(Number)
	Real-type	DOUBLE	GF(Number)
POSITION Variable		POS P(Number)	
System Variables		CNT,TMR,MVR,HERE	

-. LOCAL variables - Available for use only in the currently-running program

1) INT : Integer-type variable used for arithmetic operation and temporary input/output DATA save
2) DOUBLE : Real-type variable used for arithmetic operation and POINT DATA operation
3) POS : Position-type variable divided into a simple variable and an array variable
4) CNT : Saves the inbound pulse input through input port by UP COUNTING it
5) TMR : Increases by 1 each in the initial screen value according to parameter setup value
6) MVR : Percentage of the current position for the entire range, when moving between 2 POINTS
7) HERE : Variable with the robot's current position value
8) P(Number) : Variable with the position value taught by the user

-. GLOBAL variables - Available for use commonly in the whole program and channel

1) GI : Integer-type variable with functions identical to a LOCAL variable declared as INT (0 ≤ number ≤ 999)
2) GF : Real-type variable with functions identical to a LOCAL variable declared as REAL (0 ≤ number ≤ 999)

3.37.2 How to Use

-. Where to use

-. General variables used in the program are sure to be declared after **"main"**, the beginning part of the program.

-. Global variation is declared outside a function.

-. Configuration

-. The **variable name** of a general variable is a string of **character** combined with English characters and numbers.

-. The first letter of a variable must be either an **English character** or **'_'**. (Using number not allowed)

But, **"P", "GI", and "GF" cannot be used** because they overlap with POINT variable and GLOBAL variable.

-. Initialization

-. A variable should be used **after being initialized**.

-. Initialization of CNT, TMR variables must be done **in the previous STEP** of use of variables.

-. MVR variables are initialized to 0 once **MOVE command begins**.

3.38 Integer Type(INT, GI), Real Type(DOUBLE, GF), STRING Variables

3.38.1 LOCAL Variable

Function Declares integer-type, real-type and string variables available for use in the program.

Format
int ⟨variable name⟩,⟨variable name⟩,---
double ⟨variable name⟩,⟨variable name⟩,---
string ⟨variable name⟩,⟨variable name⟩,---

Terms ⟨variable name⟩ :

-. Character string combined with English characters and numbers.

-. The first letter of a variable must be an English character or '_'.

Description

1) Should be declared **within the function block**.
2) Integer-type variable declared as int (Range of integer value : -2.14×10^9 ~ 2.14×10^9)
3) Real-type variable declared as double (Ranage of real number : $\pm 8.4 \times 10^{-37}$ ~ $\pm 3.4 \times 10^{38}$)

CAUTION

- **"P", "GP", "GI", and "GF"** are not allowed to be used as they overlap with POINT variables and GLOBAL variables. (Alarm **"Syntax Error" , "Duplicated Symbol" occurs.)**
- When operations with real-type and integer-type variables are mixed in use, it **changes into the real-type.**
  9.5 + **10** = 19.5

3.38.2 Examples of Program Usage

1) Use of an integer-type, real-type variables

```
func void main()
int j,bot                ……  Declare integer-type variable J and BOT
double ff1               ……  Declare real-type variable FF1
while 1
bot=0H000F
j=1                      ……  Initialize variables
ff1=10.5
take 1
for j=0 to 10            ……  Move CP to AP1 position coordinate
jmov lp[1]               ……  Move CP to AP2 position coordinate
jmov lp[2]
pout(0)=bot              ……  Output using integer-type variable BOUT
end
ff1=ff1+100.0                Arithmetic operation of real-type variables
end
release
```

```
end
```

3.38.3 GLOBAL Variable

Function Declares integer-type, real-type and string variables available for use in the program

Format
int ⟨variable name⟩,⟨variable name⟩,---
double ⟨variable name⟩,⟨variable name⟩,---
string ⟨variable name⟩,⟨variable name⟩,---

Terms ⟨variable name⟩ :

-. Character string combined with English characters and numbers.

-. The first letter of a variable must begin with $.

Ex) int $int_var

Description
1) Must be declared **outside a function.**
2) Integer-type variable declared as int (Range of integer value : -2.14×10^{9} ~ 2.14×10^{9})
3) Real-type variable declared as double (Range of real number : $\pm 8.4 \times 10^{-37}$ ~ $\pm 3.4 \times 10^{38}$)

CAUTION

- **"P", "GP", "GI", and "GF"** are not allowed to be used as they overlap with POINT variables and GLOBAL variables. (Alarm **"Syntax Error" , "Duplicated Symbol" occurs.)**
- When operations with real-type and integer-type variables are mixed in use, it **changes into the real-type.**
  9.5 + **10** = 19.5

3.38.4 Examples of Program Usage

int $gint_var		Declare Global integer-type variable $gint_var
func void main()		
int I	……	Declare integer-type variable L,M,H
	…	
end		

3.38.5 Integer-type GLOBAL Variable GI

Function: These variables hold the same function as the integer-type variable declared as int and are available for use commonly in the whole program.

Format: gi[Integer/variable] : The applicable range of an integer/variable is 0~999.
→ No blank should be between gi and [Integer/variable].

Description:

1) **Used in the program without declaration** unlike LOCAL variables.
2) The program allowes direct change of values for gi[integer].
3) Available for use in such commands as VEL, ACC, DEC, FOS, and DLAY.
   Ex) vel gi[20] : Sets the velocity at the value saved to I20.
4) Available for use in IF - (ELSE) - END block and WHILE-END block.
   Ex) if gi[7] 〉A0 예) while gi[4+7] ≤ 100
5) gi(integer/variable) type can be used for Global Point Variable.
   Ex) If gi[9] is 3, P[gi[9]] is identical to p[3].

3.38.6 Examples of Program Usage

```
func void main()
int l,m,h                ……  Declare integer-type variable L,M,H
…
l=gi[0]                  ……  Allocate L=30
…
m=gi[1]                  ……  Allocate M=60
…
h=gi[2]                  ……  Allocate H=100
…
take 1
vel(1, h)                ……  Set velocity to H(100)
…
jmov lp[0]
while gi[10]〈gi[11]      ……  While satisfying GI[10]〈GI[11]
…
vel(1, m)                    Set velocity to M(60)
jmov lp[gi[25]]          ……  Move to P[GI[25]]
…
vel(1, l)                ……  Assign velocity to L(30)
…
jmov lp[gi[26]]          ……  Move to P[GI[26]]
…
gi[10]=gi[10]+1          ……  Increase counter I10 by 1
```

Variable value GI

GI0 = 30
GI1 = 60
GI2 = 100
GI10 = 1
GI11 = 4
GI25 = 1
GI26 = 2

```
...
end
release
end
```

3.38.7 Real-type GLOBAL Variable GF

Function: This variable holds the same function as real-type variable declared as double and is available for use commonly in the whole program.

Format: gf[Integer/variable] : The applicable range of an integer/variable is 0~999.
→ No blank should be between gf and Integer/variable.

Description:
1) **Used in the program without declaration** unlike LOCAL variable.
2) The program allows direct change of gf[integer/variable] value.
3) Available for use in IF - (ELSE) - END block and WHILE-END block.
   Ex) IF F7 〉A0 THEN Ex) WHILE F49 ≤ 100

CAUTION

▶ When operations with real-type and integer-type variables are mixed in use, it **changes into the real-type.**
9.5 + **10** = 19.5

3.38.8 Examples of Program Usage

```
func void main()
pos pp
take 1
pp=lp[1]
for gi[0]=0 to 10
pp.1=gf[0]          ……  Substitute value F0 in the 1st axis of position-type variable pp1
jmov pp             ……  Move to pp
gf[0]=gf[0]+10.0    ……  Increase value F0 by 10.0 each
end
release
end
```

3.39 POSITION Variable

3.39.1 POS Variable

Function: Declares position-type variable (Simple variable and array variable)

Format: Simple variable → pos ⟨variable name⟩, ⟨variable name⟩, …
Array variable → pos ⟨variable name⟩ [size], {⟨variable name⟩ [size]}, …

Terms: ⟨variable name⟩ :

-. A variable name is a character string **combined with English characters and number** and array size.

-. The 1st letter of a variable is sure to be an **English character or '_'**. (Except for using "P" independently)

[size] : The size of an array variable indicates the number (Integer) of brackets.

The number of the brackets for array variables indicates **the number of rooms.**

Ex) When declaring POS AA(3), applicable variables are 3 - AA(0), AA(1), AA(2).

Description:

1) Declaration should be made **at the start of MAIN block**
2) Variable values declared as POS are angles of each axis, lead values.
3) **Each component** value for a variable declared as POS can be used as a **⟨variable name⟩.⟨number⟩.**
   Ex) POS JP1 → JP1.1 JP1.2 JP1.3 JP1.4…
4) POS variable in the angle value **has maximum 16 types of component**, and can be initialized into the format - variable name (Simple variable, array variable) =
   ⟨1-axis component, 2-axis component, 3-axis component, 4-axis component, 5-axis component, 6-axis component>

CAUTION

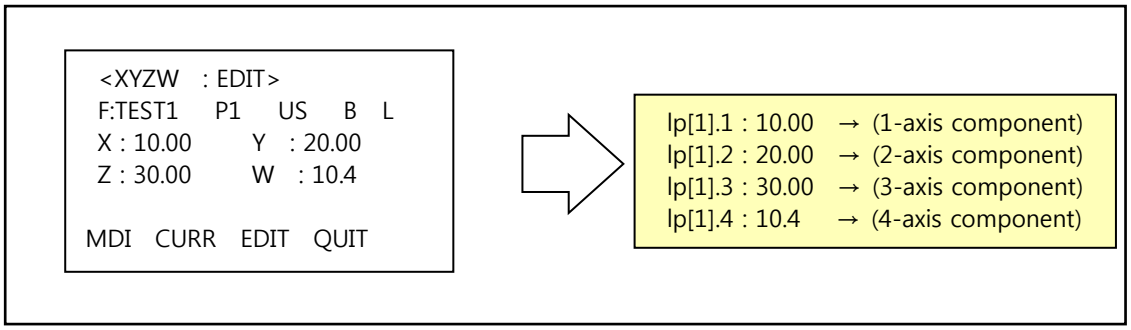
- **"P", "GP", "I", and "F" cannot be used** because they overlap with POINT variable and GLOBAL variable. (Alarm **"Syntax Error" , "Duplicated Symbol"** occurs.)
- Configured with maximum 16 axes in consideration of expandability and **value 0.0** should be entered **on an axis not in use.**

3.39.2 POINT Variable

Function Uses all points taught by the user in Job Editor

Format Variable per robot → LP(Number) (0 ≤ Number ≤ 39999)

Description
1) Variables are not declared at early stage unlike POS variables.
2) Each **component value** for POINT variables can be used **as <Point Number>.<Number>.**



3.39.3 Examples of Program Usage

1) Applies deceleration using POINT variables, POSITION variables

```
func void main()
pos mm                          ……  Declare position-type variables MM
pos a, b                            Declare position-type variables A, B
mm=⟨10.1,10.2,10.3,10.4,0,0⟩    ……  Initialize MM
a=⟨400.0,50.0,10.0,0.0,0.0,0.0⟩ ……  Initialize A
b=a                             ……  Initialize B variable identically to A value
b.3=a.3-40.0                    ……  Cut 3rd axis value in B by 40mm
take 1
vel(1000)
jmov lp[0]                      ……  Move PTP to position LP0
jmov b                          ……  Move to position coordinate B (Deceleration
                                    point)
vel(300)
jmov a                          ……  Move to position coordinate A
jmov mm                         ……  Move to position-type variable MM
vel(1000)
jmov lp[0]
release
end
```

3.40 System Variables (CNT, TMR, MVR, HERE)

3.40.1 CNT, TMR Variables

Function CNT : Saves counter values
TMR : Saves timer values

Format cnt(Pulse input BIT number)=〈Default value〉
tmr(0)=〈Default value〉
tmr(1)=〈Default value〉

Terms 〈Pulse input BIT number〉 :Input BIT number to get
〈Default value〉 : Integer from which counter(CNT), timer(TMR) starts

Description
1) System variables are used **without declaring data type**.
2) CNT variables allows **"0" to be saved at the moment the pulse input BIT is decided**, counting **each pulse input (pulse over 10m)** since then. (0 ≤ Input range ≤ 65,535)
3) TMR variable gets its value allocated from **the moment the integer value is entered** and grows by 1 each at a time, at the time interval defined in the system parameter. For applying the set value, refer to **TMR in Parameter Mode.**

⚠ CAUTION

▶ Be careful a BLANK should not be placed between CNT and pulse input port number, between TMR and "0" or "1".
Ex) CNT_(0)=0 (X), TMR_(1)=-100 (X)

3.40.2 Examples of Program Usage

1) Use of variables CNT, TMR

```
func void main()
vel(1, 100)
cnt(0)=0            ……  Initialize a counter
tmr(1)=-100         ……  Initialize a timer
while cnt(0)<20     ……  LOOP when 20 pulses are entered to input BIT "0"

jmov lp[0]
if (tmr(1)>0)       ……  When the timer runs over 1 second
goto TEST                (-100 ~> 0 : 100 units of 1ms occur.)
end
dlay 100
jmov lp[1]
end
labl TEST
end
```

3.40.3 MVR Variable

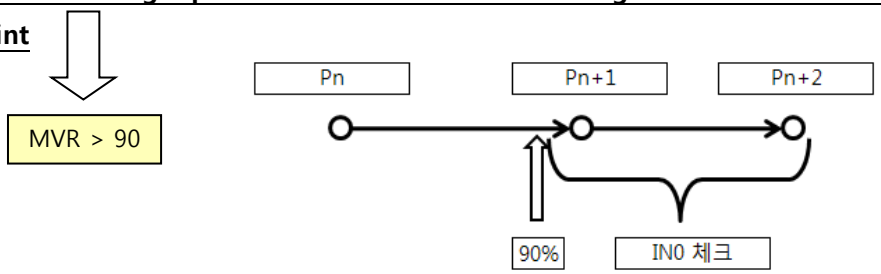
Function Rate of movement distance over the entire movement distance.

Format mvr 〈 Percentage of travel distance over the entire movement distance 〉

Description
1) Percentage of movement distance over the entire movement distance in case of movement between two points (Pn, Pn+1)
That is, MVR = (Movement distance/entire distance)*100
2) Used for conditional branch and I/O parallel processing during robot operation.
3) Be sure to use in combination with a WITH statement by dividing the movement area by the unit of 100.
4) When movement distance between two points (Pn, Pn+1) is short and when operating at high speed, pay attention to setting MVR value

CAUTION

▶ **When checking input "IN1" after 90% while moving from Pn Point to Pn+1 Point**



-. When the robot reaches about 90% of position Pn+1, an operation for moving to the next movement POINT(Pn+2) begins.
-. That is , the robot is still **operating the current #1 position** even when it is at 90% position as shown in the above figure.
★ Therefore, when the movement distance is short while operating at high speed, input Bit '0' status is occasionally unable to be checked 90% after departing Pn. In this case, reduce the robot movement speed before use.

IN0 체크 IN0 Check

CAUTION

Use of a current coordinate READ (HERE Command) function when a signal is fed during robot in motion
–. When a signal is fed during the robot in motion, the current coordinate READ function following the stop may make a deviation occur in the value for the current coordinate READ according to robot movement speed.
-. In this case, reduce the robot movement speed before use

3.40.4 Examples of Program Usage

1) Modify movement trajectory according to input Bit status during the robot in motion.

```
func void main()
vel(1, 10)
fos10
jmov lp[100]
mvr=0
with
jmov lp[0]
while mvr<100          ……  While moving point LP0 at 100%
if mvr>10              ……  Keep the output ON(1) for 0.5 seconds at 10%
                           of movement distance.
out(1)=1, 50           ……
end
in(1)=1
goto L1
end
labl L1
stop
end
jmov lp[1]
end
```

2) Start at point P1 and move to point P2 after 80% according to signal input in input "IN0"

```
func void main()
vel(1, 10)
while in(1)==1
plup 10
jmov lp[0]
mvr=0
with
jmov lp[1]
while mvr<100          ……  While moving point LP1 at 100%,
if (mvr>80)&&(in(0)==1)……  with movement distance at 80%, IN0 ON(1)
plup 0
jmov lp[22]
goto BB
end
end
end
in(0)=1
plup 0
jmov lp[2]
labl BB
end
out(1)=1, 50
end
end
```

3.40.5 HERE Variable

Function — Variable that saves the current position value

Format
HERE
Here(Integer)

Terms — 〈Integer〉 : Channel number (1 or 2)
-. Used when reading the current robot's position value between channels.

Description
1) **Saves the current robot's position value as an angle value or a lead value.**
2) **Saves the current robot position in other channels.**

3.40.6 Examples of Program Usage

1) READ the current position using HERE variables

```
func void main()
vel(1, 100)
pos curr
jmov lp[0]
mvr=0
with
jmov lp[1]
while mvr〈100
if mvr〉50
curr=here            ……  At a 50% point while moving to point LP1,
                         save the current position value to position-type
                         variable CURR
goto LLL
end
end
labl LLL
end
jmov lp[2]
jmov curr
end
```

2) READ the current robot position in other channels using a HERE<n> variable

```
func void main()
vel(1, 10)
pos ap
jmov lp[0]
while ap.1〈10.0
ap=here2             ……  Save the current position of channel 2 robot
dlay 10
end
out(0)=1
jmov lp[1]
end
```

3.41 Constant

Function Displays hexadecimal number, binary number.

Format 0H〈Number〉
0B〈Number〉

Terms 〈Number〉 : Decimal number when 0H or 0B is not placed in front of the number
* 0H : Hexadecimal number
* 0B : Binary number

Description 1) 0H〈Number〉 is hexadecimal, with numbers enabled up to 8-digit (16 bit).
2) 0B〈Number〉 is binary, with numbers enabled up to 16-digit (16 bit).
3) Either 0H or 0B is not present, it is a decimal constant or real number.

3.41.1 Examples of Program Usage

```
func void main()
int aa
vel(1, 100)
while 1
aa=pin(0) & 0H00FF          Compare the value for input port 0 to
                            hexadecimal number 00FF
if aa==1
pout(0)=0B00110100          Output binary number 00110100 to output port
                            0
md1()
end
if aa==1
pout(0)=0B01110100          Output binary number 01110100 to output port
                            0
md2()
end
end
end
```

3.42 Operator

3.42.1 Assignment Operator

Command	Function
=	Right formula is estimated and its results are converted to match the variable type at the left variables before being assigned. (Ex: IN0=1)

	Integer-type	Real-type	Position-type	Counter-type	Timer-type
Integer-type	Integer-type	Integer-type	**Not allowed**	Integer-type	Integer-type
Real-type	Real-type	Real-type	**Not allowed**	Real-type	Real-type
Position-type	**Not allowed**	**Not allowed**	Position-type	**Not allowed**	**Not allowed**
Counter-type	Integer-type	Integer-type	**Not allowed**	Integer-type	Integer-type
Timer-type	Integer-type	Integer-type	**Not allowed**	Integer-type	Integer-type

3.42.2 Arithmetic Operator

Command	Function
*,/,+,-,%	The following tables shows operation results of each operator for different data types. 'Not allowed' in the table means a formula **where an operation is unable to be made.**

▶ **+, - operators**

	Integer-type	Real-type	Position-type	Counter-type	Timer-type
Integer-type	Integer-type	Real-type	**Not allowed**	Integer-type	Integer-type
Real-type	Real-type	Real-type	**Not allowed**	Real-type	Real-type
Position-type	**Not allowed**	**Not allowed**	Position-type	**Not allowed**	**Not allowed**
Counter-type	Integer-type	Integer-type	**Not allowed**	Integer-type	Integer-type
Timer-type	Integer-type	Integer-type	**Not allowed**	Integer-type	Integer-type

▶ * Operator

	Integer-type	Real-type	Position-type	Counter-type	Timer-type
Integer-type	Integer-type	Real-type	Position-type	Integer-type	Integer-type
Real-type	Real-type	Real-type	Position-type	Real-type	Real-type
Position-type	Position-type	Position-type	**Not allowed**	Position-type	Position-type
Counter-type	Integer-type	Real-type	Position-type	Integer-type	Integer-type
Timer-type	Integer-type	Real-type	Position-type	Integer-type	Integer-type

▶ / operator

	Integer-type	Real-type	Position-type	Counter-type	Timer-type
Integer-type	Integer-type	Real-type	**Not allowed**	Integer-type	Integer-type
Real-type	Real-type	Real-type	**Not allowed**	Real-type	Real-type
Position-type	Position-type	Position-type	**Not allowed**	Position-type	Position-type
Counter-type	Integer-type	Real-type	**Not allowed**	Integer-type	Integer-type
Timer-type	Integer-type	Real-type	**Not allowed**	Integer-type	Integer-type

▶ % operator

Enables operation between integers only**, throwing away the quotient** in implementing division operation and taking the **remainder as return values.**

Ex) In AA=10%3, **AA value is 1.**

3.42.3 Relational Operator

Command	Function
〉, 〈, ≤, ≥, ==, !=	-. Mainly for integer-type and real-type data and used in conditional statements, such as an IF statement and WHILE statement. -. The results of relational operator are a logic value (true,false or 0/1).

3.42.4 Logical Operator

Command	Function
&&, ll, !	A logical operator is only for logical values. That is, it uses the results of relational operator as an operand. The results of logical operation are a logic value.

3.42.5 Bit Operator

Command	Function
&, l 〈〈, 〉〉 ~	Logical operator Mobile operator Complementary number of 1

3.43 Built-in Function

Function	Function	Usage Example
EXP(X)	Exponent e^x	EXP(3) is 20.085.
LOG(X)	Common log, $\log_{10}X$	LOG(100) is 2.
LN(X)	Natural log, $\log_e X$	LN(15) is 2.708.
SQRT(X)	Square root $\sqrt{}$	SQRT(16) is 4.
POW(x,y)	x^y	POW(2,10) is 1024.
ABS(K)	Absolute value for integer K	ABS(-12) is 12.
RND(X)	Round the real number X to integer	RND(97.62) is 98.
SIN(X)	sine, x unit : radian	SIN(RAD(30)) is 0.5.
COS(X)	cosine, x unit : radian	COS(0) is 1.
TAN(X)	tangent, x unit : radian	TAN(RAD(45)) is 1.0.
ASIN(X)	arcsine, return - π/2~π/2 values	DEG(ASIN(0.5)) is 30.
ACOS(X)	arccosine, return 0 ~ π	DEG(ACOS(0.5)) is 60.
ATAN(X)	arctangent, return -π/2 ~π/2 values	DEG(ATAN(1.0)) is 45.
ATAN2(Y,X)	2nd arctangent, return -π~π values	DEG(ATAN2(-1,-1)) is -135.
DEG(X)	Convert radian to angle value	DEG(3.1416) is 180.0.
RAD(X)	Convert angle to radian value	RAD(180.0) is 3.1416.
MIN(X,Y)	Return minimum value in x, y	MIN(2,5) is 2.
MAX(X,Y)	Return maximum value in x, y	MAX(2,5) is 5.

3.44 String

3.44.1 ASC

Function Returns the 1st character in the string to the character code

Format string variable = asc(string)

Terms string variable :

−. Means a variable declared using a string and is used to handle a string string.

(String) : Means a string variable or a string constant. The string constant is a group of characters surrounded by " ". (ex) "ABCDEF")

Description 1) Returns the first character of the string entered in ASC.

Examples of Program

```
func void main()
string aa, bb
aa = "XYZW"
bb = asc(AA)
end
```

Substitute "XYZW" in string variable

Return the 1st character "X" in string variable AA

3.44.2 BINS

Function Converts an integer to a binary number string.

Format string variable = bins(Integer)

Terms string variable :

-. Means a variable declared using a string and is used to handle a string.

Description 1) Converts the entered integer to binary (0/1) string.

Examples of Program

```
func void main()
string aa
int b
b = 10
aa = bins(b)
end
```

Save the value for integer-type variable b to string variable aa in binary string.

3.44.3 CHR

Function Converts the integer in the range of ascii codes to characters

Format string variable = chr(integer)

Terms string variable :

-. Means a variable declared using a string and is used to handle a string.

(Integer) : Integer (0~127) within ascii code values

Description 1) Returns an ascii code that corresponds to the entered integer in ASC.

Examples of Program

```
func void main()
string aa
aa = chr(65)
end
```

Substitute 'A' coming under asci code 65 in string variable AA

3.44.4 FLUSH

Function Clears input, output buffer

Format FLUSH(Integer)

Terms (Integer) : Select clear butter (1~3)

* 1: Input buffer clear
* 2: Output buffer clear
* 3: Input, output buffer clear

Description 1) Clears input/output buffers used for serial communication transmission/reception.

Examples of Program

```
func void main()
string aa
aa = strin(1000)
flush(1)
end
```

Saved to string AA by receiving the input through Serial communication

Clear input buffer

3.44.5 FTOS

Function	Converts an integer or real-number to a string
Format	string variable = ftos (integer or real number)
Terms	(integer or real number) : Integer or real number to be converted to a string
Description	1) Converts integer or real-number data to a string. ex) 1234 → "1234"

Examples of Program

```
func void main()
string aa, bb
aa = ftos(1234)
bb = ftos(-123.456)
end
```

Convert 1234 to string "1234" and substitute it to string variable AA

Convert -123.456 to string "-123.456" and substitute it to string variable BB

3.44.6 HTOS

Function Converts an integer to a hexadecimal string

Format string variable = htos(integer)

Terms

Description 1) Converts the integer entered in HTOS to a hexadecimal string

Examples of Program

```
func void main()
string aa
aa = htos(10)
end
```

Convert integer 10 to a hexadecimal string ("A") and substitute it to string AA

3.44.7 SLEFT

Function	Extracts the left string from entered string
Format	string variable = sleft(string, integer)
Terms	(integer) : The number of characters to be extracted
Description	1) Extracts the entered string from the left as much as entered integer

Examples of Program

```
func void main()
string aa, bb
aa = "XYZW"
bb = SLEFT(AA, 2)
end
```

Substitute "XYZW" to string variable AA

Return 2 characters ("XY") from the left side of string

3.44.8 SLEN

Function	Returns the length of the entered string
Format	Integer-type variable = slen(string)
terms	(integer) : The number of characters to be extracted
Description	1) Returns the length of the entered string.

Examples of Program

```
func void main()
string aa
int ll
aa = "XYZW"
ll = slen(AA)
end
```

Substitute "XYZW" to string variable AA

Substitute length(4) of string variable AA in integer-type variable LL

3.44.9 SMID

Function Extracts the string from the assigned string position as many as digits

Format Extracts the string from the assigned string position as many as digits

Terms (Assigned position) : Refers to where a string extraction begins, with a string starting at 0.

ex) In "ABCD", A is in 0^{th}, B in 1^{st}, C in 2^{nd}, and D in the 3^{rd} position.

(integer) : The number of characters to be read from the assigned position

Description 1) Extracts as many characters as user input constants from the assigned position.

Examples of Program

```
func void main()
string aa, bb
aa = "ABCDEFG"
bb = smid(AA, 2, 3)
end
```

Substitute "ABCDEFG" to string variable AA

Return 3 characters ("CDE") from the 2nd position of a string variable AA

3.44.10 SPOS

Function	Returns the start position where string 2 is matched in string 1
Format	Integer-type variable = spos(string 1, string 2)
Terms	
Description	1) Returns the start position where string 2 is matched in string 1.

Examples of Program

```
func void main()
string aa, bb
int pp
aa = "XYZW"
bb = "Z"
pp = spos(AA, BB)
end
```

Substitute "XYZW" to string variable AA
Return the first character in string variable AA
Return the position consistent with BB in string variable AA

3.44.11 SRIGHT

Function	Extracts the right part of the string entered
Format	string variable = sright(String, integer)
Terms	(integer) : The number of characters to be extracted
Description	1) Extracts the entered string from the right as many as entered integer

Examples of Program

```
func void main()
string aa, bb
aa = "ABCDEF"
bb = sright(AA, 2)
end
```

Substitute "ABCDEF" to string variable AA

Return 2 characters ("EF") on the right of string variable AA

3.44.12 STRIN

Function	Reads a string entered through serial communication to the identifier
Format	string variable = strin(integer)
Terms	(integer) : Time out duration. (unit : ms) Waits for serial input as much as time out duration. If data does not come in during time out duration, check out the time out status in the system area and carry out the following step command.
Description	1) Reads the input through serial communication while waiting as much as time assigned by the user

Examples of Program

```
func void main()
string packet
int len

packet = strin(1000)        String input through serial communication
                            (wait 1000ms when no input is made)
len = slen(packet)          Save the length of entered string


if len > 0                  When entered string exists, not TIMEOUT
...
end

end
```

3.44.13 STROUT

Function Outputs strings through serial communication

Format Integer-type variable = strout (string)

Terms

Description 1) Sends out the outputs of entered strings through serial communication, returning the number of characters unable to be transmitted

ex) When transmitting character "ABCDEFGH" is successful, the return value is 0, whereas it is 8 when partially transmitted.

Examples of Program

```
func void main()
string sendpk
int strret
sendpk = "ABCDEF"
strret = strout(sendpk)
if strret != 0
    //transmission failure..
end
end
```

3.44.14 SVAL

Function Converts a string to a number

Format (integer-type or real-type) variable = sval (string)

Terms

Description 1) Converts the entered string to an integer or real-number data.

Examples of Program

```
func void main()
int ai
double br
ai = sval("1234")
br = sval("12.234")
end
```

Convert string "1234" to integer 1234
Convert string "12.234" to real number 12.234

3.45 LOG Command

3.45.1 PRINT

Function Saves a log to a memory

Format print (memory number, log content 1, log content 2, …, log content n)

Terms Memory number : the number of memory to save (0~999)
Log content is available for use in integer-type, real-type or letter-type.

Description 1) Saves what the user has recorded to the memory.
2) When the power to the control turns off, the records saved to the memory disappear.

Examples of Program

```
func void main()
int ai
double br
ai = sval("1234")
print(0, "1234", ai)
end
```

Convert string "1234" to integer 1234

Save string "1234" and value for integer variable ai to memory

3.45.2 WLOG

Function Saves a log to a file

Format wlog(log content 1, log content 2, …, log content n)

Terms Log content is available for use in integer-type, real-type or letter-type

Description 1) Saves what the user has recorded to file.

Examples of Program

```
func void main()
int ai
double br
ai = sval("1234")
wlog("1234", ai)
end
```

Convert string "1234" to integer 1234

Save string "1234" and value for integer variable ai to file

3.46 LATCH Command (EtherCAT Communication Type)

3.46.1 LATCH_INIT

Function	Configuration for performing a latch
Format	integer-type variable = latch_init(axis, latch method, the number of sensors)
Terms	axis : axis on which latch is to be performed latch method : rising edge method, falling edge method (1 : rising edge, 2 : falling edge) the number of sensors : number of sensors to be used in latch (minimum 1, maximum 2)
Description	1) Sets an axis on which latch is to be used, latch method, and the number of sensors. 2) One-time setup is just enough unless latch setting changes.

Examples of Program

Read latch status on sensor 1 on axis 3

```
func void main()
int l_state[2]
double l_pos[2]

release
take 1
while 1
jmov lp[0]
dlay 100

latch_clear(3)                          Initialize latch setting on axis 3
latch_init(3, 2, 2)                     Latch configuration using two sensor on axis 3 with a
                                        falling edge method

latch_start()                           Start latch
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          Read latch status on sensor 1 on axis 3
l_state[1] = latch_state(3, 2)          Read latch status on sensor 2 on axis 3

if( l_staet[0] == 2 && l_state[1] ==2)  Does latch value exist in sensors 1, 2 on axis 3?
l_pos[0] = latch_pos(3, 1)              Read latch position in sensor 1 on axis 3
l_pos[1] = latch_pos(3, 2)              Read latch position in sensor 2 on axis 3
break
end
end
end
latch_stop()                            Stop latch
dlay 100
end
end
```

3.46.2 LATCH_CLEAR

Function Initializes latch setting.

Format integer-type variable = latch_clear(axis)

Terms Axis : axis on which latch setting is to be initialized
0 : initializes latch setting on all axes
1 ~ maximum axis : initializes latch setting for the corresponding axis only

Description 1) Turns OFF latch functions by initializing latch setting.

Examples of Program

```
func void main()
int l_state[2]
double l_pos[2]

release
take 1
while 1
jmov lp[0]
dlay 100

latch_clear(3)                          Initialize latch setting on axis 3
latch_init(3, 2, 2)                     Latch configuration using two sensor on axis 3 with a
                                        falling edge method

latch_start()                           Start latch
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          Read latch status on sensor 1 on axis 3
l_state[1] = latch_state(3, 2)          Read latch status on sensor 2 on axis 3

if( l_staet[0] == 2 && l_state[1] ==2)  Does latch value exist in sensors 1, 2 on axis 3?
l_pos[0] = latch_pos(3, 1)              Read latch status on sensor 1 on axis 3
l_pos[1] = latch_pos(3, 2)              Read latch status on sensor 2 on axis 3
break
end
end
end
latch_stop()                            Stop latch
dlay 100
end
end
```

3.46.3 LATCH_START

Function Starts latch.

Format integer-type variable = latch_start()

Terms

Description 1) starts latch.
2) 0 returns when latch_init has not been performed.

Examples of Program

```
func void main()
int l_state[2]
double l_pos[2]

release
take 1
while 1
jmov lp[0]
dlay 100

latch_clear(3)                          Initialize latch setting on axis 3
latch_init(3, 2, 2)                     Latch configuration using two sensor on axis 3 with a
                                        falling edge method

latch_start()                           Starts latch
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          Read latch status on sensor 1 on axis 3
l_state[1] = latch_state(3, 2)          Read latch status on sensor 2 on axis 3

if( l_staet[0] == 2 && l_state[1] ==2)  Does latch value exist in sensors 1, 2 on axis 3?
l_pos[0] = latch_pos(3, 1)              Read latch status on sensor 1 on axis 3
l_pos[1] = latch_pos(3, 2)              Read latch status on sensor 2 on axis 3
break
end
end
end
latch_stop()                            Stops latch
dlay 100
end
end
```

3.46.4 LATCH_STOP

Function Stops latch.

Format integer-type variable = latch_stop()

Terms

Description 1) Stops latch.

Examples of Program

```
func void main()
int l_state[2]
double l_pos[2]

release
take 1
while 1
jmov lp[0]
dlay 100

latch_clear(3)                          Initialize latch setting on axis 3
latch_init(3, 2, 2)                     Latch configuration using two sensor on axis 3 with a falling edge method

latch_start()                           Starts latch
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          Read latch status on sensor 1 on axis 3
l_state[1] = latch_state(3, 2)          Read latch status on sensor 2 on axis 3

if( l_staet[0] == 2 && l_state[1] ==2)  Does latch value exist in sensors 1, 2 on axis 3?
l_pos[0] = latch_pos(3, 1)              Read latch status on sensor 1 on axis 3
l_pos[1] = latch_pos(3, 2)              Read latch status on sensor 2 on axis 3
break
end
end
end
latch_stop()                            Stops latch
dlay 100
end
end
```

3.46.5 LATCH_POS

Function Reads the position where latch sensor is detected.

Format Real-type variable = latch_p
os(axis, sensor number)

Terms axis : axis on which latch is to be performed (1 ~ maximum axis)
sensor number : latch sensor number (1~2)

Description 1) When latch sensor is detected, this command involves reading the detected position.
2) It is required to call when latch status is detected by the sensor through the command latch_state An alarm occurs when calling in a situation where the sensor detection has not been made.

Examples of Program

```
func void main()
int l_state[2]
double l_pos[2]

release
take 1
while 1
jmov lp[0]
dlay 100

latch_clear(3)                          Initialize latch setting on axis 3
latch_init(3, 2, 2)                     Latch configuration using two sensor on axis 3 with a
                                        falling edge method

latch_start()                           Starts latch
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          Read latch status on sensor 1 on axis 3
l_state[1] = latch_state(3, 2)          Read latch status on sensor 2 on axis 3

if( l_staet[0] == 2 && l_state[1] ==2)  Does latch value exist in sensors 1, 2 on axis 3?
l_pos[0] = latch_pos(3, 1)              Read latch status on sensor 1 on axis 3
l_pos[1] = latch_pos(3, 2)              Read latch status on sensor 2 on axis 3
break
end
end
end
latch_stop()                            Stops latch
dlay 100
end
end
```

3.46.6 LATCH_STATE

Function Check latch operation status.

Format Integer-type variable = latch_state(axis, sensor number)

Terms axis : axis on which latch status is to be checked (1 ~ maximum axis)
sensor number : latch sensor number (1~2)

Description 1) Check latch status. Returns the following 3 values depending on condition.
0 : latch function not started.
1 : latch started but sensor detection has not been made.
2 : latch function started and sensor detection has been completed. latch_pos command can be used.

Examples of Program

```
func void main()
int l_state[2]
double l_pos[2]

release
take 1
while 1
jmov lp[0]
dlay 100

latch_clear(3)                          Initialize latch setting on axis 3
latch_init(3, 2, 2)                     Latch configuration using two sensor on axis 3 with a
                                        falling edge method

latch_start()                           Start latch
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          Read latch status on sensor 1 on axis 3
l_state[1] = latch_state(3, 2)          Read latch status on sensor 2 on axis 3

if( l_staet[0] == 2 && l_state[1] ==2)  Does latch value exist in sensors 1, 2 on axis 3?
l_pos[0] = latch_pos(3, 1)              Read latch status on sensor 1 on axis 3
l_pos[1] = latch_pos(3, 2)              Read latch status on sensor 2 on axis 3
break
end
end
end
latch_stop()                            Stop latch
dlay 100
end
end
```

3.47 LATCH Command (RS422 Communication Type)

3.47.1 LATCH_INIT

Function Does configuration for performing latch.

Format integer-type variable = latch_init(axis, sensor start number, number of sensors)

Terms Axis : axis on which latch is to be performed (integer type)
Sensor start number: sets the start position of the sensor to detect (integer in 1~4)

Range Value	1	2	3	4
Sensor used	1, 2	2, 3	3, 4	4

Number of sensors : The number of sensors to be used in latch (minimum 1 to maximum 2, integer type)

Description 1) Sets an axis on which latch is to be used, start point of a sensor to detect, and the number of sensors.
2) One-time setup is just enough unless latch setting changes.
3) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret
    double l_pos[2]

    release
    take 1
    svon(1)

    ret=latch_clear(1)
    if(ret==0)
        goto close_routine
    end

    ret=latch_init(1,1,2)
    if(ret==0)
        goto close_routine
    end

    ret=latch_start()
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)
```

Initialize latch setting on axis 1

Configure latch setting for axis 1 to read 2 sensors in position 1 (sensor 1, 2).

Start latch

When a signal is detected by reading the status value for sensor 1 on axis 1, the position value on the corresponding sensor is read.

```
            l_pos[0]=latch_pos(1,1)
        end

        if(latch_state(1,2)==2)
            l_pos[1]=latch_pos(1,2)
        end

        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

When a signal is detected by reading the status value for sensor 2 on axis 1, the position value on the corresponding sensor is read.

End latch

3.47.2 LATCH_CLEAR

Function Initializes latch setting.

Format integer-type variable = latch_clear(axis)

Terms Axis : axis on which latch setting is to be initialized (integer type)
0 : Initializes latch setting on all axes
1 ~ maximum axis : Initializes latch setting on the corresponding axis only

Description 1) Sets an axis on which latch is to be used, start point of a sensor to detect, and the number of sensors.
2) One-time setup is just enough unless latch setting changes.
3) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret
    double l_pos[2]

    release
    take 1
    svon(1)

    ret=latch_clear(1)
    if(ret==0)
        goto close_routine
    end
    ret=latch_init(1,1,2)
    if(ret==0)
        goto close_routine
    end
    ret=latch_start()
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)
            l_pos[0]=latch_pos(1,1)
        end
        if(latch_state(1,2)==2)
            l_pos[1]=latch_pos(1,2)
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
```

Clear latch on axis 1

Configure latch setting for axis 1 to read 2 sensors in position 1 (sensor 1, 2).

Start latch

When a signal is detected by reading the status value for sensor 1 on axis 1, the position value on the corresponding sensor is read

When a signal is detected by reading the status value for sensor 2 on axis 1, the position value on the corresponding sensor is read

End latch

```
end
```

3.47.3 LATCH_START

Function Starts latch.

Format integer-type variable = latch_start()

Terms

Description 1) Starts latch.
2) 0 returns when latch_init has not been performed.
3) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret
    double l_pos[2]

    release
    take 1
    svon(1)

    ret=latch_clear(1)
    if(ret==0)
        goto close_routine
    end
    ret=latch_init(1,1,2)
    if(ret==0)
        goto close_routine
    end
    ret=latch_start()
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)
            l_pos[0]=latch_pos(1,1)
        end
        if(latch_state(1,2)==2)
            l_pos[1]=latch_pos(1,2)
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

Clear latch on axis 1

Configure latch setting for axis 1 to read 2 sensors in position 1 (sensor 1, 2).

Start latch

When a signal is detected by reading the status value for sensor 1 on axis 1, the position value on the corresponding sensor is read

When a signal is detected by reading the status value for sensor 2 on axis 1, the position value on the corresponding sensor is read

End latch

3.47.4 LATCH_STOP

Function Starts latch.

Format integer-type variable = latch_stop()

Terms

Description 1) Starts latch.
2) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret
    double l_pos[2]

    release
    take 1
    svon(1)

    ret=latch_clear(1)
    if(ret==0)
        goto close_routine
    end
    ret=latch_init(1,1,2)
    if(ret==0)
        goto close_routine
    end
    ret=latch_start()
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)
            l_pos[0]=latch_pos(1,1)
        end
        if(latch_state(1,2)==2)
            l_pos[1]=latch_pos(1,2)
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

Clear latch on axis 1

Configure latch setting for axis 1 to read 2 sensors in position 1 (sensor 1, 2).

Start latch

When a signal is detected by reading the status value for sensor 1 on axis 1, the position value on the corresponding sensor is read

When a signal is detected by reading the status value for sensor 2 on axis 1, the position value on the corresponding sensor is read

End latch

3.47.5 LATCH_POS

Function Reads the position where latch sensor is detected.

Format real-type variable = latch_pos(axis, sensor index)

Terms axis : axis on which latch is to be performed (1 ~ maximum axis, integer type)
sensor index: Index of sensors to be used, which is set in latch_init function (1~2, integer type)

Range Value	1	2	3	4
Sensor used	1, 2	2, 3	3, 4	4
Index : 1	**1**	**2**	**3**	**4**
Index : 2	**2**	**3**	**4**	**X**

Description 1) When latch sensor is detected, this command involves reading the detected position.
2) It is required to call when latch status is detected by the sensor through the command latch_state An alarm occurs when calling in a situation where the sensor detection has not been made.
3) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret
    double l_pos[2]

    release
    take 1
    svon(1)

    ret=latch_clear(1)
    if(ret==0)
        goto close_routine
    end
    ret=latch_init(1,1,2)
    if(ret==0)
        goto close_routine
    end
    ret=latch_start()
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)
            l_pos[0]=latch_pos(1,1)
        end
```

Clear latch on axis 1

Configure latch setting for axis 1 to read 2 sensors in position 1 (sensor 1, 2).

Start latch

When a signal is detected by reading the status value for sensor 1 on axis 1, the position value on the corresponding sensor is read

```
        if(latch_state(1,2)==2)
            l_pos[1]=latch_pos(1,2)
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

When a signal is detected by reading the status value for sensor 2 on axis 1, the position value on the corresponding sensor is read

End latch

3.47.6 LATCH_STATE

Function Checks latch operating status.

Format integer-type variable = latch_state(axis, sensor Index)

Terms axis : axis on which latch status is to be checked (1 ~ maximum axis)
sensor index : Index of sensors to be used, which is set in latch_init function (1~2, integer type)

Range Value	1	2	3	4
Sensor used	1, 2	2, 3	3, 4	4
Index : 1	**1**	**2**	**3**	**4**
Index : 2	**2**	**3**	**4**	**X**

Description 1) Checks latch status. Returns the following 3 values depending on condition.
0 : latch not started.
1 : Latch function started but sensor was not detected.
2 : latch function started and sensor detection has been completed. latch_pos command can be used.

Examples of Program

```
func void main()
    int ret
    double l_pos[2]

    release
    take 1
    svon(1)

    ret=latch_clear(1)
    if(ret==0)
        goto close_routine
    end
    ret=latch_init(1,1,2)
    if(ret==0)
        goto close_routine
    end
    ret=latch_start()
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)
            l_pos[0]=latch_pos(1,1)
        end
```

Clear latch on axis 1

Configure latch setting for axis 1 to read 2 sensors in position 1 (sensor 1, 2).

Start latch

When a signal is detected by reading the status value for sensor 1 on axis 1, the position value on the corresponding sensor is read

```
        if(latch_state(1,2)==2)
            l_pos[1]=latch_pos(1,2)
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

When a signal is detected by reading the status value for sensor 2 on axis 1, the position value on the corresponding sensor is read

End latch

3.48 ERROR Command

3.48.1 SETERR (SYSTEM EMG ALARM COMMAND)

Function Generates System emergency alarm

Format seterr(alarm code)

Terms Alarm codes : are set at the user's discretion

Description 1) Refers to a command generating system emergency alarm in job, with alarm codes entered at the user's discretion.

Examples of Program

```
func void main()
int tmp
take 1
vel(10)
tmp = rstate(1,6)                    ..........   Check if robot 1 is servo on
if( tmp == 0 )                                    If not servo on
  seterr(10)                                      Generate system emergency alarm with alarm code
end                                               10
jmov lp[100]
release
end
```

3.48.2 RESET(ALARM CLEAR COMMAND)

Function Clears an alarm

Format Reset

Terms

Description 1) Clears the alarm generated.

Examples of Program

```
func void main()
int tmp
take 1
vel(10)
tmp = rstate(1,1)                          Check alarm status
if( tmp == 1 )          ……                 If in alarm status
  reset                                    Clear alarm
  svon(1)
end
jmov lp[100]
release
end
```

3.48.3 RERROR(ERROR CODE RETURN COMMAND)

Function Returns an error code

Format rerror()

Terms

Description 1) Returns the error code generated the last.

Examples of Program

```
func void main()
int tmp
int ecode
take 1
vel(10)
tmp = rstate(1,1)                          Check alarm status
if( tmp == 1 )                …………      If in alarm status
  ecode = rerror()                         Check alarm code
end
…
release
end
```

3.48.4 RERRCNT(Error Occurrence Count Return Command)

Function Returns the number of error occurrences

Format rerrcnt()

Terms

Description 1) Returns the number of error occurrences.

Examples of Program

```
func void main()
int tmp
int ecode
int ecount
int aa
take 1
vel(10)
tmp = rstate(1,1)                          Check alarm status
if( tmp == 1 )                  ……        If in alarm status
  ecount = rerrcnt()                       Check the number of alarms

  for aa = 1 to ecount                     Check as many error codes as the number of alarms
    ecode = rerrcode(aa)                   Read error code
    if( ecode == 0 )                       End error read when a return value of rerrcode is 0
      break
  end
end
…
release
end
```

3.48.5 RERRCODE(Error Code Return Command)

Function Returns an error code

Format rerrcode(alarm occurrence order)

Terms

Description 1) Returns an alarm code that corresponds to alarm occurrence order entered with parameters.
2) As a reference, rerror() returns the alarm code generated the last.

Examples of Program

```
func void main()
int tmp
int ecode
int ecount
int aa
take 1
vel(10)
tmp = rstate(1,1)                          Check alarm status
if( tmp == 1 )                  ……        If in alarm status
  ecount = rerrcnt()                       Check the number of alarms

  for aa = 1 to ecount                     Check as many error codes as the number of alarms
    ecode = rerrcode(aa)                   Read error code
    if( ecode == 0 )                       End error read when the return value of rerrcode is 0
      break
  end
end
…
release
end
```

3.48.6 RERRTEXT(Error Content Return Command)

Function Returns error content

Format rerrtext (alarm occurrence order)

Terms

Description 1) Returns alarm content that corresponds to alarm occurrence order entered with parameters.

Examples of Program

```
func void main()
int tmp
string etext
int ecount
int len
take 1
vel(10)
tmp = rstate(1,1)                    ……                Check alarm status
if( tmp == 1 )                                         If in alarm status
  while 1
    etext = rerrtext(aa)                               Read error content
    len = slen(etext)                                  Check the length of a string read through rerrtext function
    if(len == 0 )                                      When the length of a string read through rerrtext function is 0,
      break
  end                                                  End error read
end
…
release
end
```

3.49 MAPPING Command (RS422 Communication Type only)

3.49.1 MAP_INIT

Function Initializes mapping-related internal variables.

Format integer-type variable = map_init ()

Terms

Description 1) Refers to a function that initializes a variable prior to the start of mapping.
2) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                          Initialize a variable
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

    ret=map_start(2,1,100)                  Start mapping
    if(ret==0)
        goto close_routine
    end

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)   Read mapping data (Only until maximum time-out duration)
    dlay 1000

    ret=map_stop()                          End mapping

labl close_routine
            ret=map_stop()
end
```

3.49.2 MAP_START

Function Starts mapping.

Format integer-type variable = map_start(axis, sensor index, start local point index)

Terms axis: 1~ maximum axis (integer type)
sensor index : enters the index value of a sensor (Range: 1~2)
Start local point: Index of a start point where data will be saved, Range: 0~19999

Description 1) Starts mapping and assigns a point where data will be saved.
2) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                          Initialize a variable
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

    ret=map_start(2,1,100)                  Start mapping
    if(ret==0)
        goto close_routine
    end

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)   Read mapping data (Only until maximum time-out duration)
    dlay 1000

    ret=map_stop()                          End mapping

labl close_routine
        ret=map_stop()
end
```

3.49.3 MAP_STOP

Function Stops mapping.

Format integer-type variable = map_stop()

Terms

Description 1) Stops mapping.
2) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                                  Initialize a variable
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

    ret=map_start(2,1,100)                          Start mapping
    if(ret==0)
        goto close_routine
    end

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)  Read mapping data (Only until maximum time-
    dlay 1000                                       out duration)

    ret=map_stop()                                  End mapping

labl close_routine
        ret=map_stop()
end
```

3.49.4 MAP_READ

Function Reads mapping data.

Format integer-type variable = map_read(Time-out duration)

Terms Time-out duration: enters maximum time(ms) to wait for reading data.

Description 1) Reads mapping data to save it to the point assigned from map_start.
2) If a return value is 0, a function performed is a failure and 1 means a success.

Examples of Program

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                          Initialize a variable
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

    ret=map_start(2,1,100)                  Start mapping
    if(ret==0)
        goto close_routine
    end

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)   Read mapping data (Only until maximum time-out duration)
    dlay 1000

    ret=map_stop()                          End mapping

labl close_routine
        ret=map_stop()
end
```

Rev.	Date of Revision	Content	Revisded by	S/W Version
V.1	2015.11.16	Print First Edition		

CONTROLLER MANUAL

FIRST EDITION NOV 2015

ROBOSTAR CO, LTD

ROBOT R&D CENTER