ROBOSTAR ROBOT

N1 Series

PROGRAMMING MANUAL

- □ INSTRUCTION MANUAL
- □ OPERATION MANUAL
- ☑ PROGRAMMING MANUAL
- □ UNI-HOST MANUAL
- □ GAIN SETUP MANUAL
- □ ALARM CODE MANUAL

Robostar

www.robostar.co.kr

About Product Warranty

Products of Robostar Co., Ltd. are manufactured under the strict quality control. All the Robostar products' warranty period is one year from the date of manufacture. In this period, Robostar is only responsible for the mechanical failures due to negligence of Robostar, or the problems on design and manufacture occurring during normal use, in which the service is free of charge.

However, such free service is not possible in the following cases.

(1) after the warranty period has expired

(2) failures arising due to improper repair, alteration, redeployment, or other mishandling, when, under the instruction of you or any third party

(3) failures as a result of using parts, grease, etc. which have not been designated by Robostar.

(4) failures caused by accidents, such as fire, disaster, earthquake, storms, or other natural disasters

(5) failures caused in manure, flooding, or other environment

(6) failure caused by the consumption of consumable parts

(7) failures arising when not being operated under the instructions listed in the user or instruction manual and the maintenance manual

(8) damages in cost other than the cost of robot repairing

Address and contact points of Robostar Robostar Co., Ltd.

- Head Office & Factory
  119-38, Sasa-dong, Sangnok-gu,
  Ansan-City, Gyeonggi-do, Republic of Korea (426-220)

- 2nd Factory
  960, Gosaek-dong, Gwonseon-gu,
  Suwon-City, Gyeonggi-do, Republic of Korea (441-813)

- Request for service and Inquiry of products
  - Information on Marketing
    TEL. 031-400-3600
    FAX. 031-419-4249
  - Customer Service

Robostar

www.robostar.co.kr

Constitution of User Manual

The user manual for this product is constituted as follows. When using this product first, please fully read all the manuals before use.

- **Instruction Manual**

  A controller is generally explained. Overview of the controller, installation, and interfacing to peripherals are explained.

- **Manipulation & Operation Manual**

  As well as general use of the controller, parameter setup, JOB program editing, robot operation, etc. are explained.

- **Programming Manual**

  RRL (Robostar Robot Language) which is the robot program of Robostar and how to write a robot program by using RRL are explained.

- **Unihost Manual**

  'Unihost' which is the on-line PC program of Robostar is explained.

- **Gain Setup Manual**

  How to set up the gain necessary for trial run, and the motor response performance according to change in the gain value are explained.

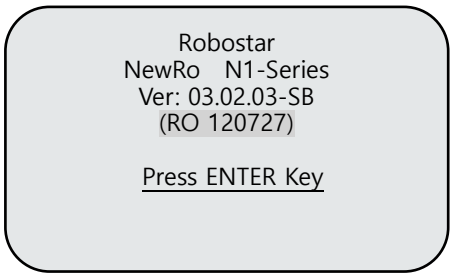
- **Alarm Code Manual**

  Reasons for and countermeasures against the alarms which can occur while operating the controller are explained.

Contents

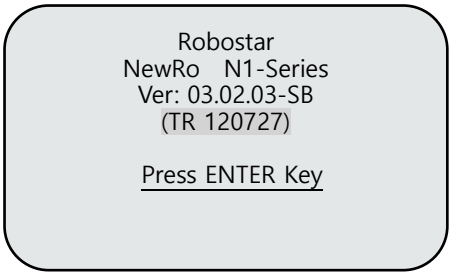
● N1 Controller Version

N1 controller is divided into two types of application software depending on robot type, each application software to be confirmed on the Teach Pendant screen when turning power to N1 controller ON.



Scara Robot, Cartesian Robot, DeskTop, Synchronous Robot Application Software (RO)

Robot Type	
SCARA	RSA-40A, RSA-50A, RSA-60A, RSA-70A, RSA-70B, RGA-80A, RGA-80B, RHA-80C, RBSA-90A, RBSA-90B, RGA-A0A, RGA-A0B, RGA-A1A, RGA-A1B, SC-TEST
XYZW	XY__, X_Z_, XYZ_, XZY_, XY_W, X_ZW, XYZW, XZYW, XYZH, XZYR, XYXY, XYZXY_, XYZXYZ, XY-TEST
DESK	RDS2_, RDS3_, RDS4_, RDT4_F, RDT4_A
DUAL	D2S, D2O, D3S, D3O, D4S, D4O, DDSO, DDOS, DDOO, DUAL
BGT	BGT



Robot application software for transferring semiconductor (TR)

Transfer Robot Type	
TR	TML__, TMS__, CML__, ROSEP300, ROSEP200
BGT	BGT

Caution ▶ Before running a robot, be sure to check application software and robot type.
▶ Normal operation is possible only when the robot in use and application software on N1 controller are consistent.

● Changes by Controller Versions

VER.	RO(Robot)	TR(Transfer Robot)
Global INT	500EA(0~499)	2000EA(0~1999)
Global FLOAT	500EA(0~499)	2000EA(0~1999)
Local Point	2000EA(0~1999)	2000EA(0~1999)
Global Point	1024EA(0~1023)	15000EA(0~14999)

What is described in this manual is based on RO(Robot) version and refer to the above table for any part to be changed when using TR(Transfer Robot) version.

Chapter 1 Introduction

1.1 Overview

1. Robot Language is a dedicated language made by Robostar, which is a command used when the robot job program is written in a system.
2. Robot Language is roughly divided into 4 groups.

- FLOW group
  Commands for setting the conditional or unconditional branch, the repeating number, the sub-program call, etc. of the program.
- MOVE group
  Commands for setting the behavioral conditions (PTP, interpolation, pallet job, etc.) of the robot.
- I/O group
  Commands (16bit, 1bit) for controlling the external I/O.
- COND(=Condition) group
  Commands for setting velocity, delay time, etc.

Write the desired job program using commands appropriated to the object of the job.

Chapter 2 Command List

2.1 Command Screen Display

group	Selection Key	FLOW	MOVE	I/O	COND
Selection Key		F1	F2	F3	F4
Item Move ▲▼	F1	MAIN	JMOV	OUT	VEL
	F2	FOR	LMOV	POUT	FOS
	F3	IF	CMOV	IN	DLAY
	F4	WHILE	AMOV	PIN	INT
	F1	EOP	IMOV	CIN	ACC
	F2	NEXT	PMOV	CWIN	PFOS
	F3	ELSE	HMOV	COUT	PLUP
	F4	ENDWL		CWOUT	REAL
	F1	SUBR	SVON	SYS	FORM
	F2	RET	SVOF		TOOL
	F3	CALL	WITH		PCLR
	F4	ENDIF	ENDWT		POS
	F1	LABL	MVR		LIMT
	F2	GOTO	HERE		OFFS
	F3	JCALL	PASS		CNT
	F4		PCNT		INPOS
	F1	STOP			TMR
	F2	EXIT			NO
	F3				LEFT
	F4				RIGHT
	F1				FIX
	F2				TRQ
	F3				TQL
	F4				XCHG
	F1				DEC
	F2				
	F3				
	F4				

2.2 Command List Table

2.2.1 Program control related commands

Command	Function	Formation	Example
MAIN	Start program	MAIN : EOP	MAIN VEL 200 JMOV P1 JMOV P2 EOP
EOP	End of program		
FOR	Repetitive execution	FOR <variable>=<starting value> TO <ending value> [BY<increment>] : NEXT *** The variable can use an integer variable only**	: FOR A=1 TO 5 BY 2 VEL 200 JMOV P1 NEXT :
NEXT			
WHILE	Conditional repetitive execution	WHILE<condition> : ENDWL	: WHILE IN0==1 JMOV P1 JMOV P2 ENDWL :
ENDWL			

IF	Conditional branch	IF<condition> THEN
		:
		(ELSE)
		:
		ENDIF
			:
			IF IN3==1 THEN
			GOTO A0
			ELSE
			JMOV P1
			ENDIF
			:

ENDIF			
LABL	Branching point designation	LABL <label name> : GOTO <label name>	: LABL A1 JMOV P1 JMOV P2 GOTO A1 :
GOTO	Branch to execution statement		
SUBR	Designate subroutine execution statement	SUBR<name of subroutine execution statement> : RET	: SUBR HON GOTO A0 ELSE JMOV P1 RET :
RET	Return to subroutine execution statement		
CALL	Call subroutine execution statement	CALL <name of subroutine execution statement>	
JCALL	Call JOB file	JCALL<JOB file name>	
STOP	Stop robot operation	STOP	
EXIT	Exit JOB execution	EXIT	

2.2.2 Robot motion related commands

Command	Function	Formation	Example
JMOV	PTP move from the current position to the target position	JMOV <Point variable>	MAIN VEL 200 JMOV P1 LMOV P2 EOP
LMOV	Linear interpolation move from the current position to the target point	LMOV <Point variable>	
CMOV	Move from the current position, drawing a circle connecting the passing point 1 and 2.	CMOV <passing point1> <passing point2>	: VEL 200 CMOV P1 P2 JMOV P3 AMOV P4 P5 :
AMOV	Move from the current position, drawing an arc connecting the passing point 1 and 2	AMOV <passing point1> <target point>	
IMOV	PTP move from the current position, by the increment	IMOV <Point variable> <Point variable> value → Axis value increment	: JMOV P1 IMOV P2 IMOV2 P3 :
IMOV2	Linear interpolation move from the current position, by the increment	IMOV <Point variable> <Point variable> value → Axis value increment	
PMOV	Perform pallet job designated at the current position	PMOV <pallet number> <reference point> data value of <pallet number> → Set in parameter	PMOV P1 P10
HMOV	Move to the origin position from the current position	HMOV <HOME position No.> Data value of <position number> → Set in ORG parameter	HMOV 1
JNTSYN	PTP synchronization mode selection	JNTSYN<Mode>	JNTSYN 1
EECH	Terminal device selection	EECH<Terminal Device>	EECH 1
TIMOV	Linear interpolation motion as much as incremental movement based on TOOL coordinate system.	TIMOV<Position-type variable>	TIMOV CURR
SVON	Servo ON	SVON → All axes 해당 SVON <Designated axis> → 1(X), 2(Y), 3(Z), 4(W)	SVON SVOF 2 SVON 2
SVOF	Servo OFF	SVOF → All axes 해당 SVOF <Designated axis> → 1(X), 2(Y), 3(Z), 4(W)	

Command	Function	Formation	Example

WITH
 	
ENDWT
	During robot operation, the execution statements are concurrently executed.	WITH
		:
		ENDWT
			:
			WITH
			JMOV P1
			MVR=0
			WHILE MVR<60
			IF IN1==1 THEN
			OUT0==1
			ENDIF
			ENDWL
			ENDWT
			:

PASS	Designate workpiece passing during pallet job	PASS <pallet No.> <order of workpiece to be passed>	PASS 1 3 PASS 2 1

2.2.3 I/O related commands

Command	Function	Formation	Example
OUT	Turn ON(=1) or OFF(=0) the output of the designated bit number on a bit-by-bit basis.	OUT<output bit number>=<0/1> [pulse effective time] [→] [pulse effective time] Unit : 10ms If there is no [pulse effective time], output signal is continuously effective. If [pulse effective time] is lapsed, return to the previous state [→] : Output the periodic waveform whose period is the pulse effective time.	OUT0=1 OUT0=1 100 OUT0=1 100 →
POUT	Output the designated value to the designated port on a port-by-port basis.	POUT<output port number>=<output designation value> <output port number> : Select one of 0~2 0 → OUT0 ~ OUT15 1 → OUT16 ~ OUT31 2 → OUT32 ~ OUT47	POUT0=0 POUT0=20 POUT1=0H000F
IN	Read the ON(=1) or OFF(=0) value of the designated bit number on a bit-by-bit basis.	IN<input bit number>=<0/1> -Wait until the designated bit input condition is satisfied <variable>=IN<input bit number> -Store the designated bit input state value in the variable	INT AA IN10=1 AA=IN0
PIN	Read the value from the designated input port on a port-by-port basis.	< variable>=PIN<input port number> <input port number> : Select one of 0~2 0 → IN0 ~ IN15 1 → IN16 ~ IN31 2 → IN32 ~ IN47	INT AA AA=PIN0
CIN	Read the ON(=1) or OFF(=0) value of the designated bit input number of a field bus on a bit-by-bit basis.	<variable>=CIN<input bit number> - Store the designated bit input state value in the variable	INT AA AA=CIN0
CBIN **CWIN** **CDIN**	Read the value from the designated field bus input port on a port-by-port basis.	<variable>=CWIN<input port number>	INT AA AA=CWIN0
COUT	Turn ON(=1) or OFF(=0) the designated bit field bus output on a bit-by-bit basis.	COUT<output bit number>=<0/1>	COUT=1
CBOUT **CWOUT** **CDOUT**	Read the value from the designated field bus input port on a port-by-port basis.	CWOUT<output port number>=<output designation value>	CWOUT=0HFF0FFF F
SYS	System control command	Execute the reserved system command	

2.2.4 Commands relating to robot motion condition

Command	Function	Formation	Example
VEL	Set up the percentage (%) of the axis moving velocity	VEL <percentage value(%)> Axis moving velocity = rated velocity X 0.001 **Rated velocity is set up in MOTION parameter.**	MAIN VEL 200 ACC 70 DEC 70 JMOV P1 VEL 1000 LMOV P2 ACC 100 DEC 100 EOP
ACC	Set up the percentage (%) of acceleration time	ACC <percentage value(%)> acceleration time = rated acceleration time X percentage value X 0.01 **Rated acceleration time is set up in MOTION parameter.**	
DEC	Set up the percentage (%) of deceleration time	DEC <percentage value(%)> deceleration time = rated deceleration time X percentage value X 0.01 **Rated declaration time is set up in MOTION parameter.**	
FOS	Change the trajectory to the next target point before the end of axis arrives at a target point.	FOS <distance rate(%)> distance rate is the percentage(%) of the total moving distance Applied to JMOV, LMOV, and AMOV	FOS 5 JMOV P1 FOS 0(disable)
PFOS	Change the trajectory to the next target point before the end of axis arrives at a target point.	PFOS <distance(mm)> Applied to LMOV	PFOS 5 LMOV P1 PFOS 0 (disable)
SPOS	Change the trajectory to the next target point before the end of axis arrives at a target point.	SFOS<distance>,<velocity level>,<arc angle> Applied to LMOV, AMOV, and CMOV	SFOS 10 0 0 LMOV P0 LMOV P1 SFOS 0 0 0(disable)
DLAY	Set up delay time	DLAY <delay time> Unit of delay time is 10ms. (If delay time is 500, then 5 seconds is delayed.)	JMOV P1 DLAY 20 JMOV P2
OFFS	Shift the target point by the designated value	OFFS <offset value> **The offset value is entered in MDI mode.** (E.g.) P100 → X:0, Y:-100, Z:10, W:0	JMOV P1 OFFS P100 JMOV P2
LIMT	Limit the moving range of each axis	LIMT <axis position value(min.)> <axis position value(max)>	JMOV P1 LIMT P3 P4 JMOV P2
PLUP	Set up Z-axis position value for pull-up operation	PLUP <Z-axis position value> Applied to Z-axis only	PLUP 5 JMOV P1 PLUP 0(disable)
FORM	Set up the form of robot arm during robot operation	FORM <form> Applied to horizontal articulated robot only <form> Select one of LEFT, RIGHT, and NO	FORM RIGHT JMOV P1 FORM NO(disable)
TOOL	Select a tool attached to the robot	TOOL <tool No.> **Data relevant to the tool No. is set in** <tool No.> : Select one of 0 ~ 3	TOOL1 JMOV P1 TOOL 0 JMOV P2

Command	Function	Formation	Example
PCLR	Initialize (Clear) a pallet job counter of PMOV	PCLR <pallet number> pallet number : 0~99 are available. When clearing, the counts (XCNT, YCNT, and ZCNT) are changed to 1.	MAIN PCLR 1 PCLR 2 PMOV P1 P10
INPOS	Set up the degree of arrival at the target point when a robot operates	INPOS <degree of arrival at target point> The degree of arrival is set by the pulse value.	INPOS 10 JMOV P2 INPOS 0(disable) JMOV P3
FIX	Select W-axis motion in the interpolation motion	FIX (0/1) 0 : W-axis rotating form, 1 : W-axis fixed form If omitted, the initial value is fixed to 1.	FIX 0 LOMV P1 FIX 1
TRQ	When the collision is detected, the torque limit alarm is generated	TRQ <axis number> <torque limit value> If a value exceeds the torque limit value of each axis, the alarm is produced.	TRQ 1 50 LMOV P1 LMOV P2
TQL	Limits the maximum thrust torque value of each axis.	TQL <axis number> <torque limit value> Each axis is driven with the limited torque.	TRQ 1 50 LMOV P1 LMOV P2
MINIT	Mapping function initialization	MINIT<Sensor Type>	MINIT 1
MSTART	Mapping function startup	MSTART<Sensor Input Pin><Axis><GP Index>	MSTART 3 3 GP10
MREAD	Mapping data renewal	MREAD<Limited Time>	MREAD 500

2.2.5 Variable declarations

Command	Function	Formation	Example
INT	Declare integer variable	INT <variable name>, REAL <variable name>, POS <variable name>, DEFSTR <variable name) Variable name is composed of 6 letters or less. Variable name is comprised of capital letters of alphabet and numerals. Variable name cannot be started with a numeral	MAIN INT N REAL A,B POS MM, XA
REAL	Declare real number variable		
POS	Declare position-type variable		
DEFSTR	Declare text string variable		

2.2.6 Program control related commands

Command	Function	Formation	Example
	Point-type variable is defined when writing the point file.	For a variable name, refer to POINT COMMAND.	

2.2.7 System variables

Command	Function	Formation	Example
CNTn **TMR0** **TMR1**	System defining variable (counter variable and timer variable)	CNT<pulse input port number>=<starting value> TMR0=<starting value> ▶ The moment the pulse input port number is entered, a value is assigned the counter variable, and since then, the counter variable counts every pulse input. ▶ From the moment the integer value is input to the timer variable, a value is assigned to it, and since then it is increased '1' by '1', with time interval defined in the system parameter	CNT0=2 TMR0=0 TMR1=-50
MVR	MOVE RATE	MVR < percentage of robot moving distance>	IF MVR<50 THEN
HERE	Angle value variable of the current axis	AP=HERE Variable, in which the angle value of the current axis is stored.	AP=HERE
RSTATE	Read the state of the robot	< integer variable >=RSTATE(Channel, STATE INDEX)	TEMP=RSTATE(1,6)
RERROR	Read the robot alarm codes	< integer variable >=RERROR(PAGE,INDEX)	ERR=RERROR(1,1)
REMCMD	System Command	REMCMD<Channel><Command>	REMCMD 1 3
GFTOGP	Store the value of GF to GP variable	GFTOGP<Gloabal Float Index><Global Point Index>	GFTOGP F10 GP50

2.2.8 Constant

	Function	Formation	Example
	Represent integer, real number, binary integer, and hexa-decimal integer	[0H/0B]<numeral> If there is not 0H or 0B in front of the numeral, it is a decimal number. 0H : hexa-decimal, 0B : binary	

2.2.9 Operators

Command	Function	Formation	Example
+, -, *, /, %	Binary operator (Add, Subtract, Multiply, Divide, and Modulus)		A = B*C
( )	Precedent operator		A = B/(A+C)
=	Assignment operator		A = B
&,|, ~, ^, <<, >>	Bit operator (BAND, BOR, Complement, BXOR, left shift, and right shift)		
&&, ||, ^^, !	Logical operator (AND, OR, XOR, and Negation)		
>, <, >=, <=, !=, ==	Comparison operator (greater than, smaller than, greater than or equal to, smaller than or equal to, different from, and equal to)		
ABS	Take the absolute value	ABS(-10.5) is 10.5	
DEG	Convert the radian value into the angle value	DEG(3.1416) is 180.0	
RAD	Convert the angle value into the radian value	RAD(180.0) is 3.1416	
POW	Exponential function	POW(2,4) is 16	
RND	Round up the decimals to convert the real number to the integer value	RND(14.8) is 15	
LOG	Common logarithm function		
SQRT	Find square root	SQRT(16) is 4	
SIN	Sine function	SIN(RAD(30)) is 0.5	
ASIN	Arc-sine function	ASIN(0.5) is 0.4794	
COS	Cosine function	COS(0) is 1.0	
ACOS	Arccosine 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	2nd arctangent function	ATAN2(Y,X) ATAN2(1,-1) is 2.3562	

2.2.10 Text string

Command	Function	Formation	Example
ASC	Return the first letter of text string as a character code	integer variable=ASC(text string)	INT AA AA=ASC("ABC")
CHR	Convert the integer into the character	text string variable=CHR(integer)	DEFSTR AA AA=CHR(65)
FLUSH	Clear the input and output buffers	FLUSH<clear buffer selection> ▶ Selection range: 1~3 1: input buffer clear 2: output buffer clear 3: input, output buffer clear	FLUSH3
FTOS	Convert the integer or real number into the text string	text string variable=FTOS(integer or real number) ▶Integer 1234 is converted into the text string "1234"	DEFSTR AA AA=FTOS(1234)
HTOS	Convert the integer to the hex-decimal text string	text string variable=HTOS(integer) ▶ Integer 10 is converted into the hexa-decimal text string "A".	DEFSTR AA AA=HTOS(10)
SLEFT	Extract the left section from the input text string	text string variable=SLEFT(text string, number) ▶ Extract from the input text string as many as the input integer from the left of the text string	DEFSTR AA AA=SLEFT("ABCDE",3)
SLEN	Return the length of the input text string	integer variable=SLEN(text string)	INT LEN LEN=SLEN("ABCDE",3)
SMID	Extract the text string from the position designated to the text string, as many as the number of digits	text string variable=SMID(text string, designated position, number of extracting letters) ▶Position of the text string initially starts with 0.	DEFSTR AA AA=SMID("ABCDEF",2,3)
SPOS	Return a starting place in the text string1, which matches with the text string2	integer variable= SPOS(text string1, text string2)	INT PP PP=SPOS("ABCDEF","B") ▶Return 1 the position of "B"
SRIGHT	Extract the right section from the input text string	text string variable=SRIGHT(text string, number) ▶ Extract from the input text string as many as the input integer from the right of the text string	DEFSTR AA AA=SRIGHT("ABCDE",3)
STRIN	Input the text string till divider	text string variable=STRIN(timeout time) ▶Unit of timeout time is msec.	DEFSTR AA AA=STRIN(1000)
STROUT	Output the text string	integer variable= STROUT(text string) ▶The number of characters, which have not been transmitted, is stored in the integer variable.	INT RT RT=STROUT("ABCDEF")
SVAL	Convert the text string into numeral	variable=SVAL(text string)	INT VA REAL VB VA=SVAL("1234") VB=SVAL("0.123")

Chapter 3 Explanation for Commands

3.1 MAIN, EOP (Start/End of program)

Function Start program, End of program

Formation
MAIN
…
EOP

Comments
1) **MAIN-EOP must be used together** to form a main program block, and only one block can be made in one JOB file.
2) EOP(End of Program) means the end of a main program block. If this command is met during JOB run, **JOB execution ends.**
However, the Job executed by JCALL finishes the job execution and **returns to the JOB called.**.
3) **In the place where ensuing EOP line, subroutine block(SUBR ~ RET) can be continuously written.**

3.1.1 Programming Example

MAIN	.....	**Start main program**
VEL 100	.....	Axis moving velocity
WHILE 1	.....	Start conditional repeat execution
JMOV P0	.....	Move to P0
JMOV P1	.....	Move to P1
CALL ON	.....	Call subroutine
ENDWL	.....	End conditional repeat execution
EOP	.....	**End of main program**
SUBR ON	.....	Designate subroutine
OUT0=1 200	.....	Output to the designated bit on a bit-by-bit basis
OUT1=1 200	.....	Output to the designated bit on a bit-by-bit basis
OUT2=1 200	.....	Output to the designated bit on a bit-by-bit basis
RET	.....	Return to subroutine

3.2 FOR, NEXT (Repetitive execution command)

Function A block is repeatedly executed until the variable is satisfied.

Formation FOR < variable>=<starting value> TO <ending value> (BY <increment>)
...
NEXT

Term <variable> : Use a variable name declared as integer type. E.g.) INT A,B,C,AA,...
<starting value> : Integer value which is set to the variable just before repeatedly execute FOR block.
<ending value> : Integer value for limiting the repeating number of FOR block.
<increment> : Integer value for regularly constantly increasing the variable value.

Comments
1) **FOR and NEXT must be used together.**
2) After the starting value is set to the integer variable, **FOR block is repeatedly executed until the ending value is met.**
3) Whenever the FOR block is repeated, the integer variable is increased by the increment. If the increment has been omitted, it increases **automatically by 1.**
4) The ending value **must always be greater than** the starting value, and the increment **must be 1 or more**.
(starting value ≤ ending value; increment ≥ 1)

3.2.1 Programming Examples

1) Sum the odd numbers from 1 to 10 by using FOR block, and save the result in the integer variable SUM.

```
MAIN
INT AA, SUM ..... Integer variable AA, SUM is declared.
SUM=0 ..... Integer variable is initialized.
FOR AA=1 TO 10 BY 2 ..... AA is increased from 1 to 10 by 2, five times.
SUM=SUM+AA ..... After 5 time executions, moved to the statement
                 ensuing the NEXT.
NEXT
EOP
```

2) Using FOR block, reciprocate 20 times, Point P1 ↔ P2

```
MAIN
INT J          ..... Integer variable J is declared.
J=0            ..... Integer variable is initialized.
FOR J=1 TO 20  ..... FOR statement is repeated 20 times, and execution is
                     moved to the statement ensuing NEXT.
JMOV P1        ..... JMOV P1
JMOV P2        ..... JMOV P2
NEXT
EOP
```

3) Move after saving the teaching point in the position variable.

```
MAIN
INT A          ..... Integer variable A is declared.
POS AP(11)     ..... Position variable AP(11) is declared. → AP(0), AP(1) ~
                     AP(10)
FOR A=1 TO 10  ..... FOR statement is repeated 10 times, and execution is
                     moved to the statement ensuing NEXT
AP(A)=P(A)     ..... In Point variable AP(A), save P(1) ~ P(10).
NEXT
EOP
```

4) Move after saving the teaching point in the position variable.

```
MAIN
INT J,K        ..... Integer variable J,K is declared.
POS TMP,AP(11) ..... Position variable AP(11) is declared. → AP(0), AP(1) ~
                     AP(10)
FOR K=1 TO 10  ..... FOR statement is repeated 10 times, and execution is
                     moved to the statement ensuing NEXT
AP(K)=P(K)     ..... In Point variable AP(K), save P(1) ~ P(10)
NEXT
IF P100.3==1.0 THEN
FOR J=1 TO 10        FOR statement is repeated 10 times, and execution is
                     moved to the statement ensuing NEXT
TMP=AP(J)      ..... AP(J)=AP(I)=P(1) ~ P(10)
TMP.3=TMP.3+10       Add 10 to Z-axis value of the position variable TMP.
AP(J)=TMP
NEXT
FOR K=1 TO 10  ..... FOR statement is repeated 10 times, and execution is
                     moved to the statement ensuing NEXT
JMOV AP(K)           PTP move to the saved AP(1) ~ AP(10).
DLAY 100
NEXT
EOP
```

3.3 WHILE, ENDWL (Conditional repetitive execution command)

Function A block is repeatedly executed only while the condition is being satisfied.

Formation
```
WHILE <condition>
 ...
ENDWL
```

Term <condition> : Referred to the logical operation equation or the comparison operation equation that can judge the true or false of the equation.

Comments
1) WHILE statement repeatedly executes WHILE block **while the condition is satisfied (the result of the condition is True or the values other than 0)**.
2) **WHILE – ENDWL must be used together**.
3) If the result of the condition is always True, infinite repeating is executed.

CAUTION

Matters that require attention in entering command
(O) WHILE_((IN0==1)&&(IN1==0))
(O) WHILE_IN0==1&&IN1==0
(O) WHILE_IN0==1
&&IN1==0

3.3.1 Programming Example

1) Repeat Move between Point P0 ↔ P1

```
MAIN
VEL 100
WHILE 1          .....   Block is infinitely repeated.
JMOV P0          .....   P0 ↔ P1 repeatedly moved.
JMOV P1          .....
ENDWL
EOP
```

3.3.2 Examples of Conditional Commands

WHILE (condition)		Comments
Constant	WHILE 1	WHILE ~ ENDWL block is infinitely repeated.
Input	WHILE IN0==1	Only while the signal is input to IN0=1, the block is repeatedly executed.
	WHILE ((IN0==1)&&(IN1==0))	Only while two input signals IN0=1, IN1=0 are simultaneously satisfied, the block is repeatedly executed.
	WHILE ((IN0==1)‖(IN1==0))	Only while only one of input signals IN0=1, IN1=0 is satisfied, the block is repeatedly executed.
	WHILE PIN0==0H000F	Only while all of IN0, IN1, IN2, and IN3 of Input PORT No. 0 are "1", the block is repeatedly executed.
	WHILE PIN0==32	Only while the input signal IN4 of Input PORT No. 0 is "1", the block is repeatedly executed.
Output	WHILE OUT0==1	Only while the output OUT0 is "1", the block is repeatedly executed.
	WHILE POUT0==0H000F	Only when all of OUT0, OUT1, OUT2, OUT3 of Output POUT No. 0 are "1", the block is repeatedly executed.
	WHILE POUT0== 0B0000000011111111	Only while all of OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7 of Output PORT No. 0 are "1", the block is repeatedly executed.
Variable	WHILE TMP==1	Only while the variable TMP is "1", the block is repeatedly executed.
	WHILE MVR>10	Only while MVR variable value is greater than "10", the block is repeatedly executed.
	WHILE TMR>10	Only while TMR(TIMER) variable value is greater than "10", the block is repeatedly executed.
	WHILE CNT>10	Only while CNT(COUNT) variable value is greater than "10", the block is repeatedly executed.
	WHILE AP10.1==1.0	Only while 1 axis value of POSITION variable AP is equal to "1.0", the block is repeatedly executed.

3.4 IF, ENDIF (Conditional branch command)

Function Judge the condition.

Formation
IF <condition> THEN
…
(ELSEIF) <condition> THEN
…
(ELSE)
…
ENDIF

Term <condition> : Referred to the logical operation equation or the comparison operation equation that can judge the true or false of the equation.

Comments
1) **IF – ENDIF must be used together.**
2) If the result of the "conditional" operation is **True (or the value other than 0), the statements following THEN are executed; if it False (or 0), the statements following ELSE are executed.**
3) ELSE-statement can be selectively used by the author of Program.
4) A different IF block can be used inside a different IF block, overlapping each other. At this time, it is important that IF and ENDIF must be in pair logically.

※ If the number of IF and the number of ENDIF is not the same, "Syntax Error" occurs.

※ The number of ELSE or ELSEIF is limited to 10 maximum.

Matters that require attention in entering command
(O) IF_((IN0==1)&&(IN1==0))
(O) IF_IN0==1&&IN1==0
(O) IF_IN0==1
&&IN1==0

3.4.1 Programming Examples

1) Point to be moved is determined depending on the state of the input IN0.

```
MAIN
VEL 100
IF IN0==1 THEN
JMOV P1 ..... If the input IN0 is 1, PTP Move to Point P1 is executed.
ELSE
JMOV P2 ..... If the input IN0 is 0, PTP Move to Point P2 is executed.
ENDIF
EOP
```

2) State of the input Port "1"(IN0 ~ 15), IN0 are judged.

```
MAIN
INT AA, BB
AA=PIN1 ..... State of Port1 (IN0~15) is saved in a variable AA.
IF (AA==0HFFFF) THEN  AA value is judged. (whether all of IN0~15 are 1.)
STOP ..... If it is "True", the robot is stopped.
ELSE
BB=IN0 ..... If it is "False", the status of IN0 is saved in BB.
IF (BB==1) THEN  BB value is judged whether it is 1.
DLAY10 .....
JMOV P0 ..... If "True(IN0=1)", the robot is moved to Point P0.
ENDIF
ENDIF
EOP
```

3) The following programs have the same context.

(Under the assumption that an input simultaneously applied to IN0~IN3 is only one.)

```
    :
LABL A0
IF IN0==1 THEN
GOTO B0
ENDIF
IF IN1==1 THEN
GOTO C0
ENDIF
IF IN2==1 THEN
GOTO D0
ENDIF
IF IN3==1 THEN
GOTO A0
ENDIF
    :
```

=

```
    :
LABL A0
IF IN0==1 THEN
GOTO B0
ELSE
IF IN1==1 THEN
GOTO C0
ELSE
IF IN2==1 THEN
GOTO D0
ELSE
IF IN3==1 THEN
GOTO A0
ENDIF
ENDIF
ENDIF
ENDIF
    :
```

3.4.2 IF(condition) THEN

IF(condition)		Comments
Input	IF_IN0==1_THEN	Is the input IN0 "1"?
	IF_((IN0==1)&&(IN==0))_THEN	Input signal IN0=1, IN1=0. Are these two conditions simultaneously satisfied?
	IF_((IN0==1)‖(IN==0))_THEN	Input signal IN0=1, IN1=0. Is at least one of these two conditions satisfied?
	IF_PIN0==0H000F_THEN	Are all of IN0, IN1, IN2, and IN3 of Input PORT0 "1"?
	IF_PIN0==32_THEN	Is the state of the input signal IN4 of the input PORT0 "1"?
Output	IF_OUT0==1_THEN	Is the output OUT0 "1"?
	IF_POUT0==0H000F_THEN	Are all of OUT0, OUT1, OUT2, OUT3 of the output POUT0 "1"?
	IF_POUT0== 0B0000000011111111_THEN	Are all of OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7 of the output POUT0 "1"?
변 수	IF_TMP==1_THEN	Is the variable TMP "1"?
	IF_MVR>10_THEN	Is MVR variable value greater than "10"?
	IF_TMR>10_THEN	Is TMR(TIMER) variable value greater than "10"?
	IF_CNT>10_THEN	Is CNT(COUNT) variable value greater than "10"?
	IF_AP10.1==1_THEN	Is POSITION variable value (per axis) equal to "1"?

3.5 LABL, GOTO (Branch command)

Function Command for designation of branching point and for move to branch execution statement.

Formation LABL <label name>
GOTO <label name>

Term <label name> : Text string of 8 letters consisting of alphabets and numbers
However, "P0, P1, ... " representing Work points cannot be used.
E.g.) LABL UNCLAMP1 **(O)**, GOTO CHECK2 **(O)**
LABL P1 **(X)**, GOTO P100 **(X)**

Comments 1) GOTO command is to branch to **the relevant label without any condition,** and can be used only in the same block.
For example, the branching from MAIN block to SUBR block, or from SUBR block to another SUBR block, cannot be performed.
2) In the case that IF block or FOR, WHILE block is used in the overlapping fashion, it is possible to branch from inside to outside, but impossible in vice versa.

3.5.1 Programming Examples

1) Repeated moving between two points P1, P0

```
MAIN
VEL 100
LABL A0
JMOV P1
JMOV P2
GOTO A0
EOP`
```

Unconditionally jump to "LABL A0".

=

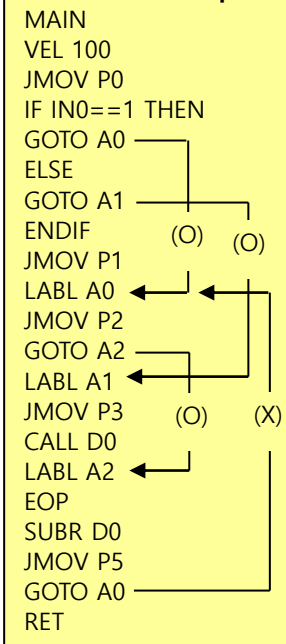
```
MAIN
VEL 100
WHILE 1
JMOV P1
JMOV P2
ENDWL
EOP
```

2) Branching depending on the state of the input IN0

```
MAIN
VEL 100
JMOV P0
IF IN0==1 THEN
GOTO A0 ..... If the input IN0 is 1, unconditional jump to LABL A0.
ELSE
GOTO A1 ..... Otherwise, unconditional jump to LABL A1
ENDIF
JMOV P1
LABL A0 ..... LABL A0, the branching point
JMOV P2
GOTO A2 ..... Unconditional jump to LABL A2
LABL A1 ..... LABL A1, the branching point
JMOV P3
LABL A2 ..... LABL A2, the branching point
EOP
```

CAUTION

Matters that require attention in entering commands



3.6 SUBR, RET (Subroutine execution command)

Function A part of program, that repeats a certain operation, being written separately and being able to be called as necessary.

Formation SUBR <name of subroutine execution statement>
...
RET

Term <name of subroutine execution statement> : Text string of 8 letters consisting of alphabets and numbers
However, "P0, P1, ... " representing Work points cannot be used.

Comments
1) **SUBR – RET must be used together.**
2) SUBR block can be **called by CALL statement only.**
3) RET statement finishes the execution of SUBR block, and **return to the step which ensues CALL.** CALL statement can be used in either MAIN or SUBR block, Recursive Call is permissible.
4) SUBR block must be defined at the place where ensues **MAIN-EOP block**.

3.6.1 Programming Example

MAIN		
VEL 100		
JMOV P0		
CALL A0	.....	CALL subroutine execution statement "A0"
CALL A1	.....	CALL subroutine execution statement "A1"
EOP		
SUBR A0		
JMOV P1		
RET	.....	Finish the execution of SUBR A0 block, and return to the step ensuing CALL
SUBR A1		
JMOV P2		
RET	.....	Finish the execution of SUBR A1 block, and return to the step ensuing CALL

3.7 CALL, JCALL (Call command)

Function Call the subroutine execution statement and JOB file.

Formation CALL <name of subroutine execution statement>
JCALL <JOB file name>

Term <name of subroutine execution statement> : Text string of 8 letters consisting of alphabets and numbers
However, "P0, P1, ... " representing Work points cannot be used.
<JOB file name> : File name of JOB program to be called, excluding the extension (.JOB).

Comments ※ **CALL**

1) When using CALL statement, there must be **the subroutine execution statement(SUBR) to be called**.
2) In CALL statement, the **name of subroutine execution statement and subroutine execution statement must be the same**.
3) If SUBR block meets RET, a step is returned to **a step ensuing CALL**.

※ **JCALL**

1) JCALL uses JOB as a subroutine.
2) In the called JOB, if EOP is met, execution is returned to a step ensuing JCALL statement of the called program.
3) Write the program so that JCALL does not become "recursive call".

Recursive Call is possible.

Note) **Maximum 8 times** of recursive call for CALL statement → if exceeding, "CALL DEPTH ERROR";
Maximum 3 times of recursive call for JCALL statement → if exceeding, "JCALL DEPTH ERROR"

⚠ CAUTION

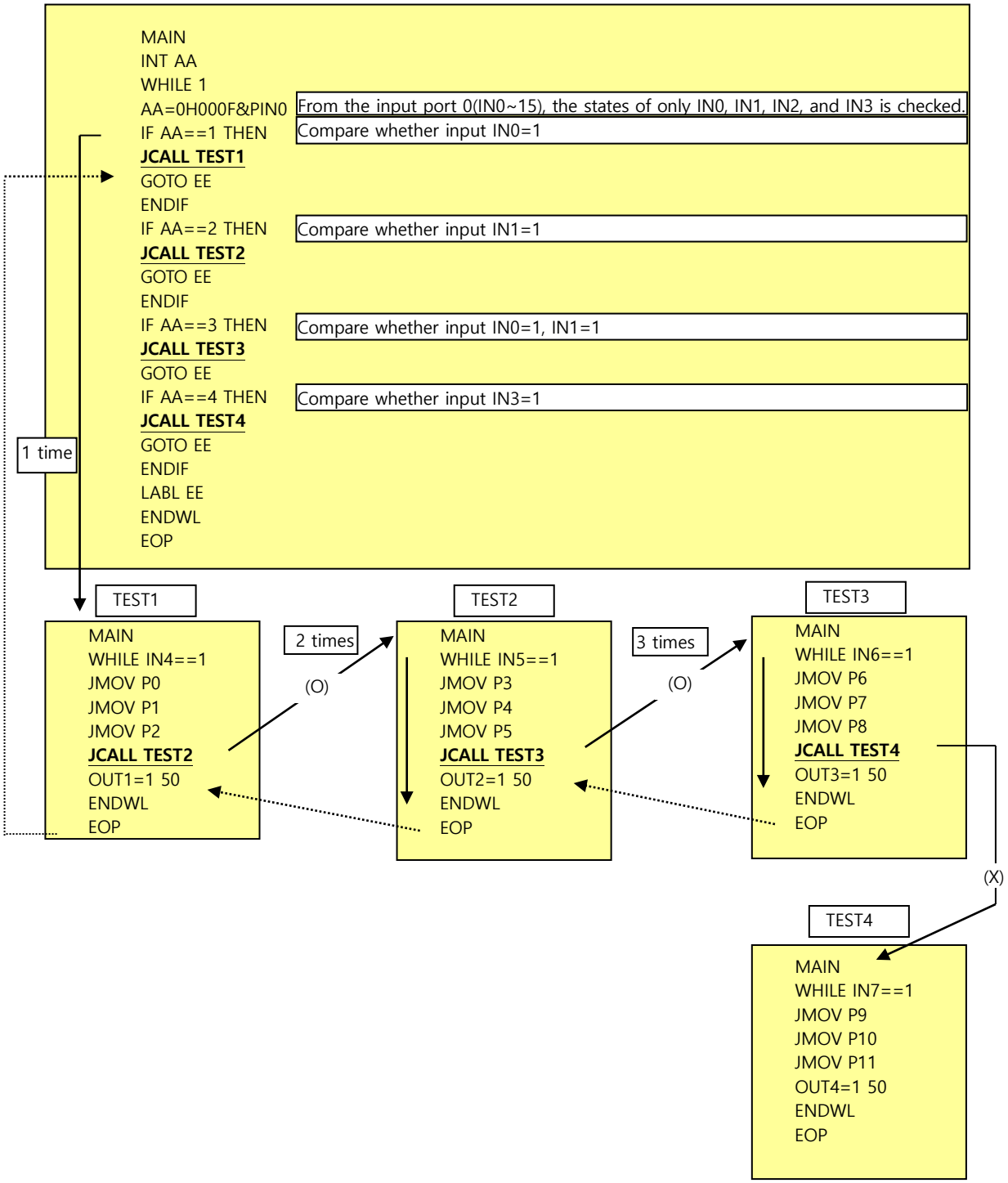
Matters that require attention in entering command

```
MAIN
VEL 100
LABL A0
JMOV P1
JMOV P2
CALL A0
GOTO A0
EOP
SUBR A0
OUT1=1
IN1=1
RET
```

(X) If the names of LABL and SUBR are the same, the error occurs.

3.7.1 Programming Example

1) Depending on the state of the input port No. 0, one of the JOB file names "TEST1"~"TEST4" is JCALL'ed.



3.8 STOP, EXIT (Robot stop & JOB exit command)

Function Stop the robot operation(STOP), Exit JOB execution(EXIT)

Formation STOP
EXIT

Comments
1) STOP command **stops the robot which is moving**, and the next step of command is handled. That is, **JOB is continuously performed after the stop.**
2) EXIT command **finishes JOB execution**. At this time, the controller generates **"EXIT Instruction"** alarm.

3.8.1 Programming Example

1) After Point moving, turn on or off the output OUT1.

```
MAIN
VEL 500
JMOV P1
MVR=0 ..... "Moving range percentage" is initialized.
WITH .....
JMOV P2 ..... Moved to Point P2.(Only JMOV is available)
WHILE MVR<60 ..... During less than 60% of the moving range
IF IN0==1 THEN ..... If IN0=1, jump to "LABL BB".
GOTO BB
ENDIF
ENDWL
OUT0=1
ENDWT

LABL BB
STOP ..... Robot moving stopped.
ENDWT ..... End of WITH statement (This must be always
                inserted. )

JMOV P3
OUT1=1
EOP
```

3.9 JMOV (PTP Move Command)

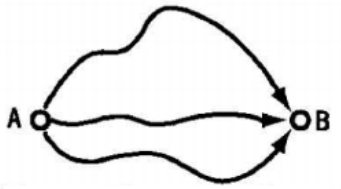
Function Axis interpolation move from the current position to the target point.

Formation JMOV P <number>
JMOV GP <number>
JMOV <position variable>

Term <number> : Set the taught position coordinate No.
P : LOCAL POINT separately used for each JOB (0 ≤ No. ≤ 1023)
GP : GLOBAL POINT used in common (0 ≤ No. ≤ 1023)
<position variable> : Means the variable name declared to POS-type.

PTP means moving point to point.

Comments Moving path depends on the root forms, and **is not defined as the linear moving.**



Moving paths are not defined as the straight line.

3.9.1 Programming Examples

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
JMOV P10  ..............  PTP move to Point P10
JMOV P11  ..............  PTP move to Point P11
JMOV P100 ..............  PTP move to Point P100
EOP
```

2) Move by using position-type variable (1)

```
MAIN
POS XA                          ..............  Point variable XA is declared.
XA=<400.0,50.0,10.4,10.0,0,0,1> ..............  Designate attribute values of XA
                                               <X,Y,Z,W,EX1,EX2,FORM>
VEL 100
JMOV P0                         ..............  PTP move to Point P10
JMOV XA                         ..............  PTP move to XA position coordinate
JMOV P100                       ..............  PTP move to Point P100
EOP
```

3) Move by using position-type variable (2)

```
MAIN
POS AP,AP1,AP2                ..............  Point variable AP,AP1,AP2 is declared.
AP=P10 +<10,10,10,10,0,0>     ..............  Each axis value of Point P10 plus 10 is saved.
AP1=P11 -<10,10,10,10,0,0>    ..............  Each axis value of Point P11 minus 10 is saved.
AP2=P100                      ..............  The value of P100 is saved in Point AP2
AP2.3=P100.3+10               ..............  Z-axis value of Point P100 plus 10 is saved.
JMOV AP                       ..............  PTP move to AP position coordinate
JMOV AP1                      ..............  PTP move to AP1 position coordinate
JMOV AP2                      ..............  PTP move to AP2 position coordinate
EOP
```

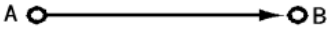
3.10 LMOV (CP Move Command)

Function Linear interpolation moving from the current position to the target point.

Formation LMOV P<number>
LMOV GP<number>
LMOV <position variable>

Term <number> : Set the taught position coordinate No.
P : LOCAL POINT separately used for each JOB (0 ≤ No. ≤ 1023)
GP : GLOBAL POINT used in common (0 ≤ No. ≤ 1023)
<position variable> : Means the variable name declared to POS-type.

Comments CP control performs **interpolation control such that the path to the target point becomes the straight line.**



The moving path becomes linear.

CAUTION

- The robot sometimes may not be operated under CP control, depending on the position of the root. This time, the controller generates "Unreachable Point" or "Inverse Error" Alarm.
- Pay special attention to using "FOS", "FIX", and "TOOL" Command.
- SCARA Robot performs the interpolation operation, with retaining the current form, and, for the first operation command of the program, PTP control must be used.

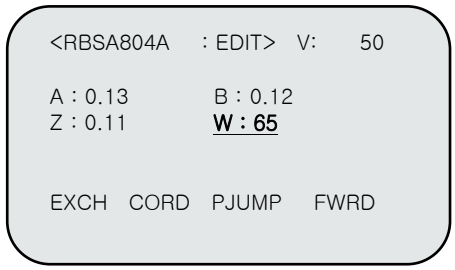
CAUTION

When, in SCARA Robot, W-axis is performed interpolation operation:

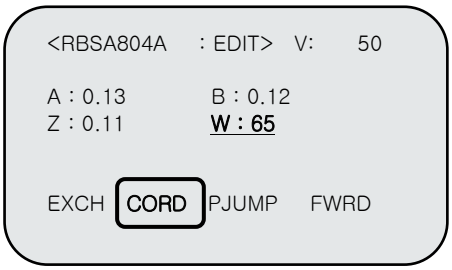
- "OFFS" and "LENG" of the parameter are to be exactly set up to the values written on a sticker of the mechanism.
- If the tool offset exists, there are two ways for W-axis interpolation operation:
  W-axis fixed form (Robot Command "FIX 1") and W-axis rotating form (Robot Command "FIX 0") (See FIX Command.)
- If the tool offset exists, be careful of Work point Teaching.
  W-axis fixed form : Teaching is performed, with the tool direction is retained to the fixed direction.
  W-axis rotating form : Perform tool selecting and work point teaching after exactly setting up the tool offset.
- Perform tool selecting and work point teaching so that W-axis form can be fixed as possible.
- Especially in W-axis rotating form, when the arc is drawn (AMOV) during the continuous interpolation operation, the robot mechanism may not follow the instruction of the controller (i.e., Unreachable Point).

3.10.1 Teaching with the tool direction being retained constantly (SCARA robot)

Step 1.

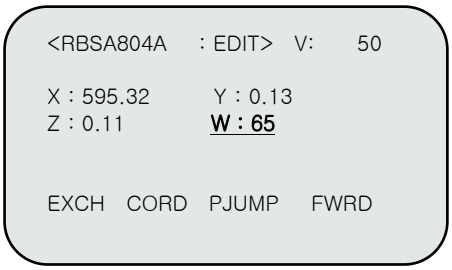


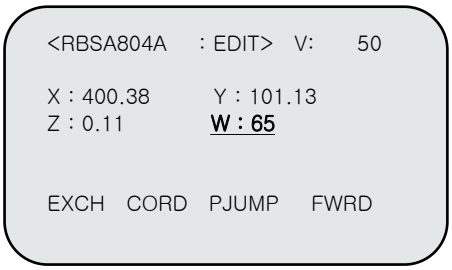
Move to Point screen, referring to Teaching Manual.



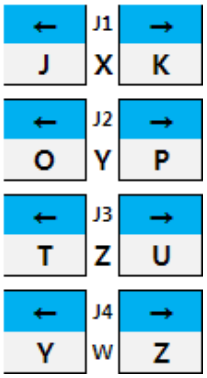
F2

Enter 「F2」 to change the coordinate system, Cylindrical → Cartesian.





If JOG operation is performed in Cartesian coordinate system screen, the robot moves, with W-axis form is being retained.



3.10.2 **Programming Examples**

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
JMOV P10 ..... PTP move to Point P10
LMOV P11 ..... CP move to Point P11
LMOV P100 ..... CP move to Point P100
EOP
```

2) Move by using position-type variable (1)

```
MAIN
POS XA ..... Point-type variable XA is declared.
XA=<400.0,50.0,10.4,10.0,0,0,1> ..... Designate attribute values of XA
                                      <X,Y,Z,W,EX1,EX2,FORM>
VEL 100
JMOV P0 ..... PTP move to Point P10
LMOV XA ..... CP move to XA position coordinate
LMOV P100 ..... CP move to Point P100
EOP
```

3) Move by using position-type variable (2)

```
MAIN
POS AP,AP1,AP2 ..... Point-type variable AP,AP1,AP2 is declared.
AP=P10 +<10,10,10,10,0,0> ..... Each axis value of Point P10 plus 10 is saved.
AP1=P11 -<10,10,10,10,0,0> ..... Each axis value of Point P11 minus 10 is saved.
AP2=P100 ..... The value of P100 is saved in Point AP2
AP2.3=P100.3+10 ..... Z-axis value of Point P100 plus 10 is saved.
JMOV AP ..... PTP move to AP position coordinate
LMOV AP1 ..... CP move to AP1 position coordinate
LMOV AP2 ..... CP move to AP2 position coordinate
EOP
```

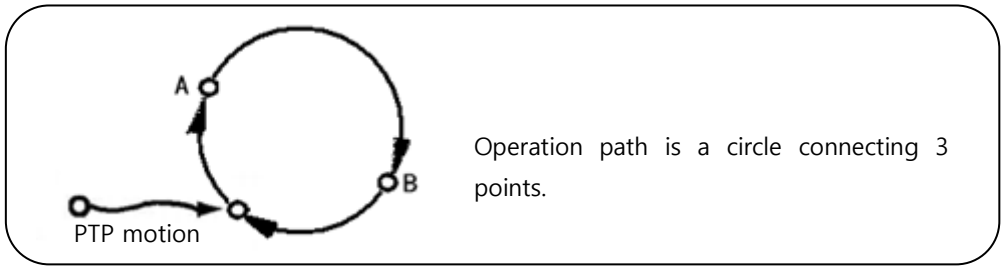
3.11 CMOV (Circular interpolation move Command)

Function Move from the current position, drawing a circle connecting the passing point 1 and 2.

Formation CMOV P<No. of passing point 1> P<No. of passing point 2>
CMOV GP<No. of passing point 1> P<No. of passing point 2>
CMOV <position variable of passing point 1> P<position variable of passing point 2>

Term <number> : Set the taught position coordinate No.
P : LOCAL POINT separately used for each JOB (0 ≤ No. ≤ 1999)
GP : GLOBAL POINT used in common (0 ≤ No. ≤ 254)
<position variable> : Means the variable name declared to POS-type.

Comments A path to reach the operational target position is **interpolation-controlled so that the path is drawn similar to a circle.**
In the circular interpolation operation, **3 Points of coordinate are needed.**



CAUTION

- Due to wrong Point Teaching, the point teaching may not be performed during Auto RUN. This time the controller generates "Unreachable Point" or "Inverse Error" Alarm.
- Pay special attention to using "FOS", "FIX", and "TOOL" Command.
- For W-axis interpolation operation, see "LMOV" Command.
- CMOV and AMOV cannot be used alone, and add JMOV as in the following example.

3.11.1 Programming Example

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
JMOV P1          .....   PTP move to Point P1
CMOV P2 P3       .....   Circular interpolation move via Point P2 and P3
EOP
```

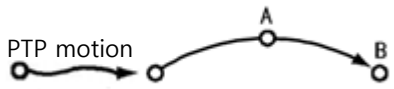
3.12 AMOV (Arc interpolation Move command)

Function Move from the current position, drawing an arc connecting the passing point 1 and 2.

Formation AMOV P<No. of passing point 1> P<No. of passing point 2>
AMOV GP<No. of passing point 1> P<No. of passing point 2>
AMOV <position variable of passing point 1> P<position variable passing point 2>

Term <number> : Set the taught position coordinate No.
P : LOCAL POINT SEPARATELY USED FOR EACH JOB. (0 ≤ No. ≤ 1023)
GP : GLOBAL POINT used in common (0 ≤ No. ≤ 1023)
<position variable> : Means the variable name declared to POS-type.

Comments A path to reach the operational target position is **interpolation-controlled so that the path is drawn similar to an arc.** In the arc interpolation operation, **3 Points of coordinate are needed.**



Operation path is a circle connecting 3 points.

CAUTION

- Due to wrong Point Teaching, the point teaching may not be performed during Auto RUN. This time the controller generates "Unreachable Point" or "Inverse Error" Alarm.
- Pay special attention to using "FOS", "FIX", and "TOOL" Command.
- For W-axis interpolation operation, see "LMOV" Command.
- CMOV and AMOV cannot be used alone, and add JMOV as in the following example.

3.12.1 Programming Example

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
JMOV P1          .....   PTP move to Point P1
AMOV P2 P3       .....   Arc interpolation move via Point P2 and P3
EOP
```

3.13 IMOV, IMOV2 (Increment move command)

Function Axis interpolation move from the current position, by the increment.
IMOV – PTP moving
IMOV2 – CP moving

Formation IMOV P<number>
IMOV GP<number>
IMOV <position variable>

Term <number> : Set the taught position coordinate No.
P : LOCAL POINT SEPARATELY USED FOR EACH JOB. (0 ≤ No. ≤ 1023)
GP : GLOBAL POINT used in common (0 ≤ No. ≤ 1023)
<position variable> : Means the variable name declared to POS-type.

Comments 1) **Relative move** as much as the designated position.
2) IMOV is for moving by the designated position in reference with the current position.

IMOV

Moving path is not defined as the straight line

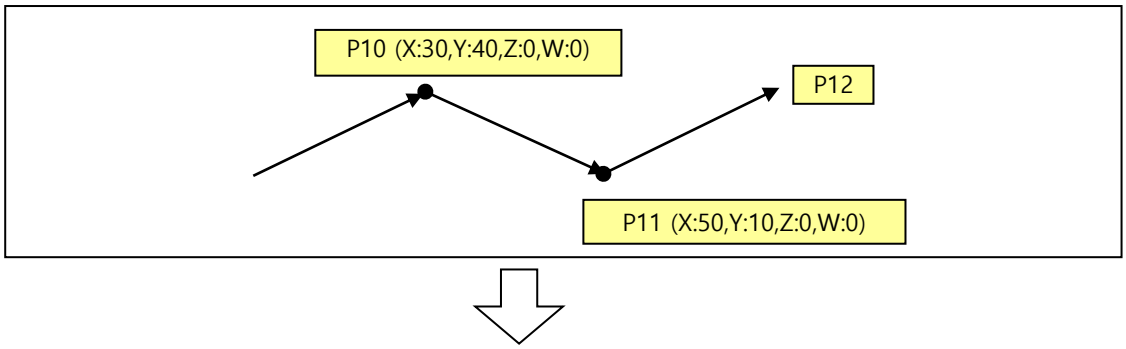
IMOV2

Moving path becomes the straight line.

CAUTION

- When the positional coordinate is designated in Point Teaching for IMOV, the designated value is stored within "RANG" of the parameter. However, the value is positional value set up with regard to the origin(ORIGIN).
- Therefore, when a robot moves (IMOV) with regard to the previous Point, "Range over error" may occur.
- Adjust the setup of the parameter "RANG".
- Since IMOV moves with regard to the previous position, the positional deviation may arise, depending on the degree of In Position of the previous position.
  For this, edit the program by adding "INPOS", "DLAY", etc. to the Job program so that the robot can exactly be In Position at the previous position.

3.13.1 Programming Example



1) Use of IMOV (Use Point P1)

```
MAIN
VEL 100
JMOV P10          .....   PTP move to Point P10
JMOV P11                  PTP move to Point P11
INPOS 10          .....   INPOS or DLAY is used to prevent positional
                          deviation.
IMOV P1           .....   PTP move by Point P1 with regard to Point P11.
EOP
```

2) Use of IMOV (Use position variable AP)

```
MAIN
POS AP            .....   Point-type variable AP is declared.
AP=<20.0,-30.0,0,0.0,0,0,1>  .....   Attribute values of AP are designated.
                          <X,Y,Z,W,EX1,EX2,FORM>
VEL 100
JMOV P10          .....   PTP move to Point P10
IMOV AP           .....   PTP move by AP value with regard to Point P10.
JMOV P100         .....   PTP move to Point P100
EOP
```

3) Use of IMOV2

```
MAIN
VEL 100
LMOV P10          .....   CP move to Point P10
LMOV P11                  CP move to Point P11
INPOS 10          .....   INPOS or DLAY is used to prevent positional
                          deviation.
IMOV2 P1          .....   CP move by Point P1 with regard to Point P11.
EOP
```

4) Use of IMOV2 (Use position-type variable AP)

```
MAIN
POS AP            .....   Point-type variable AP is declared.
AP=<20.0,-30.0,0,0.0,0,0,1>  .....   Attribute values of AP are designated.
                          <X,Y,Z,W,EX1,EX2,FORM>
VEL 100
LMOV P10          .....   CP move to Point P10
IMOV2 AP          .....   CP move by AP value with regard to Point P10.
LMOV P100         .....   CP move to Point P100
EOP
```

3.14 JNTSYN (PTP Synchronization Mode Command)

Function PTP synchronization mode selection

JNTSYN command is available only in the following version of ROSEP Robot and Ver 03.02.08.

Format JNTSYN <Format>

0 : PTP synchronization mode not in use
1 : PTP synchronization mode in use

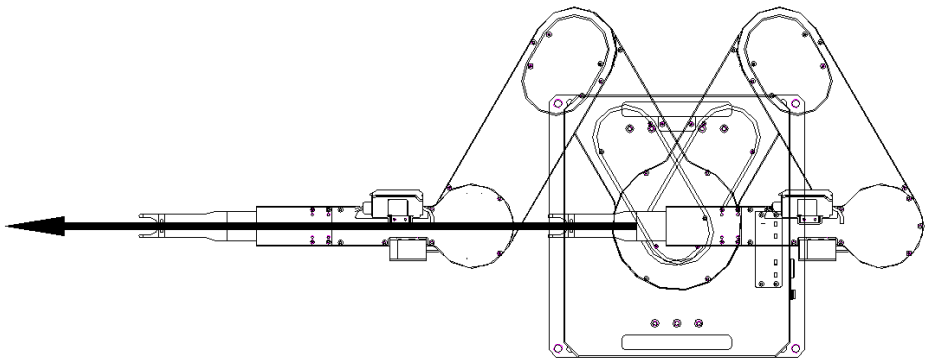
Description 1) When making a LMOV near a singularity, an 'overspeed' alarm occurs due to a rapid change in speed near the singularity. To resolve this problem, a ROSEP robot type enables a linear motion that passes a singularity by driving a PTP motion (JMOV) synchronously.

2) When set to "JNTSYN 0", operation is done in the same manner as a general JMOV.

3) When set to "JNTSYN 1", movements of J2, J4, J5 are synchronized to J1 movement to make linear motions passing the singularity. (J1(A), J2(B), J3(Z), J4(W1), J5(W2))

4) Checks that synchronization conditions are met by comparing travel distance per axis. If W1, W2 do not satisfy synchronization conditions, it is excluded from synchronization formula. If A, B do not satisfy synchronization conditions, an alarm (1422: PTP Sched. Error) goes off.

Ex) P1 =<-25, 230, 0, -115, -115, 0>, P2 =<25, 130, 0, -65, -65, 0>
P2.1 - P1.1= 50 = delta_TH1
P2.2 - P1.2= -100 = -2 * delta_TH1
P2.4 - P1.4= 50 = delta_TH1
P2.5 - P1.5= 50 = delta_TH1



[Passing ROSEP ROBOT TYPE Singularity]

3.14.1 **Examples of Program Use**

MAIN	.....	Start program
VEL 10	.....	Axis travel speed
JMOV P1	.....	Move to P1=<-25, 230, 0, -115, -115, 0>
WHILE 1	.....	Start performing conditions repeatedly
JNTSYN 1	.....	PTP synchronization mode in use
JMOV P2	.....	Move to P2=<25, 130, 0, -65, -65, 0>
JMOV P1	.....	Move to P1=<-25, 230, 0, -115, -115, 0>
JNTSYN 0	.....	PTP synchronization mode not in use
ENDWL	.....	End condition repeat statement
EOP	.....	End program

CAUTION

- For a linear motion in JMOV synchronization mode, the movements on J1(A) axis and J2(B) axis should double. If the movements are not double, a "PTP Sched. Error " alarm occurs.
- If the movements on J1(A) axis and J4(W1) axis, and J5(W2) axis are not the same, individual movement is made to the axis except for synchronous mode.

3.15 EECH (End Effector Selection Command)

Function End effector (W1, W2) selection

EECH commands are available for use only in versions after ROSEP Robot and Ver 03.00.00.

Format EECH <Format>

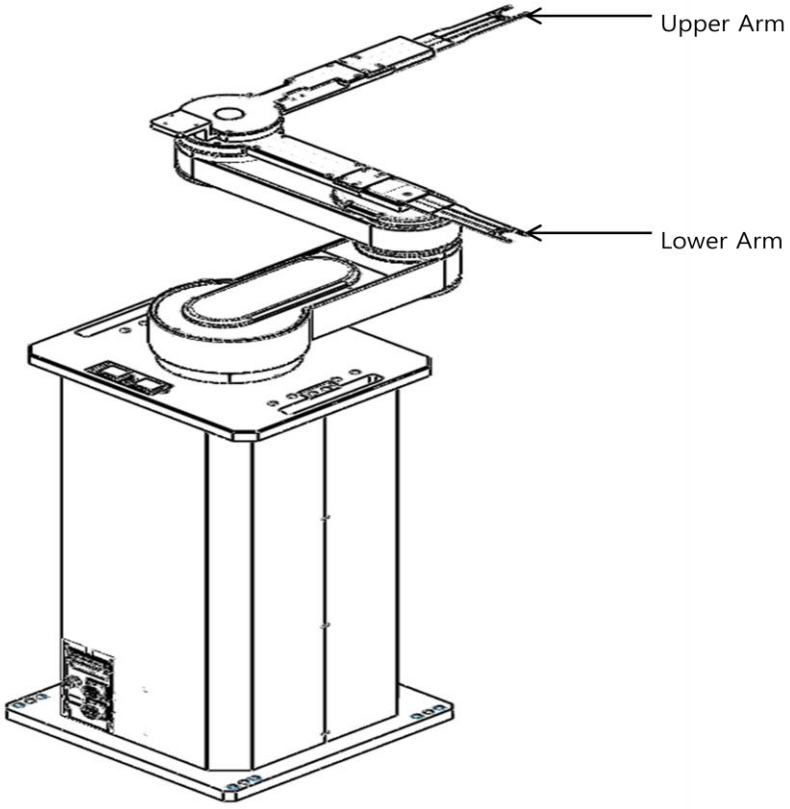
1: Select the first W1 axis and TOOL no. 0

2: Select the second W2 axis and TOOL no. 1

Description 1) A ROSEP Robot consists of two end effectors (W1, W2) which are commands and select an end effector to apply when converting XY coordinates.

2) When set to "EECH 1", W1 axis (Lower Arm) on robot and Tool 0 on the end effector are selected.

3) When set to "EECH2", W2 axis (Upper Arm) on robot and Tool 1 on the end effector are selected.



[ROSEP ROBOT Arm Selection]

3.15.1 Examples of Program Use

MAIN	.....	Start program
VEL 10	.....	Travel speed setup
WHILE 1	.....	Start performing condition repeatedly
JMOV P0	.....	Move PTP to P0
JMOV P1	.....	Move PTP to P1
EECH 1	.....	Select robot's W1 end effector
LMOV P2	.....	Move CP to P2
LMOV P1	.....	Move CP to P1
JMOV P3	.....	Move PTP to P3
EECH 2	.....	Select robot's W2 end effector
LMOV P4	.....	Move CP to P4
LMOV P3	.....	Move CP to P3
ENDWL	.....	End condition repeat statements
EOP	.....	End program

3.16 TIMOV (Incremental Move Command based on TOOL Coordinate System)

Function **From the current position make linear interpolation motions as much as the increased amount based on TOOL coordinate system. TIMOV command is only available for use only in the versions that came after ROSEP Robot and Ver 03.00.00.**

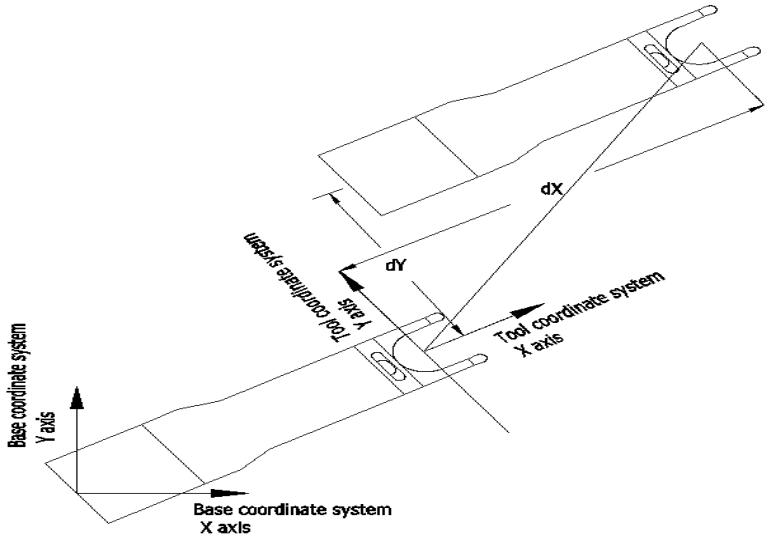
Format TIMOV position-type variable

<Position-type variable> : means the name of a variable declared as a POS format.

As a transfer is made on the basis of XY coordinates, the name of a position-type variable should start with X.

Ex) POS XP, XCUR

Description 1) TIMOV, based on TOOL coordinate system, makes a related movement from the current position to as much as the designated positions (dx, dy, dz).

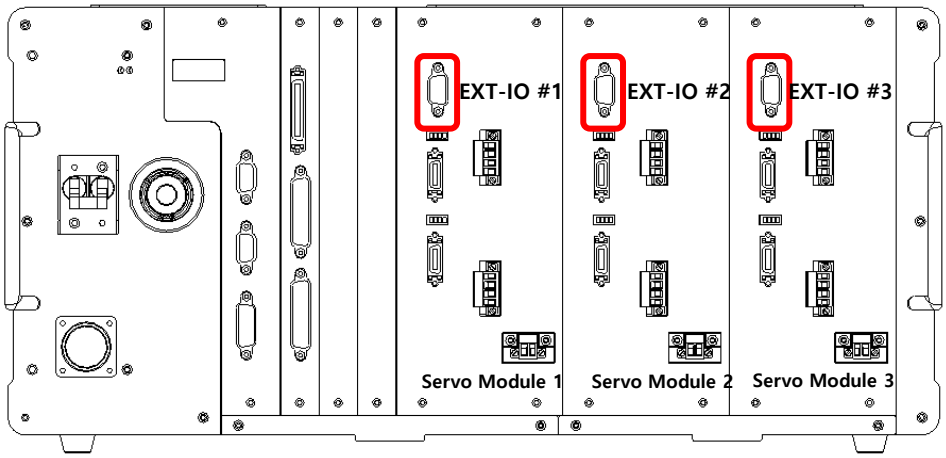


CAUTION

- When saving a position coordinate in the Point Teaching for TIMOV, the saving range should not go beyond RANG in parameter. But this is a position value set on the basis of ORIGIN.
- Therefore, when the robot moves (TIMOV) based on the previous Point, a "Range over error" may occur.
- Adjust the parameter "RANG" configuration.
- As TIMOV moves on the basis of the previous position, a deviation of position may occur according to the In Position degree in the previous position. Now is time to modify the program so TIMOV can be performed after adding "INPOS","DLAY" to the in-progress program for the robot to complete In Position accurately in the previous position.

3.16.1 **Regarding Mapping Sensor Input Port**

For an accurate use of mapping functions, the sensor input number of a servo module and driving axis should be consistent. Each servo module has 2 sensor input pins and is allocated as seen in "Sensor Input Numbers per Servo Module" in the figure below.



	Servo Module 1	Servo Module 2	Servo Module 3
IN 1	1	3	5
IN 2	2	4	6

[Sensor Input Numbers per Servo Module]

Pin No.	Function
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	P24V(Output)
9	IN1(Input)
10	IN2(Input)
11	G24V(Output)
12	-
13	-
14	-
15	-

[Servo Module EXT IO Pin Function]

Ex) In use of the mapping function in Z-axis on Scara robot,

Generally, when Scara robot allows its Z-axis to be connected to servo module 2, with mapping data saved from GP100 and its sensor connected to IN1,

MSTART 3 GP100 3
Z-axis GPNT INDEX Sensor No.

3.17 HMOV (Origin move command)

Function Move to the origin position from the current position

Formation HMOV <integer variable or number>

Term <number> : Origin coordinate number set up in the parameter. (0 ≤ No. ≤ 3)
<integer variable> : Name of integer variable declared as INT type.

Comments
1) The robot is **PTP moved to the origin coordinate** set up in HOME of ORG parameter.
2) The origin coordinate number has four values from 0 to 3 (HMOV 0, HMOV 1, HMOV 2, HMOV 3).

CAUTION

- **When moving to HMOV 0,** the moving is performed according to the origin searching sequence **set up in SEQ of ORG parameter**.
- **When moving to HMOV1 - HMOV 3,** PTP move is performed **regardless of SEQ.**

3.17.1 Programming Example

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
JMOV P1          .....   PTP move to Point P1
HMOV 0           .....   Move to HOME 0 by the origin searching sequence
HMOV 1           .....   PTP move to HOME 1
EOP
```

3.18 PMOV (Pallet move)

Function Perform pallet job designated at the current position

Formation PMOV <pallet number> < reference point for pallet job >

Term <pallet number> : Enter the pallet number set up in the parameter. (0 ≤ No. ≤ 99)
<reference point for pallet job> : Enter the first point number of four teaching points of the pallet position. (0 ≤ No. ≤ 1999)
E.g.) In the case of PMOV P0 P1: Reference point (P1), X direction (P2), Y direction (P3), Z direction (P4)

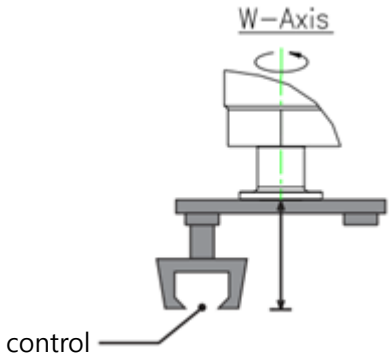
Comments
1) Move to the pallet job position on a basis of **PTP motion.**
2) Execution of PMOV is moving to the position on a pallet designated by a counter(CNT) of the parameter.
3) To perform the pallet job, the following two subjects have been to be determined.
   A. Enter the information on a pallet in a parameter mode.
      - **Initialize the pallet data before entering the first information.**
   B. Teach four points on the pallet.

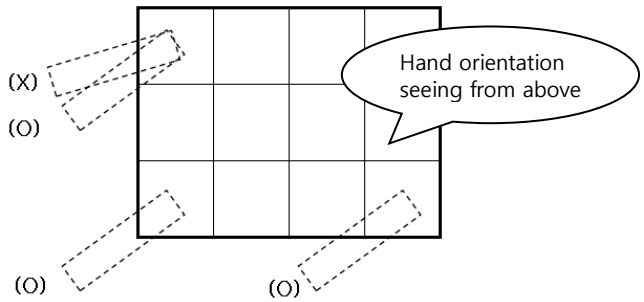
⚠CAUTION

▶ In the case that the mechanism is a horizontal multi-joint robot(SCARA robot), "OFFSET" values of A and B arm are to be set up exactly.
▶ The pallet job sequence is determined according to the types of the parameter. (Refer to the instruction manual.)

⚠CAUTION

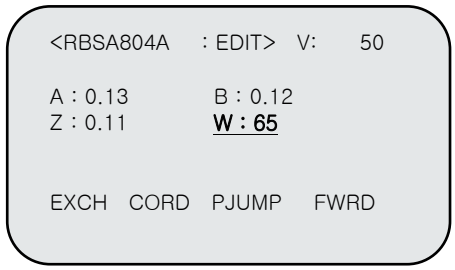
▶ If a hand is attached to an end effect of Z axis in a robot with a rotating axis (W-axis), the pallet teaching must be performed in the state that the orientation of hands are aligned with one another, as shown below.



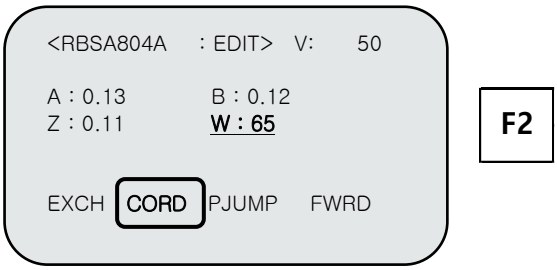


3.18.1 Teaching with the tool direction being retained constantly (SCARA robot)

Step 1.

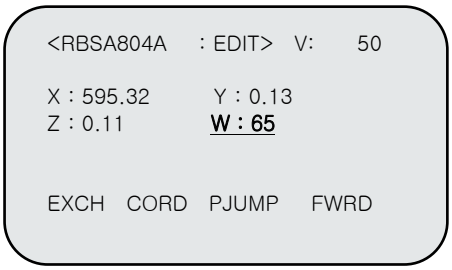


Move to Point screen, referring to Teaching Manual.

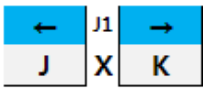


F2

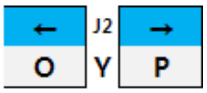
Enter 「F2」 to change the coordinate system, Cylindrical → Cartesian.



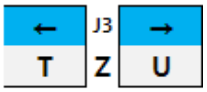
If JOG operation is performed in Cartesian coordinate system screen, the robot moves, with W-axis form is being retained.



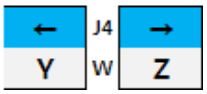
X-axis



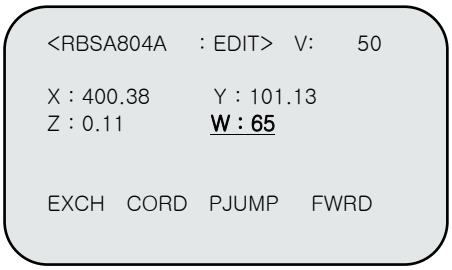
Y-axis



Z-axis



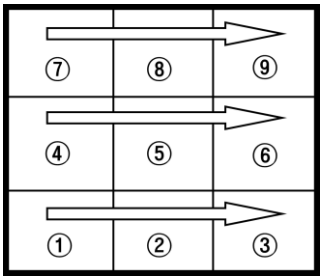
W-axis



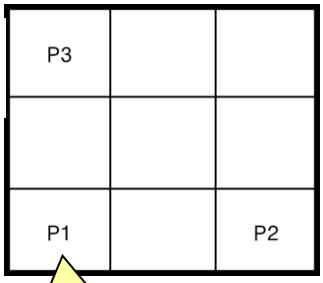
3.18.2 Examples of pallet job

Job 1 ⇨ **Command : PMOV P1 P1** → **Reference: P1**
X Direction: P2
Y Direction: P3
Z Direction: P4

① Work sequence → Point teaching



How for point teaching to palletize as this sequence?



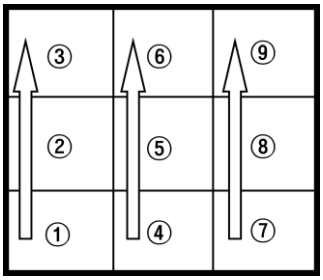
② Set up the data of the parameter setup values

```
Pallet NO = 01
1.XWM=03    3.ZWM=01
2.YWM=03
```

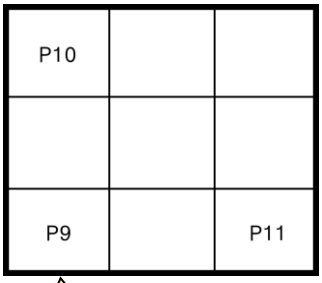
▶ **P4 Point teaching?**
- For 1 stack: P4=P1
- For 2 stacks or more:
P4 is only different from Z-axis of P1

Job 2 ⇨ **Command : PMOV P1 P9** → **Reference: P5**
X Direction: P10
Y Direction: P11
Z Direction: P12

① Work sequence → Point teaching



How for point teaching to palletize as this sequence?



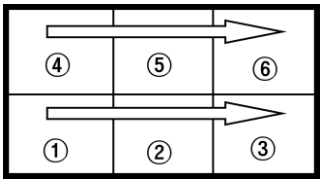
② Set up the data of the parameter setup values.

```
Pallet NO = 02
1.XWM=03    3.ZWM=01
2.YWM=03
```

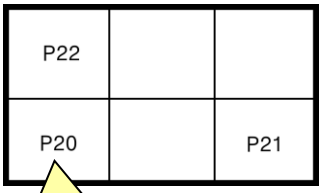
▶ **How for P12 Point teaching?**
- For 1 stack: P12=P9
- For 2 stacks or more:
P12 is only different from Z-axis of P9.

Job 3 ⇨ **Command : PMOV P20** → **Reference: P20**
X Direction: P21
Y Direction: P22
Z Direction: P23

① Work sequence → Point teaching



How for point teaching to palletize as this sequence?



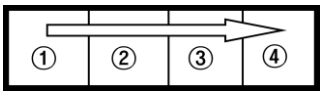
② Set up the data of the parameter setup values

```
Pallet NO = 20
1.XWM=03    3.ZWM=01
2.YWM=02
```

▶ **P23 Point teaching?**
- For 1 stack: P23=P20
- For 2 stacks or more:
P23 is only different from Z-axis of P20.

Job 4 ⇨ **Command : PMOV P9 P1** → **Reference: P1**
X Direction: P2
Y Direction: P3
Z Direction: P4

① Work sequence → Point teaching



How for point teaching to palletize as this sequence?

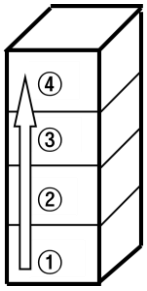
② Set up the data of the parameter setup values

```
Pallet NO = 99
1.XWM=04    3.ZWM=01
2.YWM=01
```

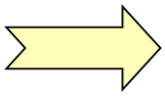
▶ **P4 Point teaching?**
- For 1 stack: P4=P1
- For 2 stacks or more:
P4 is only different from Z-axis of P1.

Job 5 ⇨ **Command : PMOV P0 P0** → **Reference: P0**
X Direction: P1
Y Direction: P2
Z Direction: P3

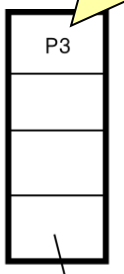
① In the case of 4 stacks at a place



How for point teaching to palletize as this sequence?



▶ **How for P3 point teaching?**
Only Z-axis value is different.



P0 = P1 = P2

② Set up the data of the parameter setup values.

```
Pallet NO = 0
1.XWM=01     3.ZWM=04
2.YWM=01
```

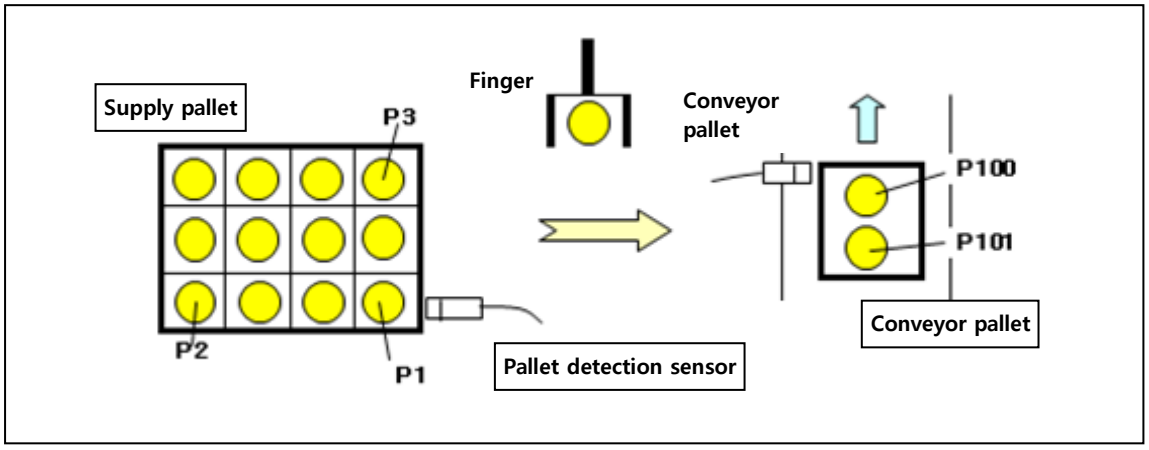
CAUTION

▶ For pallet job pattern No. 1 to 3, refer to Operation Manual, "Chapter 5 Parameter Mode'".

3.18.3 Programming Example (1)

1) Job details

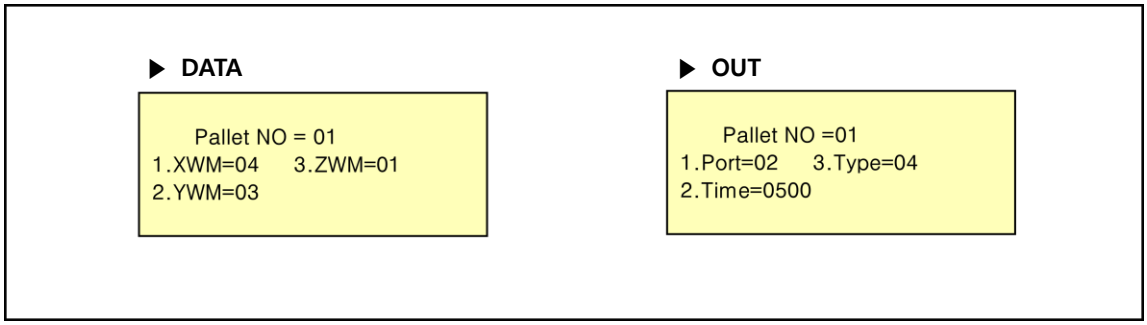
→ The robot picks up a device in a device supply pallet and provides it to a conveyor pallet.



2) User I/Os of robot

IN		OUT	
0	Supply pallet detection	0	Finger clamp start
1	Conveyor pallet detection	1	Finger unclamp start
2	Finger clamp detection	2	Pallet job completion
3	Finger unclamp detection	3	1 cycle completion
4		4	

3) Setup of pallet data parameter

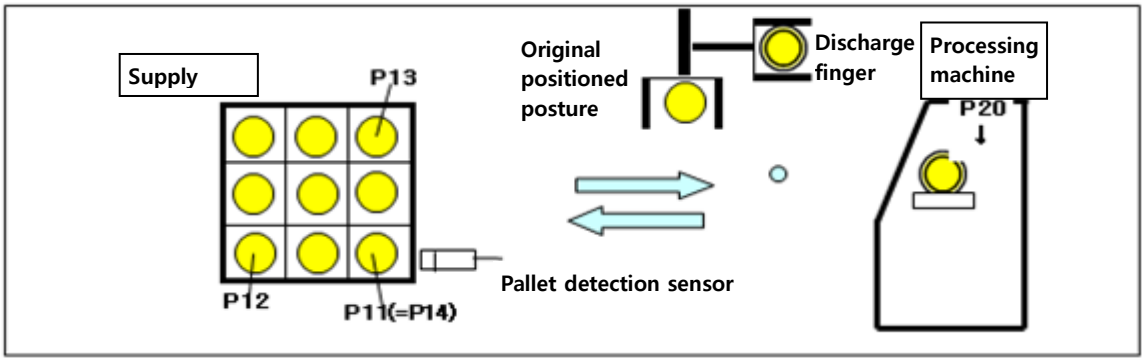


4) Program Writing

```
MAIN
VEL 500          .....   Set up Velocity 50 %
PLUP 5           .....   Set up Z-axis pull-up 5 mm
POUT0=0          .....   Output ALL OFF
CALL HOFF        .....   Finger unclamp
LABL A0
IN0=1            .....   Wait pallet arrival
CALL PALT        .....   Call the pallet job subroutine
IN1=1            .....   Wait conveyor pallet arrival
JMOV P100        .....   Move conveyor to Position P100
CALL HOFF        .....   Finger unclamp
CALL PALT        .....   Call pallet job subroutine
JMOV P101        .....   Move conveyor to Position P101
CALL HOFF        .....   Finger unclamp
OUT3=1 100       .....   1 cycle work completed
GOTO A0          .....   Jump to LABL A0
EOP
                 .....   Picking up a device in a pallet
SUBRPALT
PMOV P1 P1
CALL HON
DLAY 100         .....   Finger unclamp
RET
SUBR HOFF
OUT0=0
OUT1=1
IN3=1            .....   Finger clamp
RET
SUBR HON
OUT1=0
OUT0=1
```

3.18.4 Programming Example (2)

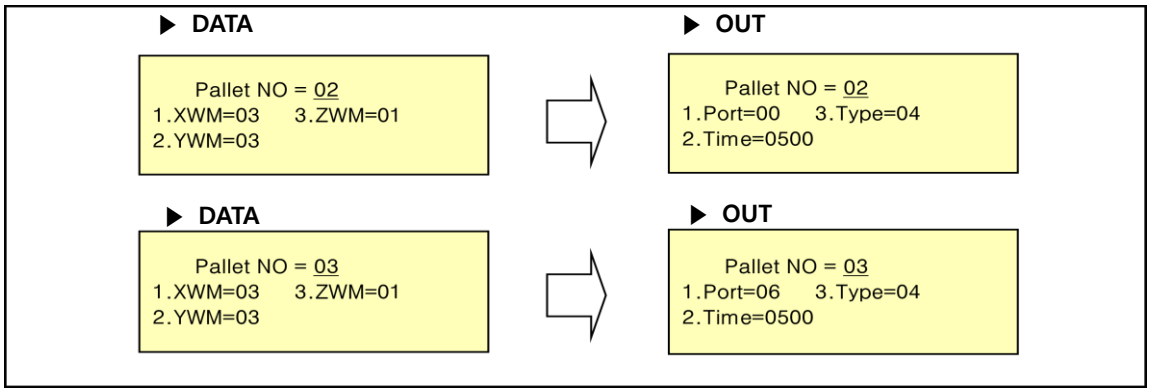
1) Job details → A robot picks up a device in a device supply pallet and provides it to a processing machine, and thereafter loads again the process-completed in the supply pallet.



2) User I/Os of robot

IN		OUT	
0	Work on supply pallet starts	0	Discharge of device in a pallet is completed
1	Processing machine starts	1	Supplying device to a processing machine is completed. (Processing starts.)
2	Supply finger clamping detection	2	Supply finger clamping starts
3	Supply finger unclamping detection	3	Supply finger unclamping starts
4	Discharge finger clamping detection	4	Discharge finger clamping starts
5	Discharge finger unclamping detection	5	Discharge finger unclamping starts
6	Detect Origin of hand	6	Supplying device in a pallet is completed
7	Detect 180-deg. rotation of hand	7	Start Origin searching of hand
8		8	Detect 180-deg. rotation of hand

3) Setup in the pallet data parameter



4) Program Writing

MAIN		
POS AP	.....	Declare POS variable AP
AP=<0,0,-100,0,0,0>		
VEL 500	.....	Set up Velocity 50 %
PLUP 5	.....	Set up Z-axis pull-up 5 mm
JMOV P1	.....	Move to waiting position
POUT0=0	.....	Reset all output signals
CALL H0	.....	Origin of robot hand (0-degree)
CALL HOFF1	.....	Robot hand supply finger unclamp
CALL HOFF2	.....	Robot hand discharge finger unclamp
LABL A0	.....	Set up Label A0
IN0=1	.....	Wait for pallet job start signal
CALL DEPALT	.....	Call subroutine execution statement "DEPALT"
JMOV P1	.....	Move to waiting position
PLUP 0	.....	Disable Z-axis pull-up
IN1=1	.....	Wait for processing machine start signal
CALL H180	.....	Call subroutine execution statement "H180"
JMOV P20	.....	Move to Point P20→021 at processing machine
JMOV P21	.....	
CALL HON2	.....	Robot hand discharge finger clamp
JMOV P20	.....	Move to Point P20 at the upper side of processing machine
CALL H0	.....	Home position of robot hand (0 degree)
JMOV P21	.....	Move to Point P21 at the lower side of processing machine
CALL HOFF1	.....	Robot hand supply finger unclamp
JMOV P20	.....	Move to Point P20→P1
JMOV P1	.....	
OUT1=1 100	.....	Start processing
CALL H180	.....	Call subroutine execution statement "H180"
CALL PALT	.....	Call subroutine execution statement "PALT"
IMOV AP	.....	Relative move (Go up 100 mm along Z-axis)
CALL H0	.....	Home position of robot hand
GOTO A0	.....	Jump to Label A0FH
EOP	.....	"Pallet job" subroutine execution statement
SUBR DEPALT		
PLUP 5		
PMOV P2 P11		
CALL HON1		
DLAY 100		

SUBR PALT	.....	"Pallet job" subroutine execution statement
PLUP 5		
PMOV P3 P11		
CALL HOFF2		
DLAY 100		
RET		
SUBR HOFF1	.....	"Robot hand supply finger unclamping" subroutine execution statement
OUT2=0		
OUT3=1		
IN3=1		
RET		
SUBR HON1	.....	"Robot hand supply finger clamping" subroutine execution statement
OUT3=0		
OUT2=1		
IN2=1		
RET		
SUBR HOFF2	.....	"Robot hand discharge finger unclamping" subroutine execution statement
OUT4=0		
OUT5=1		
IN5=1		
RET		
SUBR HON2	.....	"Robot hand discharge finger clamping" subroutine execution statement
OUT5=0		
OUT4=1		
IN4=1		
RET		
SUBR H0	.....	"Robot hand homing (0 degree)" subroutine execution statement
OUT8=0		
OUT7=1		
IN6=1		
RET		
SUBR H180	.....	"Robot hand rotating (180 degree)" subroutine execution statement
OUT7=0		
OUT8=1		
IN7=1		
RET		

3.18.5 **Programming Example (3)**

1) Job details

If power is turned off and on during a plurality of pallet jobs, number of pallet was being processed is memorized and thereby the process continues.

When in Program Writing, if the integer variable(INT variable name) is declared and PIN9 is read and set to such variable name, the pallet number which was being previously processed is memorized. Write a program, comparing the pallet number set in such variable with one in the program.

▶ E.g.) Using two pallets

```
MAIN
INT PALNO ..... Declare integer variable "PALNO"
PLUP 5
VEL 300
PALNO=PIN9 ..... Store PIN9 value in PALNO
IF PALNO=1 THEN
GOTO PAL1          } If PALNO value is "1", NO.1 pallet job is being performed.

ELSE
IF PALNO=2 THEN
GOTO PAL2          } If PALNO value is "2", NO.2 pallet job is being performed.

ENDIF
ENDIF

LABL LOOP
LABL PAL1
PMOV P1 P10
IF OUT1=0 THEN     } If NO.1 pallet job is not completed,
GOTO PAL1          } Continue NO.1 pallet job
ELSE
GOTO PAL2
ENDIF

LABL PAL2
PMOV P2 P20
IF OUT2=THEN       } If NO.2 pallet job is not completed,
GOTO PAL2          } Continue NO.2 pallet job
ELSE
GOTO LOOP
ENDIF
EOP
```

▶ Setup of parameter pallet data

3. OUT

Pallet No = 1 1. Port : 01 2. Time : 0200 3. Type : 04	Pallet No = 2 1. Port : 02 2. Time : 0200 3. Type : 04

▶ When in Program Writing, Even after the power is turned off and on, if integer variable is declared like the above program and PIN9 is read and set to such variable name, the pallet number which was being previously processed can be memorized.

3.19 PASS

Command	Function	Formation
PASS	Designate workpiece passing during pallet job	PASS_<Pallet No.>_<Order of workpiece to be passed>

3.19.1 Comments

- ▶ Related Command: PMOV
- ▶ The number of PASS able to be used for 1 pallet: 20 Pass's
  → If it is used over 20, "E150 (Passing PLT over)" occurs. .
- ▶ The of PALT using PASS Command able to be used for 1 job can be 5 maximum.
  → See Program Example (2).

3.19.2 Programming Example (1)

1) Job details

→ Perform a pallet job like the right figure. (Only sections are to be performed)

1	2	3
4	5	6
7	8	9

2) Program Writing

```
MAIN
VEL 200
PASS 1 1
PASS 1 3
PASS 1 7
PASS 1 9
LABL A0
PLUP 10
PMOVP1 P10
DLAY 10
GOTO A0
EOP
```

Pass the workpieces 1, 3, 7, and 9 of Pallet No. 1, without performing them.

3.19.3 **Programming Example (2)**

```
MAIN
VEL 200
PLUP 10
LABL A0
PASS 1 1
PASS 1 3
PMOV P1 P10
DLAY 10
PASS 2 2
PASS 2 4
PMOV P2 P10
DLAY 10
PASS 3 5
PASS 3 6
PMOV P3 P10
DLAY 10
PASS 4 1
PASS 4 4
PMOV P4 P10
DLAY 10
PASS 5 2
PASS 5 3
PMOV P5 P10
DLAY 10
GOTO A0
EOP
```

Pass the workpieces 1, and 3 of Pallet No. 1, without performing them.

Pass the workpieces 2 and 4 of Pallet No. 2, without performing them.

Pass the workpieces 5 and 6 of Pallet No. 3, without performing them.

Pass the workpieces 1 and 4 of Pallet No. 4, without performing them.

Pass the workpieces 2 and 3 of Pallet No. 5, without performing them.

3.20 WITH, ENDWT (Concurrent execution command)

Function During robot operation, the following execution statements are concurrently executed.

Formation
```
WITH
  ...
ENDWT
```

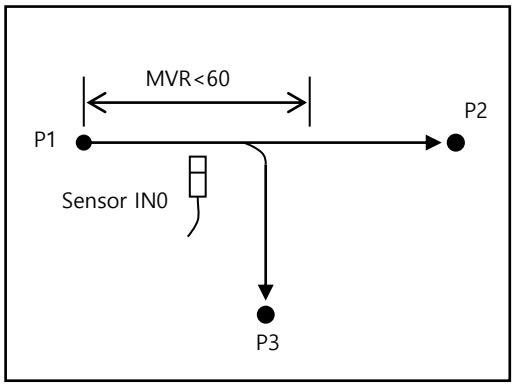
Comments
1) While a first **JMOV Command** in WITH statement is executed, the ensuring commands are concurrently executed.
2) Dividing the robot moving distance on a basis of 100 units by using MVR variable, the conditional command or the like can be executed at the desired point.

CAUTION

In WITH statement, if escaping from WITH ~ ENDWT block by using Jump command due to satisfaction of the condition, **be sure to insert ENDWT to a first jumped step.**

3.20.1 Programming Example (1)

1) Job details → If the input IN0 is turned on during moving P1 to P2, the motion is stopped and is moved to P3. (However, the sensor input signal is checked to only 60% of the moving distance P1 to P2.)



2) User I/Os of robot

IN		OUT	
0	**Sensor detection**	**0**	**Arrival at P2 is completed**
1		**1**	**Arrival at P3 is completed**

3) Program Writing

```
MAIN
VEL 500
JMOV P1
MVR=0 ..... Initialize "Moving distance percentage"
WITH .....
JMOV P2 ..... Moved to Point P2. (only for JMOV)
WHILE MVR<60 ..... During less than 60% of moving distance,
IF IN0==1 THEN
GOTO BB ..... if IN0=1, jump to "LABL BB"
ENDIF
ENDWL
OUT0=1
ENDWT

LABL BB
STOP ..... Stop the robot moving operation
ENDWT ..... End of WITH statement (Must be inserted)
JMOV P3
OUT1=1
EOP
```

3.21 OUT, POUT (Externally output command)

Function Output the designated value, on a bit-by-bit basis or port-by-port.

Formation OUT<bit output port number>=<0 or 1> [pulse effective time] [→ or ~]
OUT<(integer variable)>=<0 or 1>
POUT<output port number>=<data>
<variable>=POUT<output port number>

Term <bit output port number> : Set up the bit output port number. (0 ≤ bit output No. ≤ 95)

1) User Output : OUT0 ~ OUT15
2) Extension 1 output (Option Output 1) : OUT16 ~ OUT47 → Corresponding to one piece of Extension I/O Card mounted.
3) Extension 2 output (Option Output 2) : OUT48 ~ OUT79 → Corresponding to two pieces of Extension I/O Card mounted.

<pulse effective time> : Effective time for outputting 0/1 value (i.e., pulse output) to the relevant bit output port.

[→ or ~] :

- Periodic waveform whose period is the pulse effective time is output.
- The pulse output and periodic waveform output can be used up to maximum 32 outputs (up to one of User, Option1, and Option2).
- Port cannot be used in duplicate. (E.g.) 16 User OUTs + 16 Option OUTs)

1) "Pulse output" and "Periodic waveform output" are performed simultaneously with the step.
That is, in the case that the next step is MOVE command, the output bit is turned on or off, while the execution performing moving.
2) For "Pulse output" and "Periodic waveform output", the port to be used in the parameter has to be determined.
For determination way, refer to "Parameter Instruction".

<output port number> : Set up the output port number. (0 ≤ output port number ≤ 4)

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

<data> : Value (binary or hexa-decimal is available) to be output to the relevant output port.

E.g. 1) If POUT0=10 (10 (decimal) => 0000 0000 0000 1010 (binary)), the output port 1 and output port 3 are ON(1) and the other output ports are OFF (0).

E.g. 2) If POUT0=0H0F0F (0H0F0F (hexa decimal) => 0000 1111 0000 1111 (binary)), the output port 1 and output port 3 are ON(1) and the other output ports are OFF (0).

Comments

OUT<bit output port number>=<0 or 1> [<pulse effective time>] [→]

1) 0/1 value is output to the relevant bit output port according to the designated pulse effective time.
2) If there is no pulse effective time, execution is still valid; and if the pulse effective time has lapsed, execution returns to the previous state.
3) Unit of pulse effective time is 10ms.

OUT<(integer variable)>=<0 or 1>

For "bit output port number", the integer variable can be used.

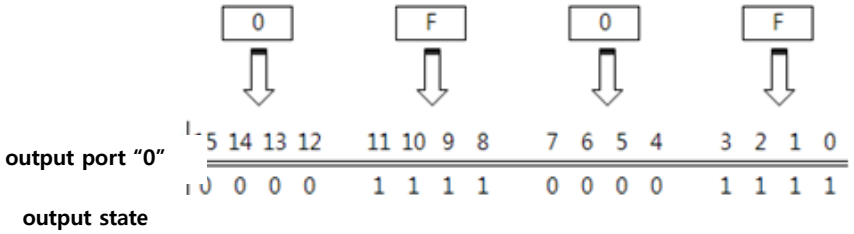
POUT<output port number>=<data>

Data value is output (16 bits) to the relevant output port.

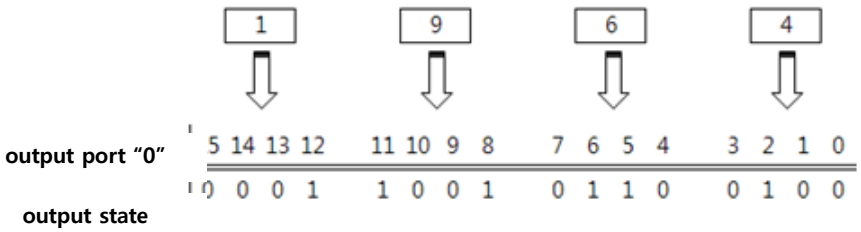
⚠ CAUTION

- ▶ Be careful that, between OUT/POUT and "bit output port number"/"output port number", blanks cannot be inserted.
  E.g.) OUT_15=1, POUT_0=0H00FF (Syntax Error occurs)
- ▶ Data value of output port number increases in the order of 0, 1, 2, ..., 15 from right to left.
- ▶ **The internal ports cannot be pulse-output-controlled.**
- ▶ The extension I/O board is an optional purchasing product.

POUT0=0H0F0F



POUT0=0H1964



3.21.1 **Programming Examples**

1) Pulse output & Data output

```
MAIN
VEL 100
JMOV P10
OUT11=1 100          ..........   Output port No. 11 is ON(1) for 100 ms.
LMOV P100
POUT0=0H1004         ..........   0H1004 (hexa-decimal) value is output to the
                                  output port No. 0~15.
EOP
```

2) Bit output port number is used as a variable

```
MAIN
INT A                ..........   Integer variable A is declared.
FOR A=0 TO 10        ..........   Repeat from 0 to 10
OUT(A)=0             ..........   All outputs of OUT0, OUT1, ..., OUT10 are turned
                                  OFF(0).
NEXT
EOP
```

3.22 IN, PIN (External input command)

Function Receive the value from the designated bit-wise input or port-wise input.

Formation IN<input bit number>=<0 or 1>
PIN<input port number>=<integer or variable>
<variable>=IN<input bit number>
<variable>=PIN<input port number>

Term <bit input port number> : Set up the bit input port number. (0 ≤ bit input No. ≤ 95)

1) User Input : IN0 ~ IN15
2) Extension 1 input (Option Input 1) : IN16 ~ IN47 → Corresponding to one piece of Extension I/O Card mounted.
3) Extension 2 input (Option Input 2) : IN48 ~ IN79 → Corresponding to two pieces of Extension I/O Card mounted.

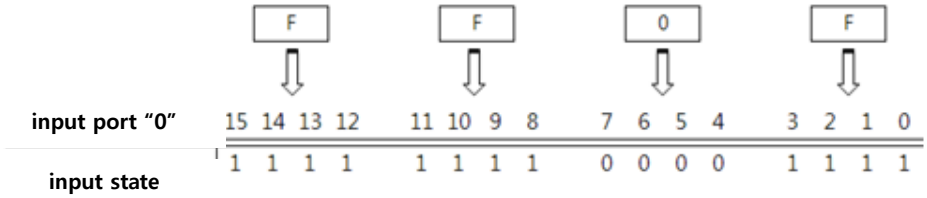
<input port number> : Set up the input port number.
For PIN and WIN : this means 16-bit word.

- If 0 : IN0 ~ IN15 are designated.
- If 1 : IN 16 ~ IN31 are designated.
- If 2 : IN 32 ~ IN 47 are designated.
- If 3 : IN 48 ~ IN 63 are designated.
- If 4 : IN 64 ~ IN 79 are designated.

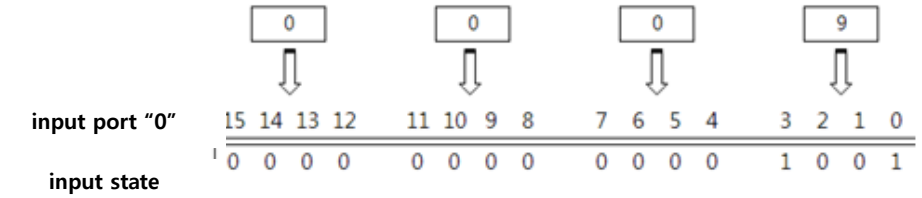
Comments **IN<bit input port number>**
ON/OFF state (1 or 0) of the designated bit input port is stored in IN <bit input port number> variable.
PIN<input port number>
The value (16 bits) of the designated input port is read and stored in PIN <input port number> variable.
IN<bit input port number>=< 0/1 >
Wait until the designated bit input port becomes ON/OFF state (1 or 0).
PIN<input port number>=<16bit value>
Wait until the designated input port value becomes the designated value (16 bits).

- ▶ Be careful that, between IN/PIN and "bit input port number"/"input port number", blanks cannot be inserted.
  E.g.) AA=IN_15, AA=PIN_0 (Syntax Error occurs)
- ▶ Data value of input port number increases in the order of 0, 1, 2, ..., 15 from right to left.
- ▶ The extension I/O board is an optional purchasing product.

PIN0=0HFF0F

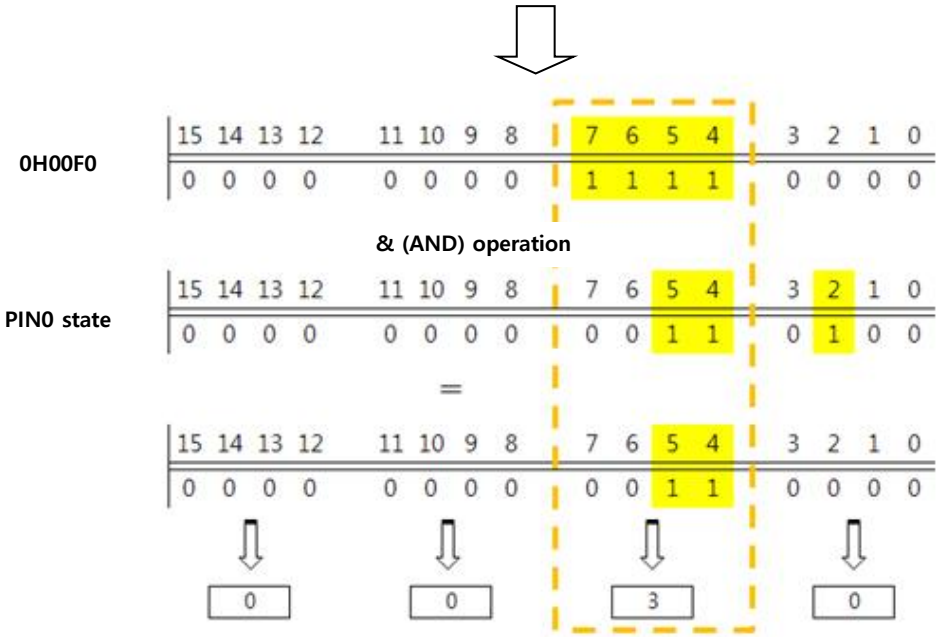


PIN0=0H0009



How to check the signals of "IN4~IN7" only among the input of Input port "0"?

AA=0H00F0&PIN0



3.22.1 **Programming Examples**

1) Waiting the input, Handling the port input on a basis of integer variable, and selecting the moving

```
MAIN
INT D1          ..............  Integer variable D1 is declared.
VEL 100
WHILE 1
JMOV P0
IN1=1           ..............  Wait until the bit input port 1 is ON(1)
IF IN0==1 THEN  ..............  If the bit input port 0 is ON(1),
D1=PIN0         ..............  The states of input ports 0~15 are stored in D1
ELSE
PIN0=0HFFFF     ..............  Wait until the input ports 0~15 are all ON(1).
ENDIF
IF D1==0HFF0F THEN  ..........  Compare the state of D1 (i.e., the state of PIN0)
                                with 0HFF0F.
JMOV P1
ELSE
JMOV P2
ENDIF
EOP
```

2) Model comparison, using BCD unit

- Example that BCD input uses inputs IN0 ~ IN7

```
MAIN
INT M1,M2,MD    ..............  Integer variable M1, M2, and MD are declared.
WHILE 1
M1=PIN0 & 0H000F  ............  Compare PIN0 and 0H000F and store to M1
M2=PIN0&0H00F0  ..............  Compare PIN0 and 0H00F0 and store to M2
M2=(M2>>4)*10   ..............  Shift the value of M2 to 1's place and multiply
                                it with 10 note)
MD=M1+M2        ..............  Sum the operated values of M1 and M2
IF MD==1 THEN
JMOV P0
ELSE
IF MD==11 THEN
JMOV P1
ENDIF
ENDIF
EOP
```

CAUTION

▶ Since, for the port input, hexa-decimal operation is performed, if 10 is input by means of BCD unit, a controller recognizes it as 16. At this time, for exact computation, decimal conversion must be done after shift operation as shown in the above example.

3.23 CIN, CBIN, CWIN, CDIN, and CFIN (Field bus input command)

Function Receive the bit-wise or port-wise input designated in a field bus card.

Formation Bit area Command

<integer variable>=CIN<input bit number>

<integer variable>=CIN<integer variable>

<integer variable>=CBIN<input port number>

Word area Command

<integer variable>=CWIN<input port number>

<integer variable>=CDIN<input port number>

<real number variable>=CFIN<input port number>

Term Bit area: Area that read out and write in on a basis of 1 bit.

<input bit number>: Set up the input bit number. (0 ≤ input bit number ≤ 127)

1) System area : CIN0~CIN31 (Cannot be used)
2) User area : CIN32 ~ CIN127

<input port number>: Set up the input port number. (0 ≤ input port number ≤ 15)

1) System area : CBIN0~CBIN3 (Cannot be used)
2) User area : CBIN4~CBIN15
   - port number 0: CIN0 ~ CIN7
   - port number 1: CIN8 ~ CIN15
   - port number 2: CIN16 ~ CIN23
   - port number 4: CIN24 ~ CIN31
   - port number 5: CIN32 ~ CIN39

Word area: Area that reads out and writes in on a basis of Word (16 bits).

<input port number>: Set up the input port number. (0≤input port number≤15).

1) User area: CWIN0 ~ CWIN15
2) CDIN<port number>: Receive the value of Word area on a basis of double word (32 bits) (integer).
   - port number 0: CWIN0 ~ CWIN1
   - port number 1: CWIN2 ~ CWIN3
   - port number 2: CWIN4 ~ CWIN5
3) CFIN<port number>: Receive the value of Word area on a basis of double word (32 bits) (real number).
   - port number 0: CWIN0 ~ CWIN1
   - port number 1: CWIN2 ~ CWIN3
   - port number 2: CWIN4 ~ CWIN5

Comments CIN<input bit number>

Store ON/OFF state (1 or 0) of the designated bit input port in IN<bit input port number> variable.

CIN<integer variable>

For "input bit number", the integer variable can be used.

CBIN<input port number>

Read the value (8 bits) of the designated input port and store it in CBIN<input port number> variable.

CWIN<input port number>

Read the value (Word) of the designated input port and store it in CWIN<input port number> variable.

CDIN<input port number>

Read the value (Double Word) of the designated input port and store it in CDIN<input port number> variable.

CFIN<input port number>

Read the value (Double Word) of the designated input port and store it in CFIN<input port number> variable.

CAUTION

- ▶ Be careful that, between CIN/CBIN/CWIN/CDIN/CFIN and "bit input port number"/" input port number", blanks cannot be inserted.
  E.g.) AA=CIN_15, AA=CBIN_0 (Syntax Error occurs)
- ▶ Data value of input port number increases in the order of 0, 1, 2, ..., 15 from right to left.
- ▶ Commands for Field bus is applied for only when PARA->PUB->HWCONF->COMM->Fieldbus->MAP is USER_PEF.

3.23.1 Programming Example

```
MAIN
INT D1                  .....   Integer variable D1 is declared.
VEL 100
WHILE 1
JMOV P0
IF CIN50==1 THEN        .....   If bit input port 51 is ON(1),
D1=CBIN5                .....   States of input ports 40~47 are stored in D1
ELSE
D1=CBIN6                .....   States of input port 48~45 are stored in D1
ENDIF
IF D1==0H0F THEN        .....   Compare the state of D1 with 0H0F
JMOV P1
ELSE
JMOV P2
ENDIF
EOP
```

3.24 COUT, CBOUT, CWOUT, CDOUT, CFOUT (Field bus output command)

Function Output the designated bit-wise or port-wise value to a filed bus card.

Formation Bit area Command

COUT<output bit number>=<0 or 1>

COUT<integer variable>=<0 or 1>

CBOUT<output port number>=<data>

Word area Command

CWOUT<output port number>=<data>

CDOUT<output port number>=<data>

CFOUT<output port number>=<data>

Term Bit area: Area that read out and write in on a basis of 1 bit.

<output bit number>: Set up the output bit number. (0≤output bit number≤127)

1) System area : COUT0~COUT31 (Cannot be used)
2) User area : COUT32 ~ COUT127

<output port number>: Set up the output port number. (0≤output port number≤15)

1) System area : CBOUT0~CBOUT3 (Cannot be used)
2) User area : CBOUT4~CBOUT15
   - port number 0: COUT0 ~ COUT7
   - port number 1: COUT8 ~ COUT15
   - port number 2: COUT16 ~ COUT23
   - port number 4: COUT24 ~ COUT31
   - port number 5: COUT32 ~ COUT39

Word area: Area that reads out and writes in on a basis of Word (16 bits).

<output port number>: Set up the output port number. (0≤output port number≤15)

1) User area: CWOUT0 ~ CWOUT15
   - CDOUT<port number>: Output the value of Word area on a basis of double word (32 bits)(integer).
   - port number 0: CWOUT0 ~ CWOUT1
   - port number 1: CWOUT2 ~ CWOUT3
   - port number 2: CWOUT4 ~ CWOUT5
2) CFOUT<port number>: Output the value of Word area on a basis of double word (32 bits) (real number).
   - port number 0: CWOUT0 ~ CWOUT1
   - port number 1: CWOUT2 ~ CWOUT3
   - port number 2: CWOUT4 ~ CWOUT5

Comments COUT<output bit number>=<0 or 1>

Output ON/OFF state (1 or 0) to the relevant output bit number.

COUT<integer variable>=<0 or 1>

For "output bit number", the integer variable can be used.

CBOUT<output port number>=<data>

Output data (8Bit) value to the relevant output port.

CWOUT<output port number>=<data>

Output data(1Word) value to the relevant output port.

CDOUT<output port number>=<data>

Output data(2Word) value to the relevant output port.

CFOUT<output port number>=<data>

Output data(2Word) value to the relevant output port.

CAUTION

- ▶ Be careful that, between COUT/CBOUT/CWOUT/CDOUT/CFOUT and "bit input port number"/" input port number", blanks cannot be inserted.
  E.g.) COUT_15=1, CBOUT_5=AA (Syntax Error occurs)
- ▶ Data value of output port number increases in the order of 0, 1, 2, ..., 15 from right to left.
- ▶ Commands for Field bus is applied for only when PARA->PUB->HWCONF->COMM->Fieldbus->MAP is USER PEF.

3.24.1 Programming Examples

1) Bit area output

```
MAIN
INT A ..... Integer variable A is declared.
FOR A=32 TO 42 ..... Repeat execution of 32 ~ 42, and
COUT(A)=0 ..... Turn OFF (0) all outputs of OUT32, OUT33, ..., OUT42
NEXT
EOP
```

2) Word area output

```
MAIN
POS CURR
WHILE 1
CURR=HERE ..... Store the current coordinate of a robot in CURR
CFOUT0=CURR.1 ..... Output No. 1 axis value of CURR to the field
                    bus (CWOUT0 and CWOUT1)
CFOUT1=CURR.2 ..... Output No. 2 axis value of CURR to the field
                    bus (CWOUT2 and CWOUT3)
ENDWL
EOP
```

3.25 VEL (Axis moving velocity setup command)

Function Set up the percentage (%) of the axis moving velocity

Formation VEL <velocity>

Term <velocity> : Set the robot moving velocity. (0 ≤ velocity ≤ 100.0%)
Robot moving velocity = Mv(maximum velocity of each axis) * 0.001 * velocity
(0 ≤ velocity Mv ≤ maximum RPM).

Comments
1) This is the velocity when a robot moves, and **the maxim velocity is set to 1000**.
2) If the velocity has not been set up in the program, the velocity preset in INIT_V parameter is **automatically set up**.
3) The variable**(integer variable) having the velocity value can be used.**

CAUTION

- Velocity value is 100.0%.
  If VEL 1000 is entered, a robot moves at the velocity of 100%. (VEL 100 => 10%)
- If the velocity exceeds the permissible maximum RPM of the mechanism, there may occur noise and damage.
  Be sure to confirm a label attached to the mechanism before use.

3.25.1 Programming Example

1) Change the velocity according to the work position

```
MAIN
WHILE 1
VEL 1000          ..............  robot moving velocity MV x 0.001 x 1000
JMOV P0
JMOV P1
VEL 200           ..............  robot moving velocity MV x 0.001 x 200
LMOV P2
LMOV P3
VEL 1000          ..............  robot moving velocity MV x 0.001 x 1000
JMOV P0
ENDWL
EOP
```

3.26 ACC, DEC (Acceleration/deceleration setup command)

Function Set up the percentage (%) of acceleration and deceleration time

Formation ACC <acceleration time rate>
DEC <deceleration time rate>

Term <acceleration time rate/deceleration time rate> : (50 ≤ acc time rate & dec time rate ≤ 200%)
- Increase or decrease the acceleration and deceleration time in a program.
- acceleration and deceleration time = At(acceleration and deceleration time) * 0.01 * acceleration and deceleration rate (%)
(0.5*(maximum acceleration and deceleration time) ≤ (acceleration and deceleration time) ≤ 2*(maximum acceleration and deceleration time))

Comments
1) Robot's acceleration and deceleration time is set up.
2) If in JOB program the acceleration and deceleration time rate have not been set up, they are set to 100%.
3) Variables (integer variable) having the acceleration and deceleration time rate values can be used.
4) Applied to all axes. (SCARA robot : applied to A, B, Z, and W-axis)

E.g.) parameter At: A-axis(0.3sec.), B-axis(0.3sec.), Z-axis(0.2sec.), W-axis(0.5sec.)
Execute ACC(DEC) 50
At : A-axis(0.15sec.), B-axis(0.15sec.), Z-axis(0.1sec.), W-axis(0.25sec.)

▶ The values themselves of parameter are not changed, but applied to only within the JOB program

5) For returning to parameter acceleration and deceleration time (At), "ACC(DEC) 100" is used and this is applied to from the next step.
(See Programming Example)

▶ The acceleration and deceleration time preset in the parameter has been set optimally according to the construction of the mechanism.

▶ If the acceleration and deceleration time rate in ACC/DEC Command are set to under 100, be careful of the noise and vibration of the mechanism.

3.26.1 **Programming Example**

1) Change the acceleration and deceleration time according to the work position

- Example when At value is set to 0.3 seconds in the parameter.

```
MAIN
WHILE 1
VEL 500                 ..............  Set the acceleration and deceleration time to
                                        0.15 seconds.
ACC 50
JMOV P1
ACC 200                 ..............  Set the acceleration and deceleration time to
                                        0.6 seconds.
LMOV P2
LMOV P3
ACC 100                 ..............  Set the acceleration and deceleration time to
                                        0.3 seconds.
JMOV P0
ENDWL
EOP
```

3.27 FOS, PFOS, (Continuous trajectory creation command)

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

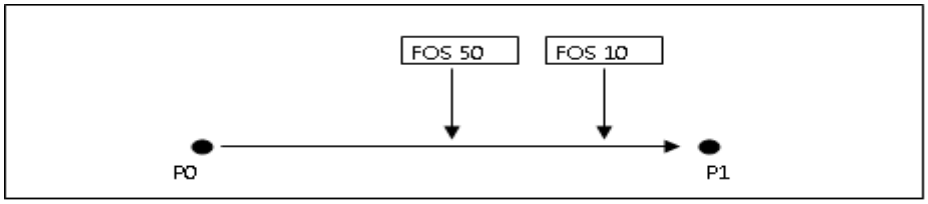
Formation FOS <distance rate>

PFOS <distance>

Term <distance rate> : (0 ≤ distance rate < 50%)

- Set up the time of moving or outputting to the next position before the robot arrives at the designated position.
- The distance rate is to be set up as the percentage value of the moving distance between two points.
- Except JMOV, 50% or more cannot be used for other motions.

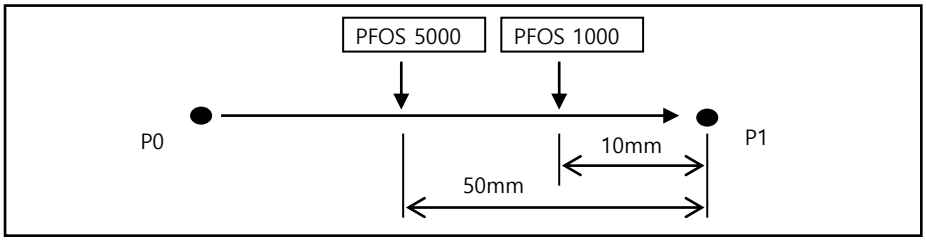
(0 ≤ distance rate < 50)



Above figure shows the time of moving or outputting to the next position according to each FOS value, when moving from P0 to P1.

<distance> : Set up the distance of moving or outputting to the next position before the robot arrives at the designated position.

(0 ≤ distance ≤ 999mm) Unit: 0.01 mm



Above figure shows the time of moving or outputting to the next position according to each PFOS value, when moving from P0 to P1.

CAUTION

- When in point teaching for continuous interpolation motion, the distance between two points must be at least 5 mm or more. (only for 100mm/s moving velocity)
- If FOS rate of the trajectory which goes on moving is shorter than the total length of the next position, "Too much FOS" Alarm occurs.
- If the moving trajectory is PTP, the trajectory is changed; if interpolation motion, the trajectory is inserted(circle, arc, or straight line).

Comments
1) If FOS or PFOS is set up, MOVE command **moves to the next position** before arriving at the designated position (the set distance rate or distance value).
2) For OUT command, it is possible for outputting while in moving.
3) PFOS is applied only to **interpolation motion(LMOV, CMOV, and AMOV)**.
4) In job program, in order to **return to the previous state** that FOS or PFOS has been applied to, **"FOS 0"** or **"PFOS 0"** is used, and the command is involved from the next step command.

```
MAIN
FOS 10
LMOV P0
LMOV P1
FOS 0
LMOV P2
LMOV P3
EOP
```

FOS 0 Command is applied from LMOV P2.

CAUTION

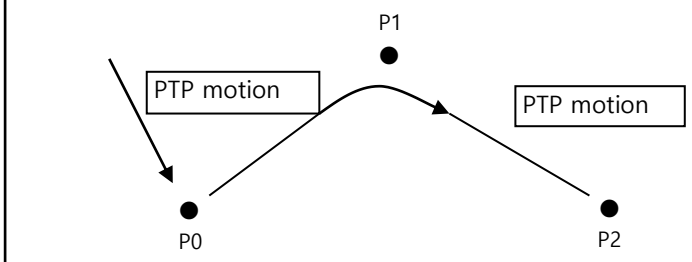
- By using FOS and PFOS, the continuous interpolation motion is performed.
- Wrong point teaching when in arc or circular motion (AMOV or CMOV) may cause, during Auto RUN, "Unreachable Point" or "Inverse Error".
- Pay special attention to using small scale AMOV or CMOV command.

3.27.1 Programming Examples

1) PTP motion (JMOV)

```
MAIN
VEL 100
JMOV P0
FOS 5
JMOV P1
FOS 0
JMOV P2
EOP
```

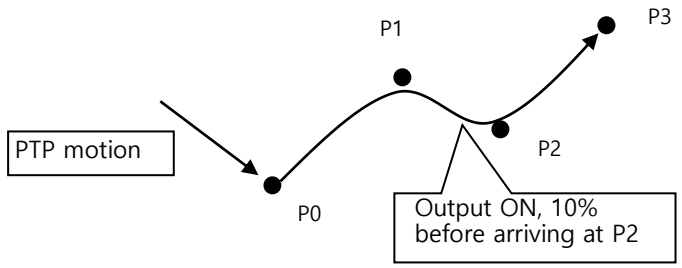
FOS 5 (5% before arriving at Point P1)



2) Turning On/Off the output signal (OUT, POUT) while robot moves

```
MAIN
VEL 100
JMOV P0
FOS 10
LMOV P1
LMOV P2
OUT0=1
FOS 0
JMOV P3
EOP
```

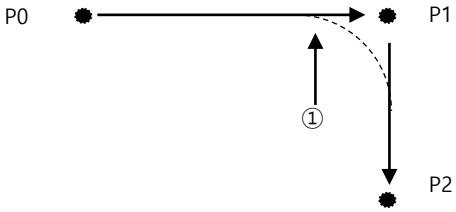
FOS 10



3) Handling next step command while robot moves

```
MAIN
WHILE 1
VEL 100
JMOV P0
FOS 10
IF IN0==1 THEN
JMOV P1
ELSE
IN1=1
JMOV P2
ENDIF
FOS 0
ENDWL
EOP
```

FOS 10



At position ①, IF command is handled, 10% before arriving at P1.
That is, if a signal enters the input "IN0", a robot moves along the trajectory ② (dotted line).

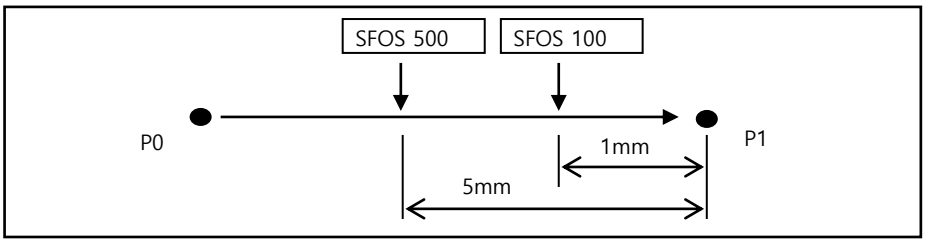
3.28 SFOS (Continuous trajectory and uniform velocity creation command)

Function This command is identical to FOS in that the trajectory is changed to the next target point before the end of axis arrives at a target point, but, when SFOS command is applied, the velocity is uniformly retained, from the time such command is applied, in reference to SFOS setup value.

Formation SFOS <distance><velocity level><arc angle>

Term <distance> : Set up the distance of moving or outputting to the next position before the robot arrives at the designated position.

(0 ≤ distance ≤ 999mm) Unit: 0.01mm



Above figure shows the time of moving or outputting to the next position according to each SFOS value, when moving from P0 to P1.

<velocity level>: Set up the velocity in the moving distance to which SFOS command is applied.

The higher the setup value, the faster the velocity. (0≤ velocity level ≤ 10)

If the setup value is 0: the same as Level 5.

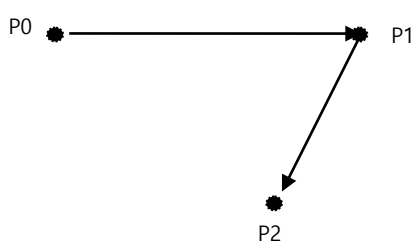
<arc angle>: Enter an arc insertion angle between one trajectory and another which are connected with each other.

If the angle is smaller than the entered angle value, the arc is not inserted; if greater, the arc is inserted as the first factor value.

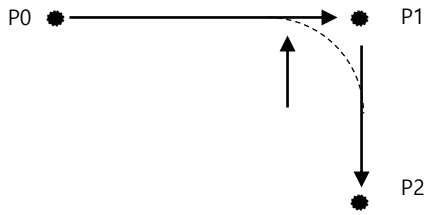
If the setup value is 0 degree: the same as 1 degree value.

If the setup value is 180 degree: the arc is not inserted between trajectories, and Too Much FOS alarm is not generated. If 180 degree is selected, deformation in trajectory and vibration during robot motion may occur.

SFOS 10 0 80



If an angle is smaller than the third factor, the arc is not inserted.



When an angle is greater than the third factor, the arc is inserted.

Comments
1) If SFOS is set up, MOVE command **moves to the next position** before arriving at the designated position (the distance value)
2) **It is impossible to use DLAY Command between MOVE Commands.**
3) In continuous interpolation motion, continuous moving (motion of uniform velocity) can be achieved without acceleration or deceleration.
4) SFOS is applied only to **interpolation motion(LMOV, CMOV, and AMOV)**.
5) In job program, in order to **return to the previous state** that SFOS has been applied to, **"SFOS 0 0 0"** is used, and the command is involved from the next step command.

```
MAIN
SFOS 10 0 0
LMOV P0
LMOV P1
SFOS 0 0 0
LMOV P2
LMOV P3
EOP
```

SFOS 0 Command is applied from LMOV P2.

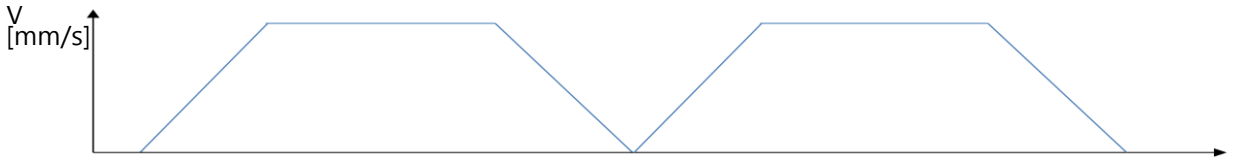
CAUTION

- By using SFOS, the continuous interpolation motion is performed.
- Wrong point teaching when in arc or circular motion (AMOV or CMOV) may cause, during Auto RUN, "Unreachable Point" or "Inverse Error".
- Pay special attention to using small scale AMOV or CMOV command.

3.28.1 Velocity profile pattern of continuous motion

1) If FOS is 0

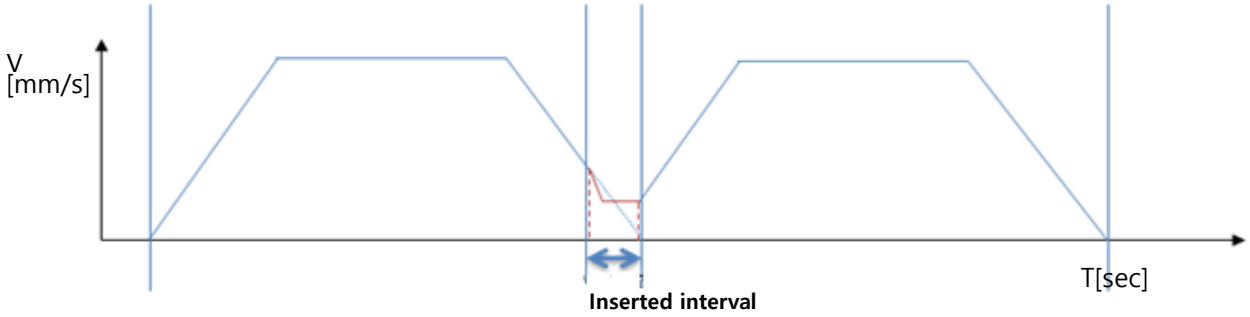
If there is no FOS Related Command, the velocity at the trajectory connected portion becomes zero (0).



[If FOS Command is not applied]

2) FOS 10 & PFOS 10

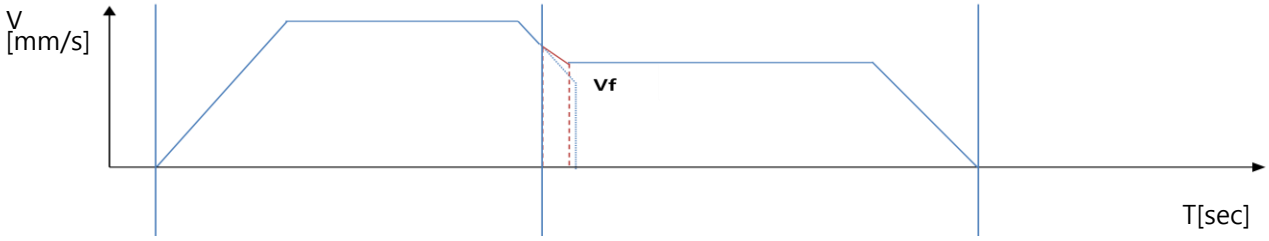
If FOS Command is inserted, the velocity at the inserted interval (arc) is retained to the entering velocity to the inserted interval.



[If FOS or PFOS Command is applied]

3) SFOS 10 0 0

The maximum velocity Vf of the next trajectory is determined by SFOS setup value and the velocity level, and thereby the moving velocity is retained.



[If SFOS Command is applied]

CAUTION

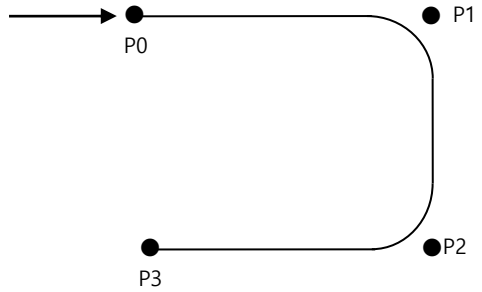
▶ If the connected trajectory is too short and so Vf cannot be retained, the uniform velocity is not achieved. To avoid that, lower the velocity level or set up the SFOS amount to the minimum length in the interval.

3.28.2 **Programming Examples**

1) Linear interpolation motion (LMOV)

```
MAIN
VEL 100
LMOV P0
SFOS 150 0 70
LMOV P1
LMOV P2
SFOS 0 0 0
LMOV P3
EOP
```

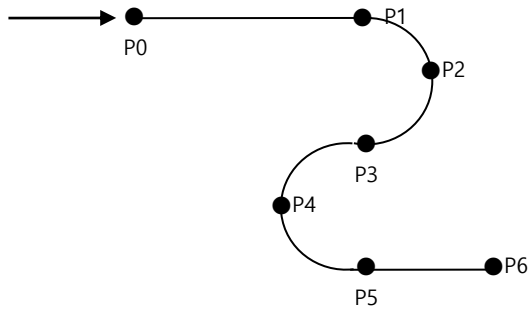
SFOS 150 0 70
Move to the next position, 15 mm before arriving at P1 and P2.



2) Arc interpolation motion (AMOV)

```
MAIN
VEL 100
JMOV P0
LMOV P1
SFOS 150 0 180
AMOV P2 P3
AMOV P4 P5
SFOS 0 0 0
LMOV P6
EOP
```

SFOS 150 0 180(0)
Move to the next position, 1.5 mm before arriving at P1 and P2.



3.29 SVON, SVOF (Servo ON/OFF command)

Function Servo ON, Servo OFF

Formation
SVON : All axes servo ON
SVON [<designated axis>] : Designated axis servo ON
SVOF : All axes servo OFF
SVOF [<designated axis>] : Designated axis servo OFF

Comments
1) SVOF :
- Turn the **servo OFF** during robot operation to change the robot body in Free mode (robot can be moved by hand).
- Turn off the servo of all axes currently being operated.

2) SVON :
- Turn ON the servo of the robot body. Related to SVOF.
- When SVON after SVOF has been not performed, robot operation (RUN) is in Dry RUN mode (the robot body does not move and JOB program is executed step-by-step).
- Turn on the servo of all axes currently turned off.

3) SVON(SVOF) <designated axis> : Servo ON/OFF for individual axis. (Range: 1 to 5)
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

3.29.1 Programming Example

1) PTP motion (JMOV)

```
MAIN
VEL 100
JMOV P0
IF IN1==1 THEN
SVOF 4 ..... 4th axis servo OFF
ENDIF
JMOV P1 ..... Move to Point P1 under the state 4th axis servo OFF
IF IN1==0 THEN
SVON 4 ..... 4th axis servo ON
ENDIF
JMOV P2 ..... Move to Point P2 under the state all axes servo ON
ENDWL
EOP
```

2) PTP motion (JMOV)

```
MAIN
VEL 500
JMOV P0
MVR=0
WITH
JMOV P1
WHILE MVR<100
IF IN0==1 THEN
STOP                ..................    If IN0=1, while moving to P1,
SVOF                                       Stop the robot and then servo OFF
GOTO L1
ELSE
EXIT
ENDIF
ENDWL

LABL L1
ENDWT
IN0=1                                      After input IN0 is "1",
SVON                ..................    Servo ON
JMOV P2
EOP
```

CAUTION

▶ If SVON and SVOF are used in a subroutine, **"Syntax Error"** arises.

E.g.)

```
MAIN
VEL 200
IN0=1
JMOV P0
CALL A0
EOP

SUBR A0
SVOF
OUT1=1
IN1=1
SVON
RET
```

Correct program ⇒

```
MAIN
VEL 200
IN0=1
JMOV P0
SVOF
CALL A0
SVON
EOP

SUBR A0
OUT1=1
IN1=1
RET
```

3.30 DLAY (Time delay command)

Function Set up delay time

Formation DLAY <delay time>

Term <delay time> : Set up the time to be delayed. (1=10ms)
(1 ≤ <delay time> ≤ 999)

Comments 1) The time is set by number.
2) 0.01 to 9.99 seconds of time can be set up.
3) Variable (integer variable) having time value can be used.

3.30.1 Programming Example

1) PTP motion (JMOV)

```
MAIN
VEL 100
JMOV P10
DLAY 10 ..... Wait at position P10 for 100ms (0.1 sec.)
LMOV P100
EOP
```

3.31 OFFS (Offset command)

Function Temporarily shift the taught target point by the designated value.

Formation OFFS P<number>
OFFS GP<number>
OFFS <position variable>

Term <number> : Set the taught position coordinate No.
P – LOCAL POINT SEPARATELY USED FOR EACH JOB. (0 ≤ No. ≤ 1999)
GP – GLOBAL POINT used in common (0 ≤ No. ≤ 254)
<position variable> : Means the variable name declared to POS-type.

Comments 1) **Store the offset value, as Cartesian coordinate, in an arbitrary point of JOB program**.
After OFFSET Command is executed and when MOVE command of the next step is executed, **a robot moves by the increased or decreased value according to the offset.**

2) This command is effective only within **JOB program which is being performed.**
OFFS value of MAIN program is not available in **Job Call(JCALL)'ed program.**

Move to offset applied position

CAUTION

- If in point teaching the point stored in JOB program is located in "RANG" of the parameter, "Range OVER" Error is not generated; but when JOB program is performed, "Range OVER" Error may be generated due to OFFS.
- To do Point Teaching, enter the offset value as ANGLE coordinate.
- To disable OFFS, assign the value of <0, 0, 0, 0, 0 ,0>.

3.31.1 Programming Example

PTP motion (JMOV) <SCARA robot>

```
P11 <10.23, 22.37, 7.96, 120.00, 0, 0>
P12 <30.23, 22.37, 20.96, 200.00, 0, 0>
P100 <100, 100, 0, 0, 0, 0>

MAIN
VEL 100
JMOV P10        .....   PTP move to Point P10
OFFS P100       .....   Offset applied to the value stored at P100
LMOV P11        .....   CP move to position of P11+P100 <110.23,122.37,7.96,120.00, 0, 0>
LMOV P12        .....   CP move to position of P12+P100<130.23, 122.37, 20.96, 200.00, 0,
                        0>
EOP
```

3.32 LIMT (Axis limitation command)

Function Limit the moving range of each axis

Formation LIMT P<number1> P(number2)
LIMT <position variable> (position-type variable)

Term <number1> : Set the negative (-) limit value in POINT of JOB program.
(0 ≤ <number> ≤ 1999)
(number2) : Set the positive (+) limit value in POINT of JOB program.
(0 ≤ <number> ≤ 1999)

Comments 1) In JOB program, **software limit is set up**.
2) This is effective in **JOB program currently being performed.**
LIMT value of MAIN program is also applied to **Job Call(JCALL)'ed program**.

3.32.1 Programming Example

1) PTP motion (JMOV) <SCARA robot>

```
MAIN
VEL 100
JMOV P0
LIMIT P100 P101
JMOV P1
EOP
```

If, in horizontal articulated robot, the limit is set to P100 and P101,
Position P100: (A:-100.00 B:-100.00 Z:000.00 W:-300.00)
Position P101: (A:100.00 B:100.00 Z:100.00 W:300.00)

The software limit in JOB program is applied as follows.
A: -100 degree ~ +100 degree
B: -100 degree ~ +100 degree
Z: 0 ~ 100 mm
W:-300 degree ~ +300 degree

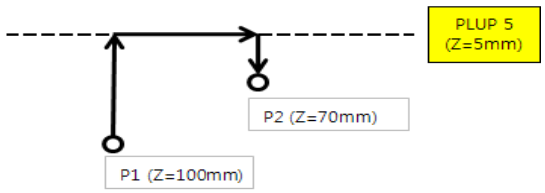
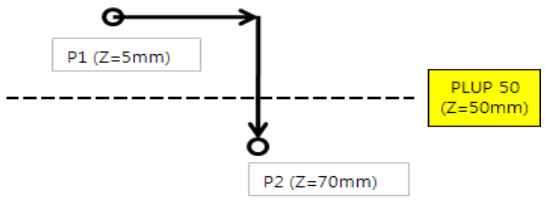
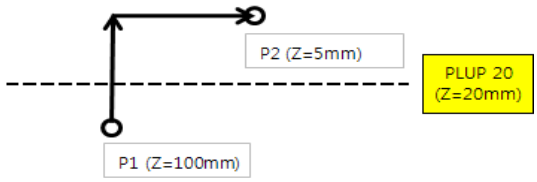
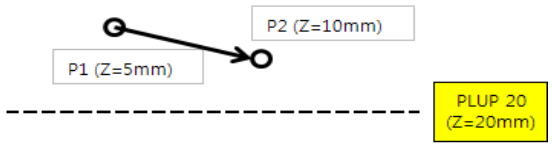
3.33 PLUP (Pull-up operation command)

Function Set up Z-axis value for pull-up operation

Formation PLUP <Z-axis value>

Term <Z-axis value> : Set up Z-axis pull-up value.
(0 ≤ Z-axis value ≤ Z-axis stroke)

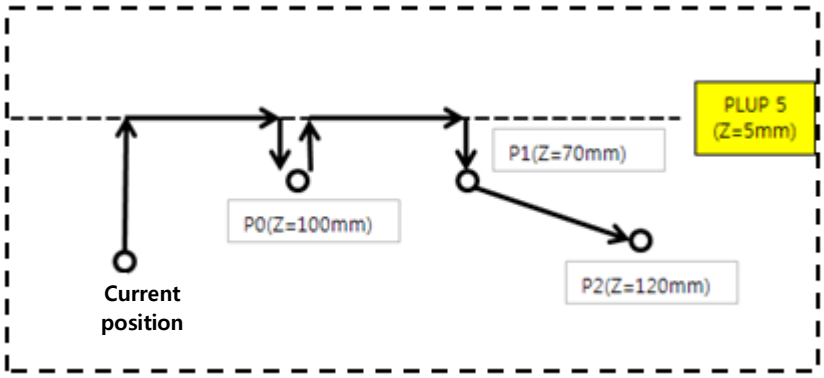
Comments
1) **Applied only to a robot having Z-axis**. If there are two or more Z-axes, **only the first Z-axis is applied**.
2) **Applied only to PTP motion (JMOV, PMOV, and IMOV)**; not applied to interpolation motion.
3) Variable (integer variable) having Z-axis pull-up value can be used.
4) To disable Z-axis pull-up value, use **"PLUP 0"**, and it is applied from the next step.
5) Robot motions according to Z-axis pull-up value are as follows.

Case division	Robot motion	Description
1. In the case that P1 and P2 values are all **greater** than the pull-up value		1. At P1, raise Z-axis up to the pull-up value. 2. Move, except Z-axis, the remaining axes to the pull-up value position over P2. 3. Lower Z-axis down to P2.
2. In the case that Z-axis value of P1 is **smaller** than the pull-up value, and Z-axis value of P2 is **greater** than the pull-up value		1. Move, except Z-axis, the remaining axes to the pull-up value position over P2. 2. Lower Z-axis down to P2.
3. P1 In the case that Z-axis value of P1 is **greater** than the pull-up value, and Z-axis value of P2 is **smaller** than the pull-up value		1. At P1, raise Z-axis up to Z-axis position P2. 2. Move to position P2.
4. In the case that P1 and P2 values are all **smaller** than the pull-up value		**Move to P2, regardless of PLUP command.**

3.33.1 **Programming Example**

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
PLUP 5          .....   Set up Z-axis pull-up value to 5mm
JMOV P0
JMOV P1
PLUP 0          .....   Disable Z-axis pull-up value
JMOV P2
EOP
```



3.34 TOOL (Axis limit command)

Function Select a tool attached to the robot

Formation TOOL <tool No.>

Term <tool No.> : In the parameter, set up TOOL Number in which "TOOL" offset value (dx, dy, and dz) is stored.

(0 ≤ <tool No.> ≤ 3)

Comments 1) Number of tool to be used is set up.

2) Shape of the tool is set up in the system parameter.

CAUTION

- ▶ In a horizontal articulated robot, the offset value for TOOL 0 is used for a reference to a jog of the teaching pedant.
- ▶ In job program, if "TOOL Command" is not used, "TOOL 0" offset value is applied.
- ▶ Use "TOOL 0" for initialization of TOOL offset, and set up TOOL offset to one of 1 to 3, as possible.

CAUTION

- ▶ Teaching the work point, in a job using a tool having the offset
  1) Set up the exact too offset (See Parameter)
  2) Before point teaching, set up W-axis to one of either fix (FIX 1) or rotation (FIX 0)

CAUTION

- ▶ Pay attention to work point teaching:
  In a job including continuous interpolation motion (sealing or dispensing), when teaching, with a tip of a tool, the work point, unreasonable motion may be burdened to A(X) and B(Y) axes.

- ■ Making and fastening the tool
  1. Make the tool so that it is possible for W-axis for fixing explained in "LMOV Command".
  2. Fasten the tool so that the tool offset can exist along one direction of either dx or dy.

3.34.1 **Programming Example**

1) Interpolation motion using TOOL

MAIN	
VEL 300	
TOOL 1	Using TOOL No. 1, start the interpolation motion
FIX 0	
LMOV P1	
LMOV P2	
OUT0=0	
FIX1	
TOOL 0	Change to the non-offset tool No. 0
EOP	

3.35 FIX (Axis limit command)

Function Select W-axis motion in the interpolation motion

Formation FIX <type>

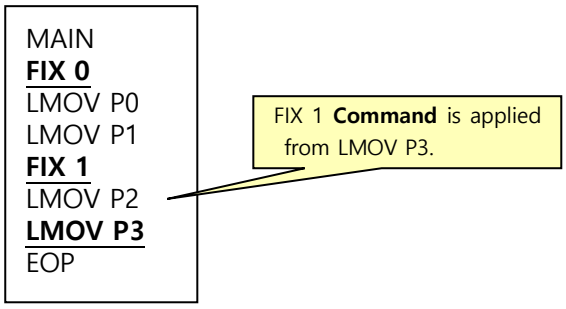
Term <type>

1 → Form of the robot hand is fixed (W-axis fixed form)

0 → Form of the robot hand is rotated (W-axis rotating form)

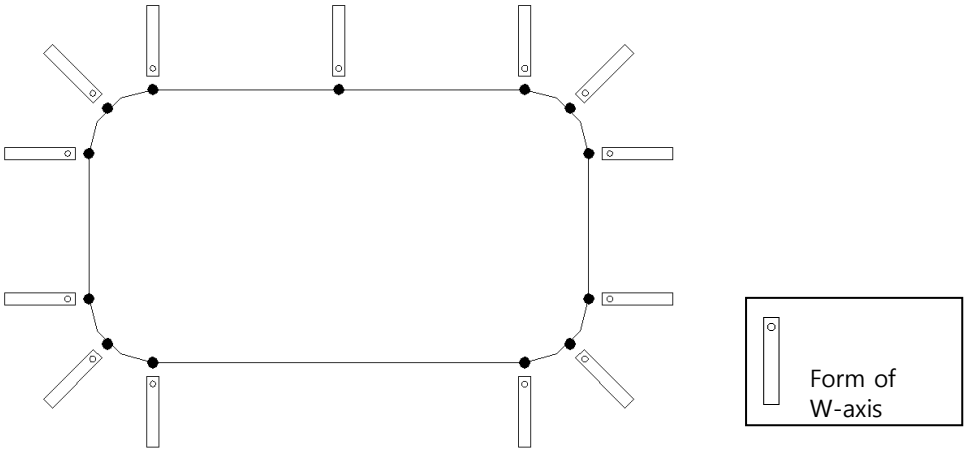
Comments

1) When a robot is in interpolation motion, the form of robot hand (W-axis) is fixed or rotated.
2) In PTP motion, the robot hand moves to the taught point, regardless of FIX setting.
3) If it is set to "FIX 0", the rotating axis of robot (W-axis) interpolation-moves to the coordinate value of W-axis.
4) If it is set to "FIX 1", the rotating axis of robot (W-axis) interpolation-moves, with keeping the form of a starting point.
5) If set to "FIX 1", the coordinate value of the rotating axis (W-axis) may have different value other than that of the first taught point.
6) In job program, if FIX command is not used, it is set to "FIX 1".
7) In job program, in order to return to the previous state that FIX has been applied, use "FIX 1". This time, the command is applied from a command of next 2 steps.



3.35.1 **Programming Example**

Perform interpolation motion, rotating the form of W-axis




```
MAIN
VEL 100
JMOV P0
FIX 0                ..............   Rotate W-axis form
LMOV P2
AMOV P3 P4
LMOV P5
AMOV P6 P7
LMOV P8
AMOV P9 P10
LMOV P11
AMOV P12 P13
FIX 1                ..............   Fix W-axis form
L.MOV P1
EOP
```

3.36 FORM (Forms of robot arm setup command)

Function Set up the form of robot arm during robot operation

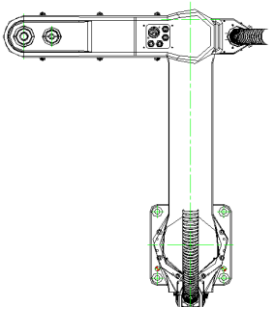
Formation FORM <form>

Term <form> : In a horizontal articulated robot (SCARA), the form of moving arm (A and B-axis) is set up (LEFT, RIGHT, and NO).

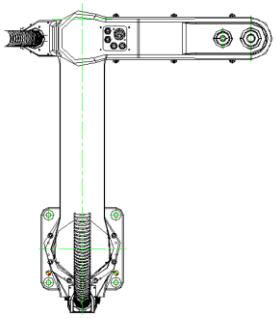
Comments **1) Applied only to a horizontal articulated robot(SCARA)**

2) Set up the form of arm when a robot moves.

3) Forms have three kinds: LEFT, RIGHT, and NO.



FORM RIGHT



FORM LEFT

CAUTION

- Only in JMOV Command, the robot form can be changed.
- A horizontal articulated robot has two forms for one coordinate value (i.e., LEFT FORM and RIGHT FORM).
- Although a point is stored as LEFT FORM, if in program "FORM RIGHT" is designated, the stored form of the point is neglected, and the robot moves to RIGHT FORM.

3.36.1 Programming Example

1) Move to the position coordinate that the point teaching has been performed

```
MAIN
VEL 100
JMOV P10
FORM LEFT ..... With a form of left arm,
JMOV P11          Move to P11
FORM RIGHT ..... With a form of right arm,
JMOV P11          Move to P11
EOP
```

3.37 TRQ (Collision detection command)

Function When the collision is detected, the torque limit alarm is generated.

Formation TRQ <designated axis> <torque limit >

Term <designated axis> : 1 → A(X)-axis, 2 → B(Y)-axis, 3 → Z-axis, 4 → W-axis,
<torque limit > : Range → 1~300[%] (300% means three-fold of the rated torque of a motor)

Comments
1) After the torque limit for each designated axis is set up during robot operation, if the torque more than the set up limit value operates for the time longer than the time set up as the parameter TOL, **Torque Limit Alarm is generated**. (See Parameter Mode.)
2) If the torque limit value is set to 240 or more, the torque value is limited but the alarm is not generated.

3.37.1 Programming Example

```
MAIN
VEL 500
FOS 10
JMOV P0
WHILE 1
LMOV P1
TRQ 1 50 ..... Set the torque limit of A(X)-axis to 50%
TRQ 2 50 ..... Set the torque limit of B(Y)-axis to 50%
LMOV P2
LMOV P1
LMOV P3
ENDWL
EOP
```

If the exterior shock is applied, during moving to P2, P1, and P3, for the time longer than the time set up as the parameter TOL, the alarm is generated.

3.38 TQL (Axis output torque limit command)

Function Limits the maximum thrust torque value of each axis.

Formation TQL <designated axis> <torque limit >

Term <designated axis> : 1 → A(X)-axis, 2 → B(Y)-axis, 3 → Z-axis, 4 → W-axis,
<torque limit > : Range → 1~300[%] (300% means three-fold of the rated torque of a motor)

Comments
1) In general, the instantaneous torque is permitted to be approximately three-fold of the rated torque. However, if this might cause the problem on the durability of a motor or other mechanics, this command is used to limit the maximum torque.
2) If the torque is insufficient and so the robot does not reach a certain position for a certain time, "In Position Error" alarm is generated.

3.38.1 Programming Example

```
MAIN
VEL 500
FOS 10
JMOV P0
WHILE 1
LMOV P1
TQL 1 50 ..... Limit the maximum output torque of A(X)-axis to 50% of the rated value.
TQL 2 50 ..... Limit the maximum output torque of B(Y)-axis to 50% of the rated value.
LMOV P2
LMOV P1
LMOV P3
ENDWL
EOP
```

When moving to P2, P1, and P3, the robot moves with the torque not more than the maximum output torque set up by TRQ.
If the robot cannot move due to the insufficient torque, the alarm is generated.

3.39 INPOS (Arrival at target point designation command)

Function Output the degree of arrival at the target point (In Position) when a robot operates.

Formation INPOS <degree of arrival at the target point>

Term <degree of arrival at the target point> : Set the permission value to check whether the robot arrives at the target point. (Unit : pulse quantity)

Comments
1) Using INPOS command, if all axes do not reach the target point, to the extent of permission, after the robot performs Move command, the next command is not executed.
2) When the difference between the current position value of the robot axis and the targeted taught point lies within the pulse quantity set up by INPOS command, it is recognized that the robot arrives at the destination and the next command is executed.
3) To disable INPOS command, use "INPOS 0" and this is applied from next step command.

▶ When INPOS command is used, the delay may occur in order to search the exact current position, and therefore the cycle time of robot moving may increase.
▶ During FOS operation or WITH execution, INPOS command is not applied to.

3.39.1 Programming Example

1) Interpolation motion using TOOL

```
MAIN
VEL 100
JMOV P10
INPOS 100 ..... Set up the permission value for arrival to 100
JMOV P1         pulses.
                Move P1 and P2 and wait until the current value of
                each axis and the taught value are within 100
                pluses
JMOV P2
INPOS 0   ..... Disable INPOS command
JMOV P3
EOP
```

3.40 MINIT (MAPPING Initialization Command)

Function **Mapping function initialization command**
MINIT command is available for use only in the version that came after Ver 03.00.00.

Format MINIT <Sensor type>

Terms <Sensor type>: Sets a sensor type used for mapping functions.
Set value 1: NC sensor
Set value 2: NO sensor

Description 1) Initializes mapping-related variables inside the controller.
2) Sets a sensor type used for mapping.

CAUTION

- ▶ Mapping functions, combined with 3 commands (MINIT, MSTART, MREAD), perform one function.
- ▶ For an accurate mapping operation, the following procedure should be taken – mapping initialization (MINIT), mapping start (MSTART), mapping data read (MREAD), otherwise a "Latch Sequence Err" alarm occurs.

3.41 MSTART(MAPPING Detection Command)

Function **Mapping start command**
This command is available for use only in the version that came after ROSEP Robot and Ver 03.00.00.

Format MSTART <Axis Assignment> <GP No.> <Sensor Input Pin>

Terms < Axis Assignment > : Assigns an axis to perform mapping

<GP No.>: Allocates the start number of a global point to store mapping data

<Sensor Input Pin> : Allocates the number of a sensor input pin used for mapping

Description 1) Sets a sensor input pin number used in performing mapping, assigned axis and the start point of a global point to store mapping-related data.
2) Saves the number of rising edges and falling edges during mapping operation to the start position of a global point.
3) **The maximum number of mappings detectable is 50.**

CAUTION

To use mapping functions, a sensor input pin number and an axis assignment number should correspond to the identical Servo Module, otherwise mapping functions becomes disabled.

CAUTION

- Mapping functions, combined with 3 commands (MINIT, MSTART, MREAD), perform one function.
- For an accurate mapping operation, the following procedure should be taken – mapping initialization (MINIT), mapping start (MSTART), mapping data read (MREAD), otherwise a "Latch Sequence Err" alarm occurs.

3.42 MREAD(Renewal Command for MAPPING Position Data)

Function **Mapping Data Renewal Command**
This command is available for use only in the version that came after ROSEP Robot and Ver 03.02.08

Format MREAD <Limit Time>

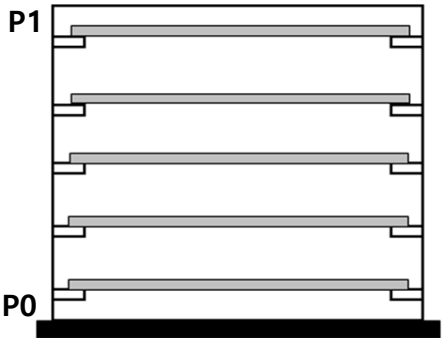
Terms <Limit Time> : Time limited for reading mapping data. (Unit : ms)

Description
1) Saves the positions of data detected from the mapping sensor, in order, from the number coming after the start number of the global point set in MSTART.
2) The minimum time should be set to be **over an expected number of detection point * 20ms** .

3.42.1 Examples of Program Use

1) Measuring Thickness of Cassette Glass

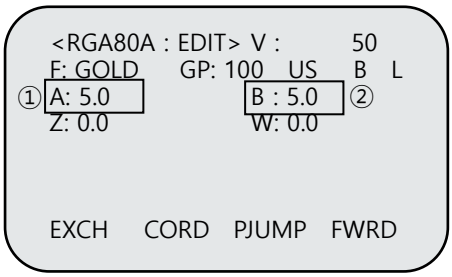
MAIN	.....	Start program
JMOV P0	.....	Move to standby position (P0)
MINIT 1	.....	Initialize mapping
MSTART 3 GP100 3	.....	Select sensor input pin no.3 and Z-axis Start mapping data from GP100
JMOV P1	.....	Move to P1
MREAD 500	.....	Save to mapping data GPNT
EOP		End program



CAUTION

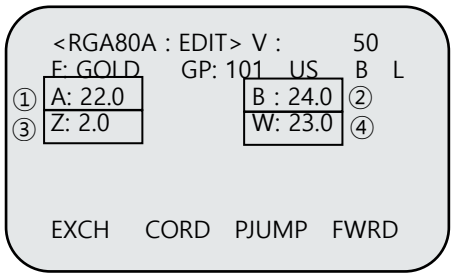
- Mapping functions, combined with 3 commands (MINIT, MSTART, MREAD), perform one function.
- For an accurate mapping operation, the following procedure should be taken – mapping initialization (MINIT), mapping start (MSTART), mapping data read (MREAD), otherwise a "Latch Sequence Err" alarm occurs.

2) Description of how to save mapping data



① Number of rising edges
② Number of falling edges

[Screen for saving number of rising/falling edges]



① Position value in rising edge
② Position value in falling edge
③ Deviation between two positions
④ Average position value between two positions

Above example shows when a cassette has 5 glasses the rising edge number of the sensor 5 is entered in the first axis of GP100, the falling edge number 5 of the sensor in the second axis. Then, mapping is conducted and when facing the MREAD command the position where the first rising edge has occurred is saved to the first axis of GP101, and the position where the first falling edge has occurred to the second axis. The deviations between the position where the falling edge has occurred and the position where the rising edge has occurred are saved to the 3rd axis of GP101, and the mean of the position where the falling edge has occurred and the position where the rising edge has occurred to the 4th axis of GP101. Mapping date is saved from GP101 to GP105 in the same format.

Terms PORT: Select COM Port (1~2)

- 1: Select RS-232 Port
- 2: Select RS-485 Port
- 3: Clear Input, output buffer

Integer: Select clear buffer (1~3)

- 1: Clear input buffer
- 2: Clear output buffer
- 3: Clear input/output buffer

Terms String Variable

Means a variable declared by use of DEFSTR and used to handle strings

PORT: Select COM Port (1~2)

- 1: Select RS-232 Port
- 2: Select RS-485 Port

Integer

- Timeout length .(Unit : ms)
- Waits for a serial input during the length of timeout.
- If data does not come back during the length of timeout, check the timeout status in the system area and perform the command for the next step.

3.43 RSTATE (robot state check command)

Function Read the state of the robot

Formation <integer variable>=RSTATE(channel, STATE INDEX)

Comments

INDEX	STATE NAME	DESCRIPTION
0	ALL	Return all state information
1	ALARM	Return alarming state
2	READY	Return operation ready state
3	ORIGIN	Return origin searching completed state (Always ON, for ABS type)
4	INPOSITION	Arrival at target point completed
5	RUN (JOB operating)	Return JOB RUNNING state
6	SERVO ON	Return SERVO MOTOR ON/OFF state
7	STOP (JOB stopped)	Return JOB STOP state
8	RUN MODE	Manual Run mode
9	SYSTEM MODE	System mode
10	TEACH MODE	Point Teaching mode
11	EMG	On when emergency stop occurs
12	PSEL	JOB LOAD completion
13	Reserved	Reserved area
14	Reserved	
15	MOVING (robot is operating)	Return the state while the relevant channel robot is moving.
16	TIMEOUT	Return the state of String Command Time out
17	1 AXIS MOVING FLAG	Return the state while 1st axis of the relevant channel is moving.
18	2 AXIS MOVING FLAG	Return the state while 2nd axis of the relevant channel is moving.
19	3 AXIS MOVING FLAG	Return the state while 3rd axis of the relevant channel is moving.
20	4 AXIS MOVING FLAG	Return the state while 4th axis of the relevant channel is moving.
21	5 AXIS MOVING FLAG	Return the state while 5th axis of the relevant channel is moving.
22	6 AXIS MOVING FLAG	Return the state while 6th axis of the relevant channel is moving.
23	Reserved	Reserved area
.....	.....	
31	Reserved	

3.43.1 **Programming Examples**

1) Check all the states of CH1 and output each of them to I/O

```
MAIN
INT TEMP
TEMP=RSTATE(1,0)          ..............  Check all states of CH1
OUT10=(TEMP>>5)&0H01      ..............  Output SVON state
OUT9=(TEMP>>4)&0H01       ..............  Output JOB RUN state
OUT8=TEMP&0H01            ..............  Output Alarm state
EOP
```

2) Read individually the states of CH2 and output them to I/O

```
MAIN
OUT10= RSTATE(2,6)        ..............  Output SVON state of CH2
OUT9= RSTATE(2,5)         ..............  Output JOB RUN state of CH2
OUT8= RSTATE(2,1)         ..............  Output Alarm state of CH2
EOP
```

3.44 RERROR (Alarm code check command)

Function Read the robot alarm codes

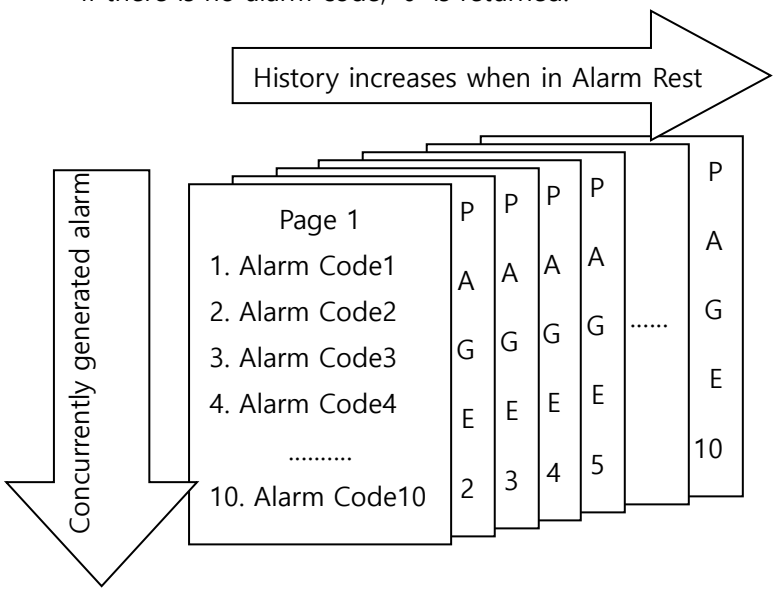
Formation <integer variable>=RERROR (PAGE NO, INDEX)

Comments PAGE NO : Number of alarm history

- Archive number of Alarm History, in which 10 indexes are stored in one page.
- If Page No. is 1, the latest generated history, while No. 10 means the oldest history number.

INDEX: Table number of the alarm concurrently generated.

- The alarm codes which can be concurrently generated are 10 maximum.
- If there is no alarm code, '0' is returned.



3.44.1 Programming Example

1) Alarm Code Read

```
MAIN
INT ERR,ERR1
WHILE 1
ERR=RERROR(1,1)        .....   Check 1st alarm code in Page 1
IF ERR != 0 THEN       .....   If there is Alarm Code,
ERR1=RERROR(1,2)       .....   Check 2nd alarm code in Page 1
ENDIF
ENDWL
EOP
```

3.45 GFTOGP (Command storing GF to GP)

Function Store the value of GF(Global Float) to GP(Global Point) variable

Formation GFTOGP <Global Float index> <Global Point index>

Term Global Float Index : 0~499
Global Point Index : 0~1023

Comments 1) If the starting number of the stored GF variable and the number of GP to be stored are designated, 6 GF data from the starting number are stored in the designated GP.

3.45.1 Programming Example

1) Store the current position of the robot in GP(Global Point) variable

MAIN		
POS CURR		
CURR=HERE	.....	Read the current position coordinate of a robot
F10=CURR.1	.....	1st axis coordinate of the robot is stored in F10.
F11=CURR.2	.....	2nd axis coordinate of the robot is stored in F11.
F12=CURR.3	.....	3rd axis coordinate of the robot is stored in F12.
F13=CURR.4	.....	4th axis coordinate of the robot is stored in F13.
F14=CURR.5	.....	5th axis coordinate of the robot is stored in F14.
F15=CURR.6	.....	6th axis coordinate of the robot is stored in F15.
GFTOGP F10 GP50	.....	Values of F10~F15 is stored in GP No. 50.
EOP		

3.46 REMCMD (SYSTEM COMMAND)

Function System Command

Formation REMCMD <channel> <command>

Comment channel: Channel to execute command

command :

Command	Name	Description
1	Servo ON	Servo ON the relevant channel robot.
2	Servo OFF	Servo OFF the relevant channel robot.
3	PSEL	Load the selected JOB program.
4	START	Start the selected JOB.
5	STOP	Stop the operating JOB.
6	RESET	Clear the alarm.
7	EMG STOP	Emergently stop the robot

3.46.1 Programming Example

1) Power ON for the CH1 JOB operation and the robot operation

```
MAIN
INT CHECK
REMCMD 1 3 ..... Load the JOB file to CH1.
CHECK=RSTATE(1,12) ..... Check the completion state of the selected JOB
IF CHECK==1 THEN
REMCMD 1 1 ..... Servo ON the CH1 robot
CHECK=RSTATE(1,6) ..... Check the state of CH1 SERVO ON/OFF
IF CHECK==1 THEN
REMCMD 1 4 ..... Run the CH1 JOB.
ENDIF
ENDIF
EOP
```

3.47 Variables

3.47.1 Types of variable

Variables include general variable, position variable, and system variable.

		LOCAL	GLOBAL
General variable	Integer type	INT	I (number)
	Real number type	REAL	F (number)
Position variable		POS, P (number)	GP (number)
System variable		CNT, TMR, MVR, HERE	

- LOCAL variable – Variable can be used only within the program currently run
  1) INT : Integer variable, which is used for arithmetic operation, temporary saving I/O data, etc.
  2) REAL : Real number variable, which is used for arithmetic operation, point data operation, etc.
  3) POS : Position variable, which is divided into the simple variable and array variable
  4) CNT : Up-count the pulse input coming through the input port, and save it.
  5) TMR : Increase the starting screen value by 1 according to the parameter setup value
  6) MVR : In the case of moving between two points, the percentage value of the current position versus the total distance
  7) HERE : Variable having the angle value of the current axis.
  8) P(number) : Variable having the position value taught by a user. (0 ≤ No. ≤ 1999)

- GLOBAL variable – Variable can be used commonly in the whole program and channels
  1) I : Integer variable, which has the same functions as LOCAL variables declared as INT. (0 ≤ No. ≤ 499)
  2) F : Real number variable, which has the same functions as LOCAL variables declared as REAL. (0 ≤ No. ≤ 499)
  3) GP : Variable having the position value taught by a user. (0 ≤ No. ≤ 1023)

3.47.2 How to use

- Location
  - General variable must be ***declared at the place next "MAIN"*** which is a portion of program start.
  - It cannot be declared at the place other than the sub-program(Subroutine) or the program start.
- Composition
  - Variable **name** is to be a text string of 8 letters consisting of alphabets and numbers
  - First letter of variable must be **English alphabet or '_'**. (Numerals cannot be used.)
    However, **"P", "GP", "I", and "F"** cannot be used because they are duplicate with Point variables and GLOBAL variables.
- Initializing
  - Variable **must be initialized** before use.
  - CNT and TMR variable must be **initialized at the previous step before using the variable.** .
  - MVR variable is **initialized to 0 when MOVE command is started.**

3.48 Integer-type (INT, I, II) & Real number-type (REAL, F) variable

3.48.1 LOCAL variable

Function Declare the integer and real number variable usable in the program

Formation INT <variable name>, <variable name>, ---
REAL <variable name>, <variable name>, ---

Term <variable name> :
- Text string of 8 letters consisting of alphabets and numbers
- Fist letter of the variable must be English alphabet or '_'.

Comments 1) They must be declared at the **start of MAIN block**, and cannot be declared in the sub-program.
2) INT-declared variable is the integer variable (Range of integer value : -2.14×10^9 ~ 2.14×10^9)
3) REAL-declared variable is the real number variable (Range of real number value : $\pm 8.4 \times 10^{-37}$ ~ $\pm 3.4 \times 10^{38}$)

CAUTION

▶ **"P", "GP", "I", "F" cannot be used** because they are duplicate with Point variables and GLOBAL variables. (**"Syntax Error" or "Duplicated Symbol"** ALARM arises.)

▶ If the operation of real number variable and that of integer variable are mixedly used, **the operation is changed to integer-type**.
9.5 + **10** = 19 **(When real number and integer is operated, error is produced.)**
9.5 + **10.0**=19.5 **(Changed to the operation of Real number + Real number)**

3.48.2 Programming Example

1) Use of integer and real number variables

```
MAIN
INT J,BOUT          ..............  Integer variable J and BOUT are declared.
REAL FF1            ..............  Real number variable FF1 is declared.
WHILE 1
BOUT=0H000F
J=1                 ..............  Initialize the variable
FF=10.5
FOR J=0 TO 10       ..............  Move to AP1 position coordinate
JMOV P1             ..............  Move to AP2 position coordinate
JMOV P2
POUT0=BOUT          ..............  Output, using integer variable BOUT
NEXT
FF1=FF1+100.0                       Arithmetic operation of the real number
                                    variable
ENDWL
EOP
```

3.48.3 Integer-type GLOBAL variable I, II

Function Variables having the same functions as the INT-declared integer variable, and commonly usable in the whole program.

Formation I<integer> : Range of the usable integer is 0~499.

II<integer> : Range of the usable integer is 0~499.

→ Between I and <integer> and also II and <integer>, blanks cannot be inserted.

→ Between I and (integer) and also I and (integer variable), blanks cannot be inserted.

Comments 1) Unlike the LOCAL variable, this variable is **used without declaration** in the program.

2) II<integer> is equal to using **I<integer> duplicate**.
E.g.) If I35 is 20, II35 is equal to I20.

3) In the program, the values of I<integer> or II<integer>, I(integer), and I(variable) can be directly changed.

4) Usable for VEL, ACC, DEC, FOS, PFOS, PLUP, DLAY, and INPOS.
E.g.) VEL I20 : Velocity is set up to the value stored in I20.

5) Able to be used in IF - (ELSE) - ENDIF block and WHILE-ENDWL block.
E.g.) IF I7 > A0 THEN E.g.) WHILE II49 ≤ 100
※ **But, cannot be used in FOR-NEXT block.**

6) I<integer> type can be used for Point Global variable(Global Point variable).
E.g.) If I9 is 3, PI9 is equal to P3.
※ **PII<integer> type cannot be used.**

3.48.4 Programming Example

```
MAIN
INT L,M,H          ..... Integer variable L, M, and H are declared.
L=I0               ..... Assign L=300
M=I1               ..... Assign M=600
H=I2               ..... Assign H=1000
VEL H              ..... Designate the velocity to H(1000)
JMOV P0
WHILE I10<I11      ..... While I10<I11 is satisfied,
FOS I4             ..... FOS I4 => FOS 10
LMOV PI25          ..... CP move to P(I25)
VEL L              ..... Designate the velocity to L(300)
AMOV PI26 PI27     ..... Arc interpolation by way of P(I26) and P(I27)
LMOV PI28          ..... CP move to P(I28)
I25=I25+10
I26=I26+10         ..... Move 1 time, and thereafter increase the values of I25, 26, and 27 by 4, respectively.
I27=I27+10
I10=I10+1          ..... Increase counter I10 by 1
ENDWL
EOP
```

I variable value

I0 = 300
I1 = 600
I2 = 1000
I10 = 1
I11 = 4
I25 = 1
I26 = 2
I27 = 3
I28 = 4

3.48.5 Real number-type GLOBAL variable F

Function Variable having the same functions as the REAL-declared real number variable, and usable commonly in the whole program.

Formation F<integer> : Range of the usable integer is 0~499.

→ Between F and <integer>, blanks cannot be inserted.

Comments **1)** Unlike the LOCAL variable, this variable is **used without declaration in the program.**

2) In the program, the values of F<integer> can be directly changed.

3) Able to be used in IF - (ELSE) - ENDIF block and WHILE-ENDWL block.

E.g.) IF F7 > A0 THEN E.g.) WHILE F49 ≤ 100

※ **But, cannot be used in FOR-NEXT block.**

CAUTION

▶ If the operation of real number variable and that of integer variable are mixedly used, **the operation is changed to integer-type**.

9.5 + **10** = 19 **(When the operation of real number and integer is performed, error is produced.)**

9.5 + **10.0**=19.5 **(Changed to the operation of Real number + Real number**

3.48.6 Programming Example

```
MAIN
POS XP
XP=P1
FOR I0=0 TO 10
XP1.1=F0          ..............  Substitute the F0 value for X-axis (1st axis) of position-
                                  type variable XP1
LMOV XP           ..............  CP move to the position of XP
F0=F0+10.0        ..............  Increase F0 value by 10.0 every once time
NEXT
EOP
```

3.49 POSITION variable

3.49.1 POS variable

Function position-type variable is declared. (Simple variable and Array variable)

Formation Simple variable → POS <variable name>, <variable name>, ...
Array variable → POS <variable name>(size), {<variable name>(size)}, ...

Term <variable name> :

- Variable name is the size of text string and array, composed of 8 letters of English **alphabet and numerals.**
- The first letter of variable must be **English alphabet**. (excepting single use of "P", "GP", "I", and "F")

(size) : The size of the array variable is represented the numeral in parentheses (integer)..

Numeral in parentheses of the declaration of the array variable denotes the number of rooms.

E.g.) When POS AA(3) is declared – Usable variables are three, i.e., AA(0), AA(1), and AA(2).

Comments

1) **Declaration must be done at the start of MAIN block,** and cannot be done in the sub-program.
2) POS-declared variable value is the angle of each axis, read value or Cartesian coordinate value.
   - Variable value **starting with X** (E.g.: XA, XA1, ...) → Cartesian coordinate value variable
   - Variable starting with letters other than **X** → angle of each axis, read value variable, and increment variable of each axis
3) **Each component value of** POS-declared variable can be used as **<variable name>.<number>**.
   E.g.) POS XP1 → XP1.1 XP1.2 XP1.3 XP1.4...
4) The angle value POS variable has **maximum 6 kinds of components**, and can be initialized in the formation of
   variable name(Simple variable, Array variable)=
   <1-axis component, 2-axis component, 3-axis component, 4-axis component, 5-axis component, 6-axis component>.
5) The Cartesian coordinate value POS variable has **maximum 7 kinds of components**, and can be initialized in the formation of
   variable name(Simple variable, Array variable)=<X-axis component, Y-axis component, Z-axis component, 4-axis component, 5-axis component, 6-axis component, Arm Form value>. → This is possible only when the mechanism is

SCARA robot.

- Arm Form value : 0 → LEFT form, 1 → RIGHT form, 2 → NO form

6) The angle value POS variable and the Cartesian coordinate value POS variable can be operated and designated with each other, and at this time, they are converted according to the value being stored.

CAUTION

- **"P", "GP", "I", "F" cannot be used** because they are duplicate with Point variables and GLOBAL variables. (**"Syntax Error" or "Duplicated Symbol"** ALARM arises.)
- **If there is no arm form in the component value, it is judged as the angle value, and is converted to XY coordinate system when it is stored in XP variable.**
- No. 5 and 6 axes **are unused** axes by taking the extensibility into consideration. **0.0 value is to be entered.**

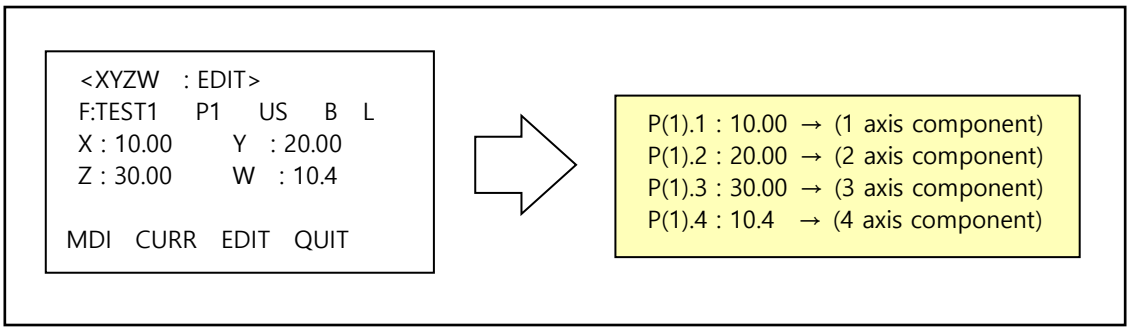
3.49.2 POINT variable

Function Use all points that a user taught in Job Editor

Formation LOCAL variable → P(number) (0 ≤ No. ≤ 1999)
GLOBAL variable → GP(number) (0 ≤ No. ≤ 1023)

Comments 1) Unlike the POS variable, it is not necessary to declare the variable at first.

2) **Each component value of** POINT variable can be used as **<Point number>.<number>**.



3.49.3 Programming Example

1) Application of the decelerated point, using POINT variable and POSITION variable

```
MAIN
POS MM,XA,XB                ..... Position-type variable XA, MM, and XB are declared.
MM=<10.1,10.2,10.3,10.4,0,0>  ..... MM initialization
XA=<<400.0,50.0,10.0,0.0,0.0,0.0,1>  ..... XA initialization
XB=XA                       ..... Initialize XB to be equal to XA value
XB.3=XA.3-40.0              ..... Subtract 40mm from Z(3rd axis) value of XB
VEL 1000
JMOV P0                     ..... PTP move to P0
LMOV XB                     ..... CP move to XB position coordinate (decelerated point)

VEL 300
LMOV XA                     ..... CP move to XA position coordinate
IMOV2 MM                    ..... Incremental move by position-type variable MM value

VEL 1000
JMOV P0
EOP
```

CAUTION

▶ **Component value is converted and stored, depending on the variable name.**
XP=<angle value>: Converted to Cartesian coordinate system and stored.
XP=<Cartesian coordinate value>: Stored as the component value is.
AP=<angle value>: Stored as the component value is.
AP=<Cartesian coordinate value>: Converted to the angle value and stored.

3.50 System variables (CNT, TMR, MVR, HERE)

3.50.1 CNT & TMR variable

Function CNT : Counter value is stored.
TMR : Timer value is stored.

Formation CNT<pulse input BIT number>=<starting value>
TMR0=<starting value>
TMR1=<starting value>

Term <pulse input BIT number> : Input BIT number to be received as an input
<starting value> : Integer value that counter(CNT) and timer(TMR) start

Comments 1) System variables are used **without declaration of their type**.
2) The moment the **pulse input BIT is determined, "0" is stored in CNT variable**, and since then, the CNT variable counts **every pulse input (pulse of 10m or more)**. (0 ≤ input range ≤ 65,535)
3) From the moment the **integer value is input to TMR variable,** a value is assigned to the TMR, and since then it is increased '1' by '1', with time interval defined in the system parameter.
For application of the setup value, see TMR in Parameter Mode.

CAUTION

▶ Be careful that, between CNT and pulse input port number and between TMR and "0" or "1", the blank cannot be inserted.
E.g.) CNT_0=0 (X), TMR_1=-100 (X)

3.50.2 Programming Example

1) Use of CNT and TMR variable

```
MAIN
VEL 100
CNT0=0          ..............  Initialize counter
TMR1=-100       ..............  Initialize timer
WHILE CNT0<20   ..............  LOOP until 20 inputs are input to BIT "0"
JMOV P0
IF(TMR1>0) THEN ..............  If the timer is 1 sec. or higher,
GOTO TEST                      (-100 ~> 0 : 100 pieces of 1ms are produced)
ENDIF
DLAY 100
JMOV P1
ENDWL
LABL TEST
EOP
```

3.50.3 MVR variable

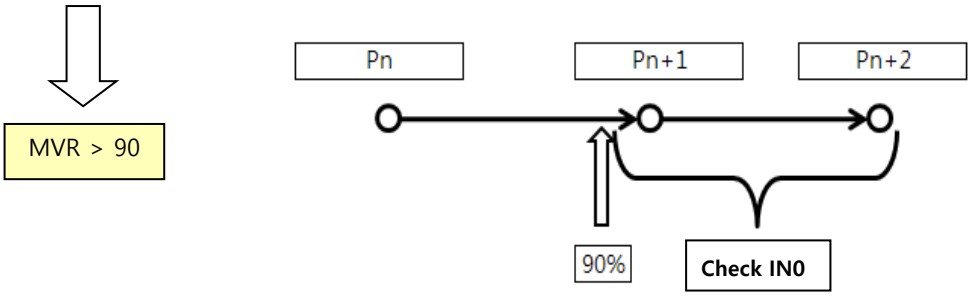
Function Ratio of the moving distance to the total distance is stored

Formation MVR <percentage value of the moving distance to the total distance>

Comments
1) Percentage value of the moving distance to the total distance, for movement between two points (Pn, Pn+1).
   I.e., MVR = {(Moving distance)/(Total distance)}*100
2) This is used for conditional branch and I/O parallel processing while in robot operation
3) This must be used along with WITH statement by dividing the traveling interval by 100 units.
4) Pay attention to MVR value setup when the moving distance between two points (Pn, Pn+1) is short and when the robot is moved at high velocity.

CAUTION

▶ **Checking the input "IN1" at the time of 90% after, while moving from Pn to Pn+1**



- When a robot reaches about 90% before arriving at Point Pn+1, the computation for moving to next Point Pn+2 starts.
- That is, as shown in the above figure, even through the robot is at 90% position, **#1point is now being computed.**

★ Therefore, when in short moving distance and high speed moving, the state of the input Bit ' 0' may not sometimes be checked after 90% after having started from Pn. At this time, try to lower the robot moving velocity..

CAUTION

Use of function of reading the current coordinate (HERE command) when a signal is input while a robot is moving

- When using a function of reading the current coordinate when a signal is input during robot motion and it is stopped, there may arise the deviation in the read value depending on the robot moving velocity.
- At this time, try to lower the robot moving velocity.

3.50.4 **Programming Examples**

1) Correction of moving trajectory in response to the state of the input bit while in moving.

```
MAIN
VEL 100
FOS 10
JMOV P100
MVR=0
WITH
JMOV P0
WHILE MVR<100          ..............  During 100% moving to P0 Point
IF MVR>10 THEN         ..............  At the place where the moving distance is 10%
OUT1=1 50              ..............  Output is retained ON(1) for 0.5 sec.
ENDIF
IN1=1
GOTO L1
ENDWL
LABL L1
STOP
ENDWT
JMOV P1
EOP
```

2) Move to P2 Point after 80% distance from P1 Point, in response to the signal input to input "IN0"

```
MAIN
VEL 100
WHILE IN1==1
PLUP 10
JMOV P0
MVR=0
WITH
JMOV P1
WHILE MVR<100          ..............  During 100% moving to P1 Point
IF (MVR>80)&&(IN0==1)  ..............  If moving distance is 80% and IN0 is ON(1)
THEN
PLUP 0
JMOV P22
GOTO BB
ENDIF
ENDWL
ENDWT
IN0=1
PLUP 0
JMOV P2
LABL BB
ENDWT
OUT1=1 50
ENDWL
EOP
```

3.50.5 HERE variable

Function The current position value is stored

Formation HERE
HERE<integer>

Term <integer> : channel number (1 or 2)
- Used to read the robot position between channels.

Comments **1) The current position value of the robot is stored as the angle value or the read value.**
2) The current position of another channel robot is stored.

3.50.6 Programming Examples

1) Read the current position, using HERE variable

```
MAIN
VEL 100
POS CURR
JMOV P0
MVR=0
WITH
JMOV P1
WHILE MVR<100
IF MVR>50 THEN                    At the point of 50% during moving to P1 Point
CURR=HERE              .....      Store the current position value in position-type
                                  variable CURR
GOTO LLL
ENDIF
ENDWL
LABL LLL
ENDWT
JMOV P2
JMOV CURR
EOP
```

2) Read the current position of another channel robot, using HERE<n> variable

```
MAIN
VEL 100
POS AP
JMOV P0
WHILE AP.1<10.0
AP=HERE2               .....      Store the No. 2 channel robot
DLAY 10
ENDWL
OUT0=1
JMOV P1
EOP
```

3.51 Constant

Function Representation of hexa-decimal and binary number.

Formation 0H<numeral>

0B<numeral>

Term <numeral> : If there is not 0H or 0B in front of the numeral, it is a decimal number

* 0H : hexa-decimal

* 0B : binary

Comments 1) 0H<numeral> is a hexa-decimal, and the numeral can be used up to 8 places.

2) 0B<numeral> is a binary, and the numeral can be used up to 16 places (16 bits).

3) If there is not 0H or 0B, it is a decimal integer or real number.

3.51.1 Programming Example

```
MAIN
INT AA
VEL 100
WHILE 1
AA=PIN0 & 0H00FF                Compare the value of input port0 with hexa-decimal 00FF
IF AA==1 THEN
POUT0=0B00110100                Output binary 00110100 to output port0
JCALL MD1
ENDIF
IF AA==1 THEN
POUT0=0B01110100                Output binary 01110100 to output port0
JCALL MD2
ENDIF
ENDWL
EOP
```

3.52 Operators

3.52.1 Assignment operator

Command	Function
=	Evaluate the right-side equation and convert the result to be proper to the left-side variable type, and thereafter assign the converted. (E.g.: IN0=1)

	Integer type	Real number type	Position type	Counter type	Timer type
Integer type	Integer type	Integer type	**Improper**	Integer type	Integer type
Real number type	Real number type	Real number type	**Improper**	Real number type	Real number type
Position type	**Improper**	**Improper**	Position type	**Improper**	**Improper**
Counter type	Integer type	Integer type	**Improper**	Integer type	Integer type
Timer type	Integer type	Integer type	**Improper**	Integer type	Integer type

3.52.2 Arithmetic operator

Command	Function
*, /, +, -, %	Operation results of each operator for different data types are shown in the table below. In the table, 'Improper' means the **inoperable equation**.

▶ **'+' & '-' operator**

	Integer type	Real number type	Position type	Counter type	Timer type
Integer type	Integer type	Real number type	**Improper**	Integer type	Integer type
Real number type	Real number type	Real number type	**Improper**	Real number type	Real number type
Position type	**Improper**	**Improper**	Position type	**Improper**	**Improper**
Counter type	Integer type	Integer type	**Improper**	Integer type	Integer type
Timer type	Integer type	Integer type	**Improper**	Integer type	Integer type

▶ *** operator**

	Integer type	Real number type	Position type	Counter type	Timer type
Integer type	Integer type	Real number type	Position type	Integer type	Integer type
Real number type	Real number type	Real number type	Position type	Real number type	Real number type
Position type	Position type	Position type	**Improper**	Position type	Position type
Counter type	Integer type	Real number type	Position type	Integer type	Integer type
Timer type	Integer type	Real number type	Position type	Integer type	Integer type

▶ **/ operator**

	Integer type	Real number type	Position type	Counter type	Timer type
integer type	Integer type	Real number type	**Improper**	Integer type	Integer type
real number type	Real number type	Real number type	**Improper**	Real number type	Real number type
위치 type	Position type	Position type	**Improper**	Position type	Position type
Counter type	Integer type	Real number type	**Improper**	Integer type	Integer type
Timer type	Integer type	Real number type	**Improper**	Integer type	Integer type

▶ **% operator**

This is possible for only integer-to-integer operation. In using this operator, perform division operation, **discard the quotient and take the remainder as a result value.**

E.g.) If AA=10%3, **the value of AA is 1**

3.52.3 Relational operator

Command	Function
>, <, ≤, ≥, ==, !=	- These are mainly for integer type and real number type data, which are used in conditional commands, such as IF statement and WHILE statement. - The result of the relational operation is a logical value (true/false, or 0/1).

3.52.4 Logical operator

Command	Function
&&, ll, !	Logical operator is only for the logical value. That is, it uses the result of the relational operation as an operand. The result of logical operation is a logical value.

3.52.5 Bit operator

Command	Function
&, I	Logical operator
<<, >>	Shift operator
~	1's complement

3.53 Built-in Functions

Function name	Description	Use example
EXP(X)	Exponential e^x	EXP(3) is 20.085
LOG(X)	Common logarithms $\log_{10}X$	LOG(100) is 2
LN(X)	Natural logarithms $\log_eX$	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 of integer K	ABS(-12) is 12
RND(X)	Round the real number X to make the integer value	RND(97.62) is 98
SIN(X)	Sine; unit of X : radian	SIN(RAD(30)) is 0.5
COS(X)	Cosine; unit of X : radian	COS(0) is 1
TAN(X)	Tangent; unit of X : radian	TAN(RAD(45)) is 1.0
ASIN(X)	Arcsine; Return the value of - π/2~π/2	DEG(ASIN(0.5)) is 30
ACOS(X)	Arccosine; Return the value of 0 ~ π	DEG(ACOS(0.5)) is 60
ATAN(X)	Arctangent; Return the value of -π/2 ~π/2	DEG(ATAN(1.0)) is 45
ATAN2(Y,X)	2nd arctangent; Return the value of -π~π	DEG(ATAN2(-1,-1)) is -135
DEG(X)	Convert radian into angle value	DEG(3.1416) is 180.0
RAD(X)	Convert angle into radian value	RAD(180.0) is 3.1416

3.54 ASC

Function Return the first letter of text string as a character code

Formation integer variable = ASC(text string)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".
- EX) "ABCDEF"

character code

- Numerals (integer) relevant to the characters.
  Ex) Character 'A' is integer(65), and character 'B' is integer(66)입니다.
- See ASC code table.

Comments **1)** First letter of the text string input to ASC is returned.

3.54.1 Programming Example

```
MAIN
DEFSTR  AA
INT BB
AA = "XYZW"  ..........  Substitute "XYZW" for text string variable AA
BB = ASC(AA)  ..........  In text string variable AA, return the first letter "X" to character
EOP                      code(88)
```

3.55 CHR

Function Convert the integer in range of ASC code into the character

Formation text string variable = CHR(integer)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

integer

- Integer(0~127) beloing to ASC code value

Comments **1)** Return ASC code relevant to the integer input to ASC.

3.55.1 Programming Example

```
MAIN
DEFSTR  AA
AA = CHR(65)        ..........  Substitute "A" relevant to ASC code 65 for text
EOP                             string variable AA
```

3.56 FLUSH

Function Clear the input and output buffers

Formation FLUSH(integer)

Term integer: clear buffer selection(1~3)

- 1: Clear input buffer
- 2: Clear output buffer
- 3: Clear input and output buffer

Comments **1)** Clear I/O buffer used for transmission and reception in serial communication.

3.56.1 Programming Example

```
MAIN
DEFSTR  AA
AA = STRIN(1000)  ..........  Receive the input via serial communication and
                              store it in text string AA
FLUSH(1)          ..........  Clear input buffer
EOP
```

3.57 FTOS

Function Convert the integer or real number into the text string

Formation text string variable = FTOS(integer or real number)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

integer or real number

- Integer or real number to be converted into the text string

Comments **1)** Convert the numerical data of integer or real number into the text string.
EX) FTOS(1234) → "1234"

3.57.1 Programming Example

```
MAIN
DEFSTR   AA,BB
AA = FTOS(1234)        ..... Convert 1234 into text string "1234" and substitute it for
                             text string variable AA
BB = FTOS(-123.456)    ..... Convert -123.456 into text string "-123.456" and
                             substitute it for text string variable BB
EOP
```

3.58 HTOS

Function Convert the integer to the hexa-decimal text string

Formation text string variable= HTOS(integer)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

integer

- Integer to be converted into the hexa-decimal text string

Comments **1)** Convert the integer input to HTOS into the hexa-decimal text string.

3.58.1 Programming Example

```
MAIN
DEFSTR  AA
AA = HTOS(10)        ..... Convert the integer 10 into the hexa-decimal text
                           string("A") and substitute it for text string AA
EOP
```

3.59 SLEFT

Function Extract the left section from the input text string

Formation text string variable = SLEFT(text string, integer)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

text string

- Means the text string variable or the text string constant.
- The text string constant is a set of characters surrounded by " "

integer

- The number of letters to be extracted

Comments **1)** Extract from the input text string as many as the input integer from the left of the text string

3.59.1 Programming Example

```
MAIN
DEFSTR  AA,BB
AA = "XYZW'          ..........  Substitute "XYZW" for text string variable AA
BB = SLEFT(AA,2)     ..........  Extract the left two letters ("XY") from the text string
                                 variable AA.
EOP
```

3.60 SLEN

Function Return the length of the input text string

Formation integer variable=SLEN(text string)

Term text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".
- EX) "ABCDEF"

Comments **1)** The length of the input text string is returned.

3.60.1 Programming Example

```
MAIN
DEFSTR  AA
INT LL
AA = "XYZW"          .....  Substitute "XYZW" for text string variable AA
LL = SLEN(AA)        .....  Substitute the length(4) of the text string variable AA for
                            integer variable LL
EOP
```

3.61 SMID

Function Extract the text string from the position designated to the text string, as many as the number of digits

Formation text string variable = SMID(text string, designated position, integer)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".

designated position

- Start position that the text string will be extracted. Start of the text string is 0.

integer

- The number of letters to be read from the designated position

Comments **1)** Extract letters from the designated position, as many as the integer that a user entered.

3.61.1 Programming Example

```
MAIN
DEFSTR   AA, BB
AA = "ABCDEFG"        .....   Substitute "ABCDEFG" for text string variable AA
BB = SMID(AA,2,3)     .....   Return 3 letters ("CDE") from the second left place of the
                              text string variable AA
EOP
```

3.62 SPOS

Function Return a starting place in the text string1, which matches with the text string2

Formation integer variable = SPOS(text string1, text string2)

Term text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".
- EX) "ABCDEF"

Comments **1)** Return a starting place in the text string1, which matches with the text string2.

3.62.1 Programming Example

```
MAIN
DEFSTR   AA, BB
INT PP
AA = "XYZW"          ..... Substitute "XYZW" for text string variable AA
BB = "Z"             ..... Substitute "Z" for text string variable BB
PP = SPOS(AA, BB)    ..... Return a place in text string variable AA, matching with
                           BB
EOP
```

3.63 SRIGHT

Function Extract the right section from the input text string

Formation text string variable = SRIGHT(text string, integer)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".

integer

- The number of letters to be extracted

Comments **1)** Extract from the input text string as many as the input integer from the right of the text string

3.63.1 Programming Example

```
MAIN
DEFSTR   AA, BB
AA = "ABCDEF"          .....   Substitute "ABCDEF" for text string variable AA
BB = SRIGHT(AA,2)      .....   Return the right two letters ("EF") of text string variable
                               AA
EOP
```

3.64 STRIN

Function Read the text string coming via serial communication

Formation text string variable = STRIN(integer)

Term text string variable

- Means the variable declared by using DEFSTR, and is used to handle the text string.

integer

- Timeout time (Unit: ms)
- Wait the serial input for timeout time. If the data does not come till the timeout time, the timeout state in a system area is checked and the next step command is executed.

Comments **1)** Read the input coming via serial communication, waiting for user defined time..

3.64.1 Programming Example

```
MAIN
DEFSTR  PACKET
INT LEN, OVERTM
PACKET = STRIN(1000)        ..... Text string input via serial communication (If no input, wait for 1 sec)
LEN = SLEN(PACKET)          ..... Store the length of the text string input
SS1=SYS1
OVERTM=(SS1>12)             ..... Read whether timed out
&0H01
IF OVERTM==0&&LEN>0         ..... When it is not timed out and the input text string exists
THEN
.....
ENDIF
EOP
```

3.65 STROUT

Function Output the text string via serial communication

Formation integer variable = STROUT(text string)

Term text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".

Comments **1)** Transmit the input text string via serial communication, and return the number of untransmitted letters
EX) When transmitting "ABCDEFGH", if successful, the return value is 0; if no data have been transmitted, the return value is 8

3.65.1 Programming Example

```
MAIN
DEFSTR   SENDPK
INT STRRET
SENDPK="ABCDEF"          .....   Substitute "ABCDEF" for SENDPK text string variable
STRRET=                  .....   Output via serial communication port the text
STROUT(SENDPK)                   string("ABCDEF")
IF STRRET !=0 THEN       .....   Check the transmission fail
.....
ENDIF
EOP
```

3.66 SVAL

Function Convert the text string into numerals

Formation (integer variable or real number variable) = SVAL(text string)

Term text string

- Means the text string variable or the text string constant. The text string constant is a set of characters surrounded by " ".

Comments **1)** Convert the input text string into integer or real number data.

3.66.1 Programming Example

```
MAIN
INT AI
REAL BR
AI=SVAL("1234")        .....   Convert text string "1234" into integer 1234
BR=SVAL("12.234")      .....   Convert text string "12.234" into real number 12.234
EOP
```

Rev.	Revised Date	Description	Reviser	S/W Version
V.1	**2012.07.30**	**First Edition Printed**		
V.2	**2012.12.31**	**Add content**	**Kimjs**	**Ver3.02.08**

N1 ROBOT CONTROLLER

CONTROLLER MANUAL

FIRST EDITION JULY 2012

ROBOSTAR CO, LTD

ROBOT R&D CENTER