로보스타 제어기

T Series

프로그래밍 설명서

- ☐ 제어기 설명서
- ☐ 조작 및 운용 설명서
- ☑ 프로그램밍 설명서
- ☐ 유니호스트 설명서
- ☐ 알람코드 설명서

(주) 로보스타

Robostar

www.robostar.co.kr

목차

제1장 개요

1.1 개요

i. 로봇언어는 Robostar에서 만든 전용 언어로 시스템 내에서 로봇 작업 프로그램을 작성할 때 사용하는 명령어 입니다.

ii. 2. 로봇 언어는 크게 4개의 그룹으로 분류됩니다.

- FLOW 그룹
  프로그램의 조건 또는 무조건 분기, 반복 횟수, 부 프로그램 호출 등을 설정하는 명령어로 구성되어 있습니다.
- MOVE 그룹
  로봇의 동작 조건을 설정하는 명령어로 구성되어 있습니다.
- I/O 그룹
  외부 입,출력을 제어(16bit, 1bit)하는 명령어로 구성되어 있습니다.
- COND(=Condition) 그룹
  속도, 지연시간 등을 설정하는 명령어로 구성되어 있습니다.

작업 용도에 맞는 적절한 명령어를 사용하여 원하는 작업 프로그램을 작성 하십시오.

제2장 명령어 일람

2.1 명령어 일람표

2.1.1 프로그램 제어 관련 명령어

명령어	기 능	형 식	사용 예
INCLUDE	JOB 포함	include 〈job 이름〉	Include "aa"
FUNC	함수 선언	func void main() : end	func void main() vel 20 jmov lp[1] jmov lp[2] end
VOID	함수의 리턴값이 없음		
END	함수 종료		
FOR	스텝반복수행	for 〈변수〉=〈초기값〉 to 〈종료값〉 [step〈증분량〉] : end	: for a=1 to 5 step 2 vel 20 jmov lp[1] end :
TO			
STEP			
END			
WHILE	조건반복수행	while〈조건식〉 : end	: while in(0)==1 jmov lp[1] jmov lp[2] end :
END			
IF	조건판단수행	if〈조건식〉 : (elif 〈조건식〉) : (else) : end	: if in(3)==1 goto A0 elif in(4) == 1 goto B0 else jmov lp[1] end :
ELIF			
ELSE			
END			
LABL	분기점 위치 지정	labl 〈레이블명〉 : goto 〈레이블명〉	: labl A1 jmov lp[1] jmov lp[2] goto A1 :
GOTO	수행문 분기		
CONTINUE	반복문의 처음으로 되돌림	while 〈조건식〉 : continue end	while 1 if in(1) == 1 continue end jmov lp[0] end

명령어	기 능	형 식	사용 예

BREAK	반복문을 종료함.	while 〈조건식〉
		  :
		  break
		end
			while 1
			  if in(1) == 1
			  break
			  end
			  jmov lp[0]
			end

RETURN	함수를 종료함.	func int main()
		  :
		  return 0
		end
			func int add(int a, int b)
			  int c
			  c = a + b
			  return c
			end

STOP	로봇 동작 정지	stop	
EXIT	JOB 수행 정지	exit	

2.1.2 로봇 동작 관련 명령어

명령어	기 능	형 식	사용 예
JMOV	현 위치에서 목표점으로 PTP이동	jmov 〈포인트 변수〉, 〈FOS〉, 〈VEL〉, 〈ACC〉, 〈DEC〉 ▶ FOS, VEL, ACC, DEC,는 생략 가능	func void main(0 vel 20 jmov lp[1] jmov lp[2] end
SVON	서보 ON	svon(로봇) → 모든 축 해당 svon(로봇, 지정 축) → 1(X), 2(Y), 3(Z), 4(W)	svon(1) svof(2, 2) svon(2, 2)
SVOF	서보 OFF	svof(로봇) → 모든 축 해당 svof(로봇, 지정 축) → 1(X), 2(Y), 3(Z), 4(W)	
WITH	로봇 동작 중 다음 수행문 열을 동시처리	with : end	: with jmov lp[1] mvr=0 while mvr〈60 if in(1)==1 out(0)=1 end end end :
END			

2.1.3 입출력 관련 명령어

명령어	기 능	형 식	사용 예
IN	1비트 단위로, 지정된 비트 번호의 ON(=1), OFF(=0) 값을 읽는다.	〈변수〉=in(입력비트 번호) -지정된 비트 입력 상태 값을 변수에 저장	int aa aa=in(0)
PIN	포트 단위로, 지정된 입력 포트의 값을 읽는다.	〈변수〉=pin(입력포트 번호) 〈입력포트번호〉: 0 ~ 2 중 선택 0 → IN0 ~ IN15 1 → IN16 ~ IN31 2 → IN32 ~ IN47	int aa aa=pin(0)
OUT	1비트 단위로, 지정된 비트번호의 출력을 ON(=1), OFF(=0)	out(출력비트번호)=〈0/1〉, [펄스유효시간], [→] [펄스유효시간] 단위 : 10ms [펄스유효시간]이 없으면 : 출력신호 계속 유효 [펄스유효시간]이 지나면 이전상태로 복귀 [→] : 펄스유효시간을 주기로 갖는 주기파형 출력	out(0)=1 out(0)=1, 100 out(0)=1, 100, →
POUT	포트 단위로, 지정된 포트로 지정된 값을 출력한다.	pout(출력포트번호)=〈출력지정 값〉 〈출력포트번호〉: 0 ~ 2 중 선택 0 → OUT0 ~ OUT15 1 → OUT16 ~ OUT31 2 → OUT32 ~ OUT47	pout(0)=0 pout(0)=20 pout(1)=0H000F
BIN	바이트 단위로, 지정된 바이트의 값을 읽는다.	〈변수〉 = bin(입력 바이트 번호)	int aa aa = bin(1)
BOUT	바이트 단위로, 지정된 바이트로 지정된 값을 출력한다.	bout(출력 바이트 번호) = 〈출력 지정 값〉	bout(1) = 10
WIN	워드 단위로, 지정된 워드의 값을 읽는다.	〈변수〉 = win(입력 워드 번호〉	int aa aa = win(1)
WOUT	워드 단위로, 지정된 워드로 지정된 값을 출력한다.	wout(출력 워드 번호) = 〈출력 지정 값〉	wout(1) = 0hFFFF
DIN	더블워드 단위로, 지정된 더블워드의 값을 읽는다.	〈변수〉 = din(입력 더블워드 번호〉	int aa aa = din(1)
DOUT	더블워드 단위로, 지정된 더블워드로 지정된 값을 출력한다.	dout(출력 더블워드 번호) = 〈출력 지정 값〉	dout(0) = 0hFFFFFFFF

명령어	기 능	형 식	사용 예
CIN	cclink 입력을 비트 단위로 읽는다.	〈변수〉=cin(입력비트 번호) -지정된 비트 입력 상태 값을 변수에 저장	int aa aa = cin(0)
COUT	cclink로 비트 단위로 출력한다.	cout(출력비트번호)=〈0/1〉	cout(1) = 1
CBIN	cclink 입력을 바이트 단위로 읽는다.	〈변수〉=cbin(입력포트 번호)	int aa aa = cbin(1)
CBOUT	cclink로 바이트 단위로 출력한다.	cbout(출력포트번호)=〈출력지정 값〉	cbout(0) = 123
CWIN	cclink 입력을 워드 단위로 읽는다.	〈변수〉=cwin(입력포트 번호)	int aa aa = cwin(1)
CWOUT	cclink로 워드 단위로 출력한다.	cwout(출력포트번호)=〈출력지정 값〉	cwout(0) = 1234
CDIN	cclink 입력을 더블워드 단위로 읽는다.	〈변수〉=cdin(입력포트 번호)	int aa aa = cdin(1)
CDOUT	cclink로 더블워드 단위로 출력한다.	cdout(출력포트번호)=〈출력지정 값〉	cdout(0) = 1234
CFIN	cclink 입력을 float 단위로 읽는다.	〈변수〉=cfin(입력포트 번호)	double da da = cfin(1)
CFOUT	cclink로 float 단위로 출력한다.	cfout(출력포트번호)=〈출력지정 값〉	cfout(0) = 123.56

2.1.4 로봇 동작 조건 관련 명령어

명령어	기 능	형 식	사용 예
VEL	축 이동 속도의 천분율을 설정	vel 〈천분율 값〉 축 이동속도=정격속도 X 0.01 X Level X 10 **정격속도는 MOTION 파라미터에서 설정**	func void main() vel(1, 200) acc 70 dec 70 jmov lp[1] vel 100 jmov lp[2] acc 100 dec 100 end
ACC	가속시간의 천분율을 설정	acc 〈백분율 값(%)〉 가속시간 = 정격가속시간 X 백분율 값 X 0.01 **정격가속시간은 MOTION 파라미터에서 설정**	
DEC	감속시간의 천분율을 설정	dec 〈백분율 값(%)〉 감속시간 = 정격감속시간 X 백분율 값 X 0.01 **정격감속시간은 MOTION 파라미터에서 설정**	
FOS	축 선단이 목표점에 도달하기 전에 다음 목표점으로 궤적변경 함.	fos 〈거리비율(%)〉 거리비율은 전체 이동거리의 백분율(%) JMOV에 적용	fos 5 jmov lp[1] fos 0(해제)
DLAY	지연시간 설정	dlay 〈지연시간〉 지연시간 단위는 10ms (지연시간이 500이면 5초 지연됨)	jmov lp[1] dlay 20 jmov lp[2]

명령어	기 능	형 식	사용 예
RSTATE	로봇의 상태를 읽어옴.	〈변수〉 = rstate(robot id, state) ▶ state 1 : ALARM 상태 3: Origin 완료 상태 6 : SERVO ON/OFF 상태 9 : AUTO/MANUAL 모드 상태 11 : EMG 상태 14: Origin 기능 사용 여부 15: 로봇 모션 수행 여부	int r_st r_st = rstate(1, 6)
SWLIMIT	파라미터에 설정된 swlimit 값을 읽어옴.	〈변수〉 = swlimit(robot id, axis id, limit type) ▶ limit type 0 : low 1 : high	double limit_low limit_low = swlimit(1, 1, 0)
GETLP	local point 데이터 읽기	〈변수〉 = getlp(robot id, local point index)	pos t_lp t_lp = getlp(1, 0)
SETLP	local point 데이터 쓰기	〈변수〉 = setlp(robot id, local point index, pos)	int ret pos t_lp ret = setlp(1, 0, t_lp)

2.1.5 변수 선언

명령어	기 능	형 식	사용 예
INT	정수형 변수를 선언	int 〈변수명〉, double 〈변수명〉, pos 〈변수명〉, string 〈변수명〉 변수명은 소문자 알파벳과 숫자로 구성 변수명은 숫자로 시작할 수 없다.	func void main() int n double a,b pos mm string sa
DOUBLE	실수형 변수를 선언		
POS	위치형 변수를 선언		
STRING	문자열 변수를 선언		

2.1.6 시스템 변수

명령어	기 능	형 식	사용 예
CNT(n) TMR(0) TMR(1)	시스템정의변수 (카운터변수,타이머 변수)	cnt(펄스입력포트번호)=〈초기값〉 TMR(0)=〈초기값〉 ▶ 카운터변수는 펄스입력포트 번호를 입력하는 순간부터 값이 할당되고 그 후 매 펄스입력을 카운트 한다. ▶ 타이머 변수는 정수 값을 입력하는 순간부터 값이 할당되어 시스템 파라미터에 정의된 시간 간격으로 1씩 증가한다.	CNT(0)=2 TMR(0)=0 TMR(1)=-50
MVR	MOVE RATE	mvr 〈로봇이동구간의 백분율〉	if mvr 〈 50
HERE	현재축 각도 값 변수	here	ap=here

2.1.7 상수

	기 능	형 식	사용 예
	정수, 실수, 2진정수, 16진 정수를 표시	[0H/0B]〈숫자〉 숫자 앞에 0H 또는 0B가 없으면 10진수 0H : 16진수, 0B : 2진수	
문자열 상수	문자열을 표시	문자열 상수는 “”를 사용하여 나타낸다 문자열의 최대 길이는 100자 이하이다. Ex) “string constant”	

2.1.8 연산자

명령어	기 능	형 식	사용 예
+, -, *, /, %	이항연산자 (가,감,승,제,modulus)		A = B*C
( )	우선 연산자		A = B/(A+C)
=	대입연산자		A = B
&, I , ~, ^, 〈〈, 〉〉	비트연산자 (BAND, BOR, Complement, BXOR, left shift, right shift)		
&&, ‖ , ^^, !	논리연산자 (AND, OR, XOR, Negation)		
〉, 〈, 〉=, 〈=, !=, ==	비교연산자(초과, 미만, 이상, 이하, 상이, 상동)		
ABS	절대값을 취함	abs(-10.5)는 10.5	
DEG	라디안 값을 각도 값으로 변환	deg(3.1416)는 180.0	
RAD	각도 값을 라디안 값으로 변환	rad(180.0)는 3.1416	
POW	지수함수	pow(2,4)는 16	
RND	소수부를 반올림하여 실수값을 정수 값으로 변환	rnd(14.8)은 15	
EXP	지수 e^x	EXP(3) 은 20.085	
LN	자연로그 $\log_e X$	ln(15)는 2.708	
LOG	상용로그 $\log_{10} X$	log(100) 은 2	
SQRT	평방제곱근을 구함	sqrt(16)은 4	
SIN	사인 함수	sin(rad(30))는 0.5	
ASIN	아크 사인 함수	asin(0.5)는 0.4794	
COS	코사인 함수	cos(0)은 1.0	
ACOS	아크 코사인 함수	acos(0.5)은 1.0472	
TAN	탄젠트함수	tan(rad(45))는 1.0	
ATAN	아크탄젠트 함수	atan(-1.0)은 -0.7854	
ATAN2	제2 아크탄젠트 함수	Atan2(Y,X) Atan2(1,-1)은 2.3562	
MIN	최소값 함수	min(1, 2)의 값은 2	int mv mv = min(1,2)
MAX	최대값 함수	max(1.1, 2.2)의 값은 2.2	double db db = max(1.1, 2.2)
DIST	위치 데이터의 거리차이 계산		pos tmp tmp = dist(lp[0], lp[1])

2.1.9 문자열

명령어	기 능	형 식	사용 예
ASC	문자열 첫문자를 캐릭터 코드로 반환	문자열변수 = asc(문자열)	string aa aa = asc("ABC")
BINS	정수를 이진수 문자열로 변환	문자열 변수 = bins(정수)	string aa aa = bins(13)
CHR	정수를 문자로 변환	문자열변수=chr(정수) ▶ Ex) aa = chr(65) 65에 해당하는 ASCII Code인 문자 A가 AA 변수에 저장됨.	string aa aa = chr(65)
FLUSH	입력, 출력 버퍼 클리어	flush 〈클리어 버퍼 선택〉 ▶ 버퍼 선택 범위 : 1~3 1: 입력 버퍼 클리어 2: 출력 버퍼 클리어 3: 입력,출력 버퍼 클리어	
FTOS	정수나 실수를 문자열로 변환	문자열변수 = ftos(정수or실수) ▶ 정수 1234를 문자열 "1234"로 변환	string aa aa = ftos(1234)
HTOS	정수를 16진수의 문자열로 변환	문자열변수 = htos(정수) ▶ 정수 10을 16진수 문자열 "A"로 변환	string aa aa = htos(10)
SLEFT	좌측부분 문자열 추출	문자열변수 = sleft(문자열, 숫자) ▶ 문자열의 왼쪽부터 숫자만큼 문자를 추출해 문자열 변수에 저장	string aa aa = sleft("ABCDE",3)
SLEN	문자열 길이를 반환	정수형변수 = slen(문자열)	int len len = slen("ABCDEF")
SMID	문자열 지정위치부터 자릿수 만큼 문자열 추출	문자열변수 = smid(문자열,지정위치,추출문자수) ▶ 문자열 위치는 맨 처음이 0부터 시작함. Ex) "ABCDEFG"에서 A는 0번째, B는 1번째, C는 2번째 순서임	string aa //"CDE" 문자열을 AA에 저장 aa = smid("ABCDEFG",2,3)
SPOS	문자열1 내에서 문자열2가 들어가 있는 위치 반환	정수형변수 = spos(문자열1, 문자열2)	int pp //"B"의 위치 1 반환 pp = spos("ABCDEF","B")

SRIGHT	우측부분 문자열 추출	문자열변수 = sright(문자열, 숫자)
		▶ 문자열의 오른쪽부터 숫자만큼 문자를 추출해 문자열 변수에 저장
			string aa
			//AA에 "CDE" 저장
			aa = sright("ABCDE",3)

STRIN	구분자까지 문자열 입력	문자열변수 = strin(타임아웃시간) ▶ 타임아웃시간은 ms단위임.	string aa aa = strin(1000)
STROUT	문자열 출력	정수형변수 = strout(문자열) ▶ 정수형변수에 전송하지 못한 문자수 저장	int rt rt = strout("ABCDEF")
SVAL	문자열을 숫자로 변환	변수 = sval(문자열)	int va double vb va = sval("1234") vb = sval("0.123")

2.1.10 THREAD 관련 명령어

명령어	기 능	형 식	사용 예
CTHREAD	thread를 생성	cthread(함수, thread 번호) ▶ thread 번호는 1~3까지만 사용 가능	int t_ret t_ret = cthread(aa, 1) cthread(bb, 2)
ETHREAD	thread를 종료	ethread(thread 번호)	ethread(1) ethread(2)
TH_STATE	thread 상태 정보 읽기	th_state(thread 번호) 리턴값. ▶TH_ERROR : thread 번호 오류 ▶TH_IDLE : thread 사용 가능 ▶TH_STOP : thread가 정지 ▶TH_RUN : thread가 구동중	int t_state t_state = th_state(1)
TH_SUSPEND	구동중인 thread 정지	th_suspend(thread 번호)	th_suspend(1)
TH_RESUME	정지중인 thread 구동	th_resume(thread 번호)	th_resume(1)
TAKE	thread에서 구동할 robot 선택	take 〈robot id〉	take 1
RELEASE	take로 선택한 로봇 해제	release	release

2.1.11 LOG 관련 명령어

명령어	기 능	형 식	사용 예
PRINT	로그를 파일로 저장	print(index, 로그 내용) ▶ index는 0~999까지 사용 가능	int t_ret t_ret = cthread(aa, 1) print(1, “1st thread create : ”, t_ret)
PRINT_READ	PRINT 함수로 저장한 로그를 읽어옴	print_read(index) ▶ index는 0~999까지 사용 가능	string str str=print_read(100)
WLOG	시간 정보를 포함하여 로그를 순서대로 파일로 저장	wlog(로그 내용)	int t_ret t_ret = cthread(aa, 1) wlog(“1st thread create : ”, t_ret)
CLOG	시간 정보를 포함하여 통신 관련 로그를 저장	clog(로그 내용)	int c_ret c_ret=cout(0) clog("cout[0]:",c_ret)

2.1.12 LATCH 관련 명령어

1) EtherCAT 타입 (제어기 : T1S,T1M,T1L)

명령어	기 능	형 식	사용 예
LATCH_INIT	latch 환경 설정	latch_init(축, latch 방식, 센서 갯수) ▶ 축 : 1 ~ 최대축 ▶ latch 방식 : 1 : rising edge 2 : falling edge ▶ 센서 개수 : latch에 사용할 센서의 개수 (1 ~ 2)	int ret ret = latch_init(3, 1, 2)
LATCH_CLEAR	latch 환경 초기화	latch_clear(축)	int ret ret = latch_clear(3)
LATCH_START	latch 시작	latch_start()	int ret ret = latch_start()
LATCH_STOP	latch 정지	Latch_stop()	int ret ret = latch_stop()
LATCH_POS	latch 위치	latch_pos(축, 센서 번호)	float l_pos[2] l_pos[0] = latch_pos(3, 1) l_pos[1] = latch_pos(3, 2)
LATCH_STATE	latch 상태	latch_state(축, 센서 번호)	int ret ret = latch_state(3, 1) ret = latch_state(3, 2)
LATCH_SENSOR	latch 센서의 현재 상태를 읽는 함수	latch_sensor(축, 센서 번호)	int ret ret = latch_sensor(3, 1) ret = latch_sensor(3, 2)

2) RS422 타입 (제어기 : T1X)

명령어	기 능	형 식	사용 예
LATCH_INIT	latch 환경 설정	int latch_init(int 축, int 센서 시작 번호, int 센서 수) ▶ 축 : 1 ~ 최대축 ▶ 센서 시작 번호 – Range: 1~4 까지 Range Value / 사용되는 센서 1 / 1, 2번 2 / 2, 3번 3 / 3, 4번 4 / 4번 ▶ 센서 수 – Range: 1~2 까지 ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = latch_init(3, 1, 2)
LATCH_CLEAR	latch 환경 초기화	int latch_clear(int 축) ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = latch_clear(3)
LATCH_START	latch 시작	int latch_start() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = latch_start()
LATCH_STOP	latch 정지	int latch_stop() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = latch_stop()
LATCH_POS	latch 위치	double latch_pos(int 축, int 센서 Index) ▶ 센서 Index – Range: 1~2 까지 ▶ 리턴 값 - double 형태, Position 값	float l_pos[2] l_pos[0] = latch_pos(3, 1) l_pos[1] = latch_pos(3, 2)

LATCH_STATE	latch 상태	int latch_state(int 축, int 센서 Index) ▶ 센서 Index – Range: 1~2 까지 ▶ 리턴 값 - 0: FAIL, 1: 신호 감지 못함, 2: 신호 감지	int ret ret = latch_state(3, 1) ret = latch_state(3, 2)
LATCH_SENSOR	latch 센서의 현재 상태를 읽는 함수	latch_sensor(축, 센서 Index)	int ret ret = latch_sensor(3, 1) ret = latch_sensor(3, 2)

2.1.13 ERROR 관련 명령어

명령어	기 능	형 식	사용 예
SETERR	에러 설정	seterr(사용자 정의 error 번호)	if in(0) == 1 seterr(1) end
RESET	알람 클리어	reset	reset
RERROR	가장 마지막 에러 코드 반환	정수형 변수 = rerror()	int ecode ecode = rerror()
RERRCNT	현재 발생한 알람 개수 반환	정수형 변수 = rerrcnt()	int ecount ecount = rerrcnt()
RERRCODE	INDEX에 해당하는 에러 코드 반환	정수형 변수 = rerrcode(index) ▶ index : 알람 발생 순서	Int ecode Ecode = rerrcode(1) //첫번째 에러 //코드 반환
RERRTEXT	INDEX에 해당하는 에러 내용 반환	문자형 변수 = rerrtext(index) ▶ index : 알람 발생 순서	string etext etext = rerrtext(1) //첫번째 에러 //내용 반환
READ_DERR	Driver error 코드 읽기	정수형 변수 = read_derr(int 축) ▶ 축: 축 번호	Int value value = read_derr (1)

2.1.14 MAPPING 관련 명령어 (RS422 통신타입 only)

명령어	기 능	형 식	사용 예
MAP_INIT	Mapping 관련 내부 변수 초기화	int map_init()	int ret ret = map_init()
MAP_START	Mapping 시작	int map_start(int 축, int 센서인덱스, int 시작 로컬 포인트 인덱스) ▶ 축 : 1 ~ 최대축 ▶ 센서 인덱스 : 센서의 인덱스 값(Range: 1~2) ▶ 시작 로컬 포인트: mapping 데이터를 저장할 시작 포인트 인덱스, Range: 0~19999	int ret ret = map_start(2, 3,100)
MAP_STOP	Mapping 정지	int map_stop()	int ret ret = map_stop()
MAP_READ	Mapping 데이터 읽기	int map_read(int 타임아웃 시간) ▶ 타임아웃 시간(ms)	int ret ret = map_read(1000)

2.1.15 INSHAPE 관련 명령어

명령어	기 능	형 식	사용 예
INSHAPE_SET	InShape 기능을 사용할 축을 설정 함	Int inshape_set(int 축, int 최대 index 개수) ▶ 축: 1~ 최대 축, INSHAPE 기능을 사용할 축 입력 ▶ 최대 index 개수: 사용할 배열의 인덱스 개수를 입력 (Range: 1~10) ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = inshape_set(1, 6)
INSHAPE_CLEAR	InShape 기능 사용을 해제함	int inshape_clear() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = inshape_clear()
INSHAPE_READ	해당 축의 InShape 관련 파라미터 값을 읽어 옴	double inshape_read(int 축, int param, int index) ▶ 축: 1~ 최대 축 ▶ param: 값을 읽어 올 해당 파라미터를 명시. 1 (inshape 발생 위치), 2(보상 값) ▶ index: index number 지정 (Range: 1~10)	double occur_pos occur_pos = inshape_read(1, 1, 1)
INSHAPE_WRITE	해당 축의 InShape 파라미터 값을 설정 함	int inshape_write(int 축, int param, int index, double 설정 값) ▶ 축: 1~ 최대 축 ▶ param: 값을 읽어 올 해당 파라미터를 명시. 1 (inshape 발생 시간을 비율로 표현), 2(보상 값) ▶ index: index number 지정 (Range: 1~10) ▶ 설정 값: double 형. 원하는 값을 입력 [주의] Param 이 1일 경우(inshape 발생위치)의 Range: 0.0 ~ 1.0 사이의 값 ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret double value1 double value 2 value1 = 0.1 Value2 = 10 ret = inshape_write(1, 1, 1, value1) ret = inshape_write(1, 2, 1, value2)
INSHAPE_START	InShape 기능 사용 시작을 알림	int inshape_start() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = inshape_start()

2.1.16 ARM FOLD 관련 명령어

명령어	기 능	형 식	사용 예
INITARM_WRITE	ARM FOLD 기능 사용 시 핸드 위치 값 저장	int initarm_write(int robot_id, int 인덱스, double 위치) ▶ 인덱스 - 1: 핸드 위치, 2: 부가축 위치 ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret double hand_pos hand_pos = -1300.0 ret = initarm_write(1, 1, hand_pos)
INITARM_READ	ARM FOLD 기능 사용 시 도달하는 핸드 위치 값 읽기	double initarm_write(int robot_id, int 인덱스) ▶ 인덱스 - 1: 핸드 위치, 2: 부가축 위치 ▶ 리턴 값 - (Double) 핸드 복귀 위치 값	double hand_pos hand_pos = initarm_read(1,1)
INITARM_CLEAR	핸드 도달 위치 삭제	int initarm_clear(void) ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret=initarm_clear()

2.1.17 Z축 보상 관련 명령어

1) 초기 설정 관련 명령어

명령어	기 능	형 식	사용 예
ZCAL_DATA_SET	Z축 보상 함수 생성에 사용되는 위치 및 오차 정보 입력	int zcal_data_set(int 인덱스, int 핸드 축, float 핸드 위치 값, float 오차) ▶ 인덱스: 핸드위치 값과 오차를 저장할 배열의 위치를 지정(Range: 0~30) ▶ 핸드 축: 대상 핸드 축 지정 ▶ 핸드 위치 값: 지정된 인덱스에 저장할 핸드의 위치 값을 입력, 범위는 sw_limit 까지로 제한됨 ▶ 오차: 핸드 축의 Z 축 방향 처짐 정도를 입력 (Range: -50 ~ 50) ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret, index, arm float range, err index=1 arm =3 range = 115.21 err=0.24 ret = zcal_data_set(index, arm, range, err)
ZCAL_DATA_CLR	Z축 보상 함수 생성에 사용되는 위치 및 오차 정보 초기화	int zcal_data_clr() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_data_clr()
ZCAL_COEF_GEN	Z축 보상 다항식 계수 생성	int zcal_coef_gen() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_coef_gen ()
ZCAL_DATA_READ	입력한 위치 및 오차 정보 읽기	float zcal_data_read(int 인덱스, int 옵션) ▶ 인덱스: 읽어올 데이터 배열의 위치를 지정 (Range: 0~30) ▶ 옵션: 데이터 셋 여부(1), 위치 값(2), 오차 값(3) 중 선택 ▶ 리턴 값 - (double) 선택한 인덱스와 옵션에 해당하는 데이터 값	int ret ret = zcal_data_read(0,2)

2) 실행 관련 명령어

명령어	기 능	형 식	사용 예
ZCAL_INIT	Z 보상 대상 축 및 참조할 핸드 축을 설정	int zcal_init(int 보상대상 축, int 핸드 축) ▶ 보상대상 축: hand의 직진도 테스트 중 보상을 해야하는 축 입력(예를들어, Z축) ▶ 핸드 축: 진진도를 측정하는 대상 핸드 축 ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_init(2,3)
ZCAL_READ	핸드 위치 값에 따른 Z축 보상 값을 반환	double zcal_read(double 핸드 위치 값) ▶ 핸드 위치 값: z축 보상 값을 반환받고자 하는 hand의 위치 값 ▶ 리턴 값 - (double) z 축 보상 값	double delta_z pos m_pos m_pos = lp[0] delta_z = zcal_read(m_pos.3)
ZCAL_OPT	사용할 부가축 설정	int zcal_opt(int 부가 축) ▶ 부가 축: stroke를 늘리기 위해 하단에 별도로 설치된 축 입력 ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_opt(6)
ZCAL_CLEAR	설정 값 해제 및 초기화	int zcal_clear() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_clear()

ZCAL_START	Z축 보상 시작	int zcal_start() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_start()
ZCAL_STOP	Z축 보상 정지	int zcal_stop() ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = zcal_clear()
ZCAL_STATE	Z축 보상 사용 여부 확인	int zcal_state() ▶ 리턴 값 - 0: STOP, 1: RUN	int ret ret = zcal_state()

2.1.18 그 외

명령어	기 능	형 식	사용 예
OVERRIDE	전체 실행 속도를 변경	void override(int robot_id, int 속도) ▶robot id: 실행 속도를 변경하고자 하는 로봇 번호 ▶ 속도: 변경 속도 값 ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = override(1,10)
XENB	특정 축의 사용 여부를 결정함	void xenb(int 축, int 상태) ▶ 축: 사용 여부를 결정할 축 번호 ▶ 상태: ON – 축 사용 안함 (Disable) OFF – 축 사용 함 (Enable)	int axis axis = 3 xenb(axis, ON)
READ_DIST	누적거리 및 초과 카운트 읽기	double read_dist(int robot_id, int 타입, int 축) ▶ robot_id: 로봇 번호 ▶ 타입: 읽어 올 데이터 타입을 선택 1: 누적거리 값 2: 초기화 카운트 (제한거리 초과 시) ▶ 축: 축 번호 ▶ 리턴 값 - (double) 누적 거리 혹은 초기화 카운트 값	doube value value = read_dist(1, 1, 1)
ORGEXEC	Origin 기능 수행	int orgexec(void) ▶ 리턴 값 - 0: FAIL, 1: SUCCESS	int ret ret = orgexec()
LP_USED	로컬 포인트 사용 여부 확인	int lp_used(int robot_id, int 포인트번호) ▶ 리턴 값 - 0: 사용 안 함, 1: 사용 함	int ret ret = lp_used(1, 10000)
READ_RPM	RPM 읽기	double read_rpm(int robot_id, int 축) ▶ robot_id: 로봇 번호 ▶ 축: 축 번호 ▶ 리턴 값 - (double) RPM 값	doube value value = read_dist(1, 1)
READ_LOAD	LOAD 읽기	double read_load(int robot_id, int 축) ▶ robot_id: 로봇 번호 ▶ 축: 축 번호 ▶ 리턴 값 - (double) LOAD 값	doube value value = read_load(1, 1)
READ_TRQMAX	MAX TORQUE 읽기	double read_trqmax(int robot_id, int 축) ▶ robot_id: 로봇 번호 ▶ 축: 축 번호 ▶ 리턴 값 - (double) MAX TORQUE 값	doube value value = read_trqmax(1, 1)

READ_VEL	VELOCITY 읽기	double read_vel(int robot_id, int 축) ▶ robot_id: 로봇 번호 ▶ 축: 축 번호 ▶ 리턴 값 - (double) VELOCITY 값	doube value value = read_vel(1, 1)

제3장 명령어 해설

3.1 FUNC, END (함수 선언/종료)

기 능 함수 시작 및 종료를 나타냅니다.

형 식
```
func 함수리턴타입 함수명(매개변수)
…
end
```

설 명
1) **FUNC-END는 반드시 같이 사용**하여 함수 블록을 형성합니다. 프로그램의 시작은 맨처음 나오는 함수에서 시작합니다.
2) END 함수 블록의 끝을 의미하며 함수 블록 수행 시 이 명령을 만나면 **함수 수행이 종료**됩니다.
   단, 함수 호출에 의해 수행된 함수는 그 JOB의 수행을 종료하고 **호출한 함수로 복귀** 합니다.
3) **END 다음 라인**에 필요한 **함수 블록(FUNC ~ END)을 계속 작성**할 수 있습니다.
4) 리턴타입이 VOID가 아닌 경우에는 함수 종료 전에 반드시 RETURN 명령어가 수행되어야 합니다.
5) 처음 실행하는 job의 맨 처음 함수가 주 program 이며, 주 program의 형식은 반드시 func void 함수이름() 형태여야 합니다.

3.1.1 프로그램 사용 예제

func void main()	……	**주 Program 시작**
take 1	……	로봇 지정
vel(100)	……	축 이동 속도
while 1	……	조건반복수행 시작
jmov lP[0]	……	P0로 이동
jmov lP[1]	……	P1으로 이동
biton()	……	부 프로그램 호출
End	……	조건반복수행 종료
release	……	로봇 지정 해제
End	……	**주 Program 종료**
func int biton()	……	**함수 지정**
int a	……	
out(0)=1, 200	……	1 비트 단위로, 지정된 비트에 출력
out(1)=1, 200	……	1 비트 단위로, 지정된 비트에 출력
out(2)=1, 200		1 비트 단위로, 지정된 비트에 출력
a = in(0)		
return (a)	……	**함수 리턴**
end	……	**함수 종료**

3.2 INCLUDE

기 능 다른 JOB을 포함시킨다.

형 식 include “job이름”

설 명 1) JOB의 맨 처음에 나와야 하며, 다른 JOB의 이름을 호출하여야 합니다.

3.2.1 프로그램 사용 예제

include "inc.job"		다른 JOB파일을 포함시킴
func void main()	……	주 Program 시작
…		
end	……	함수 종료

3.3 FOR, END (반복 수행문)

기 능 변수 값이 만족할 때 까지 블록을 반복 수행합니다

형 식 for 〈변수〉=〈초기값〉 to 〈종료값〉 (step 〈증분량〉)
…
end

용 어 〈변수〉 : 정수나 실수형으로 선언된 변수이름을 사용합니다. 예) int a,b,c,aa,…
〈초기값〉 : FOR 블록을 반복 수행하기 직전 변수에 설정되는 정수/실수 값입니다.
〈종료값〉 : FOR 블록의 반복 수행 횟수를 제한하기 위한 정수/실수 값입니다.
〈증분값〉 : 변수의 값을 규칙적으로 일정하게 증가시키기 위해 사용되는 정수/실수 값입니다

설 명
1) **FOR - END 는 반드시 같이 사용**되어야 합니다.
2) 정수형이나 실수형 변수에 초기값이 설정된 후 **종료값을 만족할 까지 FOR 블록을 반복 수행** 합니다.
3) FOR 블럭을 반복 수행할때마다 정수/실수형 변수의 값은 증분값에 의해 증가되고, 증분값이 생략되었을 경우에는 **자동적으로 1씩 증가**합니다.
4) 종료값은 초기값보다 **항상 큰 값**이어야 합니다. (초기값≤종료값)

3.3.1 프로그램 사용 예제

1) FOR 블록을 이용하여, 1부터 10까지 홀수의 합을 연산하여 정수형 변수 SUM에 저장.

```
func void main()
int aa, sum ……… 정수변수 AA, SUM 선언
sum=0 ……… 정수변수 초기화
for aa=1 to 10 step 2 ……… AA=1,부터 +2씩 증가 정수 10까지 5회 수행
sum=sum+aa ……… 5회 수행 후 END 다음 이동
end
end
```

2) FOR 블록을 이용하여, 포인트 P1 ↔ P2 간의 20회 왕복운동

```
func void func()
int j                ……  정수변수 J선언
j=0                  ……  정수변수 초기화
take 1
for j=1 to 20        ……  FOR문 20회 반복동작 후 END 다음 이동
jmov lp[1]           ……  JMOV P1
jmov lP[2]           ……  JMOV P2
end
release
End
```

3) Position 변수에 티칭포인트를 저장한 후 이동

```
func void main()
int a                ……  정수변수 A선언
pos ap[11]           ……  위치변수 AP(11)선언 → AP(0),AP(1) ~ AP(10)
for a=1 to 10        ……  FOR문 10회 반복동작 후 END 다음 이동
ap[a]=getlp(1, a)    ……  포인트 변수 AP(A)에 P(1) ~ P(10) 저장
end
end
```

4) Position 변수에 티칭포인트를 저장한 후 이동

```
func void main()
int j,k              ……  정수변수 J,K선언
pos tmp,ap[11]       ……  위치변수 AP(11)선언 → AP(0),AP(1) ~ AP(10)
take 1               ……
for k=1 to 10        ……  FOR문 10회 반복동작 후 END 다음 이동
ap[k]=lp[k]          ……  포인트 변수 AP(K)에 LP(1) ~ LP(10) 저장
end
if lp[100].3==1
for j=1 to 10                FOR문 10회 반복동작 후 END 다음 이동
tmp=ap[j]            ……  AP(J)=AP(I)=LP(1) ~ LP(10)
tmp.3=tmp.3+10               위치변수 TMP의 Z축 값에 10을 더함.
ap[j]=tmp
end
for k=1 to 10        ……  FOR문 10회 반복동작 후 END 다음 이동
jmov ap[k]                   저장된 AP(1) ~ AP(10)으로 PTP 이동
dlay 100
end
release
end
```

3.4 WHILE, END (조건 반복수행 명령문)

기 능 조건식이 만족 되는 동안만 블록을 반복수행합니다

형 식 while 〈조건식〉
…
end

용 어 〈조건식〉 : 수식의 진위를 판단할 수 있는 논리연산식이나 비교연산식을 말합니다.

설 명 1) WHILE 문은 **조건이 만족(조건식의 결과가 참, 또는 0이외의 값)되는 동안** WHILE 블록을 **반복 수행**합니다.
2) **WHILE - ENDWL은 반드시 같이** 사용되어야 합니다.
3) 조건식의 결과가 **항상 참인 경우에는 무한 반복 수행**을 하게됩니다.

CAUTION

명령어 입력시 주의 사항
(O) WHILE_((IN(0)==1)&&(IN(1)==0))
(O) WHILE_IN(0)==1&&IN(1)==0
(X) WHILE_IN(0)==1
&&IN(1)==0

3.4.1 프로그램 사용 예제

1) 포인트 P0 ↔ P1을 반복이동

```
func void main()
take 1 ……… robot 1 선택
vel(100) ……
while 1 …… 블록 무한 반복 실행
jmov lp[0] ……
jmov lp[1] …… LP0 ↔ LP1 반복이동
end
release
end
```

3.4.2 조건식 사용 예제

WHILE(조건식)		설 명
상 수	WHILE 1	WHILE ~ END 블록을 무한 반복 실행
입 력	WHILE IN(0)==1	입력신호 IN0=1로 입력되는 동안만 블록 반복 실행
	WHILE ((IN(0)==1)&&(IN1==(0)))	입력신호 IN0=1, IN1=0 두 가지 조건이 동시에 만족하는 동안만 블록을 반복실행
	WHILE ((IN(0)==1) || (IN(1)==0))	입력신호 IN0=1, IN1=0 두 가지 조건 중 한가지만 만족하는 동안만 블록을 반복실행
	WHILE PIN(0)==0H000F	입력PORT 0번의 IN0, IN1, IN2, IN3 “1”인 동안만 블록을 반복 실행
	WHILE PIN(0)==32	입력 PORT 0번의 입력신호상태가 IN4가 “1”인 동안만 블록을 반복 실행
출 력	WHILE OUT(0)==1	출력 OUT0이 “1”인 동안만 블록을 반복실행
	WHILE POUT(0)==0H000F	출력 POUT0번의 OUT0, OUT1, OUT2, OUT3이 모두 “ 1”인 동안만 블록을 반복실행
	WHILE POUT(0)==0B0000000011111111	출력 POUT0번의 OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7이 모두 “1”인 동안만 블록을 반복실행
변 수	WHILE TMP==1	변수 TMP가 “1”인 동안만 블록을 반복실행
	WHILE MVR〉10	MVR변수 값이 “10”보다 큰 동안만 블록반복 실행
	WHILE TMR〉10	TMR(TIMER)변수값이 “10”보다 큰 동안만 블록반복 실행
	WHILE CNT〉10	CNT(COUNT)변수 값이 “10”보다 큰 동안만 블록반복 실행
	WHILE AP[10].1==1	POSITION 변수 AP값의 1축 값이 “1”과 같은 동안만 블록을 반복 실행

3.5 CONTINUE, BREAK(반복문 제어)

기 능 for나 while의 반복문을 제어 합니다.

형 식
```
while 1
  if 〈조건식〉
    continue
  else
    break
end
```

용 어 〈조건식〉 : 수식의 진위를 판단할 수 있는 논리연산식이나 비교연산식을 말합니다.

설 명
1) CONTINUE문을 만나면 반복문의 맨 처음으로 이동합니다.
2) BRAKE문을 만나면 반복문을 탈출합니다.

3.5.1 프로그램 사용 예제

```
func void run()
take 1
vel(100)
while 1 ……… 블록 무한 반복 실행
if in(0) == 1
dlay 1000 1초 대기
continue 반복문 처음으로 이동
elif in(1) == 1
break 반복문 종료
end
jmov lp[0] ……… LP0 ↔ LP1 반복이동
jmov lp[1] ……… LP0 ↔ LP1 반복이동
end
release
end
```

3.6 IF, END (조건 분기 명령어)

기 능 조건식의 조건 판단을 수행합니다.

형 식
```
if 〈조건식〉
...
(else)
...
end
```

용 어 〈조건식〉 : 수식의 진위를 판단할 수 있는 논리연산식이나 비교연산식을 말합니다.

설 명
1) **IF - END 는 반드시 같이** 사용하여야 합니다.
2) 조건식의 연산 결과가 **참(또는 0이외의 값)이면 IF 이하의 문장**들이 실행되고, **거짓(또는 0)이면 ELSE 다음의 문장**들이 실행됩니다.
3) ELSE문은 프로그램 작성자가 **선택적으로 사용**할 수 있습니다.
4) IF 블록 내부에서 다시 IF 블록을 사용하는 중첩 IF 블록을 사용할 수 있습니다. 이때 주의할 것은 IF와 END가 논리적으로 쌍을 이룰수 있도록 작성해야 합니다.

※ IF의 개수와 ENDIF의 개수가 다를경우 "Syntax Error"가 발생 됩니다.

⚠ CAUTION

명령어 입력시 주의 사항
(O) IF_((IN(0)==1)&&(IN(1)==0))
(O) IF_IN(0)==1&&IN(1)==0
(X) IF_IN(0)==1
&&IN(1)==0

3.6.1 프로그램 사용 예제

1) 입력 IN0의 상태에 따라 이동할 포인트 결정

```
func void main()
take 1
vel(1, 10)
if in(0)==1
jmov lp[1]          ……          입력 IN0 가 1인 경우 포인트 LP1로 PTP 이동
else
jmov lp[2]          ……          입력 IN0 가 0인 경우 포인트 LP2로 PTP 이동
end
release
end
```

2) 입력 Port "1"(IN0 ~ 15), IN0의 상태 검사

```
func void main()
int aa, bb
take 1
aa=pin(1)           ……          Port1(IN0~15) 상태를 변수 AA에 저장
if (aa==0HFFFF)                 AA값 판단 (IN0~15 모두 1인지 여부)
stop                ……          "참" 일 경우, 로봇 스톱
else
bb=in(0)            ……          "거짓" 일 경우, IN0의 상태를 BB에 저장
if (bb==1)                      BB가 1인지 판단
dlay10              ……
jmov lp[0]          ……          "참(IN0=1)" 일 경우, 포인트 LP0로 이동
end
end
release
end
```

3) 아래 프로그램은 동일한 내용이다.

(단, IN0~IN3중 동시에 가해지는 입력이 하나뿐이란 가정하에 동일 내용)

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

=

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

3.6.2 IF(조건식) THEN

IF(조건식)		설 명
입 력	IF_IN(0)==1	입력 IN0이 “1”인가?
	IF_((IN(0)==1)&&(IN(1)==0))	입력신호 IN0=1, IN1=0 두 가지 조건이 동시에 만족하는가?
	IF_((IN(0)==1) || (IN(1)==0))	입력신호 IN0=1, IN1=0 두 가지 조건 중 최소 한가지 만족하는가?
	IF_PIN(0)==0H000F	입력PORT0번의 IN0, IN1, IN2, IN3이 모두 “1”인가?
	IF_PIN(0)==32	입력 PORT0번의 입력신호상태가 IN4가 “1”인가?
출 력	IF_OUT(0)==1	출력 OUT0이 “1”인가?
	IF_POUT(0)==0H000F	출력 POUT0번의 OUT0, OUT1, OUT2, OUT3이 모두 “ 1”인가?
	IF_POUT(0)==0B0000000011111111	출력 POUT0번의 OUT0, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7이 모두 “1”인가?
변 수	IF_TMP==1	변수 TMP가 “1”인가?
	IF_MVR〉10	MVR변수 값이 “10”보다 큰가?
	IF_TMR〉10	TMR(TIMER)변수 값이 “10”보다 큰가?
	IF_CNT〉10	CNT(COUNT)변수 값이 “10”보다 큰가?
	IF_AP[10].1==1	POSITION 변수 값(1축 값) “1”과 같은가?

3.7 LABL, GOTO (분기 명령어)

기 능 분기점 위치 지정, 수행문 분기 이동 명령어 입니다.

형 식 labl 〈레이블명〉
goto 〈레이블명〉

용 어 〈레이블명〉 : 영문자와 숫자로 구성된 문자 스트링입니다.
단, 작업 포인트를 표시하는 "lp[0], lp[1], … "는 사용할 수 없습니다.
예) labl UNCLAMP1 **(O)**, GOTO CHECK2 **(O)**
labl p[1] **(X)**, goto p[100] **(X)**

설 명 1) GOTO명령은 **조건 없이 해당 레이블로 분기**하며 동일 블록 내에서만 가능합니다.
예를 들면, main 블록 내에서 다른 func블록으로 분기할 수 없습니다.
2) IF블록이나 FOR, WHILE블록을 중첩하여 사용할 때, 내부에서 외부로의 분기는 가능 하지만, **반대의 경우는 불가능**합니다.

3.7.1 프로그램 사용 예제

1) 두 포인트 P1, P0를 반복 이동

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

"LABL A0"로 무조건 점프

=

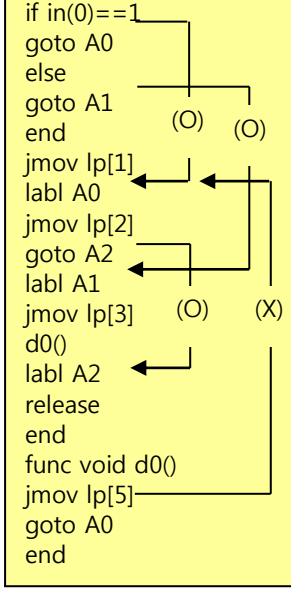
```
func void main()
take 1
vel(100)
while 1
jmov lp[1]
jmov lp[2]
end
release
end
```

2) 입력 IN0의 상태에 따라 분기

```
func void main()
take 1
vel(100)
jmov lp[0]
if in(0)==1
goto A0            ……  입력 IN0이 1이면  LABL A0로 무조건 점프
Else
goto A1            ……  아니면 LABL A1로 무조건 점프
End
jmov lp[1]
labl A0            ……  LABL A0 분기지점
jmov lp[2]
goto A2            ……  LABL A2로 무조건 점프
labl A1            ……  LABL A1 분기지점
jmov lp[3]
labl A2            ……  LABL A2 분기지점
release
end
```

CAUTION

명령어 입력시 주의 사항



3.8 STOP, EXIT (로봇 및 JOB 정지 명령어)

기 능 로봇 동작 정지(STOP), JOB 수행정지(EXIT)

형 식
stop
exit

설 명
1) STOP 명령어는 STOP 명령어가 사용된 thread에서 **구동중인 로봇을 정지**시키고, 다음 STEP의 명령어를 처리합니다. 즉, **정지후 JOB을 계속 수행**합니다.
2) 다른 thread에서 구동중인 robot은 정지 시킬수 없습니다.
3) EXIT 명령어는 EXIT 명령어가 수행되는 thread의 **JOB 수행을 중지**시킵니다.

3.8.1 프로그램 사용 예제

1) 포인트이동 후 출력 OUT1을 ON, OFF

```
func void main()
take 1
vel(500)
jmov lp[1]
mvr=0 ……                    "이동구간 백분율"을 초기화
with ……
jmov lp[2] ……                포인트 P2로 이동(JMOV만 가능함)
while mvr<60 ……              이동구간의 60%미만 동안에
if in(0)==1 ……               IN0=1이면 "LABL BB"로 점프
goto BB
End
End
out(0)=1
End

labl BB
stop ……                     로봇이동 동작 정지
end ……                      WITH문 종료 (반드시 삽입)
jmov lp[3]
out(1)=1
release
end
```

3.9 CTHREAD (thread 생성 명령어)

기 능 thread를 생성합니다.

형 식 cthread(함수, THREAD ID)

용 어 함수 : thread로 구동 시킬 함수로 리턴 타입은 void형, 매개변수는 없어야 합니다.
THREAD ID : 구동할 thread의 ID
1 ~ 3까지만 입력 가능합니다.
선택한 Thread ID가 이미 다른 thread에 의해 사용된 경우에는
thread가 생성되지 않습니다.

반환값 결과값
0 : thread 생성 실패
1 : thread 생성 성공

설 명 모션 명령어는 사용 할 수 없습니다.
thread로 데이터를 전달하는 경우 글로벌 변수를 사용합니다.
thread는 최대 3개까지 생성 가능합니다.

3.9.1 프로그램 사용 예제

```
func void main()
int t_ret
t_ret=cthread(sum, 1)        ……  1~10까지 누적 합을 구하는 함수를 thread로 실행
                             ….
if t_ret == 0
print(0,  “failed  to
create sum thread”)
end
end

func void sum()              ……  thread 함수(반환값이 void, 매개변수는 없어야 합니다.)
                             ….
int a,b
for a = 1 to 10
    b = b + a
end
end
```

3.10 ETHREAD (thread 종료 명령어)

기 능 thread를 종료합니다.

형 식 ethread(thread ID)

용 어 thread ID : cthread에서 입력한 thread의 ID입니다.

반환값 반환값은 없습니다.

설 명 실행 중인 thread를 종료합니다.

3.10.1 프로그램 사용 예제

```
func void main()
int t_id, t_ret
t_id = 1
t_ret=cthread(sum, t_id)  ……  1~10까지 누적 합을 구하는 함수를 thread로 실행
dlay 1000
ethread(t_id)
end

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

3.11 TH_STATE(thread 상태 명령어)

기 능 실행중인 thread의 상태를 확인합니다.

형 식 th_state(thread ID)

용 어 thread ID : cthread에서 입력한 thread의 ID입니다.

반환값 Thread 상태를 반환합니다.
0 : thread id를 잘못 입력(thread ID의 범위는 0~3 사이입니다.)
1 : thread가 동작하고 있지 않습니다.
2 : thread가 정지하고 있습니다. (th_suspend 명령으로 thread가 정지한 경우)
3 : thread가 구동중입니다.

설 명 실행 중인 thread의 상태를 확인합니다.

3.11.1 프로그램 사용 예제

```
func void main()
int t_id
int th_st
t_id = 1
cthread(sum, t_id)                ……                1~10까지 누적 합을 구하는 함수를 thread로 실행
                                  ……
while 1
dlay 1000
th_st = th_state(t_id)                              thread 상태 확인
if (th_st == 1)                                     thread가 동작 하지 않는 경우
   cthread(sum, t_id)                               thread 생성
elif (th_st == 2)                                   thread가 정지 중인 경우
   th_resume(t_id)                                  thread 구동
elif (th_st == 3)                                   thread가 구동 중인 경우
   th_suspend(t_id)                                 thread 정지
end
dlay 1000
th_st = th_state(t_id)
if ( th_st != 1 && th_st != 0)
   ethread(t_id)
end
end

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

```
end
end
```

3.12 TH_SUSPEND(thread 정지 명령어)

기 능 실행중인 thread를 정지 시킵니다.

형 식 th_suspend(thread ID)

용 어 thread ID : cthread에서 입력한 thread의 ID입니다.

반환값 반환값은 없습니다.

설 명 실행 중인 thread를 정지 시킵니다.

3.12.1 프로그램 사용 예제

```
func void main()
int t_id
int th_st
t_id = 1
cthread(sum, t_id)                ……                1~10까지 누적 합을 구하는 함수를 thread로 실행
                                  ……
while 1
dlay 1000
th_st = th_state(t_id)                              thread 상태 확인
if (th_st == 1)                                     thread가 동작 하지 않는 경우
   cthread(sum, t_id)                               thread 생성
elif (th_st == 2)                                   thread가 정지 중인 경우
   th_resume(t_id)                                  thread 구동
elif (th_st == 3)                                   thread가 구동 중인 경우
   th_suspend(t_id)                                 thread 정지
end
dlay 1000
th_st = th_state(t_id)
if ( th_st != 1 && th_st != 0)
   ethread(t_id)
end
end

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

3.13 TH_RESUME(thread 재구동 명령어)

기 능 Th_suspend 명령에 의해 정지중인 thread를 다시 구동 합니다.

형 식 th_ suspend (thread ID)

용 어 thread ID : cthread에서 입력한 thread의 ID입니다.

반환값 반환값은 없습니다.

설 명 실행 중인 thread의 상태를 확인합니다.

3.13.1 프로그램 사용 예제

```
func void main()
int t_id
int th_st
t_id = 1
cthread(sum, t_id)          …… 1~10까지 누적 합을 구하는 함수를 thread로 실행
                            ……
while 1
dlay 1000
th_st = th_state(t_id)                  thread 상태 확인
if (th_st == 1)                         thread가 동작 하지 않는 경우
  cthread(sum, t_id)                    thread 생성
elif (th_st == 2)                       thread가 정지 중인 경우
  th_resume(t_id)                       thread 구동
elif (th_st == 3)                       thread가 구동 중인 경우
  th_suspend(t_id)                      thread 정지
end
dlay 1000
th_st = th_state(t_id)
if ( th_st != 1 && th_st != 0)
  ethread(t_id)
end
end

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

3.14 TAKE (로봇 선택 명령어)

기 능 로봇 선택

형 식 take 〈robot_id〉

용 어 robot_id : 구동할 robot의 id입니다. (1~3)

반환값 반환값은 없습니다.

설 명 thread에서 구동할 로봇을 선택합니다.
다른 thread에서 선택한 로봇 ID는 선택 할 수 없습니다.

3.14.1 프로그램 사용 예제

```
func void main()
take 1                          1번 로봇을 선택
while 1
jmov lp[0]
dlay 100
jmov lp[1]
dlay 100
end
release                         선택한 로봇을 해제합니다.
end
```

3.15 RELEASE(로봇 선택 해제 명령어)

기 능 로봇 선택 해제

형 식 release

용 어

반환값 반환값은 없습니다.

설 명 thread에서 take 명령을 통해 선택한 로봇을 해제 합니다.

3.15.1 프로그램 사용 예제

```
func void main()
take 1                          1번 로봇을 선택
while 1
jmov lp[0]
dlay 100
jmov lp[1]
dlay 100
end
release                         선택한 로봇을 해제합니다.
end
```

3.16 JMOV (PTP 이동 명령)

기 능 현위치에서 목표점으로 축보간 이동

형 식 jmov lp[번호], FOS, VEL, ACC, DEC
jmov 위치형변수

용 어 [번호] : 티칭한 위치 좌표 번호를 설정합니다.
lp - 로봇별로 개별 사용 되는 POINT
위치형 변수 : POS형으로 선언된 변수이름을 의미합니다.
다음의 데이터는 생략 가능하며, 현재 모션 명령이 종료되면 원래의 값으로 되돌아 갑니다.
FOS : fos 량
VEL : 이동시 속도
ACC : 이동시 가속도
DEC : 이동시 감속도

설 명 PTP(Point to Point)란 **점에서 점으로의 이동**을 의미합니다.
이동하는 경로는 로봇의 자세에 의존하며 **직선동작에 한정되지는 않습니다**.

3.16.1 프로그램 사용 예제

1) 포인트 티칭한 위치좌표로 이동

```
func void main()
take 1
vel(100)
jmov lp[10]          ……  포인트 LP10 으로 PTP 이동(fos 0, 속도 10%)
jmov lp[11], 10, 1000 ……  포인트 LP11 으로 PTP 이동(fos 10, 속도 100%)
jmov lp[100]         ……  포인트 LP100 으로 PTP 이동(fos 0, 속도 10%)
release
end
```

2) 위치형 변수를 이용한 이동 (1)

```
func void main()
pos a                          ……    포인트형 변수 A 선언
a=〈400.0,50.0,10.4,10.0,0,0〉  ……    A의 속성값을 지정
take 1
vel(100)
jmov lp[0]                     ……    포인트 LP10 으로 PTP 이동
jmov a                         ……    A 위치 좌표로 PTP 이동
jmov lp[100]                   ……    포인트 LP100 으로 PTP 이동
release
end
```

3) 위치형 변수를 이용한 이동(2)

```
func void main()
pos ap,ap1,ap2                 ……    포인트형 변수 AP,AP1,AP2를 선언
take 1
ap=lp[10] +〈10,10,10,10,0,0〉  ……    포인트 LP10의 각축값에 10을 더한 값을 저장
ap1=lp[11]-〈10,10,10,10,0,0〉  ……    포인트 LP11의 각축값에 10을 뺀 값을 저장
ap2=lp[100]                    ……    포인트 AP2에 LP100의 값을 저장
ap2.3=lp[100].3+10             ……    포인트 P100의 Z축값에 10을 더한 값을 저장
jmov ap                        ……    AP 위치 좌표로 PTP 이동
jmov ap1                       ……    AP1 위치 좌표로 PTP 이동
jmov ap2                       ……    AP2 위치 좌표로 PTP 이동
release
end
```

3.17 WITH, ENDWT (동시처리 명령어)

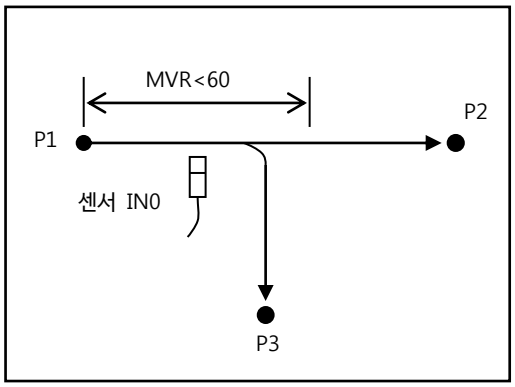
기 능 로봇 동작중 다음 수행문열을 동시처리 합니다.

형 식
```
with
  ...
end
```

설 명
1) WITH 문 내의 최초 **JMOV 명령어**를 수행하면서 이후 명령어를 병렬처리합니다.
2) MVR 변수를 사용하여 로봇이동거리를 백단위로 나누어 원하는 지점에서 조건식 등을 처리할 수 있습니다.

3.17.1 프로그램 사용 예제 (1)

1) 작업내용 →
P1에서 P2로 이동중 입력 IN0 신호가 ON되면 이동을 멈추고, P3로 이동.
(단, 센서입력신호는 P1에서 P2까지의 이동거리의 60% 까지만 검사한다.)



2) 로봇 사용자(USER) I/O

IN		OUT	
0	센서 감지	0	P2 도착완료
1		1	P3 도착완료

3) 프로그램 작성

```
func void main()
take 1
vel(500)
jmov lp[1]
mvr=0 ……                    "이동구간 백분율"을 초기화
with ……
jmov lp[2] ……                포인트 LP2로 이동(JMOV만 가능함)
while mvr<60 ……              이동구간의 60%미만 동안에
if in(0)==1 ……               IN0=1이면 "LABL BB"로 점프
goto BB
end
end
out(0)=1
end

labl BB
stop ……                     로봇이동 동작 정지
jmov lp[3]
out(1)=1
release
end
```

3.18 OUT, POUT (외부 출력 명령어)

기 능 비트단위 또는 포트단위로 지정된 값을 출력한다.

형 식 OUT(비트출력포트번호)=〈0 또는 1〉 [펄스유효시간] [→ 또는 ~]
OUT((정수형변수))=〈0 또는 1〉
POUT(출력포트번호)=〈데이터〉

용 어 〈비트출력포트번호〉: 비트출력 포트번호를 설정합니다. (0 ≤ 비트출력번호 ≤ 95)
1) 사용자 출력(User Output) : OUT(0) ~ OUT(15)
2) 확장1 출력(Option Output1) : OUT(16)~OUT(47) → 확장 I/O Card 1장 장착시 해당
3) 확장2 출력(Option Output2) : OUT(48)~OUT(79) → 확장 I/O Card 2장 장착시 해당
〈펄스유효시간〉: 해당 비트출력 포트에 0/1값을 출력(펄스출력)하는 유효시간.

[→ 또는 ~] :
-. 펄스유효시간을 주기로 갖는 주기파형 출력
-. 펄스출력, 주기파형 출력은 최대 32점(User, Option1, Option2 중 1개)까지 사용가능.
-. Port를 중복 사용할 수는 없음. (예) User OUT 16점 + Option OUT 16점)
1) "펄스 출력"과 "주기 파형출력"은 스텝 진행과 동시에 진행 됩니다.
즉, 다음 스텝이 이동명령(MOVE)인 경우 이동하면서 출력비트를 ON 또는 OFF 시킵니다.
2) "펄스출력"과 "주기파형출력"은 파라미터에서 사용할 Port를 정해야 합니다.
설정방법은 "파라미터 설명서를 참조하십시오.

〈출력포트번호〉: 출력포트 번호를 설정합니다. (0 ≤ 출력포트번호 ≤ 4)
-. 0일 경우 : OUT0 ~ OUT15 를 지정
-. 1일 경우 : OUT16 ~ OUT31 을 지정
-. 2일 경우 : OUT32 ~ OUT47 을 지정
-. 3일 경우 : OUT48 ~ OUT63 을 지정
-. 4일 경우 : OUT64 ~ OUT79 를 지정

〈데이터〉: 해당 출력포트에 출력할 값 (2진수, 16진수가능)을 의미합니다.
예1) POUT0=10 일 경우 (10(10진수) =〉 0000 0000 0000 1010 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.
예2) POUT0=0H0F0F 일 경우 (0H0F0F (16진수) =〉 0000 1111 0000 1111 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.

설 명 **OUT(비트출력포트번호)=〈0 또는 1〉 [〈펄스유효시간〉] [→]**

1) 지정된 펄스 유효시간 만큼 해당비트출력 포트에 0/1값을 출력합니다.
2) 펄스 유효시간이 없으면 계속 유효하며 펄스유효시간이 지나면 이전상태로 복귀합니다.
3) 펄스 유효시간의 단위는 10ms 입니다.

OUT((정수형 변수))=〈0 또는 1〉

"비트출력포트번호"에 정수형 변수를 사용할 수 있습니다.

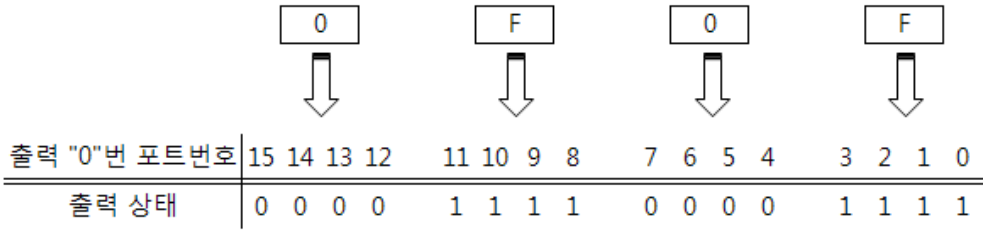
POUT(출력포트번호)=〈데이타〉

데이터 값을 해당 출력포트에 출력(16비트)합니다.

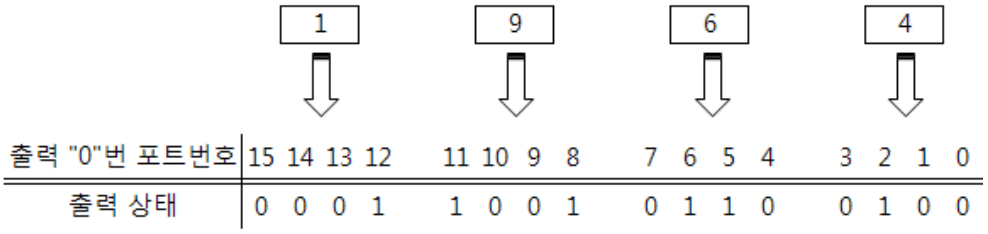
⚠CAUTION

- ▶ 출력포트 번호의 데이터값은 우측에서 좌측으로 0,1,2 ~ 15 의 순으로 증가 합니다.
- ▶ **내부 접점은 펄스 출력 제어가 되지 않습니다.**
- ▶ 확장 I/O 보드는 옵션 구매품 입니다.

[POUT0=0H0F0F]



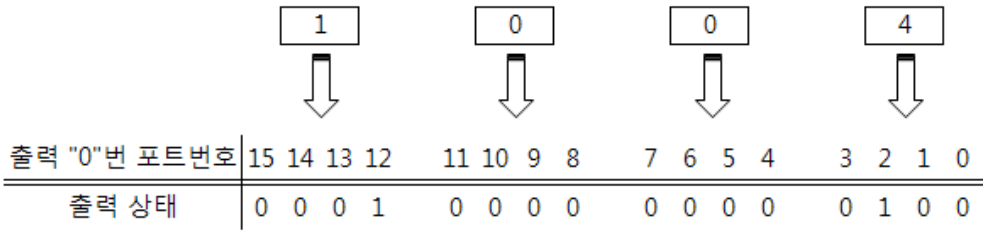
[POUT0=0H1964]



3.18.1 프로그램 사용 예제

1) 펄스출력, 데이터출력

```
func void main()
take 1
vel(100)
jmov lp[10]
out(11)=1, 100          ……….    출력 포트 11번을 100ms 동안 ON(1) 시킴
jmov lp[100]
pout(0)=0H1004          ……….    출력포트 0~15번에 0H1004(16진수)값으로 출력
release
end
```



2) 비트출력포트번호를 변수로 사용

```
func void main()
int a                   ……….    정수형 변수 A를 선언
for a=0 to 10           ……….    0 ~ 10까지 반복 수행을 하며
out(a)=0                ……….    OUT0,OUT1 ~ OUT10 의 출력을 모두 OFF(0)
end
end
```

3.19 IN, PIN (외부 입력 명령어)

기 능 지정된 비트단위 또는 포트단위의 입력에서 값을 받는다.

형 식 〈변수〉=IN〈입력비트번호〉
〈변수〉=PIN〈입력포트번호〉

용 어 (비트입력포트번호) : 비트입력 포트번호를 설정합니다.(0 ≤ 비트입력번호 ≤ 31)
1) 사용자 입력(User Input) : IN(0) ~ IN(31)
2) 확장1 입력(Option Input1) : IN(32) ~ IN(63) → 확장 I/O Card 장착시 해당

〈입력포트번호〉 : 입력포트 번호를 설정합니다.
PIN, WIN 인 경우 : 16bit 묶음(word)을 의미합니다.
-. 0일 경우 : IN0 ~ IN15 를 지정
-. 1일 경우 : IN 16 ~ IN31 을 지정
-. 2일 경우 : IN 32 ~ IN 47 을 지정
-. 3일 경우 : IN 48 ~ IN 63 을 지정

설 명 **IN(비트입력포트번호)**
지정된 비트입력포트의 ON/OFF상태(1또는 0)를 IN(비트입력포트번호)변수에 저장합니다.
PIN(입력포트번호)
지정된 입력포트의 값(16비트)을 읽어서 PIN〈입력포트번호〉변수에 저장합니다.

⚠CAUTION

▶ 입력포트 번호의 데이터값은 우측에서 좌측으로 0,1,2 ~ 15 의 순으로 증가 합니다.
▶ 확장 I/O 보드는 옵션 구매품 입니다.

[PIN(0)=0HFF0F]

	F			F			0			F		
입력 "0번" 포트 번호	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
입력 상태	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1

[PIN(0)=0H0009]

	0			0			0			9		
입력 "0번" 포트 번호	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
입력 상태	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

입력 "0"번 포트의 입력중 "IN4~IN7" 사이의 신호만 체크하려면?

[AA=0H00F0&PIN(0)]

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0H00F0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
& (AND 연산)																
PIN0 상태	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0
=																
변수"AA" 값	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
				0				0				3				0

3.19.1 프로그램 사용 예제

1) 입력 대기, Port입력을 정수변수처리 이동선택

```
func void main()
int d1                  ……  정수형 변수 D1 선언
take 1
vel(100)
while 1
jmov lp[0]
if in(0)==1             ……  비트 입력 포트 0번이 ON(1) 일경우
d1=pin(0)               ……  입력포트 0~15 까지의 상태를 D1에 저장
else
end
if d1==0HFF0F           ……  D1의 상태 (즉 PIN0의 상태)를 0HFF0F와 비교
jmov lp[1]
else
jmov lp[2]
end
release
end
```

2) BCD 유닛을 이용한 MODEL 비교

- BCD 입력은 IN0 ~ IN7까지 입력을 사용 한 예

```
func void main()
int m1,m2,md            ……  정수형 변수 M1,M2,MD를 선언
take 1
while 1
m1=pin(0) & 0H000F      ……  M1에 PIN0와 0H000F를 비교 하여 저장
m2=pin(0)&0H00F0        ……  M2에 PIN0와 0H00F0를 비교 하여 저장
m2=(m2)4)*10            ……  M2의 값을 1의 자리로 쉬프트 후 10을 곱함 주)
md=m1+m2                ……  연산된 M1과 M2의 값을 더함
if md==1
jmov lp[0]
else
if md==11
jmov lp[1]
end
end
release
end
```

CAUTION

▶ 포트 입력의 경우 16진수 연산을 하기 때문에 BCD 유닛등으로 10을 입력 시, 제어기에서는 16으로 인식 합니다.
이때는 상기 예제와 같이 쉬프트 연산 후 10진수 변환을 해주어야 정확히 계산 됩니다.

3.20 BIN (외부 입력 명령어)

기 능 지정된 바이트 단위의 입력에서 값을 받는다.

형 식 〈변수〉=bin〈입력바이트번호〉

용 어 〈입력바이트번호〉: 입력바이트 번호를 설정합니다.
8bit 묶음(byte)을 의미합니다.
-. 0일 경우 : IN0 ~ IN7 를 지정
-. 1일 경우 : IN 8 ~ IN15 을 지정
-. 2일 경우 : IN 16 ~ IN 23 을 지정
-. 3일 경우 : IN 24 ~ IN 31 을 지정

설 명 **bin(입력바이트번호)**
지정된 입력바이트의 값을 읽어서 변수에 저장합니다.

⚠ CAUTION

▶ 입력포트 번호의 데이터값은 우측에서 좌측으로 0,1,2 ~ 15 의 순으로 증가 합니다.
▶ 확장 I/O 보드는 옵션 구매품 입니다.

3.21 BOUT (외부 출력 명령어)

기 능 바이트단위로 지정된 값을 출력한다.

형 식 bout(출력바이트번호)=〈데이터〉
〈정수형 변수〉 = bout(출력바이트번호)

용 어 〈출력바이트번호〉 : 출력바이트 번호를 설정합니다. (0 ≤ 출력포트번호 ≤ 3)
확장 I/O를 사용하는 경우 (0 ≤ 출력포트번호 ≤ 7)
-. 0일 경우 : OUT0 ~ OUT7 를 지정
-. 1일 경우 : OUT8 ~ OUT15 을 지정
-. 2일 경우 : OUT16 ~ OUT23 을 지정
-. 3일 경우 : OUT24 ~ OUT31 을 지정

〈데이터〉 : 해당 출력바이트에 출력할 값 (2진수, 16진수가능)을 의미합니다.
예1) bout(0)=10 일 경우 (10(10진수) =〉 0000 1010 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.
예2) bout(0)=0H0F0F 일 경우 (0H0F (16진수) =〉 0000 1111 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.

설명 bout(출력바이트번호)=〈데이터〉
출력 바이트 번호에 데이터를 출력합니다.
〈정수형 변수〉 = bout(출력바이트번호)
출력 바이트 번호의 데이터를 읽어옵니다.

3.22 WIN(외부 입력 명령어)

기 능 지정된 워드 단위의 입력에서 값을 받는다.

형 식 〈변수〉=win〈입력워드번호〉

용 어 〈입력워드번호〉 : 입력워드 번호를 설정합니다.
16bit 묶음(word)을 의미합니다.
-. 0일 경우 : IN 0 ~ IN 15 를 지정
-. 1일 경우 : IN 16 ~ IN 31 을 지정
-. 2일 경우 : IN 32 ~ IN 47 을 지정
-. 3일 경우 : IN 48 ~ IN 63 을 지정

설 명 **win(입력워드번호)**
지정된 입력워드의 값을 읽어서 변수에 저장합니다.

3.23 WOUT (외부 출력 명령어)

기 능 워드단위로 지정된 값을 출력한다.

형 식 wout(출력워드번호)=〈데이터〉
〈정수형 변수〉 = wout(출력워드번호)

용 어 〈출력워드번호〉 : 출력워드 번호를 설정합니다. (0 ≤ 출력포트번호 ≤ 1)
확장 I/O를 사용하는 경우 (0 ≤ 출력포트번호 ≤ 3)
-. 0일 경우 : OUT0 ~ OUT15 를 지정
-. 1일 경우 : OUT16 ~ OUT31 을 지정
-. 2일 경우 : OUT32 ~ OUT47 을 지정
-. 3일 경우 : OUT48 ~ OUT63 을 지정

〈데이터〉 : 해당 출력바이트에 출력할 값 (2진수, 16진수가능)을 의미합니다.
예1) wout(0)=10 일 경우 (10(10진수) =〉 0000 0000 0000 1010 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.
예2) wout(0)=0H0F0F 일 경우 (0H0F0F (16진수) =〉 0000 1111 0000 1111 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.

설명 wout(출력워드번호)=〈데이터〉
출력 워드 번호에 데이터를 출력합니다.
〈정수형 변수〉 = wout(출력워드번호)
출력 워드 번호의 데이터를 읽어옵니다.

3.24 DIN(외부 입력 명령어)

기 능 지정된 더블워드 단위의 입력에서 값을 받는다.

형 식 〈변수〉=win〈입력더블워드번호〉

용 어 〈입력더블워드번호〉: 입력워드 번호를 설정합니다.
32bit 묶음(double word)을 의미합니다.
-. 0일 경우 : IN 0 ~ IN 31 를 지정
-. 1일 경우 : IN 32 ~ IN 63 을 지정

설 명 **din(입력더블워드번호)**
지정된 입력더블워드의 값을 읽어서 변수에 저장합니다.

3.25 DOUT (외부 출력 명령어)

기 능 더블워드단위로 지정된 값을 출력한다.

형 식 dout(출력더블워드번호)=〈데이터〉
〈정수형 변수〉 = dout(출력워드번호)

용 어 〈출력워드번호〉 : 출력워드 번호를 설정합니다. (0 ≤ 출력포트번호 ≤ 1)
확장 I/O를 사용하는 경우 (0 ≤ 출력포트번호 ≤ 2)
-. 0일 경우 : OUT0 ~ OUT31 를 지정
-. 1일 경우 : OUT32 ~ OUT63 을 지정

〈데이터〉 : 해당 출력바이트에 출력할 값 (2진수, 16진수가능)을 의미합니다.
예1) dout(0)=10 일 경우 (10(10진수) =〉 0000 0000 0000 1010 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.
예2) dout(0)=0H0F0F 일 경우 (0H0F0F (16진수) =〉 0000 1111 0000 1111 (2진수))
출력포트 1 번과 출력포트 3번을 ON(1) 시키며 다른 출력 포트는 OFF (0) 된다.

설명 dout(출력더블워드번호)=〈데이터〉
출력 더블워드 번호에 데이터를 출력합니다.
〈정수형 변수〉 = dout(출력더블워드번호)
출력 더블워드 번호의 데이터를 읽어옵니다.

3.26 필드버스용 출력 명령어(COUT, CBOUT, CWOUT, CDOUT, CFOUT)

기 능 필드 버스 카드로 비트단위 또는 포트단위로 지정된 값을 출력한다.
→ CC-LINK Card, ProfiBus Card 등.

형 식 COUT〈비트출력포트번호〉=〈0 또는 1〉 [펄스유효시간] [→ 또는 ~]
COUT〈(정수형변수)〉=〈0 또는 1〉
CBOUT〈출력포트번호〉=〈데이터〉
CWOUT〈출력포트번호〉=〈데이터〉
CDOUT〈출력포트번호〉=〈데이터〉
CFOUT〈출력포트번호〉=〈데이터〉

용 어 〈비트출력포트번호〉: 비트출력 포트번호를 설정합니다.(0 ≤ 비트출력번호 ≤ 799)
→ 사용자 출력(User Output) : COUT0 ~ COUT799

〈펄스유효시간〉: 해당 비트출력 포트에 0/1값을 출력(펄스출력)하는 유효시간.

[→ 또는 ~] :
-. 펄스유효시간을 주기로 갖는 주기파형 출력
-. 펄스출력, 주기파형 출력은 최대 32점(User, Option1, Option2 중 1개)까지 사용가능.
-. Port를 중복 사용할 수는 없음. (예) User OUT 16점 + Option OUT 16점)
1) "펄스 출력"과 "주기 파형출력"은 스텝 진행과 동시에 진행 됩니다.
즉, 다음 스텝이 이동명령(MOVE)인 경우 이동하면서 출력비트를 ON 또는 OFF 시킵니다.
2) "펄스출력"과 "주기파형출력"은 파라미터에서 사용할 Port를 정해야 합니다.
설정방법은 "파라미터 설명서를 참조하십시오.

〈출력포트번호〉: 출력포트 번호를 설정합니다. (0 ≤ 출력포트번호 ≤ 5)

1) CBOUT 인 경우 : 8bit 묶음(byte)을 의미합니다.
-. 0일 경우 : COUT0 ~ COUT7을 지정
-. 1일 경우 : COUT8 ~ COUT15를 지정
-. 2일 경우 : COUT16 ~ COUT23을 지정
-. 3일 경우 : COUT24 ~ COUT31을 지정
-. 4일 경우 : COUT32 ~ COUT39를 지정
-. 5일 경우 : COUT40 ~ COUT47을 지정

2) CWOUT인 경우 : 16bit 묶음(word)을 의미합니다.
-. 0일 경우 : COUT0 ~ COUT15 를 지정
-. 1일 경우 : COUT16 ~ COUT31 을 지정
-. 2일 경우 : COUT32 ~ COUT47 을 지정
-. 3일 경우 : COUT48 ~ COUT63 을 지정

3) CDOUT 인 경우 : 32bit 묶음(double word)을 의미합니다.
-. 0일 경우 : COUT0 ~ COUT31을 지정
-. 1일 경우 : COUT32 ~ COUT63을 지정
-. 2일 경우 : COUT64 ~ COUT95를 지정
-. 3일 경우 : COUT96 ~ COUT127을 지정

4) CFOUT 인 경우 : 실수영역의 저장공간을 의미합니다. (32bit)

〈데이터〉 : 해당 출력포트에 출력할 값 (2진수, 16진수가능)을 의미합니다.

설명

CAUTION

▶ Word 영역이 따로 존재하는 CC-LINK인 경우, CWOUT은 COUT 영역의 16bit 묶음이 아닌, Word 영역의 인덱스를 나타내므로 사용에 착오 없으시기 바랍니다.

3.27 필드버스용 입력 명령어(CIN, CBIN, CWIN, CDIN, CFIN)

기 능 필드 버스 카드에서 지정된 비트단위 또는 포트단위의 입력에서 값을 받는다.
→ CC-LINK Card, ProfiBus Card 등.

형 식 〈변수〉=CIN〈입력비트번호〉
〈변수〉=CBIN〈입력포트번호〉
〈변수〉=CWIN〈입력포트번호〉
〈변수〉=CDIN〈입력포트번호〉
〈변수〉=CFIN〈입력포트번호〉

용 어 〈비트입력포트번호〉: 비트입력 포트번호를 설정합니다.(0 ≤ 비트입력번호 ≤ 799)
→ 사용자 입력(User Input) : CIN0 ~ CIN799

〈입력포트번호〉: 입력포트 번호를 설정합니다.

1) CBIN 인 경우 : 8bit 묶음(byte)을 의미합니다.
-. 0일 경우 : CIN0 ~ CIN7을 지정
-. 1일 경우 : CIN8 ~ CIN15를 지정
-. 2일 경우 : CIN16 ~ CIN23을 지정
-. 3일 경우 : CIN24 ~ CIN31을 지정
-. 4일 경우 : CIN32 ~ CIN39를 지정
-. 5일 경우 : CIN40 ~ CIN47를 지정

2) CWIN 인 경우 : 16bit 묶음(word)을 의미합니다.
-. 0일 경우 : CIN0 ~ CIN15 를 지정
-. 1일 경우 : CIN 16 ~ CIN31 을 지정
-. 2일 경우 : CIN 32 ~ CIN 47 을 지정
-. 3일 경우 : CIN 48 ~ CIN 63 을 지정

3) CDIN 인 경우 : 32bit 묶음(double word)을 의미합니다.
-. 0일 경우 : CIN0 ~ CIN31을 지정
-. 1일 경우 : CIN32 ~ CIN63을 지정
-. 2일 경우 : CIN64 ~ CIN95를 지정
-. 3일 경우 : CIN96 ~ CIN127을 지정

4) CFIN 인 경우 : 실수영역의 저장공간을 의미합니다.(32bit)

설명

CAUTION

▶ Word 영역이 따로 존재하는 CC-LINK인 경우, CWIN은 CIN 영역의 16bit 묶음이 아닌, Word 영역의 인덱스를 나타내므로 사용에 착오 없으시기 바랍니다.

3.28 VEL(축 이동 속도 설정 명령어)

기 능　　축 이동 속도의 천분율을 설정

형 식　　vel(속도)

용 어　　〈속도〉: 로봇의 이동속도를 설정합니다. (0 ≤ 〈속도〉 ≤ 1000)
로봇이동속도 = Mv(각 축의 최대속도) * 0.001 * 속도
(0 ≤ 〈이동속도〉 ≤ 최대 RPM).

설 명　　1) 로봇이 이동할 때의 속도이며, **최대속도를 1000으로 설정**합니다.
2) 프로그램에서 속도 설정을 하지 않은 경우 INIT_V 파라미터에 설정된 속도로 **자동 설정** 됩니다.
3) 속도값을 갖는 **변수(정수형 변수)를 사용**할 수 있습니다.
4) **take 명령을 통해 로봇을 선택한 상태에서만 동작합니다.**

CAUTION

▶ 속도값은 1000 입니다.
VEL 1000을 입력 시 100%의 속도로 이동 됩니다. (VEL 1000 => 100%)
▶ 기구부의 허용 최대 RPM을 초과해서 사용시 소음 및 파손의 위험이 있습니다.
반드시 기구부에 부착된 라벨을 확인 후 사용 하시길 바랍니다.

3.28.1 프로그램 사용 예제

1) 작업위치에 따라 속도변경

```
func void main()
take 1                          로봇 선택
while 1
vel(1000)          ……           로봇 1 이동 속도 MV x 0.001 x 1000
jmov p[0]
jmov p[1]
vel(200)           ……           로봇 1 이동 속도 MV x 0.001 x 200
jmov p[2]
jmov p[3]
vel(1000)          ……           로봇 1 이동 속도 MV x 0.001 x 1000
jmov p[0]
end
release
end
```

3.29 ACC, DEC (가감속 설정 명령어)

기 능 가속/감속 시간의 천분율을 설정

형 식 acc 〈가속시간비율〉
dec 〈감속시간비율〉

용 어 〈가속시간비율/감속시간비율〉 : (1 ≤ 〈가감속시간비율〉 ≤ 2000)
-. 가감속시간을 프로그램에서 증가시키거나 감소시킵니다.
-. 가감속시간 = At(가감속시간) * 0.001 * 가감속비율
(0.5*최대가감속시간 ≤ 〈가감속시간〉 ≤ 2*최대가감속시간)

설 명 1) 로봇의 가감속시간을 설정합니다.
2) JOB프로그램 내부에서 가감속시간비율을 설정하지 않은 경우에는 100%로 설정 됩니다.
3) 가감속비율값을 갖는 변수(정수형 변수)를 사용할 수 있습니다.
4) 모든 축에 적용됩니다. (SCARA 로봇 : A, B, Z, W축 적용)

예) 파라미터 At : A축(0.3초), B축(0.3초), Z축(0.2초), W축(0.5초)
ACC(DEC) 50 실행
At : A축(0.15초), B축(0.15초), Z축(0.1초), W축(0.25초)
▶ 파라미터 값 자체가 수정되는 것이 아니고 JOB프로그램 내에서만 적용

5) 파라미터 가감속 시간(At)으로 복귀는 "ACC(DEC) 1000"을 사용하고 다음스텝부터 적용됩니다.
(예제 프로그램 참조)
6) **take 명령을 통해 로봇을 선택한 상태에서만 동작합니다.**

CAUTION

▶ 파라미터에 설정된 가감속 시간은 기계부 구성에 따라 최적으로 설정된 값입니다.

▶ ACC/DEC명령어의 가감속 시간비율이 100보다 작게 설정될 때는 기계부의 소음 및 진동이 발생할 수 있으니 주의하여 사용하십시오.

3.29.1 프로그램 사용 예제

1) 작업위치에 따라 가감속시간 변경

- 파라미터 에서 At 값이 0.3초로 설정 되어있을 경우의 예

```
func void main()
take 1                          로봇 선택
while 1
vel(500)
acc(500)            ……          가감속 0.15초로 설정
jmov lp[1]
acc(2000)           ……          가감속 0.6초로 설정
jmov lp[2]
jmov lp[3]
acc(1000)           ……          가감속 0.3초로 설정
jmov lp[0]
end
release
end
```

3.30 FOS (연속궤적생성 명령어)

기 능 축선단이 목표점에 도달하기전에 다음 목표점으로 궤적변경함

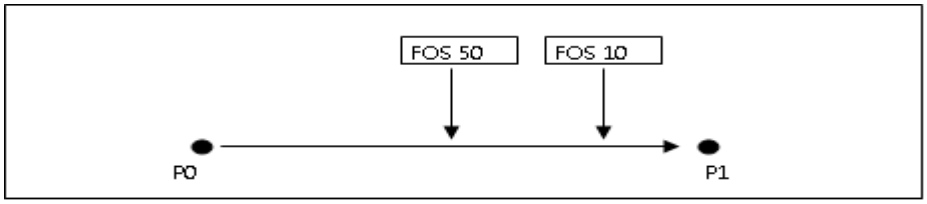
형 식 FOS 〈거리비율〉

용 어 〈거리비율〉 : (0 ≤ 〈가감속시간비율〉 〈 50%)

-. 로봇이 지정된 위치에 도착하기 전에 다음 위치로 이동시키거나 출력할 때의 시점을 설정합니다.

-. 거리비율은 두점(Point) 이동 거리에 대한 백분율값으로 설정합니다.

-. JMOV를 제외한 다른 모션에서는 50% 이상 사용금지
(0 ≤ 〈거리비율〉 〈 50)



위 그림은 P0에서 P1으로 이동할 때 각 FOS값에 따라 다음 위치로 이동하거나 출력할 때의 시점을 표시.

CAUTION

- ▶ 연속된 보간이동위한 포인트 티칭시 두 점 사이의 거리는 최소 5mm이상일 것 (단 100mm/s 이동 속도 인 경우)
- ▶ 이동중인 궤적의 FOS 비율이 다음 위치의 전체길이 보다 짧을경우 "Too much FOS" Alarm 이 발생 됩니다.

설 명

1) FOS가 설정되면, MOVE 명령은 지정된 위치에 도착하기 전 (설정된 거리 비율또는 거리값)에 **다음 위치로 이동**합니다.
2) OUT 명령인 경우에는 이동중에 출력이 가능합니다.
3) 작업 프로그램에서 FOS 적용 **이전의 상태로 복귀**하기위해 **"FOS 0"** 를 사용하고, 명령어는 1스텝 다음 명령어부터 적용됩니다.

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

FOS 0 명령어 적용은 jmov lp[2] 부터 됩니다.

CAUTION

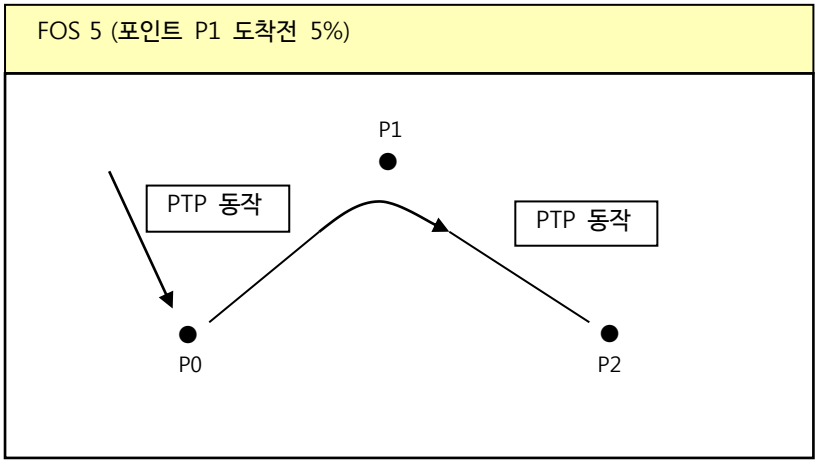
- FOS를 사용함으로 연속된 보간동작이 이루어 집니다.
- 연속궤적생성을 위한 두개의 모션의 속도차이가 심한경우 Auto RUN중 "Unreachable Point","Inverse Error" 가 발생할 수 있습니다.

3.30.1 프로그램 사용 예제

1) PTP 동작 (JMOV)

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

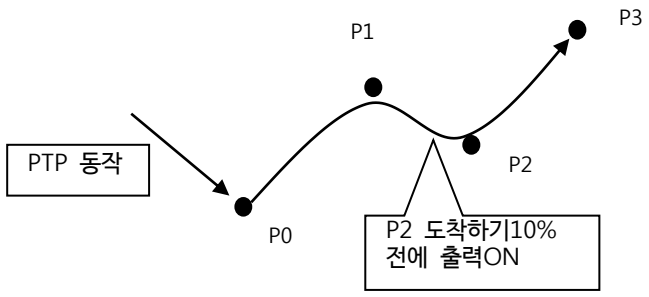
FOS 5 (포인트 P1 도착전 5%)



2) 로봇 이동중 출력신호 (OUT, POUT) On, Off

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

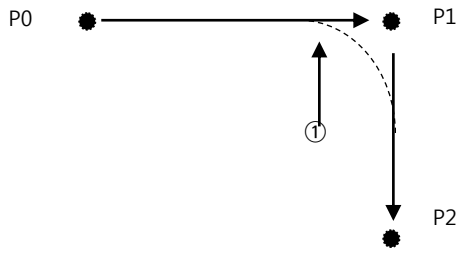
FOS 10



3) 로봇이동 중 다음스텝 명령어 처리

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

FOS 10



①위치에서
P1도착하기 10%전에 IF 명령어 처리
즉, 입력 "IN0"에 신호가 들어오면 ②(점선)와 같은
궤적을 그리며 이동

3.31 SVON, SVOF (서보 ON/OFF 명령어)

기 능 서보 ON, 서보 OFF

형 식
SVON(로봇) : 모든 축 서보 ON
SVON(로봇,지정축) : 지정 축 서보 ON
SVOF(로봇) : 모든 축 서보 OFF
SVOF(로봇,지정축) : 지정 축 서보 OFF

설 명
1) SVOF(로봇) :
-. 로봇 동작중 **서보를 OFF** 시켜 로봇 Body를 Free(손으로 로봇을 이동시킬 수 있는상태) 상태로 전환함.
-. 현재 구동중인 모든 축을 서보 OFF 시킴
2) SVON(로봇) :
-. 로봇 Body를 서보 ON 시킴, SVOF와 관련
-. SVOF 후 SVON시키지 않은 상태에서 로봇동작(RUN)은 Dry RUN(로봇 Body는 움직이지 않고 JOB 프로그램이 1Step씩 실행) 상태임.
-. 현재 서보 OFF된 모든 축을 서보 ON 시킴
3) SVON(SVOF)(로봇, 지정축) : 개별 축을 서보 ON/OFF 시킴. (범위:1~16)
SCARA 로봇 → A축 : 1, B축 : 2, Z축 : 3, W축 : 4
직각 로봇 → X축 : 1, Y축 : 2, Z축 : 3, W축 : 4

3.31.1 프로그램 사용 예제

1) PTP 동작 (JMOV)

```
func void main()
take 1
vel(500)
jmov lp[0]
if in(1)==1
svof(1, 4)            ……  4번째 축을 서보 OFF 시킴
end
jmov lp[1]            ……  4번째 축 서보 OFF 상태에서 포인트 LP1으로 이동
if in(1)==0
svon(1, 4)            ……  4번째 축을 서보 ON 시킴
end
jmov lp[2]            ……  모든 축이 서보 ON 상태로 포인트 LP2로 이동
end
release
end
```

2) PTP 동작 (JMOV)

```
func void main()
take 1
vel(500)
jmov lp[0]
mvr=0
with
jmov lp[1]
while mvr<100
if in(0)==1 ……                    LP1으로 이동중 IN0=1이면
stop
svof(2)                           로봇 정지 후 서보 OFF
goto L1
else
exit
end
end

labl L1
end
in(0)=1                           입력 IN0 가 "1"이 된 후
svon(2) ……                        서보 ON
jmov lp[2]
release
end
```

3.32 DLAY (시간지연 명령어)

기 능 지연시간 설정

형 식 dlay 〈지연시간〉

용 어 〈지연시간〉 : 지연하고자 하는 시간을 설정합니다. (ms)
(1 ≤ 〈지연시간〉 ≤ 999)

설 명 1) 설정시간을 숫자로 설정합니다.
2) 설정시간은 0.001초~9.99초까지 가능합니다.
3) 시간값을 가진 변수(정수형 변수)를 사용할 수 있습니다.

3.32.1 프로그램 사용 예제

1) PTP 동작 (JMOV)

```
func void main()
take 1
vel(100)
jmov lp[10]
dlay 100                ……        100ms(0.1초) 동안 LP10 위치에서 대기
jmov lp[100]
release
end
```

3.33 RSTATE (robot 상태 확인 명령어)

기 능 robot 상태 확인

형 식 정수형 변수 = rstate(robot, state index)

설 명

INDEX	STATE NAME	설명
1	ALARM	알람 상태 반환
3	ORIGIN	Origin 완료 상태 반환
6	SERVO ON	servo motor on/off 상태 반환
9	AUTO MODE	auto mode 상태 반환
11	EMG	Emergency 반환
14	ORIGIN ENABLE	Origin 기능 사용 여부 반환
15	ROBOT MOVING	로봇 모션 수행 여부 반환

3.33.1 프로그램 사용 예제

2) PTP 동작 (JMOV)

```
func void main()
int tmp
take 1
vel(100)
tmp = rstate(1,6)                    robot 1이 servo on 상태 확인
if( tmp == 0 )          ……….        servo on 상태가 아니면
  svon(1)                            servon on
end
jmov lp[100]
release
end
```

3.34 DIST(두 위치 변수의 차이값 명령어)

기 능 두 위치 변수 사이의 차이값 구하는 명령

형 식 위치형 변수 = dist(위치변수1, 위치변수2)

용 어

설 명 1) 위치변수1에서 위치변수2를 뺀 값을 반환합니다.
2) 반환된 데이터는 위치 데이터형 입니다.

3.34.1 프로그램 사용 예제

```
func void main()
pos tmp
take 1
tmp = dist(lp[0], lp[1])          lp[0]에서 lp[1]의 값을 뺀 값을 위치 변수 tmp에 저장
release
end
```

3.35 SWLIMIT(파라미터에 설정된 SW LIMIT 값을 읽어오는 명령어)

기 능 파라미터의 SW limit 값 읽기

형 식 실수형 변수 = swlimit(robot id, 축 번호, 최대최소여부)

용 어 robot id : sw limit을 읽어올 로봇 ID(1~3)
축 번호 : sw limit을 읽어올 축 번호(1~16)
최대최소여부 : sw limit의 최소값을 읽어올지 최대값을 읽어올지 선택
(0 : 최소값, 1 : 최대값)

설 명 1) 로봇 파라미터에 있는 BODY-RANGE에 설정된 sw limit 값을 읽어옵니다.

3.35.1 프로그램 사용 예제

```
func void main()
double tmp
tmp = swlimit(1, 1, 0)                    1번 로봇의 1번축의 sw limit 최소값을 읽어옵니다.
end
```

3.36 GETLP(Local point를 읽어오는 명령어)

기 능 Local point 값 읽기

형 식 위치형 변수 = getlp(robot id, local point index)

용 어 robot id : local point를 읽어올 로봇 ID(1~3)
local point index : local point의 index(0~39999)

설 명 1) 지정한 로봇의 local point를 읽어옵니다.
2) lp[index]를 통한 위치 데이터 읽기는 먼저 take 명령을 통해 로봇을 선택하여야 하지만, getlp 명령어는 로봇 선택이 안되어 있더라도 robot id 입력을 통해 local point의 데이터를 읽어 옵니다.

3.36.1 프로그램 사용 예제

```
func void main()
pos tmp
tmp = getlp(1, 120)          1번 로봇의 local point index 120번의 위치 데이터를 읽어옵니다.
end
```

3.37 SETLP(Local point에 저장하는 명령어)

기 능 Local point 값 쓰기

형 식 정수형 변수 = setlp(robot id, local point index, 위치형 변수)

용 어 robot id : local point를 저장할 로봇 ID(1~3)
local point index : local point의 index(0~39999)
위치형 변수:저장할 로봇 포인트 정보

설 명 1) 지정한 로봇의 local point에 포지션 정보를 저장합니다.
2) lp[index]를 통한 위치 데이터 쓰기는 먼저 take 명령을 통해 로봇을 선택하여야 하지만, setlp 명령어는 로봇 선택이 안되어 있더라도 robot id 입력을 통해 local point를 설정할 수 있습니다.

3.37.1 프로그램 사용 예제

```
func void main()
int ret
pos c_pos

c_pos.1=0.0
c_pos.2=0.0
c_pos.3=0.0
c_pos.4=0.0

ret = getlp(1, 120, c_pos)          1번 로봇의 local point index 120번지에 c_pos 위치 데이터를 저장합니다.

end
```

3.38 변수

3.38.1 변수의 종류

변수는 일반 변수, POSITON 변수, 시스템 변수로 나눌 수 있습니다.

		LOCAL	GLOBAL
일반 변수	정수형	INT	GI(번호)
	실수형	DOUBLE	GF(번호)
POSITION 변수		POS P(번호)	
시스템 변수		CNT,TMR,MVR,HERE	

-. LOCAL 변수 - 현재 구동중인 프로그램 내에서만 사용 가능 한 변수

1) INT : 정수형 변수로 산술연산, 입/출력 DATA 임시저장 등에 사용
2) DOUBLE : 실수 형 변수로 산술 연산, POINT DATA 연산 등에 사용
3) POS : 위치형 변수로 단순 변수와 배열 변수로 구분
4) CNT : 입력 포트를 통해 들어오는 펄스 입력을 업카운트(UP COUNT) 하여 저장
5) TMR : 파라미터 설정값에 따라 초기 화면 값에서 1씩 증가
6) MVR : 두 POINT 사이를 이동 하는 경우, 전체 거리에 대한 현재 위치의 백분율 값
7) HERE : 현재 축의 각도 값을 갖고 있는 변수
8) P(번호) : 사용자가 티칭한 위치 값을 가지고 있는 변수

-. GLOBAL 변수 - 전체 프로그램 및 채널에서 공통으로 사용 가능 한 변수

1) GI : 정수형 변수로 INT로 선언된 LOCAL 변수와 동일한 기능 (0 ≤ 번호 ≤ 999)
2) GF : 실수형 변수로 REAL로 선언된 LOCAL 변수와 동일한 기능 (0 ≤ 번호 ≤ 999)

3.38.2 사용 방법

-. 사용위치

-. 프로그램에서 사용하는 일반변수는 프로그램 시작부분인 **"main" 다음에 선언**
-. 글로벌 변수는 함수 밖에서 선언합니다.

-. 구성

-. 일반변수의 **변수명**은 영문자와 숫자로 구성된 **문자** 스트링입니다.
-. 변수의 첫자는 반드시 **영문자이거나 '_'**입니다. (숫자 사용 불가)
단, **"P", "GI", "GF"** 등은 Point 변수 및 GLOBAL 변수와 중복되므로 **사용할 수 없습니다**.

-. 초기화

-. 변수는 **초기화 후 사용**해야 합니다.
-. CNT, TMR 변수는 변수사용 **이전 STEP에서 초기화**를 해야합니다.
-. MVR 변수는 **MOVE 명령어가 시작되면 0으로 초기화** 됩니다.

3.39 정수형(INT, GI), 실수형(DOUBLE, GF), 문자열(STRING) 변수

3.39.1 LOCAL 변수

기 능 프로그램 내부에서 사용 가능한 정수형, 실수형, 문자열 변수의 선언

형 식
int 〈변수명〉,〈변수명〉,---
double 〈변수명〉,〈변수명〉,---
string 〈변수명〉,〈변수명〉,---

용 어
〈변수명〉 :
-. 영문자와 숫자로 구성된 문자 스트링입니다.
-. 변수의 첫자는 반드시 영문자 이거나 '_'입니다.

설 명
1) **함수 블록내** 에서 선언하여야 합니다.
2) int로 선언된 변수는 정수형 변수 (정수값 범위 : $-2.14 \times 10^{9} \sim 2.14 \times 10^{9}$)
3) double로 선언된 변수는 실수형 변수 (실수값 범위 : $\pm 8.4 \times 10^{-37} \sim \pm 3.4 \times 10^{38}$)

CAUTION

▶ **"P", "GP", "GI", "GF"** 등은 Point 변수 및 GLOBAL 변수와 중복되므로 **사용할 수 없습니다**.
(**"Syntax Error" , "Duplicated Symbol"** 등 ALARM 발생)

▶ 실수형 변수와 정수형 변수의 연산을 혼합 사용시 **실수형으로 변경** 됩니다.
9.5 + **10** = 19.5

3.39.2 프로그램 사용 예제

1) 정수형, 실수형 변수 사용

```
func void main()
int j,bot            ……  정수형 변수 J 와 BOT를 선언
double ff1           ……  실수형 변수 FF1을 선언
while 1
bot=0H000F
j=1                  ……  변수 초기화
ff1=10.5
take 1
for j=0 to 10        ……  AP1 위치 좌표로 CP 이동
jmov lp[1]           ……  AP2 위치 좌표로 CP 이동
jmov lp[2]
pout(0)=bot          ……  정수형 변수 BOUT을 사용 하여 출력
end
ff1=ff1+100.0            실수형 변수 산술 연산
end
release
end
```

3.39.3 GLOBAL 변수

기 능 프로그램 내부에서 사용 가능한 정수형, 실수형, 문자열 변수의 선언

형 식 int 〈변수명〉,〈변수명〉,---
double 〈변수명〉,〈변수명〉,---
string 〈변수명〉,〈변수명〉,---

용 어 〈변수명〉 :
-. 영문자와 숫자로 구성된 문자 스트링입니다.
-. 변수의 첫자는 반드시 $로 시작하여야 합니다.
Ex) int $int_var

설 명 1) **함수 블록밖**에서 선언하여야 합니다.
2) int로 선언된 변수는 정수형 변수 (정수값 범위 : $-2.14 \times 10^{9} \sim 2.14 \times 10^{9}$)
3) double로 선언된 변수는 실수형 변수 (실수값 범위 : $\pm 8.4 \times 10^{-37} \sim \pm 3.4 \times 10^{38}$)

CAUTION

▶ **"P", "GP", "GI", "GF"** 등은 Point 변수 및 GLOBAL 변수와 중복되므로 **사용할 수 없습니다**.
(**"Syntax Error" , "Duplicated Symbol"** 등 ALARM 발생)

▶ 실수형 변수와 정수형 변수의 연산을 혼합 사용시 **실수형으로 변경** 됩니다.
9.5 + **10** = 19.5

3.39.4 프로그램 사용 예제

```
int $gint_var                          Global 정수형 변수 $gint_var 선언
func void main()
int l                    ……          정수형 변수 L,M,H를 선언
                         …
end
```

3.39.5 정수형 GLOBAL 변수 GI

기 능 int로 선언된 정수형 변수와 동일한 기능을 가지며 전체 프로그램에서 공통으로 사용 가능한 변수 입니다.

형 식 gi[정수/변수] : 사용 가능한 정수/변수의 범위는 0~999 까지입니다.
→ gi와 [정수/변수] 사이에 공란(BLANK) 이 있으면 안됩니다.

설 명
1) LOCAL 변수와 달리 **프로그램에서 선언하지 않고** 사용합니다.
2) 프로그램에서 직접 gi[정수]의 값을 변경할 수 있습니다.
3) VEL, ACC, DEC, FOS, DLAY 에 사용할 수 있습니다.
   예) vel gi[20] : I20 에 저장된 값으로 속도(Velocity) 를 설정합니다.
4) IF – (ELSE) – END 블록과 WHILE-END 블록에서 사용할 수 있습니다.
   예) if gi[7] 〉A0 예) while gi[4+7] ≤ 100
5) gi(정수/변수)형은 포인트 전역변수(Global Point Variable)에 사용할 수 있습니다.
   예) gi[9] 가 3 이라면 P[gi[9]] 는 p[3] 과 같습니다.

3.39.6 프로그램 사용 예제

```
func void main()
int l,m,h                ……        정수형 변수 L,M,H를 선언
                         …
l=gi[0]                  ……        L=30 할당
                         …
m=gi[1]                  ……        M=60 할당
                         …
h=gi[2]                  ……        H=100 할당
                         …
take 1
vel(1, h)                ……        속도를 H(100) 으로 지정
                         …
jmov lp[0]
while gi[10]〈gi[11]     ……        GI[10]〈GI[11] 을 만족하는 동안
                         …
vel(1, m)                          속도를 M(60) 으로 지정
jmov lp[gi[25]]          ……        P[GI[25]] 로 이동
                         …
vel(1, l)                ……        속도를 L(30) 으로 지정
                         …
jmov lp[gi[26]]          ……        P[GI[26]] 로 이동
                         …
gi[10]=gi[10]+1          ……        카운터 I10을 1증가
                         …
end
release
end
```

GI 변수 값

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

3.39.7 실수형 GLOBAL 변수 GF

기 능 double로 선언된 실수형 변수와 동일한 기능을 가지며 전체 프로그램에서 공통으로 사용 가능한 변수 입니다.

형 식 gf[정수/변수] : 사용 가능한 정수/변수의 범위는 0~999 까지입니다.
→ gf와 [정수/변수] 사이에 공란(BLANK) 이 있으면 안됩니다.

설 명
1) LOCAL 변수와 달리 **프로그램에서 선언하지 않고** 사용합니다.
2) 프로그램에서 직접 gf[정수/변수]의 값을 변경할 수 있습니다.
3) IF - (ELSE) - END 블록과 WHILE-END 블록에서 사용할 수 있습니다.
예) IF F7 〉A0 THEN 예) WHILE F49 ≤ 100

CAUTION

▶ 실수형 변수와 정수형 변수의 연산을 혼합 사용시 **실수형으로 변경** 됩니다.
9.5 + **10** = 19.5

3.39.8 프로그램 사용 예제

```
func void main()
pos pp
take 1
pp=lp[1]
for gi[0]=0 to 10
pp.1=gf[0]          ……  위치형 변수 pp1 의 1번째축에 F0의 값을 대입
jmov pp             ……  pp의 위치로 이동
gf[0]=gf[0]+10.0    ……  F0 값을 10.0씩 증가
end
release
end
```

3.40 POSITION 변수

3.40.1 POS 변수

기 능 위치형 변수 선언 (단순변수 및 배열변수)

형 식 단순변수 → pos 〈변수명〉, 〈변수명〉, …
배열변수 → pos 〈변수명〉[크기], {〈변수명〉[크기]}, …

용 어 〈변수명〉 :
-. 변수명은 **영문자와 숫자로 구성**된 문자 스트링 및 배열의 크기입니다.
-. 변수의 **첫자는 반드시 영문자이거나 '_'**입니다.("P" 문자 단독사용 제외)
[크기] : 배열변수의 크기는 대괄호안의 숫자(정수)로 나타냅니다.
배열 변수의 선언의 괄호안의 숫자는 **방의 개수**를 나타냅니다.
예) POS AA[3] 선언시 - 사용 가능한 변수는 AA[0], AA[1], AA[2] 3개.

설 명 1) **MAIN 블록의 시작** 에서 선언하여야 합니다.
2) POS로 선언된 변수값은 각축의 각도, 리드값입니다.
3) POS로 선언된 변수는 각 **성분값**은 **〈변수명〉.〈번호〉**로 사용할 수 있습니다.
예) POS JP1 → JP1.1 JP1.2 JP1.3 JP1.4…
4) 각도값 POS 변수는 **최대 16가지 성분**을 가지며,
변수명(단순변수, 배열변수)=
〈1축성분, 2축성분, 3축성분, 4축성분, 5축성분, 6축성분〉
형식으로 초기화할 수 있습니다.
5)

CAUTION

▶ **"P", "GP", "GI", "GF"** 등은 Point 변수 및 GLOBAL 변수와 중복되므로 **사용할 수 없습니다**.
(**"Syntax Error" , "Duplicated Symbol"** 등 ALARM 발생)

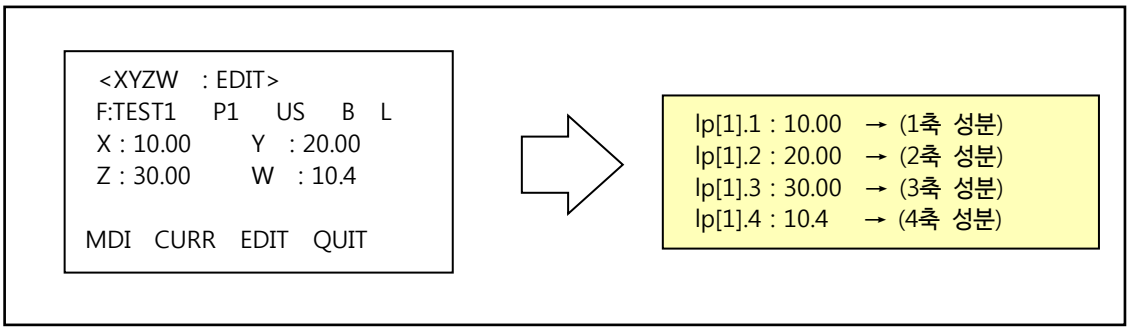
▶ 확장성을 고려한 최대 16축으로 구성되어 있으며, **사용 하지 않는 축**에 대해서는 **0.0값을 입력** 하여야 합니다.

3.40.2 **POINT 변수**

기 능 Job Editor에서 사용자가 Teaching한 모든 Point를 사용

형 식 로봇별 변수 → LP(번호) (0 ≤ 번호 ≤ 39999)

설 명 1) POS변수와 달리 초기에 변수 선언을 하지 않습니다.
2) POINT 변수는 각 **성분값**은 **〈포인트번호〉.〈번호〉**로 사용할 수 있습니다.



3.40.3 **프로그램 사용 예제**

1) POINT변수, POSITION변수를 활용한 감속 포인트 적용

```
func void main()
pos mm                                  ……  위치형 변수 MM을 선언
pos a, b                                    위치형 변수 A, B를 선언
mm=〈10.1,10.2,10.3,10.4,0,0〉           ……  MM을 초기화
a=〈400.0,50.0,10.0,0.0,0.0,0.0〉        ……  A를 초기화
b=a                                     ……  B를 A 값과 동일하게 초기화
b.3=a.3-40.0                            ……  B의 3번째축값을 40mm 감산
take 1
-------------------------------------------------------------
vel(1000)
jmov lp[0]                              ……  LP0 위치로 PTP 이동
jmov b                                  ……  B 위치 좌표로 이동 (감속 포인트)
vel(300)
jmov a                                  ……  A 위치 좌표로 이동
jmov mm                                 ……  위치형 변수 MM으로 이동
vel(1000)
jmov lp[0]
-------------------------------------------------------------
release
end
```

3.41 시스템 변수 (CNT, TMR, MVR, HERE)

3.41.1 CNT, TMR 변수

기 능 CNT : 카운터값 저장.
TMR : 타이머값 저장

형 식 cnt(펄스입력 BIT 번호)=〈초기값〉
tmr(0)=〈초기값〉
tmr(1)=〈초기값〉

용 어 〈펄스입력 BIT 번호〉: 입력으로 받고자 하는 입력 BIT 번호
〈초기값〉: 카운터(CNT), 타이머(TMR)가 시작되는 정수값

설 명
1) 시스템 변수는 **자료형을 선언하지 않고** 사용합니다.
2) CNT 변수는 **펄스 입력 BIT가 결정되는 순간 "0"이 저장**되며
그후 **매 펄스입력(10m 이상의 펄스)**을 카운트합니다. (0 ≤ 입력범위 ≤ 65,535)
3) TMR변수는 **정수값을 입력하는 순간**부터 값이 할당되어 시스템 파라미터에서 정의된 시간간격으로 1씩 증가합니다.
설정 값 적용은 **파라미터 모드의 TMR을 참조** 바랍니다.

CAUTION

▶ CNT와 펄스입력포트번호 사이 ,TMR와 "0" 또는 "1" 사이에 공란(BLANK)이 들어가지 않도록 주의 하시기 바랍니다.
예) CNT_(0)=0 (X), TMR_(1)=-100 (X)

3.41.2 프로그램 사용 예제

1) CNT, TMR 변수 사용

```
func void main()
vel(1, 100)
cnt(0)=0            ……  카운터 초기화
tmr(1)=-100         ……  타이머 초기화
while cnt(0)<20     ……  입력 BIT "0"번에 펄스가 20개 입력될때가 LOOP
jmov lp[0]
if (tmr(1)>0)       ……  타이머가 1초 이상 이면
goto TEST                (-100 ~> 0 : 1ms가 100개 발생)
end
dlay 100
jmov lp[1]
end
labl TEST
end
```

3.41.3 MVR 변수

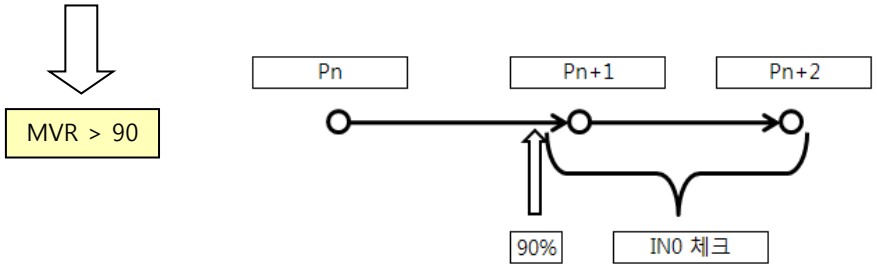
기 능 전체거리에 대해 이동거리에 대한 비율 저장.

형 식 mvr 〈전체거리에 대한 이동거리의 백분율값〉

설 명
1) 두 Point (Pn, Pn+1) 사이를 이동하는 경우 전체거리에 대한 이동거리의 백분율 값
즉, MVR = (이동거리/전체거리)*100
2) 로봇동작중 조건분기 및 I/O병렬 처리에 사용.
3) 이동구간을 100단위로 나누어 반드시 WITH문과 병행하여 사용.
4) 두 Point (Pn, Pn+1)의 이동거리가 짧고 고속으로 운전할 때는 MVR값 설정에 유의바랍니다.

⚠ CAUTION

▶ **Pn Point에서 Pn+1 Point로 이동 중 90% 이후에 입력 "IN1"를 확인하는 경우**



-. 로봇이 Pn+1위치의 90%정도 도달하였을 때는 다음 이동 POINT(Pn+2) 이동을 위한 연산을 시작합니다.

-. 즉,위의 그림과 같이 로봇이 90%위치에 있어도 **현재 #1 위치를 연산중**입니다.

★ 따라서 이동거리가 짧고 고속으로 운전할 때에는 Pn출발 후 90% 이후에는 입력 Bit'0'의 상태를 확인할 수 없는 경우도 발생합니다.
이경우 로봇 이동 속도를 낮추어 사용해 보십시오.

⚠ CAUTION

이동 중 신호가 입력 되었을 때 현재좌표 READ(HERE 명령어)기능 사용

-. 로봇 이동중 신호가 입력되었을 때 정지 후 현재좌표 READ 기능은 로봇이동 속도에 따라 현재좌표 READ한 값에 편차가 발생할 수 있습니다.

-. 이 경우 로봇의 이동 속도를 낮추어 사용하십시오.

3.41.4 프로그램 사용 예제

1) 이동중 입력Bit 상태에 따라 이동 궤적 수정.

```
func void main()
vel(1, 10)
fos10
jmov lp[100]
mvr=0
with
jmov lp[0]
while mvr<100          ……  LP0포인트를 100% 이동중
if mvr>10              ……  이동거리가 10% 위치에서
out(1)=1, 50           ……  출력을 0.5초 동안 ON(1)으로 유지
end
in(1)=1
goto L1
end
labl L1
stop
end
jmov lp[1]
end
```

2) P1 포인트 출발하여 80%이후에 입력 “IN0”의 신호입력에 따라 P2 포인트로 이동

```
func void main()
vel(1, 10)
while in(1)==1
plup 10
jmov lp[0]
mvr=0
with
jmov lp[1]
while mvr<100                  ……  LP1포인트를 100% 이동중
if (mvr>80)&&(in(0)==1)        ……  이동거리가 80% 이고, IN0가 ON(1)된 경우
plup 0
jmov lp[22]
goto BB
end
end
end
in(0)=1
plup 0
jmov lp[2]
labl BB
end
out(1)=1, 50
end
end
```

3.41.5 HERE 변수

기 능 현재위치 값을 저장하는 변수

형 식 HERE
Here(정수)

용 어 〈정수〉: 채널 번호 (1 또는 2)
-. 채널간 로봇의 위치를 읽어올 경우 사용 합니다.

설 명 **1) 로봇의 현재 위치 값을 각도값 또는 리드값 으로 저장**
2) 타 채널에 있는 로봇의 현재 위치를 저장.

3.41.6 프로그램 사용 예제

1) HERE 변수를 활용 하여 현재 위치를 READ

```
func void main()
vel(1, 100)
pos curr
jmov lp[0]
mvr=0
with
jmov lp[1]
while mvr〈100
if mvr〉50                          ……  LP1 포인트로 이동 중 50% 지점에서
curr=here                          ……  현재 위치값을 위치형 변수 CURR에 저장
goto LLL
end
end
labl LLL
end
jmov lp[2]
jmov curr
end
```

2) HERE〈n〉변수 활용 타 채널 로봇의 현재위치를 READ

```
func void main()
vel(1, 10)
pos ap
jmov lp[0]
while ap.1〈10.0
ap=here2                           ……  2번 채널 로봇의 현재위치를 저장
dlay 10
end
out(0)=1
jmov lp[1]
end
```

3.42 상수

기 능 16진수, 2진수 표시.

형 식 0H〈숫자〉
0B〈숫자〉

용 어 〈숫자〉: 숫자 앞에 0H 또는 0B가 없으면 10진수
* 0H : 16진수
* 0B : 2진수

설 명 1) 0H〈숫자〉는 16진수이며, 숫자는 8자리까지 가능합니다.
2) 0B〈숫자〉는 2진수이며, 숫자는 16자리(16비트)까지 가능합니다.
3) 0H나 0B가 없으면, 10진 정수 또는 실수입니다.

3.42.1 프로그램 사용 예제

```
func void main()
int aa
vel(1, 100)
while 1
aa=pin(0) & 0H00FF                  입력포트0번의 값을 16진수 00FF 와 비교
if aa==1
pout(0)=0B00110100                  출력포트0번에 2진수 00110100 출력
md1()
end
if aa==1
pout(0)=0B01110100                  출력포트0번에 2진수 01110100 출력
md2()
end
end
end
```

3.43 연산자

3.43.1 배정 연산자

명령어	기 능
=	오른쪽 수식을 평가하여 그결과를 왼쪽변수의 변수형에 맞게 변환하여 **배정**합니다. (예: IN0=1)

	정 수 형	실 수 형	위 치 형	카운터형	타이머형
정수형	정 수 형	정 수 형	불 가	정 수 형	정 수 형
실수형	실 수 형	실 수 형	불 가	실 수 형	실 수 형
위치형	불 가	불 가	위 치 형	불 가	불 가
카운터형	정 수 형	정 수 형	불 가	정 수 형	정 수 형
타이머형	정 수 형	정 수 형	불 가	정 수 형	정 수 형

3.43.2 산술 연산자

명령어	기 능
*,/,+,-,%	서로 다른 자료형에 대한 각 연산자의 연산결과는 다음표와 같습니다. 표에서 '불가'는 **연산이 이루어질수 없는 수식**의 경우를 말합니다.

▶ **+, - 연산자**

	정 수 형	실 수 형	위 치 형	카운터형	타이머형
정수형	정 수 형	실 수 형	불 가	정 수 형	정 수 형
실수형	실 수 형	실 수 형	불 가	실 수 형	실 수 형
위치형	불 가	불 가	위 치 형	불 가	불 가
카운터형	정 수 형	정 수 형	불 가	정 수 형	정 수 형
타이머형	정 수 형	정 수 형	불 가	정 수 형	정 수 형

▶ * 연산자

	정 수 형	실 수 형	위 치 형	카운터형	타이머형
정수형	정 수 형	실 수 형	위 치 형	정 수 형	정 수 형
실수형	실 수 형	실 수 형	위 치 형	실 수 형	실 수 형
위치형	위 치 형	위 치 형	불 가	위 치 형	위 치 형
카운터형	정 수 형	실 수 형	위 치 형	정 수 형	정 수 형
타이머형	정 수 형	실 수 형	위 치 형	정 수 형	정 수 형

▶ / 연산자

	정 수 형	실 수 형	위 치 형	카운터형	타이머형
정수형	정 수 형	실 수 형	불 가	정 수 형	정 수 형
실수형	실 수 형	실 수 형	불 가	실 수 형	실 수 형
위치형	위 치 형	위 치 형	불 가	위 치 형	위 치 형
카운터형	정 수 형	실 수 형	불 가	정 수 형	정 수 형
타이머형	정 수 형	실 수 형	불 가	정 수 형	정 수 형

▶ % 연산자

정수간 연산만이 가능하며 나눗셈 연산에서 **몫은 버리고 나머지를 결과 값**으로 갖습니다.
예) AA=10%3 일경우 **AA의 값은 1**

3.43.3 관계 연산자

명령어	기 능
〉, 〈, ≤, ≥, ==, !=	-. 주로 정수형과 실수형 자료를 대상으로 하며 IF 문과 WHILE 문의 조건식에 사용됩니다. -. 관계 연산의 결과는 논리값 (true,false 또는 0/1)입니다

3.43.4 논리 연산자

명령어	기 능
&&, ||, !	논리연산자는 논리값만을 대상으로 합니다. 즉 관계연산의 결과를 피연산자로 사용합니다. 논리연산의 결과는 논리값 입니다.

3.43.5 비트 연산자

명령어	기 능
&, | 〈〈, 〉〉 ~	논리연산자 이동연산자 1 의 보수

3.44 내장 함수

함 수	기 능	사 용 예
EXP(X)	지수 e^x	EXP(3) 은 20.085
LOG(X)	상용로그 $\log_{10}X$	LOG(100) 은 2
LN(X)	자연로그 $\log_eX$	LN(15)는 2.708
SQRT(X)	제곱근 $\sqrt{}$	SQRT(16)은 4
POW(x,y)	x^y	POW(2,10)은 1024
ABS(K)	정수 K 의 절대값	ABS(-12)는 12
RND(X)	실수 X 를 반올림하여 정수값	RND(97.62)는 98
SIN(X)	sine, x 단위 : radian	SIN(RAD(30))는 0.5
COS(X)	cosine, x 단위 : radian	COS(0)는 1
TAN(X)	tangent, x 단위 : radian	TAN(RAD(45))는 1.0
ASIN(X)	arcsine, - π/2~π/2 의 값 반환	DEG(ASIN(0.5))는 30
ACOS(X)	arccosine, 0 ~ π의 반환	DEG(ACOS(0.5))는 60
ATAN(X)	arctangent, -π/2 ~π/2 의 값 반환	DEG(ATAN(1.0))은 45
ATAN2(Y,X)	제 2 arctangent, -π~π 의 값 반환	DEG(ATAN2(-1,-1))은 -135
DEG(X)	radian 을 angle 값으로 변환	DEG(3.1416)은 180.0
RAD(X)	angle 을 radian 값으로 변환	RAD(180.0)는 3.1416
MIN(X,Y)	x, y중 최소값을 반환	MIN(2,5)는 2
MAX(X,Y)	x, y중 최대값을 반환	MAX(2,5)는 5

3.45 문자열

3.45.1 ASC

기 능　　문자열 첫문자를 캐릭터 코드로 반환

형 식　　문자열변수 = asc(문자열)

용 어　　문자열변수 :
　　-. string을 사용하여 선언한 변수를 의미하며, 문자열을 다루기 위해 사용합니다.

　　(문자열) : 문자열 변수나 문자열 상수를 의미합니다. 문자열 상수는 “”로 둘러싸인 문자들의 집합입니다.(ex) “ABCDEF”)

설 명　　1) ASC에 입력된 문자열의 첫문자를 반환합니다.

프로그램 예제

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

문자열 변수 AA에 “XYZW” 대입

문자열 변수 AA에서 첫번째 문자 ‘X’ 반환

3.45.2 BINS

기 능　　정수를 이진수 문자열로 변환

형 식　　문자열변수 = bins(정수)

용 어　　문자열변수 :
　　-. string을 사용하여 선언한 변수를 의미하며, 문자열을 다루기 위해 사용합니다.

설 명　　1) 입력한 정수를 이진수(0/1) 문자열로 변환합니다.

프로그램 예제

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

정수형 변수 b의 값을 이진수 문자열 형태로 문자열 변수 aa에 저장합니다.

3.45.3 CHR

기 능　　아스키 코드 범위의 정수를 문자로 변환

형 식　　문자열변수 = chr(정수)

용 어　　문자열변수 :

-. string을 사용하여 선언한 변수를 의미하며, 문자열을 다루기 위해 사용합니다.

(정수) : 아스키 코드값 내의 정수(0~127)

설 명　　1) ASC에 입력된 정수에 해당하는 아스키 코드를 반환합니다.

프로그램 예제

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

문자열 변수 AA에 아스키코드 65에 해당하는 'A' 대입

3.45.4 FLUSH

기 능　　입력, 출력 버퍼 클리어

형 식　　FLUSH(정수)

용 어　　(정수) : 클리어 버퍼 선택(1~3)
- * 1: 입력 버퍼 클리어
- * 2: 출력 버퍼 클리어
- * 3: 입력, 출력 버퍼 클리어

설 명　　1) 시리얼 통신 송수신에 사용하는 입출력 버퍼를 클리어합니다.

프로그램 예제

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

Serial 통신으로 입력을 받아 문자열 AA에 저장

입력 버퍼 클리어

3.45.5 FTOS

기 능　　정수나 실수를 문자열로 변환

형 식　　문자열변수 = ftos(정수 or 실수)

용 어　　(정수 or 실수) : 문자열로 변환할 정수나 실수

설 명　　1) 정수나 실수 수치 데이터를 문자열로 변환합니다.
ex) 1234 → “1234”

프로그램 예제

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

1234를 문자열 “1234”로 변환후 문자열 변수 AA에 대입

-123.456을 문자열 “-123.4.56”로 변환후 문자열 변수 BB에 대입

3.45.6 HTOS

기 능　　정수를 16진수의 문자열로 변환

형 식　　문자열변수 = htos(정수)

용 어

설 명　　1) HTOS에 입력된 정수를 16진수 문자열로 변환합니다.

프로그램 예제

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

정수 10을 16진수 문자열(“A”)로 변환하여 문자열 AA에 대입

3.45.7 SLEFT

기 능　　입력된 문자열의 좌측부분 추출

형 식　　문자열변수 = sleft(문자열, 정수)

용 어　　(정수) : 추출할 문자의 수

설 명　　1) 입력된 문자열을 왼쪽에서부터 입력된 정수만큼 추출

프로그램 예제

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

문자열 변수 AA에 "XYZW" 대입

문자열 변수 AA의 왼쪽부터 2문자("XY")를 반환

3.45.8 SLEN

기 능　　입력된 문자열의 길이를 반환

형 식　　정수형변수 = slen(문자열)

용 어　　(정수) : 추출할 문자의 수

설 명　　1) 입력된 문자열의 길이를 반환합니다.

프로그램 예제

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

문자열 변수 AA에 “XYZW” 대입

문자열 변수 AA의 길이(4)를 정수형 변수 LL에 대입

3.45.9 SMID

기 능　　문자열 지정위치부터 자릿수 만큼 문자열 추출

형 식　　문자열 지정위치부터 자릿수 만큼 문자열 추출

용 어　　(지정위치) : 문자열 추출할 시작 위치. 문자열의 시작은 0입니다.
ex) “ABCD”에서 A는 0번째, B는 1번째, C는 2번째, D는 3번째 위치임.
(정수) : 지정위치부터 읽어올 문자 개수

설 명　　1) 지정위치에서부터 사용자 입력 정수 만큼 문자를 추출합니다.

프로그램 예제

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

문자열 변수 AA에 “ABCDEFG” 대입

문자열 변수 AA의 2번째위치부터 문자 3개(“CDE”) 반환

3.45.10 SPOS

기 능 문자열1에서 문자열2가 매칭되는 시작위치 반환

형 식 정수형변수 = spos(문자열1, 문자열2)

용 어

설 명 1) 문자열1에서 문자열2와 매칭되는 시작위치를 반환합니다.

프로그램 예제

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

문자열 변수 AA에 “XYZW” 대입
문자열 변수 AA에서 첫번째 문자 반환
문자열 변수 AA에서 BB와 일치하는 위치 반환

3.45.11 SRIGHT

기 능 입력된 문자열의 우측부분 추출

형 식 문자열변수 = sright(문자열, 정수)

용 어 (정수) : 추출할 문자의 수

설 명 1) 입력된 문자열을 오른쪽에서부터 입력된 정수만큼 추출.

프로그램 예제

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

문자열 변수 AA에 “ABCDEF” 대입

문자열 변수 AA의 오른쪽 두문자를(“EF”) 반환

3.45.12 STRIN

기 능　　시리얼통신을 통하여 들어온 문자열을 구분자까지 읽어옴.

형 식　　문자열변수 = strin(정수)

용 어　　(정수) : 타임아웃시간.(단위 : ms)
타임아웃시간 만큼 시리얼 입력을 기다립니다. 타임아웃시간동안 데이터가 들어오지 않으면, 시스템 영역에 타임아웃상태를 체크하고 다음 스텝 명령을 수행합니다.

설 명　　1) 사용자가 지정한 시간만큼 대기하면서 시리얼 통신을 통한 입력을 읽어옵니다.

프로그램 예제

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

packet = strin(1000)        시리얼 통신을 통한 문자열 입력(입력 없을시 1000ms 대기)
len = slen(packet)          입력받은 문자열 길이 저장


if len > 0                  TIMEOUT이 아니고, 입력받은 문자열이 존재한 경우
...
end

end
```

3.45.13 STROUT

기 능 시리얼 통신을 통하여 문자열 출력

형 식 정수형변수 = strout(문자열)

용 어

설 명 1) 입력한 문자열을 시리얼 통신을 통해 출력을 내보내며, 전송하지 못한 문자수를 반환.
ex) “ABCDEFGH”의 문자를 전송하는 경우 성공시 반환값은 0
전부 전송하지 못한 경우 반환값은 8

프로그램 예제

```
func void main()
string sendpk
int strret
sendpk = “ABCDEF”
strret = strout(sendpk)
if strret != 0
    //전송 실패..
end
end
```

3.45.14 SVAL

기 능 문자열을 숫자로 변환

형 식 (정수 or 실수형)변수 = sval(문자열)

용 어

설 명 1) 입력한 문자열을 정수 또는 실수 데이터로 변환합니다.

프로그램 예제

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

문자열 "1234"를 정수 1234로 변환

문자열 "12.234"를 실수 12.234로 변환

3.46 LOG 명령어

3.46.1 PRINT

기 능　　로그를 파일로 저장

형 식　　print(인덱스 번호, 로그 내용1, 로그 내용2, …, 로그 내용 n)

용 어　　인덱스 번호 : 저장할 인덱스 번호(0~999)
로그 내용은 정수형, 실수형, 문자형등 사용가능.

설 명　　1) 사용자가 기록한 내용을 파일로 저장합니다.
2) 해당 기록은 팬던트의 RUN화면의 DISP 창 8. ULOG에서만 확인 가능합니다.

프로그램 예제

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

문자열 "1234"를 정수 1234로 변환

문자열 "1234"와 정수 변수 ai 값을 저장

3.46.2 PRINT_READ

기 능　　PRINT 함수로 저장한 로그 내용을 읽어 옴

형 식　　문자열변수 = print_read(인덱스 번호)

용 어　　인덱스 번호 : 저장할 메모리의 번호(0~999)
반환 값은 string 형태로 반환 됨

설 명　　1) 해당 인덱스 위치에 저장된 로그를 읽어옵니다.

프로그램 예제

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

문자열 "1234"를 정수 1234로 변환
문자열 "1234"와 정수 변수 ai 값을 메모리에 저장
0번 인덱스에 저장된 문자열을 읽어 옴

3.46.3 WLOG

기 능　　시간정보가 포함된 로그를 순서대로 파일에 저장

형 식　　wlog(로그 내용1, 로그 내용2, …, 로그 내용 n)

용 어　　로그 내용은 정수형, 실수형, 문자형등 사용가능.

설 명　　1) 사용자가 기록한 내용을 파일로 저장합니다.
2) 저장한 기록을 읽어 올 수는 없습니다.
3) 해당 기록은 팬던트의 6. INFO -〉 3. LOG -〉 3. USER 에서 확인 가능합니다.

프로그램 예제

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

문자열 "1234"를 정수 1234로 변환

문자열 "1234"와 정수 변수 ai 값을 파일에 저장

3.47 LATCH 명령어 (EtherCAT 통신타입)

3.47.1 LATCH_INIT

기 능 latch를 수행하기 위한 환경 설정

형 식 정수형 변수 = latch_init(축, latch 방식, 센서 갯수)

용 어 축 : latch를 수행할 축
latch 방식 : rising edge 방식, falling edge 방식 (1 : rising edge, 2 : falling edge)
센서 개수 : latch에 사용할 센서의 개수 (최소 한 개에서 최대 두 개까지)

설 명 1) latch를 사용할 축 및 latch 방식, 센서의 갯수를 설정합니다.
2) latch 환경이 바뀌지 않는 한 한번만 설정해도 됩니다.

프로그램 예제

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

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

latch_clear(3)                          3번 축에 대하여 latch 환경 초기화
latch_init(3, 2, 2)                     3번 축에 falling edge 방식으로 두개의 센서를 사용하는 latch 환경 설정.

latch_start()                           latch 시작
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          3번축의 1번 센서의 latch 상태 읽기
l_state[1] = latch_state(3, 2)          3번축의 2번 센서의 latch 상태 읽기

if( l_staet[0] == 2 && l_state[1] ==2)  3번축의 1번 센서와 2번 센서의 latch값이 존재하는가?
l_pos[0] = latch_pos(3, 1)              3번축의 1번 센서의 latch 위치를 읽어옵니다.
l_pos[1] = latch_pos(3, 2)              3번축의 2번 센서의 latch 위치를 읽어옵니다.
break
end
end
end
latch_stop()                            latch 정지
dlay 100
end
end
```

3.47.2 LATCH_CLEAR

기 능　　latch 환경을 초기화 합니다.

형 식　　정수형 변수 = latch_clear(축)

용 어　　축 : latch 환경 초기화를 수행할 축
0 : 전 축에 대하여 latch 환경 초기화
1 ~ 최대축 : 해당 축만 latch 환경 초기화

설 명　　1) latch 환경을 초기화 하여 latch 기능을 OFF 시킵니다.

프로그램 예제

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

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

latch_clear(3)          3번 축에 대하여 latch 환경 초기화
latch_init(3, 2, 2)     3번 축에 falling edge 방식으로 두개의 센서를 사용하는 latch 환경 설정.

latch_start()           latch 시작
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)   3번축의 1번 센서의 latch 상태 읽기
l_state[1] = latch_state(3, 2)   3번축의 2번 센서의 latch 상태 읽기

if( l_staet[0] == 2 && l_state[1] ==2)   3번축의 1번 센서와 2번 센서의 latch값이 존재하는가?
l_pos[0] = latch_pos(3, 1)       3번축의 1번 센서의 latch 위치를 읽어옵니다.
l_pos[1] = latch_pos(3, 2)       3번축의 2번 센서의 latch 위치를 읽어옵니다.
break
end
end
end
latch_stop()            latch 정지
dlay 100
end
end
```

3.47.3 LATCH_START

기 능 latch를 시작합니다.

형 식 정수형 변수 = latch_start()

용 어

설 명 1) latch를 시작 합니다.
2) latch_init을 수행하지 않았을 경우 0 이 리턴됩니다.

프로그램 예제

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

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

latch_clear(3)                    3번 축에 대하여 latch 환경 초기화
latch_init(3, 2, 2)               3번 축에 falling edge 방식으로 두개의 센서를 사용하는 latch 환경 설정.

latch_start()                     latch 시작
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)    3번축의 1번 센서의 latch 상태 읽기
l_state[1] = latch_state(3, 2)    3번축의 2번 센서의 latch 상태 읽기

if( l_staet[0] == 2 && l_state[1] ==2)   3번축의 1번 센서와 2번 센서의 latch값이 존재하는가?
l_pos[0] = latch_pos(3, 1)        3번축의 1번 센서의 latch 위치를 읽어옵니다.
l_pos[1] = latch_pos(3, 2)        3번축의 2번 센서의 latch 위치를 읽어옵니다.
break
end
end
end
latch_stop()                      latch 정지
dlay 100
end
end
```

3.47.4 LATCH_STOP

기 능 latch를 정지합니다.

형 식 정수형 변수 = latch_stop()

용 어

설 명 1) latch를 정지합니다.

프로그램 예제

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

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

latch_clear(3)                          3번 축에 대하여 latch 환경 초기화
latch_init(3, 2, 2)                     3번 축에 falling edge 방식으로 두개의 센서를 사용하는 latch 환경 설정.

latch_start()                           latch 시작
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          3번축의 1번 센서의 latch 상태 읽기
l_state[1] = latch_state(3, 2)          3번축의 2번 센서의 latch 상태 읽기

if( l_staet[0] == 2 && l_state[1] ==2)  3번축의 1번 센서와 2번 센서의 latch값이 존재하는가?
l_pos[0] = latch_pos(3, 1)              3번축의 1번 센서의 latch 위치를 읽어옵니다.
l_pos[1] = latch_pos(3, 2)              3번축의 2번 센서의 latch 위치를 읽어옵니다.
break
end
end
end
latch_stop()                            latch 정지
dlay 100
end
end
```

3.47.5 LATCH_POS

기 능 latch sensor가 검출된 위치를 읽어옵니다.

형 식 실수형 변수 = latch_pos(축, 센서 번호)

용 어 축 : latch를 수행 할 축(1 ~ 최대축)
센서번호 : latch 센서 번호(1~2)

설 명 1) latch 센서가 검출 된 경우, 검출 된 위치를 읽어오는 명령입니다.
2) latch_state 명령을 통해 latch 상태가 센서 검출이 되어 있는 경우에 호출 하여야 합니다. 센서 검출이 되지 않은 상태에서 호출시 알람이 발생합니다.

프로그램 예제

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

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

latch_clear(3)                    3번 축에 대하여 latch 환경 초기화
latch_init(3, 2, 2)               3번 축에 falling edge 방식으로 두개의 센서를 사용하는 latch 환경 설정.

latch_start()                     latch 시작
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)    3번축의 1번 센서의 latch 상태 읽기
l_state[1] = latch_state(3, 2)    3번축의 2번 센서의 latch 상태 읽기

if( l_staet[0] == 2 && l_state[1] ==2)   3번축의 1번 센서와 2번 센서의 latch값이 존재하는가?
l_pos[0] = latch_pos(3, 1)        3번축의 1번 센서의 latch 위치를 읽어옵니다.
l_pos[1] = latch_pos(3, 2)        3번축의 2번 센서의 latch 위치를 읽어옵니다.
break
end
end
end
latch_stop()                      latch 정지
dlay 100
end
end
```

3.47.6 LATCH_STATE

기 능 latch 동작 상태를 확인합니다.

형 식 정수형 변수 = latch_state(축, 센서 번호)

용 어 축 : latch 상태를 확인 할 축(1 ~ 최대축)
센서번호 : latch 센서 번호(1~2)

설 명 1) latch 상태를 확인합니다. 상태에 따라 다음 3가지 값을 리턴합니다.
0 : latch 기능을 시작하지 않은 상태입니다.
1 : latch 기능이 시작되었으나, 센서 검출은 되지 않은 상태입니다.
2 : latch 기능이 시작되었고, 센서 검출도 완료된 상태입니다. latch_pos 명령을 사용 할 수 있습니다.

프로그램 예제

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

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

latch_clear(3)                          3번 축에 대하여 latch 환경 초기화
latch_init(3, 2, 2)                     3번 축에 falling edge 방식으로 두개의 센서를 사용하는 latch 환경 설정.

latch_start()                           latch 시작
with
jmov lp[1]

while 1
l_state[0] = latch_state(3, 1)          3번축의 1번 센서의 latch 상태 읽기
l_state[1] = latch_state(3, 2)          3번축의 2번 센서의 latch 상태 읽기

if( l_staet[0] == 2 && l_state[1] ==2)  3번축의 1번 센서와 2번 센서의 latch값이 존재하는가?
l_pos[0] = latch_pos(3, 1)              3번축의 1번 센서의 latch 위치를 읽어옵니다.
l_pos[1] = latch_pos(3, 2)              3번축의 2번 센서의 latch 위치를 읽어옵니다.
break
end
end
end
latch_stop()                            latch 정지
dlay 100
end
end
```

3.47.7 LATCH_SENSOR

기 능　　latch 센서의 현재 상태를 확인합니다.

형 식　　정수형 변수 = latch_sensor (축, 센서 번호)

용 어　　축 : latch 센서 상태를 확인 할 축(1 ~ 최대축)
센서번호 : latch 센서 번호(1~2)

설 명　　1) latch 센서의 현재 입력 값을 확인 합니다. 다음 2가지 값을 리턴합니다.
0 : 센서 OFF
1 : 센서 ON
2) latch_init을 수행하지 않아도 사용 가능합니다.

프로그램 예제

```
func void main()
    int stop_flag, ret1, ret2

    stop_flag=1
    ret1=0
    ret2=0

    while(stop_flag)
        ret1 = latch_sensor(1,1)      1번축 1번 latch 센서의 현재 상태를 읽기
        ret2 = latch_sensor(1,2)      1번축 2번 latch 센서의 현재 상태를 읽기
        if(ret1==1 || ret2==1)
            stop_flag=0               두 센서 중 하나라도 ON 되는 경우 함수 종료
        end
    end
end
```

3.48 LATCH 명령어 (RS422 통신타입)

3.48.1 LATCH_INIT

기 능 latch를 수행하기 위한 환경 설정을 합니다.

형 식 정수형 변수 = latch_init(축, 센서 시작 번호, 센서 갯수)

용 어 축 : latch를 수행할 축(정수형)
센서 시작 번호 : 감지할 센서의 시작 위치를 설정 (1~4 사이의 정수)

Range Value	1	2	3	4
사용되는 센서	1, 2 번	2, 3 번	3, 4 번	4 번

센서 수 : latch에 사용할 센서의 개수 (최소 한 개에서 최대 두 개까지, 정수형)

설 명 1) latch를 사용할 축, 감지할 센서의 시작 위치 및 센서의 갯수를 설정합니다.
2) latch 환경이 바뀌지 않는 한 한번만 설정해도 됩니다.
3) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공 입니다.

프로그램 예제

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

    release
    take 1
    svon(1)

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

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

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

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

1번 축에 대하여 latch clear

1번 축에서 1번 위치(1, 2 번 센서)의 센서 2개를 읽어들이도록 latch 환경 설정

latch 시작

1번 축 1번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

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

        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

1번 축 2번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

latch 종료

3.48.2 LATCH_CLEAR

기 능 latch 환경을 초기화 합니다.

형 식 정수형 변수 = latch_clear(축)

용 어

3.48.3축 OVERRIDE

기 능 전체 실행 속도를 변경합니다.

형 식 정수형 변수 = override(robot_id, 속도)

용 어 robot id : local point를 읽어올 로봇 ID(1~3)
속도:변경하고자 하는 속도 값(1~100)

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    release

    override(1,20)          전체 실행 속도를 20으로 변경

    take 1

    dlay 100

    svon(1)

    jmov lp[0]
    dlay 1000

    override(1,5)           전체 실행 속도를 5로 변경

    jmov lp[1]
    dlay 1000
end
```

0 : 전 축에 대하여 latch 환경 초기화
1 ~ 최대축 : 해당 축만 latch 환경 초기화

설 명 1) latch를 사용할 축, 감지할 센서의 시작 위치 및 센서의 갯수를 설정합니다.
2) latch 환경이 바뀌지 않는 한 한번만 설정해도 됩니다.
3) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

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

    release
    take 1
    svon(1)

    ret=latch_clear(1)          1번 축에 대하여 latch clear
    if(ret==0)
        goto close_routine
    end
    ret=latch_init(1,1,2)       1번 축에서 1번 위치(1, 2 번 센서)의 센서 2개를
    if(ret==0)                  읽어들이도록 latch 환경 설정
        goto close_routine
    end
    ret=latch_start()           latch 시작
    if(ret==0)
        goto close_routine
    end

    while(1)
        if(latch_state(1,1)==2)     1번 축 1번 센서의 상태 값 읽어 신호를 감지하면,
            l_pos[0]=latch_pos(1,1) 해당 센서의 포지션 값을 읽어옴
        end
        if(latch_state(1,2)==2)     1번 축 2번 센서의 상태 값 읽어 신호를 감지하면,
            l_pos[1]=latch_pos(1,2) 해당 센서의 포지션 값을 읽어옴
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine              latch 종료
    ret=latch_stop()
end
```

3.48.4 LATCH_START

기 능　　latch를 시작합니다.

형 식　　정수형 변수 = latch_start()

용 어

설 명　　1) latch를 시작 합니다.
2) latch_init을 수행하지 않았을 경우 0 이 리턴됩니다.
3) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

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

    release
    take 1
    svon(1)

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

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

labl close_routine
    ret=latch_stop()
end
```

1번 축에 대하여 latch clear

1번 축에서 1번 위치(1, 2 번 센서)의 센서 2개를 읽어들이도록 latch 환경 설정

latch 시작

1번 축 1번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

1번 축 2번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

latch 종료

3.48.5 LATCH_STOP

기 능 latch를 정지합니다.

형 식 정수형 변수 = latch_stop()

용 어

설 명 1) latch를 정지합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

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

    release
    take 1
    svon(1)

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

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

labl close_routine
    ret=latch_stop()
end
```

1번 축에 대하여 latch clear

1번 축에서 1번 위치(1, 2 번 센서)의 센서 2개를 읽어들이도록 latch 환경 설정

latch 시작

1번 축 1번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

1번 축 2번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

latch 종료

3.48.6 LATCH_POS

기 능 latch sensor가 검출된 위치를 읽어옵니다.

형 식 실수형 변수 = latch_pos(축, 센서 인덱스)

용 어 축 : latch를 수행 할 축(1 ~ 최대축, 정수형)
센서 인덱스: latch_init 함수에서 설정한 사용할 센서의 인덱스(1~2, 정수형)

Range Value	1	2	3	4
사용되는 센서	1, 2 번	2, 3 번	3, 4 번	4 번
Index : 1	**1**	**2**	**3**	**4**
Index : 2	**2**	**3**	**4**	**X**

설 명 1) latch 센서가 검출 된 경우, 검출 된 위치를 읽어오는 명령입니다.
2) latch_state 명령을 통해 latch 상태가 센서 검출이 되어 있는 경우에 호출 하여야 합니다. 센서 검출이 되지 않은 상태에서 호출시 알람이 발생합니다.
3) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

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

    release
    take 1
    svon(1)

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

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

1번 축에 대하여 latch clear

1번 축에서 1번 위치(1, 2 번 센서)의 센서 2개를 읽어들이도록 latch 환경 설정

latch 시작

1번 축 1번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

1번 축 2번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

```
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

latch 종료

3.48.7 LATCH_STATE

기 능 latch 동작 상태를 확인합니다.

형 식 정수형 변수 = latch_state(축, 센서 Index)

용 어 축 : latch 상태를 확인 할 축(1 ~ 최대축)
센서 인덱스 : latch_init 함수에서 설정한 사용할 센서의 인덱스(1~2, 정수형)

Range Value	1	2	3	4
사용되는 센서	1, 2 번	2, 3 번	3, 4 번	4 번
Index : 1	**1**	**2**	**3**	**4**
Index : 2	**2**	**3**	**4**	**X**

설 명 1) latch 상태를 확인합니다. 상태에 따라 다음 3가지 값을 리턴합니다.
0 : latch 기능을 시작하지 않은 상태입니다.
1 : latch 기능이 시작되었으나, 센서 검출은 되지 않은 상태입니다.
2 : latch 기능이 시작되었고, 센서 검출도 완료된 상태입니다. latch_pos 명령을 사용 할 수 있습니다.

프로그램 예제

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

    release
    take 1
    svon(1)

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

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

1번 축에 대하여 latch clear

1번 축에서 1번 위치(1, 2 번 센서)의 센서 2개를 읽어들이도록 latch 환경 설정

latch 시작

1번 축 1번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

1번 축 2번 센서의 상태 값 읽어 신호를 감지하면, 해당 센서의 포지션 값을 읽어옴

```
        end
        jmov lp[1]
        dlay 100
    end

labl close_routine
    ret=latch_stop()
end
```

latch 종료

3.48.8 LATCH_SENSOR

기 능 latch 현재 상태를 확인합니다.

형 식 정수형 변수 = latch_sensor(축, 센서 Index)

용 어 축 : latch 상태를 확인 할 축(1 ~ 최대축)
센서 인덱스 : latch_init 함수에서 설정한 사용할 센서의 인덱스(1~2, 정수형)

Range Value	1	2	3	4
사용되는 센서	1, 2 번	2, 3 번	3, 4 번	4 번
Index : 1	**1**	**2**	**3**	**4**
Index : 2	**2**	**3**	**4**	**X**

설 명 1) latch 센서의 현재 입력 값을 확인 합니다. 다음 2가지 값을 리턴합니다.
0 : 센서 OFF
1 : 센서 ON
2) latch_init을 수행하지 않아도 사용 가능합니다.

프로그램 예제

```
func void main()
    int stop_flag, ret1, ret2

    stop_flag=1
    ret1=0
    ret2=0

    while(stop_flag)
        ret1 = latch_sensor(1,1)      1번축 1번 latch 센서의 현재 상태를 읽기
        ret2 = latch_sensor(1,2)      1번축 2번 latch 센서의 현재 상태를 읽기
        if(ret1==1 || ret2==1)        두 센서 중 하나라도 ON 되는 경우 함수 종료
            stop_flag=0
        end
    end
end
```

3.49 ERROR 명령어

3.49.1 SETERR (SYSTEM EMG 알람 명령어)

기 능 System emergency 알람을 발생

형 식 seterr(알람코드)

용 어 알람코드 : 사용자가 임의로 설정

설 명 1) job에서 system emergency 알람을 발생시키는 명령어로 알람 코드는 사용자가 임의로 입력 합니다.

프로그램 예제

```
func void main()
int tmp
take 1
vel(10)
tmp = rstate(1,6)                    ……        robot 1이 servo on 상태 확인
if( tmp == 0 )                                 servo on 상태가 아니면
  seterr(10)                                   10번 알람 코드로 system emergency 알람 발생
end
jmov lp[100]
release
end
```

3.49.2 RESET(알람 해제 명령어)

기 능 알람 해제

형 식 reset

용 어

설 명 1) 발생한 알람을 해제합니다.

프로그램 예제

```
func void main()
int tmp
take 1
vel(10)
tmp = rstate(1,1)                          alarm 상태 확인
if( tmp == 1 )                …………     alarm 상태이면
  reset                                    알람 해제
  svon(1)
end
jmov lp[100]
release
end
```

3.49.3 RERROR(에러 코드 반환 명령어)

기 능 에러 코드 반환

형 식 rerror()

용 어

설 명 1) 가장 마지막에 발생한 에러 코드를 반환합니다.

프로그램 예제

```
func void main()
int tmp
int ecode
take 1
vel(10)
tmp = rstate(1,1)                          alarm 상태 확인
if( tmp == 1 )          ……….....           alarm 상태이면
  ecode = rerror()                         알람 코드 확인
end
…
release
end
```

3.49.4 RERRCNT(에러 발생 개수 반환 명령어)

기 능 에러 발생 개수 반환

형 식 rerrcnt()

용 어

설 명 1) 발생한 에러의 개수를 반환한다.

프로그램 예제

```
func void main()
int tmp
int ecode
int ecount
int aa
take 1
vel(10)
tmp = rstate(1,1)                          alarm 상태 확인
if( tmp == 1 )                  ……        alarm 상태이면
  ecount = rerrcnt()                       알람 개수 확인

  for aa = 1 to ecount                     알람 개수만큼 에러 코드 확인
    ecode = rerrcode(aa)                   에러 코드 읽기
    if( ecode == 0 )                       rerrcode의 반환값이 0이면 에러 읽기 종료
      break
  end
end
...
release
end
```

3.49.5 RERRCODE(에러 코드 반환 명령어)

기 능 에러 코드 반환

형 식 rerrcode(알람 발생 순서)

용 어

설 명 1) 파라미터로 입력된 알람 발생 순서에 해당하는 알람 코드를 반환합니다.
2) 참고로 rerror()는 가장 마지막에 발생한 알람 코드를 반환합니다.

프로그램 예제

```
func void main()
int tmp
int ecode
int ecount
int aa
take 1
vel(10)
tmp = rstate(1,1)                          alarm 상태 확인
if( tmp == 1 )                  ……        alarm 상태이면
  ecount = rerrcnt()                       알람 개수 확인

  for aa = 1 to ecount                     알람 개수만큼 에러 코드 확인
    ecode = rerrcode(aa)                   에러 코드 읽기
    if( ecode == 0 )                       rerrcode의 반환값이 0이면 에러 읽기 종료
      break
  end
end
…
release
end
```

3.49.6 RERRTEXT(에러 내용 반환 명령어)

기 능 에러 내용 반환

형 식 rerrtext(알람 발생 순서)

용 어

설 명 1) 파라미터로 입력된 알람 발생 순서에 해당하는 알람 내용을 반환합니다.

프로그램 예제

```
func void main()
int tmp
string etext
int ecount
int len
take 1
vel(10)
tmp = rstate(1,1)                          alarm 상태 확인
if( tmp == 1 )                  ……        alarm 상태이면
  while 1
    etext = rerrtext(aa)                   에러 내용 읽기
    len = slen(etext)                      rerrtext함수를 통해 읽어온 문자열의 길이 확인
    if(len == 0 )                          rerrtext함수를 통해 읽어온 문자열의 길이가 0이면에러
      break                                읽기 종료
  end
end
…
release
end
```

3.49.7 READ_DERR(Driver 에러 코드 반환 명령어)

기 능 에러 내용 반환

형 식 read_derr(축)

용 어

설 명
1) 파라미터로 입력된 축에 해당하는 드라이버 알람 내용을 반환합니다.
2) 알람 미발생시 0을 반환합니다.

프로그램 예제

```
func void main()
int ecode1, ecode2

take 1


ecode1= read_derr(1)                    1번축 error 읽기
ecode2= read_derr(4)                    4번축 error 읽기


release
end
```

3.50 MAPPING 명령어 (RS422 통신타입 only)

3.50.1 MAP_INIT

기 능 mapping 관련 내부 변수를 초기화 합니다.

형 식 정수형 변수 = map_init ()

용 어

설 명 1) mapping 시작에 앞서 관련 변수를 초기화 하는 함수입니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                                  변수 초기화
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

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

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)  mapping 데이터 읽기(최대 타임아웃 시간까지만)
    dlay 1000

    ret=map_stop()                                  mapping 종료

labl close_routine
        ret=map_stop()
end
```

3.50.2 MAP_START

기 능 mapping을 시작합니다.

형 식 정수형 변수 = map_start(축, 센서 인덱스, 시작 로컬 포인트 인덱스)

용 어 축: 1~ 최대 축 (정수형)
센서 인덱스 : 센서의 인덱스 값을 입력(Range: 1~2)
시작 로컬 포인트: mapping 데이터를 저장할 시작 포인트 인덱스, Range: 0~19999

설 명 1) mapping을 시작하고, 데이터를 저장할 포인트를 지정합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                                  변수 초기화
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

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

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)  mapping 데이터 읽기(최대 타임아웃 시간까지만)
    dlay 1000

    ret=map_stop()                                  mapping 종료

labl close_routine
        ret=map_stop()
end
```

3.50.3 MAP_STOP

기 능 mapping을 정지합니다.

형 식 정수형 변수 = map_stop()

용 어

설 명 1) mapping을 정지합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                                  변수 초기화
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

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

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)  mapping 데이터 읽기(최대 타임아웃 시간까지만)
    dlay 1000

    ret=map_stop()                                  mapping 종료

labl close_routine
        ret=map_stop()
end
```

3.50.4 MAP_READ

기 능 mapping을 데이터를 읽어옵니다.

형 식 정수형 변수 = map_read(타임아웃 시간)

용 어 타임아웃 시간: 데이터 읽기에 대기할 최대 시간(ms)을 입력합니다.

설 명 1) mapping을 데이터를 읽어 map_start 에서 지정한 포인트에 저장합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret

    release
    take 1
    svon(1)
    dlay 1000

    ret=map_init()                                  변수 초기화
    if(ret==0)
        goto close_routine
    end

    jmov lp[0]
    dlay 100

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

    jmov lp[1]
    dlay 100

    ret=map_read(1000)          //1 sec(1000 msec)  mapping 데이터 읽기(최대 타임아웃 시간까지만)
    dlay 1000

    ret=map_stop()                                  mapping 종료

labl close_routine
        ret=map_stop()
end
```

3.51 INSHAPE 명령어

3.51.1 INSHAPE_SET

기 능 InShape 기능을 사용할 축을 설정합니다.

형 식 정수형 변수 = inshape_set(축, 최대 index 개수)

용 어 축: 1~최대 축(정수형), InShape 기능을 사용할 축을 입력합니다.
최대 index 개수: 사용할 배열의 index 개수를 입력합니다. (Range: 1~10)

설 명 1) InShape 기능 시작에 앞서 사용할 축과, 보상 포인트를 지정합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret, i, axis, max_num

    max_num=5
    init_01_inshape_table()
    axis=1
    release
    take 1
    svon(1)

    ret=inshape_clear()                                          InShape 기능사용 정지, 변수 초기화
    for i=1 to max_num step 1
        ret=inshape_write(axis,1,i,$inshape_time_table[i-1])     파라미터 설정
        ret=inshape_write(axis,2,i,$inshape_dist_table[i-1])
    end

    ret=inshape_set(axis,max_num)
    if(ret==0)                                                   InShape 기능을 사용할 축 선택
        goto exit_routine
    end
    ret=inshape_start()
    if(ret==0)                                                   InShape 기능을 사용
        goto exit_routine
    end

    jmov lp[0]                                                   보상 포함, JMOV 실행

    ret=inshape_clear()

labl close_routine
        ret=inshape_clear()
end
```

3.51.2 INSHAPE_CLEAR

기 능 InShape 기능을 사용을 해제합니다.

형 식 정수형 변수 = inshape_clear()

용 어

설 명 1) 함수 사용 시 InShape 관련 설정 데이터가 전부 초기화 됩니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret, i, axis, max_num

    max_num=5
    init_01_inshape_table()
    axis=1
    release
    take 1
    svon(1)

    ret=inshape_clear()                                          InShape 기능사용 정지, 변수 초기화
    for i=1 to max_num step 1
        ret=inshape_write(axis,1,i,$inshape_time_table[i-1])     파라미터 설정
        ret=inshape_write(axis,2,i,$inshape_dist_table[i-1])
    end

    ret=inshape_set(axis,max_num)
    if(ret==0)                                                   InShape 기능을 사용할 축 선택
        goto exit_routine
    end
    ret=inshape_start()
    if(ret==0)
        goto exit_routine                                        InShape 기능을 사용
    end

    jmov lp[0]                                                   보상 포함, JMOV 실행

    ret=inshape_clear()

labl close_routine
        ret=inshape_clear()
end
```

3.51.3 INSHAPE_READ

기 능 해당 축의 InShape 관련 파라미터 값을 읽어 옵니다.

형 식 실수형 변수 = inshape_read(축, param, index)

용 어 축: 1~최대 축(정수형), 파라미터 값을 읽어올 축을 선택합니다.
param: 값을 읽어올 해당 파라미터를 명시합니다. 1로 설정할 경우 InShape 발생 시간(전체 시간을 1로 생각했을 때의 비율) , 2로 설정할 경우 보상 값(mm 단위)이 반환됩니다.
index: 읽어올 index number를 지정합니다. (Range: 1~10)

설 명 1) 파라미터 데이터를 읽어 옵니다.

프로그램 예제

```
func void main()
    int ret, i, axis, max_num
    double param

    max_num=5
    init_01_inshape_table()
    axis=1
    param = 0

    release
    take 1

    ret=inshape_clear()                                          InShape 기능사용 정지, 변수 초기화
    for i=1 to max_num step 1
        ret=inshape_write(axis,1,i,$inshape_time_table[i-1])     파라미터 설정
        ret=inshape_write(axis,2,i,$inshape_dist_table[i-1])
    end

    for i=1 to max_num step 1                                    파라미터 값 읽기
        param=inshape_read(axis,1,i)
        print(i*2,param)
        param=inshape_read(axis,2,i)
        print(i*2+1,param)
    end

 end
```

3.51.4 INSHAPE_WRITE

기 능 해당 축의 InShape 관련 파라미터 값을 설정합니다.

형 식 정수형 변수 = inshape_write(축, param, index, 설정 값)

용 어 축: 1~최대 축(정수형), 파라미터 값을 읽어올 축을 선택합니다.
param: 값을 설정할 해당 파라미터를 명시합니다. 1로 설정할 경우 InShape 발생 시간(전체 시간을 1로 생각했을 때의 비율) , 2로 설정할 경우 보상 값(mm 단위)을 설정합니다.
index: 설정할 index number를 지정합니다. (Range: 1~10)
설정 값: 원하는 값을 입력합니다.(실수형)
[주의] Param 이 1일 경우(inshape 발생위치)의 Range: 0.0 ~ 1.0 사이의 값

설 명 1) 파라미터 데이터를 설정합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret, i, axis, max_num

    max_num=5
    init_01_inshape_table()
    axis=1
    release
    take 1
    svon(1)

    ret=inshape_clear()                                  InShape 기능사용 정지, 변수 초기화
    for i=1 to max_num step 1
        ret=inshape_write(axis,1,i,$inshape_time_table[i-1])   파라미터 설정
        ret=inshape_write(axis,2,i,$inshape_dist_table[i-1])
    end

    ret=inshape_set(axis,max_num)
    if(ret==0)                                           InShape 기능을 사용할 축 선택
        goto exit_routine
    end
    ret=inshape_start()
    if(ret==0)                                           InShape 기능을 사용
        goto exit_routine
    end

    jmov lp[0]                                           보상 포함, JMOV 실행

    ret=inshape_clear()

labl close_routine
        ret=inshape_clear()
end
```

3.51.5 INSHAPE_START

기 능 InShape 기능 사용시작을 알립니다.

형 식 정수형 변수 = inshape_start()

용 어

설 명 1) InShape 기능을 수행합니다. 모션 종료 시 사용시작 플래그가 초기화 됩니다. 따라서, 모션 명령어 이전에 반복사용해 주어야 합니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.
3) JMOV 명령어에 한정되며, FOS 모드에서는 사용 불가합니다.

프로그램 예제

```
func void main()
    int ret, i, axis, max_num

    max_num=5
    init_01_inshape_table()
    axis=1
    release
    take 1
    svon(1)

    ret=inshape_clear()                                          InShape 기능사용 정지, 변수 초기화
    for i=1 to max_num step 1
        ret=inshape_write(axis,1,i,$inshape_time_table[i-1])     파라미터 설정
        ret=inshape_write(axis,2,i,$inshape_dist_table[i-1])
    end

    ret=inshape_set(axis,max_num)
    if(ret==0)                                                   InShape 기능을 사용할 축 선택
        goto exit_routine
    end
    ret=inshape_start()
    if(ret==0)                                                   InShape 기능을 사용
        goto exit_routine
    end

    jmov lp[0]                                                   보상 포함, JMOV 실행

    ret=inshape_clear()

labl close_routine
        ret=inshape_clear()
end
```

3.52 ARM FOLD 명령어

3.52.1 INITARM_WRITE

기 능 Arm Fold 기능 사용 시 핸드가 이동할 위치 파라미터 값을 입력합니다.

형 식 정수형 변수 = initarm_write(robot_id, 인덱스, 실수형 변수)

용 어 robot_id: 로봇 번호를 입력합니다.
인덱스: 1일 경우 핸드 위치 값을, 2 일경우 부가 축 위치 값을 선택합니다.
실수형 변수: 핸드 위치 값을 입력합니다.

설 명 1) Arm Fold 기능은 티칭 화면에서 실행할 수 있습니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int ret
    double hand_pos

    ret = initarm_write(1, 1,-1300.0)      핸드 복귀 위치 저장
    if( ret =0)
        initarm_clear()                    저장 실패 시, 복귀 위치 값 삭제
    end

    hand_pos=initarm_read(1,1)             핸드 복귀 위치 값 읽기
    print(0, hand_pos)
end
```

3.52.2 INITARM_READ

기 능 Arm Fold 기능 사용 시 핸드가 이동할 위치 파라미터 값을 읽어옵니다.

형 식 실수형 변수 = initarm_read(robot_id, 인덱스)

용 어 robot_id: 로봇 번호를 입력합니다.
인덱스: 1일 경우 핸드 위치 값을, 2 일경우 부가 축 위치 값을 선택합니다.

설 명 1) 파라미터 데이터를 읽어옵니다.

프로그램 예제

```
func void main()
    int ret
    double hand_pos

    ret = initarm_write(1, 1,-1300.0)        핸드 복귀 위치 저장
    if( ret =0)
        initarm_clear()                      저장 실패 시, 복귀 위치 값 삭제
    end

    hand_pos=initarm_read(1,1)               핸드 복귀 위치 값 읽기
    print(0, hand_pos)
end
```

3.52.3 INITARM_CLEAR

기 능 Arm Fold 기능 사용 시 핸드가 이동할 위치 파라미터 값을 삭제합니다.

형 식 정수형 변수 = initarm_clear()

용 어

설 명 1) 파라미터 데이터를 삭제하고 초기화 합니다.

프로그램 예제

```
func void main()
    int ret
    double hand_pos

    ret = initarm_write(1, 1,-1300.0)        핸드 복귀 위치 저장
    if( ret =0)
        initarm_clear()                      저장 실패 시, 복귀 위치 값 삭제
    end

    hand_pos=initarm_read(1,1)               핸드 복귀 위치 값 읽기
    print(0, hand_pos)
end
```

3.53 Z축 보상 명령어

3.53.1 ZCAL_DATA_SET

기 능 Z축 보상 함수 생성에 사용되는 위치 및 오차 정보를 입력합니다.

형 식 정수형 변수 = zcal_data_set(인덱스, 핸드 축, 핸드 위치 값, 오차)

용 어 인덱스: 핸드위치 값과 오차를 저장할 배열의 위치를 지정합니다.(Range: 0~30)
핸드 축: 대상 핸드 축을 지정합니다.
핸드 위치 값: 지정된 인덱스에 저장할 핸드의 위치 값을 입력합니다. (Range: sw_limit)
오차: 핸드 축의 Z 축 방향 처짐 정도를 입력 (Range: -50 ~ 50)

설 명 1) Z보상 기능 사용에 앞서 실행해 주어야 합니다. 보상 다항식의 계수가 생성, 저장되었다면 생략해도 됩니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int i, arm
    double read_range, read_err

    i=0
    arm=3

    zcal_data_clr()                                  Z보상 함수 생성에 사용되는 위치 및 오차 정보 초기화

    for i=0 to 20 step 1
        zcal_data_set(i,arm,$range[i],$err[i])       위치 및 오차 정보 입력

        read_range=zcal_data_read(2,i)               위치 값 읽어 오기
        if($range[i]!=read_range)
            goto EXIT
        end

        read_range=zcal_data_read(3,i)               오차 값 읽어 오기
        if($err[i]!=read_err)
            goto EXIT
        end
    end

    zcal_coef_gen()                                  Z보상 다항식 계수 생성

labl EXIT
end
```

3.53.2 ZCAL_DATA_CLR

기 능 Z축 보상 함수 생성에 사용되는 위치 및 오차 정보 등을 초기화 합니다.

형 식 정수형 변수 = zcal_data_clr()

용 어

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int i, arm
    double read_range, read_err

    i=0
    arm=3

    zcal_data_clr()                          Z보상 함수 생성에 사용되는 위치 및 오차 정보 초기화

    for i=0 to 20 step 1
        zcal_data_set(i,arm,$range[i],$err[i])   위치 및 오차 정보 입력

        read_range=zcal_data_read(2,i)       위치 값 읽어 오기
        if($range[i]!=read_range)
            goto EXIT
        end

        read_range=zcal_data_read(3,i)       오차 값 읽어 오기
        if($err[i]!=read_err)
            goto EXIT
        end
    end

    zcal_coef_gen()                          Z보상 다항식 계수 생성

labl EXIT
end
```

3.53.3 ZCAL_COEF_GEN

기 능 Z축 보상 다항식의 계수를 생성합니다.

형 식 정수형 변수 = zcal_coef_gen()

용 어

설 명 1) Z보상 기능 사용에 앞서 실행해 주어야 합니다. 보상 다항식의 계수가 생성, 저장되었다면 생략해도 됩니다.
2) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int i, arm
    double read_range, read_err

    i=0
    arm=3

    zcal_data_clr()                                  Z보상 함수 생성에 사용되는 위치 및 오차 정보 초기화

    for i=0 to 20 step 1
        zcal_data_set(i,arm,$range[i],$err[i])       위치 및 오차 정보 입력

        read_range=zcal_data_read(2,i)               위치 값 읽어 오기
        if($range[i]!=read_range)
            goto EXIT
        end

        read_range=zcal_data_read(3,i)               오차 값 읽어 오기
        if($err[i]!=read_err)
            goto EXIT
        end
    end

    zcal_coef_gen()                                  Z보상 다항식 계수 생성

labl EXIT
end
```

3.53.4 ZCAL_DATA_READ

기 능　　입력한 위치 및 오차 정보를 읽어옵니다.

형 식　　실수형 변수 = zcal_data_read(인덱스, 옵션)

용 어　　인덱스: 읽어올 데이터 배열의 위치를 지정합니다.(Range: 0~30)
옵션: 데이터 셋 여부(1), 위치 값(2), 오차 값(3)

설 명　　1) 입력한 위치 및 오차 데이터를 읽어옵니다.

프로그램 예제

```
func void main()
    int i, arm
    double read_range, read_err

    i=0
    arm=3

    zcal_data_clr()                          Z보상 함수 생성에 사용되는 위치 및 오차 정보 초기화

    for i=0 to 20 step 1
        zcal_data_set(i,arm,$range[i],$err[i])   위치 및 오차 정보 입력

        read_range=zcal_data_read(2,i)       위치 값 읽어 오기
        if($range[i]!=read_range)
            goto EXIT
        end

        read_range=zcal_data_read(3,i)       오차 값 읽어 오기
        if($err[i]!=read_err)
            goto EXIT
        end
    end

    zcal_coef_gen()                          Z보상 다항식 계수 생성

labl EXIT
end
```

3.53.5 ZCAL_INIT

기 능 Z보상 대상 축 및 참조할 핸드 축을 설정합니다.

형 식 정수형 변수 = zcal_init(보상대상 축, 핸드 축)

용 어 보상대상 축: hand의 직진도 테스트 중 보상을 해야하는 축 입력(예를들어, Z축)
핸드 축: 진진도를 측정하는 대상 핸드 축

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                          Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)                Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                          Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)          Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                           Z보상 정지
    while(ret)
        ret=zcal_state()                  Z보상 정지 상태 확인
    end
end
```

3.53.6 ZCAL_READ

기 능　　핸드 위치 값에 따른 Z축 보상 값을 반환합니다.

형 식　　실수형 변수 = zcal_read(핸드 위치 값)

용 어　　핸드 위치 값: Z축 보상 값을 반환받고자 하는 hand의 위치 값

설 명

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                          Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)                Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                          Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)          Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                           Z보상 정지
    while(ret)
        ret=zcal_state()                  Z보상 정지 상태 확인
    end
end
```

3.53.7 ZCAL_OPT

기 능 Z축 보상에 사용할 부가축을 설정합니다.

형 식 정수형 변수 = zcal_opt(부가 축)

용 어 부가 축: Stroke를 늘리기 위해 하단에 별도로 설치된 축 입력

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                    Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)          Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                    Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)    Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                     Z보상 정지
    while(ret)
        ret=zcal_state()            Z보상 정지 상태 확인
    end
end
```

3.53.8 ZCAL_CLEAR

기 능 Z축 보상 설정 값을 해제하고 초기화 합니다.

형 식 정수형 변수 = zcal_clear()

용 어

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                    Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)          Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                    Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)    Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                     Z보상 정지
    while(ret)
        ret=zcal_state()            Z보상 정지 상태 확인
    end
end
```

3.53.9 ZCAL_START

기 능 Z축 보상을 시작합니다.

형 식 정수형 변수 = zcal_start()

용 어

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                    Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)          Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                    Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)    Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                     Z보상 정지
    while(ret)
        ret=zcal_state()            Z보상 정지 상태 확인
    end
end
```

3.53.10 ZCAL_STOP

기 능 Z축 보상을 정지합니다.

형 식 정수형 변수 = zcal_stop()

용 어

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                    Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)          Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                    Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)    Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                     Z보상 정지
    while(ret)
        ret=zcal_state()            Z보상 정지 상태 확인
    end
end
```

3.53.11 ZCAL_STATE

기 능 Z축 보상 사용 여부 확인합니다.

형 식 정수형 변수 = zcal_state()

용 어

설 명 1) 반환되는 값이 0 일 경우 Z축 보상을 사용하지 않는 상태고, 1일 경우 Z축 보상을 사용하는 상태입니다.

프로그램 예제

```
func void main()
    int arm, opt_arm, z_axis
    double delta_z

    arm=3
    opt_arm=6
    z_axis=2

    stop_zcal()

    zcal_clear()                    Z축 보상 관련 설정 값 해제 및 데이터 초기
    zcal_init(z_axis, arm)          Z축 보상 대상 축 및 참조할 핸드 축을 설정
    zcal_opt(opt_arm)

    svon(1)
    dlay 500

    zcal_start()                    Z축 보상 시작

    jmov lp[0]
    dlay 1000

    delta_z = zcal_read(lp[0].3)    Z축 보상 시작

end

func void stop_zcal()
    int ret
    ret=1

    zcal_stop()                     Z보상 정지
    while(ret)
        ret=zcal_state()            Z보상 정지 상태 확인
    end
end
```

3.54 그 외

3.54.1 OVERRIDE

기 능 전체 실행 (외부 속도) 속도를 변경합니다.

형 식 override(robot_id, 속도)

용 어 robot id : local point를 읽어올 로봇 ID(1~3)
속도:변경하고자 하는 속도 값(1~100)

설 명

프로그램 예제

```
func void main()
    release

    override(1,20)          전체 실행 속도를 20으로 변경

    take 1

    dlay 100

    svon(1)

    jmov lp[0]
    dlay 1000

    override(1,5)           전체 실행 속도를 5로 변경

    jmov lp[1]
    dlay 1000
end
```

3.54.2 XENB

기 능 특정 축의 사용 여부를 결정합니다.

형 식 xenb(축, 상태)

용 어 축: 사용 여부를 결정할 축 번호 (정수형)
상태: ON - 축 사용 안함(Disable)
OFF - 축 사용 함 (Enable)

설 명

프로그램 예제

```
func void main()
    release
    take 1

    xenb(1,ON)
end
```

1번 축, 사용 안함

3.54.3 READ_DIST

기 능　　누적거리 및 초과 카운터를 읽어옵니다.

형 식　　실수형 변수 = read_dist(robot_id, 타입, 축)

용 어　　robot_id: 로봇 번호를 입력합니다.
타입: 읽어 올 데이터의 타입을 설정합니다.
1: 누적거리 값
2: 초기화 카운트 (제한거리 초과 시)
축: 데이터를 읽어 올 축 설정 (정수형)

설 명　　1) 실수형 변수에 누적 거리 혹은 초기화 카운트 값을 가져옵니다.

프로그램 예제

```
func void main()
    double dist, cnt

    dist = 0
    cnt = 0

    release
    take 1

    dist = read_dist(1,1,4)
    cnt= read_dist(1,2,4)
end
```

1번 로봇 4번 축 누적거리 값 읽어오기
1번 로봇 4번 축 카운트 값 읽어오기

3.54.4 ORGEXEC

기 능 Origin 기능을 수행합니다.

형 식 정수형 변수 = orgexec()

용 어

설 명 1) 반환되는 값이 0 일 경우 함수 수행 실패, 1일 경우 수행 성공입니다.

프로그램 예제

```
func void main()
    int state
    int ret

    release
    dlay 100

    state=rstate(1,3)

    if(state==0)
        take 1
        dlay 500

        ret=orgexec()
        print(1,ret)

        jmov lp[0]
    end
end
```

Origin 완료 여부 확인

Origin 수행

3.54.5 LP_USED

기 능 특정 로컬 포인트의 사용 여부를 확인합니다.

형 식 정수형 변수 = lp_used(robot_id, 포인트 번호)

용 어

설 명 1) 반환되는 값이 0 일 경우 사용안하는 상태이며, 1일 경우 포인트가 저장되어 사용하는 상태입니다.

프로그램 예제

```
func void main()

    int ret

    release
    dlay 100

    ret = lp_used(1, 10000)
    print(0, ret)

end
```

1번 로봇의 10,000번째 로컬 포인트의 사용 여부를 확인

3.54.6 READ_RPM

기 능 RPM을 읽어옵니다.

형 식 실수형 변수 = read_rpm(robot_id, 축)

용 어 robot_id: 로봇 번호를 입력합니다.
축: 데이터를 읽어 올 축 설정 (정수형)

단 위 RPM

설 명 1) 실수형 변수에 현재 RPM 값을 실시간으로 가져옵니다.

프로그램 예제

```
func void main()
    double rpm

    rpm = 0

    release
    take 1

    rpm = read_rpm(1,4)        1번 로봇 4번 축 RPM 값 읽어오기
end
```

3.54.7 READ_LOAD

기 능 LOAD를 읽어옵니다.

형 식 실수형 변수 = read_load(robot_id, 축)

용 어 robot_id: 로봇 번호를 입력합니다.
축: 데이터를 읽어 올 축 설정 (정수형)

설 명 1) 실수형 변수에 LOAD 값을 가져옵니다.

프로그램 예제

```
func void main()
    double load

    load = 0

    release
    take 1

    load = read_load(1,1)
end
```

1번 로봇 1번 축 LOAD 값 읽어오기

3.54.8 READ_TRQMAX

기 능 MAX TORQUE 를 읽어옵니다.

형 식 실수형 변수 = read_trqmax(robot_id, 축)

용 어 robot_id: 로봇 번호를 입력합니다.
축: 데이터를 읽어 올 축 설정 (정수형)

설 명 1) 실수형 변수에 MAX TORQUE 값을 가져옵니다.

프로그램 예제

```
func void main()
    double trqmax

    trqmax = 0

    release
    take 1

    trqmax = read_trqmax(1,1)
end
```

1번 로봇 1번 축 MAX TORQUE 값 읽어오기

3.54.9 READ_VEL

기 능 VELOCITY 를 읽어옵니다.

형 식 실수형 변수 = read_vel(robot_id, 축)

용 어 robot_id: 로봇 번호를 입력합니다.
축: 데이터를 읽어 올 축 설정 (정수형)

단 위 mm/s, degree/s

설 명 1) 실수형 변수에 현재 VELOCITY 값을 실시간으로 가져옵니다.

프로그램 예제

```
func void main()
    double velocity

    velocity = 0

    release
    take 1

    velocity = read_vel(1,1)
end
```

1번 로봇 1번 축 VELOCITY 값 읽어오기

Rev.	수정일자	내용	수정자	S/W Version
V.1	2015.11.16	초판 인쇄		

CONTROLLER MANUAL

FIRST EDITION NOV 2015

ROBOSTAR CO, LTD

ROBOT R&D CENTER