Chapter 8 Command Reference

8-1 BL/1 Grammar

BL/1 is a BASIC-like interpreter. Its basic operation and description method are based on BASIC interpreters.

Program configuration

A program is managed by the unit of line and described in the following order.
Attention should be paid to the limitations on the components and the number of characters.

470	IF	HPT(XIN0)==0	THEN	:	RMVS	X_A 5000	:	END_IF
Statement number	Command	Argument(s)	Reserved word	Colon	Command	Argument(s)	Colon	Command

Line	The maximum number of characters which can be input in a line is 255 bytes.
Command line	Statement number, command, and argument(s)
Number of arguments	Although the number of arguments is determined by the command specification, the maximum number is 14.
Number of characters in an argument	Arguments provide formulas, arithmetic formulas, constants, variables, character strings, labels, and the like. Maximum 100 characters.
Multistatement	Multiple command lines can be described in a line by adding colons.
Program and command	If the statement number is omitted in an executable unit, it is instantly executed. Each executable unit is given its program order according to a statement number.
Comment	The portion after a ‘ (single quote) is not executed as a comment.
Label	A character string starting with a * (asterisk) becomes a label statement, which is not an executable statement but defines an execution starting point for GOTO, GOSUB, FORK, and the like. The number of characters in a label is within the maximum number of characters in an argument.

Variables and constants

Variables	Four-byte integer within 15 characters. A-z, 0-9, $, _, and @ can be used. Up to 2000 variables can be used. The initial value of a variable is ixed to the flash ROM. Although all variables are set to 0 when a program is loaded and RUN after NEW, if it is modified and RUN is repeated in the middle, values at that point of time are retained. Therefore, initialization of variables in a program should be kept in mind.
Task-local variables	Variables with _ added to the end, having independent values in different tasks. AB_ for example. Up to 32 can be defined. The initial value is uncertain.

Character string variables	Variables with $ added to the end, which become character string variables. A$ for example. Up to 128 character strings can be used, and the maximum length of a character string is 255.
Array variables	Array variables, each of which can be declared by giving a label with DIM command. Up to 20000 in total can be used. In addition, a two-dimensional array is also available. Array elements are secured every time a DIM command is executed after RUN. Therefore, all DIM declarations should be placed in a group at the top of a program, so that they can be used as backup variables and the like.
Reserved arrays	Point data P(n)={X(n),Y(n),Z(n),U(n)} n=1 ~ 7000* * 16000 points in MPC-2100. Touch panel shared variables MBK(m)m=0 ~ 8099 Each can also be used partially as a character string array.
Constants	Numerical values BL/1 has in advance. They are used as input options of a command for example.
Character string constants	Character strings surrounded with "" (double quotes). They can be used for communication and character string processing for example. Rem) In a character string constant, control codes become ¥n (LF), ¥r (CR), and ¥t (TAB). ¥ is the same character as backslash, having the same significance.

Formulas and conditional formulas

In BL/1 there is no distinction between a formula and a conditional formula. A conditional formula is a collection of functions and arithmetic operations having a logical value of 1 or 0.
The result of A==B is 1 if it holds true, and 0 if not.
The result of SW(n) is 1 in the ON state, and 0 in the OFF state.
The result of A=B+C is an integer because A is substituted by the sum of B and C.

Therefore, if an ordinary conditional formula includes variables and functions with an integer value mixed, appropriate arithmetic formula and functions must be used so that the result becomes 1 or 0.

C*(A+B)>=1000 →Although arithmetic, the result becomes 1 or 0 by a comparative operator >=.

Comparison operators are as follows:

>	Left-hand side is larger	<	Left-hand side is smaller
>=	Left-hand side is equal or larger	<=	Left-hand side is equal or larger
==	Equal	!=	Not equal (<> can also be used)

In MPC, comparison operators have no distinction from arithmetic operators described later and are handled as binary operators having only 0 or 1 as the operation result.
Therefore, although comparison operators can be used as coexistent with arithmetic operators, attention should be paid to the fact that the operators have no precedence and are executed in order from the left.

1+2>3+4: The result is 4. 1+2>3 is executed first, and 4 is added to the result.
1+2>(3+4): The result is 0, where comparison of both sides is performed last.

As this example demonstrates, although the left-hand side is executed from the left, () become necessary for the right-hand side calculation to be executed first.

Ordinarily, the description becomes simpler by placing complex computation formulas on the left-hand side and only numerical values and variables on the right-hand side.
For example, the following description examples have the same meaning as the comparison, the upper example does not need to define the operation precedence using ().

```
A*B+C*D>E
E<(A*B+C*D)
```

By this specification, MPC can describe the following complex logical process in one formula.

```
((SW(0)==1)&(SW(1)==1)&(DAT>1000))|(SW(2)==1)
```

In addition, in order to clarify the meaning of the formula, there are cases where logical conjunctions such as OR and AND should better be used.

```
(SW(0)==1)&(SW(1)==1)&(DAT>1000) OR SW(2)==1
```

Although arithmetic operations are executed in order from the left, only multiplication and division are given priority over addition and subtraction. If C+A*B, A*B is executed first, then C is added.
Here, if addition is given priority, the priority operation should be explicitly specified by describing it as (C+A)*B. Dyadic operations are prepared as follows. Among them, "," and ";" are word composition operators. In addition, "," is enabled only inside ( ).

```
#prx 1,2
00000001 00000002
#prx (1,2)
00010002
##prx 1;2
01000002
```

Dyadic operators

+	Addition	<<	Left shift
-	Subtraction	>>	Right shift
*	Multiplication	,	Word composition
/	Division	;	Upper byte
%	Remainder	&	Logical product
^	Exclusive Logical sum	|	Logical sum

Character string operations

In character string operations, only addition and comparison (coincidence) are allowed.

```
A$=C$+"1234"
IF A$==C$ THEN
```

In other character string processing, the concept of point is introduced, allowing an efficient character string processing. See functions such as SERCH, SUBST, and VAL. There is a character string array P$() which uses the point data area as character strings.

Vector arguments

In XY robot commands, four-dimensional vector quantities are often dealt with. The four-dimensional elements are the orthogonal three-dimensional coordinates X, Y, and Z, and U which corresponds to the attitude axis. There are two kinds of expressions of this vector quantity: One which uses P(n), and the other which directly specifies the coordinate values. In the case of coordinate specification,

Point expression	Coordinate value expression
JUMP P(n)	JUMP VALx VALy VALu VALz
MOVS P(n)	MOVS VALx VALy VALu VALz
BACKLASH P(n)	BACKLASH VALx VALy VALu VALz

Special rules in expressing the coordinate values:	Examples:	Significances:
The axis with VOID as its argument is not operated	MOVS VOID 100 200 VOID	X and Z are not operated.
VOID is given to any missing argument.	MOVS 1900 200 300	Z is no operated.
Axes specified with one value are given the same value.	MOVS X_A|Y_A 100	Both X and Y are given 100, others are not operated.

Further, the following specifications are possible in point expression.
Vector argument = { Axis specification + P(n) + AD_P( ) }

MOVS VOID_U P(1)	Argument is P(1). U axis is not operated due to VOID_U.
MOVS P(1) AD_P(X_A,VAL)	P(1) as an argument. VAL is added to the X value.
MOVS P(1) AD_P(P(100))	P(1) as an argument. P(100) is added to P(1).
MOVS VOID_U P(1) AD_ P(P(100))	P(1) as an argument. P(100) is added to P(1). U axis is not operated.

Commands to which this vector argument is applicable are STPS, BACKLASH, JUMP, JMPZ, MOVS, and MOVL.
Because RMVS, RMVL, and RMVC are relative movement commands, vector arguments cannot be used.

8-2 Command Reference

@

Operation — Function

■ Format
@(arg)

■ Usage
IF @(A==1) THEN
IF @((A!=1) &(B!=1)) THEN

■ Function
Logical inversion

■ Explanation
Logical inversion which converts 1 into 0 and 0 into 1.
While NOT() is for inversion of the long type, @() returns only 0 or 1.
* Rem: Although @(&h101) was converted into &h100 in ver. 1.908 or older, the upper bit is masked from ver. 1.909.

@SW

IO — Function

■ Format
@SW(arg)

■ Usage
IF (@SW(0)&SW(1))==1

■ Function
Inverted reading of the input port

■ Explanation
The value of SW is logically inverted and returned.

```
LIST
10          ON  -1 -3 -5
20          PRINT  @(SW(-1))|@(SW(-3))|@(SW(-5))
30          ON  -1 -3 -5
40          PRINT  @SW(-1)|@SW(-3)|@SW(-5)
50          PRINT  SW(-1)|@SW(-3)|@SW(-5)
#run

*
Compiling
---------------------
0
0
1
#
```

ABS

Operation — Function

■ Format
ABS(arg)

■ Usage
A=ABS(-100)

■ Function

Obtaining the absolute value

■ Explanation

The argument is converted into a positive integer and returned.

```
A=-123
A=ABS(A)
A becomes 123.
```

ACCEL

Pulse generation Command

■ Format

ACCEL [axis] PPS [leng,lo_pps]

■ Usage

ACCEL 4000
ACCEL 4000 1000 100
ACCEL Z_A 8000
ACCEL SACL 4000
ACCEL X_A|SACL 2000
ACCEL X_A|OUTSL 30000

■ Function

Setting an acceleration

■ Explanation

PG acceleration is set. If the axis specification is omitted, it applies to all of the axes. Specified parameters are maximum speed (pps), acceleration distance (pulse), and self start (pps).
If the acceleration distance and the following are omitted, default values are set.
This command cannot be used during pulse generation. The SPEED command should be used during pulse generation. In addition, if the self-start speed is set low, the whole movement slows down.
Recommended minimum speed is about 100 pps for a stepping motor, and about 1 kpps for a servo motor. (If the minimum speed is set to 1 pps, outputting one pulse takes one second. If the beginning and end of a movement take 1 pps each, they require two seconds.)

1) If OR is taken between an axis specification parameter and a constant SACL, S-curve acceleration/deceleration is realized.

```
Example: ACCEL X_|SACL 80000
```

(In both MPG-2314 and MPG-2541 since 1.11_58 2009/04/30, even if S-curve is speci-fied, trapezoidal acceleration/deceleration speed and acceleration/ deceleration time would not change.)

2) If OR is taken between an axis specification parameter and a constant OUTSL, the RANGE setting value and the current position are compared and reflected on the output port. (MPG-2314 CEP128D or later)

```
Case of RANGE X_A 10000 XXX:     O0 is OFF if X(0) is up to 9999, ON if 1000 or larger.
Case of RANGE X_A -10000 XXX:    O0 is OFF if X(0) is up to -1001, ON if -10000 or larger.
```

*O0 is MPG-2314 J4-19.
If executed without any argument, set parameters and set ACCEL statement number are displayed. If the statement number is 0, it means it is not set by the program.

```
#accel
X=> Max=3000 Length=150 Min=300 Feed=100 Set@20
Y=> Max=3000 Length=150 Min=300 Feed=100 Set@30
U=> Max=8000 Length=400 Min=800 Feed=100 Set@0
Z=> Max=3000 Length=150 Min=300 Feed=100 Set@40
#
```

ACOS,ATAN

Floating point — Function

■ Format

ATAN(v)
ACOS(v)

■ Usage

FP(0)=DEG(ACOS(1/SQR(2)))
FP(1)=DEG(ATAN(1))

■ Function

Inverse trigonometric functions

■ Explanation

Double-precision inverse trigonometric functions which output an argument in radians. These have a meaning in FLOAT command only.

```
FP(0)=DEG(ACOS(1/SQR(2)))
FP(1)=DEG(ATAN(1))
```

AD

AD_DA — Function

■ Format

AD(ch)
AD(fnc,ch)

■ Usage

A=AD(0)
IF AD(1,7)>500 THEN

■ Function

Acquiring MPC-AD12 data

■ Explanation

[1 msec sampling]

Function AD(ch) returns the converted value of the AD converter. The ch numbers are 0~7. These data are updated at every 1 msec. If one more MPC-AD12 board is added, 8~15 should be specified as the ch numbers. The returned values are 0~4095 1mV/1digit with AD7890-4 (standard shipped state) installed, and -2048~2047 1mV/1digit with AD7890-10 installed.

[How to obtain the average value] (Value averaged over eight data by default)

AD(1,ch) returns the average value. The number of data to be averaged is specified with the SET_AD command.

[Automatic continuous data acquisition]

MPC-AD12 acquires data at every 1 msec and can acquire and refer to the data 832 times continuously.

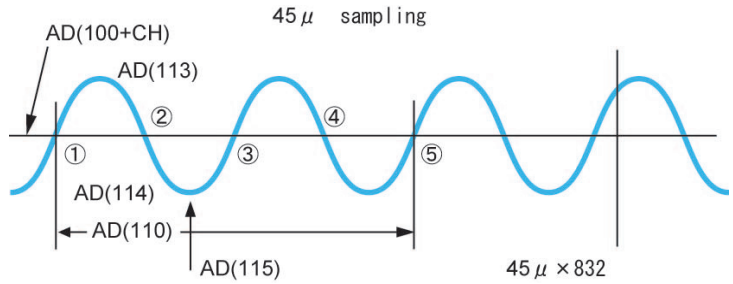
AD(2,ch) starts continuous data acquisition at 1 msec sampling rate.
AD(4,ch) starts continuous data acquisition at 2 msec sampling rate.
AD(3,ch) waits for the completion of acquisition.
Data are extracted with AD_D(0,n), where n is 0~831.

[8-CH automatic continuous data acquisition]

If ch is specified to 8, all ch simultaneous sampling is performed. In this case, the rate is fixed to 1 msec.
For each ch 104 data are acquired. (0.1 sec)
AD(2,8) starts continuous data acquisition at 1 msec sampling rate.
AD(3,0) waits for the completion of acquisition.
Data are extracted with AD_D(ch,n), where ch is 0~7, and n is 0~103.

[45-µsec sampling] See figure.

AD12 also has a 45-µsec high-speed sampling function.
AD(100+CH) has a specified channel acquire 832 data at a 45 µsec cycle (for about 37 msec). The returned value is the average of the first 400 data. Data are extracted with AD_D(0,n), where n is 0~831.
The following results can also be obtained by sampling a periodic signal of about 1~20 msec with 45-µ sampling.



This function is built-in for AC signal analysis. It is not appropriate for aperiodic, noise signals.

AD(110): Returns the number of two periods. (The number of samples in the section (1)-(2) with the average value as the boundaries)

AD(112): Calculates the average, maximum, and minimum values, and the position of the minimum value of the acquired samples based on the AD(100+CH) value. The directly returned value is the average value.

AD(113): Maximum value

AD(114): Minimum value

AD(115): Position of the minimum value

AD(n,100+CH): Performs sampling of n data 45 µsec, and returns the average value. (Data acquisition after the period is found)

```
'1msec SAMPLING
 SYCSCLK=0
   FOR   i=0 TO 1440 STEP 2
    WAIT   i==SYSCLK
    X(i+1000)=AD(0)
   NEXT

'Bulk Sampling
SYSCLK=0
 dmy=AD(2,ch)
  PRINT   AD(3,ch) SYSCLK "1msec"
```

```
 PRINT   "dump"
 FOR   i=0 TO 800 STEP 50
  PRINT   i AD_D(0,i)
 NEXT
'8CH Bulk sampling
dmy=AD(2,8)
 PRINT   AD(3,0) SYSCLK
 FOR   i=0 TO 100 STEP 10
  PRINT   i AD_D(0,i) AD_D(1,i) AD_D(2,i) AD_D(3,i)
 NEXT
'45usec sampling
 140     *get45
160      PRINT   " GET " AD(100) cyc=AD(110) a "AV= " AD(cyc,100) " Hz=" 2000000/(45*cyc)
190     dmy=AD(112)
200      PRINT   " MAX=" AD(113) " MIN=" AD(114)
205      PRINT   "from MIN=>"
210      FOR   i=AD(115) TO AD(115)+10
220       PRINT   i AD_D(0,i)
230      NEXT
#run 140
140-
 GET  1059 357 8 AV=  874  Hz= 124 125
 MAX= 1835  MIN= 64
 from MIN=>
 166   64
 167   111
 168   192
 169   269
 170   341
 171   410
 172   476
 173   539
 174   600
 175   657
 176   712
#
```

ADD_MBK

Touch panel Command

■Format

ADD_MBK add_value adrs

■Usage

add_mbk 1000 1

■Function

Direct addition of MBK() array

■Explanation

Data in array MBK() are directly added.

```
#pr mbk(1)
1000
#add_mbk 1000 1
#pr mbk(1)
2000
#
```

ADD_STR

Character string Command

■Format

ADD_STR Str [Str]

■Usage

ADD_STR "Win" a$
ADD_STR "7"

■Function

Appending a character string

■Explanation

ADD_STR appends a character string to a specified character string.
At first, the character string variable to append to is specified, and the initial value is given.

```
ADD_STR "Win" a$
```

At a point in time, Win is copied to a$. Afterwards, only by specifying a character string to append, the characters are added.

```
ADD_STR "7"
```

As a result, a$ becomes Win7. ADD_STR can also append a null code by the following description.

```
ADD_STR chr$(0)
```

This sample program describes a case for outputting 01, 03, 00, 00, 01, and 01.

```
CNFG#  2 "38400b8pns1NONE"
    CH=2
     ADD_STR  CHR$(1) SEND$
     ADD_STR  CHR$(3)
     ADD_STR  CHR$(0)
     ADD_STR  CHR$(0)
     ADD_STR  CHR$(1)
     ADD_STR  CHR$(1)
     PRINT#  CH STR_LEN|6 SEND$
    END
```

AD_D

AD_DA Function

■Format

AD_D(ch,n)

■Usage

a=AD_D(0,1)

■Function

Reading out data which are continuously taken in

■Explanation

Continuously sampled data are taken out.
In the case of a single channel, AD_D(0,n), where n is 0~831.
In the case of all channels, AD_D(ch,n), where ch is 0~7 and n is 0~103.

AD_P

Pulse generation — Function

■Format

AD_P(axs,n) n=+/-32767
AD_P(P(n))

■Usage

MOVS P(n) AD_P(X_A,1000)
MOVS P(n) AD_P(X_A,1000) AD_P(Z_A,-1000)
JUMP P(n) AD_P(P(m))

■Function

Moving point correction

■Explanation

A correction value is added to a point data argument (coordinate values) of MOVS and the like. It is used for stopping at a specified point. It can also be used for pausing at an X, Y point in image processing.
When a point data are specified, 4-axis coordinate values are added as they are.
In this operation, point data themselves are not modified. The correction range for the case of AD_P(axs,n) is +/-32767.

```
MOVS P(5) AD_P(X_A,1000) =>MOVS X(5)+1000 Y(5) U(5) Z(5)
MOVS P(6) AD_P(X_A,1000) AD_P(Z_A,-1000) =>MOVS X(6)+1000 Y(6) U(6) Z(6)-1000
```

AFFIN

Floating point — Command

■Format

AFFIN n m k deg

■Usage

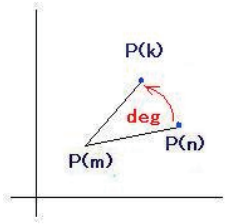
AFFIN 2 1 3 i*10000

■Function

Point rotation operation

■Explanation

P(n) is rotated centering on P(m) by deg degrees, and the result substitutes P(k).
The angle is given by a value multiplied by 10000.
In this example, the X-direction horizontal line is rotated in the ccw direction by 30 degrees.




```
#setp 1 10000 20000  0 0
#setp 2 1010000  20000   0 0
#affin 2 1 3  300000
#pr p(3)
 876025 520000 0 0
#
```

ALL_A

Pulse generation — Reserved constant

■Format

ALL_A

■Function

All axis specification

■Explanation

Applicable boards: MPG-2314/2541

```
ACCEL ALL_A 30000 1000 500                  /* Acceleration/deceleration setting
FEED ALL_A 100                              /* Speed setting
INSET ALL_A MD_2PLS|ALM_OFF|LMT_OFF         /* In port set
STOP ALL_A STP_D                            /* Moving stop with deceleration
WAIT RR(ALL_A)==0                           /* Wait until moving complete
  etc
```

ALL_E

Pulse generation — Reserved constant

■Format

ALL_E

■Function

All axis error specification

■Explanation

Applicable boards: MPG-2314

Presence/absence of an error after a movement is examined. It indicates that one of the following bits stood up.

RR1 register (Driving completion status) ENG, ALARM, LMT-, and LMT+

RR2 register (Error information) ENG, ALARM, HLMT-, HLMT+, SLMT-, and ALMT+

```
100  MOVL P(1)
110  WAIT RR(ALL_A)=0
120  IF RR(ALL_E) !=0 THEN          /* Confirming error status
130    PRINT "ERROR STOP"
140  ELSE
150    PRINT "NORMAL STOP"
160  END_IF
170  PRX RR(ALL_E)
```

ALM

Pulse generation — Reserved constant

■Format

ALM

■Functin

Error bit specification

■Explanation

Applicable boards: MPG-2314

Alarm signal bit

```
IF LMT(X_A,ALM) !=0 THEN          /* confirming reason for stop
```

ALM_OFF

Pulse generation — Reserved constant

■Format
ALM_OFF

■Function
Alarm setting

■Explanation
Applicable boards: MPG-2314
Enabled with alarm OFF

```
INSET X_A ALM_OFF              /* X-axis 'ALARM' enabled on signal 'OFF"
```

ALM_ON

Pulse generation — Reserved constant

■Format
ALM_ON

■Function
Alarm setting

■Explanation
Applicable boards: MPG-2314
Enabled with alarm ON

```
INSET X_A ALM_ON               /* X-axis 'ALARM' enabled on signal 'ON'
```

APPEND

USB — Command

■Format
APPEND [USB] Str

■Usage
APPEND "data1.txt"
APPEND USB1 "data1.txt"

■Function
Open for writing USB data (Appending)

■Explanation
Setting a USB memory open for writing.
If the file exists, it is appended. If not, it is newly created and appended.
Writing is performed with PRINT# USB. After the writing is complete, CLOSE is executed.
If CTRL_A is given during the execution, CLOSE processing is automatically performed.

```
10      USB_DEL  "AA.TXT"
20      APPEND  "AA.TXT"
30      FOR  I=0 TO 10
40       PRINT#  USB "TEST=" STR$(I) "
"
50      NEXT
60      CLOSE
LIST
10      USB_DEL  "AA.TXT"
20      APPEND  "AA.TXT"
```

```
30      FOR   I=0 TO 10
40       PRINT#   USB "TEST=" STR$(I) "
"
50      NEXT
60      CLOSE
#run

A:>
#type "AA.TXT"
TEST=0
TEST=1
TEST=2
TEST=3
TEST=4
TEST=5
TEST=6
TEST=7
TEST=8
TEST=9
TEST=10

A:>
```

ASC

Character string Function

■Format

ASC(str)
ASC(arg)

■Usage

ASC(a$)
ASC(4)

■Function

Obtaining the ASCII code of a character string

■Explanation

When a character string is given as the argument, the character code of its top character is returned.
If a numerical value of 0~4 is given, the character codes of the given number of characters starting from the ptr_ position are read out. Therefore, comparison of character strings of four characters or shorter can be easily performed.

```
10      a$="123abcABC456"
20       PRINT   ASC(a$)
30       SERCH   a$ "abc"
40       FOR   i=0 TO 4
50        PRX   ASC(i)
60       NEXT   i
#run

49
00000041
00000041
00004241
00434241
34434241
#
```

ATAN

Floating point Command

■Format

atan y r var [x]

■Usage

atan 10000 1000 a
atan 100000 1000 a 173205

■Fonction

ATAN operation

■Explanation

ATAN floating-point operation is performed. The result is converted into an integer in degrees.The magnification in the integer conversion can be decided using r.

ver=r×atan(y/x)

Rem 1) If x is omitted, x is set to 10000.
Rem 2) The range of result (degrees) is -90 ~ +90.

Example 1) atan 10000 1 a
This calculation is the ATAN value of an isosceles right triangle, and the result is 45 (degrees).

a=1×atan(10000/10000)

Example 2) atan 17321 1000 a 10000
This calculation is the ATAN value of a 60-degree right triangle.

a=1000×atan(17321/10000)

Because 60 degrees is magnified 1000 times, it becomes a value of 60000.

```
#atan 10000 1000 a
#pr a
 45000
#atan 100000 1000 a 173205
#pr a
 30000
#atan 17321 1000 a 10000
#pr a
 60000
#
```

ATAN2

Floating point Command

■Format

atan2 y x var [r]

■Usage

atan2 100000 100000 a3
atan2 100000 173205 a3 10000

■Function

ATAN operation

■Explanation

ATAN floating-point operation is performed. The difference from ATAN is only the order of the arguments. The result is converted into an integer in degrees. The magnification in the integer conversion can be decided using r.

var = r×atan2(y/x)

Rem 1) If r is omitted, r is set to 10000.
Rem 2) If y > x, atan2(x/y) is calculated, and an angle is calculated based from its supplementary angle. Therefore, a correct value can be returned even if x = 0.
Rem 3) The range of the result (degrees) is -90 ~ +90.

Example 1) atan2 10000 10000 a
The ATAN value of an isosceles right triangle is calculated, and the result is 45 (degrees). Because r is omitted, the result is multiplied by 10000.

a=10000×atan(10000/10000)

Example 2) atan2 173205081 100000000 a 100000
The ATAN value of a 60-degree right triangle is calculated.

a=100000×atan(173205081/100000000)

Because 60 degrees are multiplied by 100000, it becomes a value of 6000000.

```
#atan2 100000 100000 a
#pr  a
 450000
#atan2 100000 173205 a 10000
#pr  a
 300000
#
```

AVOID

IO — Reserved constant

■Format
AVOID

■Function
Disabling a command

■Explanation
Disabling a command.

```
10      CONST   sol1 AVOID      /* not use
20      CONST   sol2 1
30      ON   sol1 sol2          /* sol1 disable, sol2 enable
```

BACKLASH

Pulse generation — Command

■Format
BACKLASH Xb Yb Ub Zb

■Usage
BACKLASH 111 121 0 0

■Function
Backlash correction setting

■Explanation
Backlash correction is given to the pulse output of MPG-2314.
The backlash correction is enabled only for a single-axis, linear interpolation. It is not applied to circular interpolation.

It is 0 after a power-on reset, and it needs to be set every time after power cut-off.
Backlash correction is performed by adding a backlash-set pulse at a point where the pulse generation direction changes.
Its use requires the caution that the backlash status of the mechanical system needs to be initialized in advance.
For example, after returning to the origin, a CW-direction dummy movement is performed by more than the backlash amount to set the backlash value to a positive value.
As far as the pulse generation direction is the same as the backlash value, backlash pulse addition is not performed. However, when the pulse generation becomes the negative direction, the backlash value is converted into a negative value, and pulse addition is performed.
Therefore, the backlash value is internally inverted from negative to positive by an operation which also monitors the direction.

The backlash correction is not almighty.
The backlash amount of the mechanical system varies due to conditions such as moving speed, load, and vibration.
Its use requires a good grasp of the characteristics of the mechanical system.

BAT

Maintenance — Command

■Format
BAT(arg)

■Usage
IF BAT(0)==1 THEN : PRINT "Battery error" : END_IF

■Function
Obtaining battery error number

■Explanation
This is a function which indicates whether the CPU correctly entered the retreat state at the last powering off. If 0 returns, it is normal. If 1 returns, the CPU has an abnormality at the power shut-off, or the backup battery expired. If there is a battery error, there is a possibility that point data, MBK data, or the like have been destroyed.

BATTERY

Maintenance — Reserved variable

■Format
BATTERY

■Usage
IF BATTERY != 0 THEN
MBK(20)=BATTERY

■Function
Battery status

■Explanation
In MPC-2000 with a battery built-in, if the battery voltage drops at the time of powering off, an indication of BATTERY OUT or BATTERY LOW is displayed immediately after powering on.
BATTERY OUT indicates that either the battery is completely consumed or the battery itself is taken out.

BATTERY LOW indicates that the battery voltage is low.
Reserved variable BATTERY becomes 1 in the case of BATTERY OUT and 2 in the case of BATTERY LOW.

BREAK

Control statement — Statement

■Format
BREAK

■Usage
```
DO
IF SW(0)==1 THEN : BREAK : END_IF
LOOP
```

■Function
Cancelling an iterative execution in FOR-NEXT, DO-LOOP, or WHILE-WEND.

■Explanation
In cancelling an endless execution with multiple conditions, a clearer expression can be made by describing it with DO-LOOP and executing BREAK in an IF statement. The REAK statement can be described anywhere in the loop multiple times.

BREAK_POINT {BKP}

Maintenance — Command

■Foemat
BKP [args]

■Usage
```
BKP 100
BKP 100 110
BKP *aa
BKP 0
```

■Function
Setting a break point

■Explanation
With BREAK_POINT command, a program can be stopped at up to eight specified statement numbers. (Label specification can also be made.)
As shown in the example program, when a program number is specified, the specified line is displayed.
Afterwards, the statement number of the specified line is displayed inverted on FTMW. As break points, statement numbers should be specified in order. To release a specified statement number, the same number should be input.
To view registered statement numbers, the BKP command should be executed with no argument. In addition, to release all the break points, BKP 0 should be entered.

If a break has occurred,
1) If RUN is issued with a break point actually specified, execution is paused at the specified position. Then, the paused line and the task number are displayed. Execution is resumed until the next break point with the n key. In this program, a break occurs every time it passes the statement number 30 before the execution.
2) To perform step forwarding (executing continuously line by line), t should be pressed. To release step forwarding, n should be pressed.

3) During a break, values of variables and functions can be referred to.
   After pressing 'p' a variable or function name should be entered.
4) A break point may be added.
   After pressing 'b' and entering a statement number, a break point can be added.
5) To release a break point during the break point, "u" should be entered.
6) To stop the program execution, 'e' should be pressed.

With FTMW6.39s or later, break points can be used in the menu.

```
30      FORK   2 *bb
40      END
110    *bb
120     DO
130      FOR   i_=8 TO 15
140       ON   i_ : TIME   50 : OFF   i_
150      NEXT
160     LOOP
#bkp 110 140

110    *bb
140     ON   i_ : TIME   50 : OFF   i_
#bkp
BREAK_POINT 0=110
BREAK_POINT 1=140
#bkp 110

110    *bb
#bkp
BREAK_POINT 0=140
#
```

CANCEL_RETURN

Control statement Statement

■Format

CANCEL_RETURN

■Usage

CANCEL_RETURN : GOTO *AAAA

■Function

Discarding the RETURN stack

■Explanation

This is a prohibited tactic.
The RETURN stack is discarded. It is used when jumping to a label and the like in the parent routine instead of returning from a subroutine with a RETURN statement. It should not be used indiscriminately..

```
FOR   i=1 TO 100
 s=0
  GOSUB   *aho
 NEXT
 PRINT   "normal" i j s
 END
*baka
 PRINT   i j s
 GOTO   *init
*aho
 FOR   j=0 TO 100
```

```
s=s+j
 IF  j==50 THEN  : CANCEL_RETURN  : GOTO  *baka : END_IF
NEXT  j
RETURN
```

CCW

Pulse generation — Reserved constant

■Format

CCW

■function

Origin return search direction specification
Circular interpolation specification

■Explanation

In SHOM the Z-phase search direction for origin search is specified. In MOVT the rotation direction for circular interpolation is specified.

```
SHOM X_A|Y_A IN0_ON|CCW  /* set HOME condition. CCW movement until the sensor turns on
MOVT X_A|Y_A P(102) CCW      /* continuous interpolation. CCW revolution.
RMVC X_A CCW                          /* infinite pulse generation. CCW movement.
```

CHR$

Character string — Function

■Format

CHR$(arg)

■Usage

a$=CHR$(15)
print# 1 chr$(10)

■Function

Generating one character

■Explanation

It generates a character which cannot be expressed by a$=”slkd” and the like.
CHR$(1) ==> SOH for example.

CHR_C

Communication — Reserved constant

■Format

CHR_C

■Function

Setting the number of received characters

■Explanatin

The number of received characters is set.

```
10      CNFG#  1 "9600b8pns1NONE"
20      INPUT#  1 CHR_C|1 a$                    /* receive 1 character
30      PRINT  a$
#RUN

 a                                              /* send 'a' from the terminal soft
```

CK_Z,CK_NZ

Operation — Function

■Format

CK_Z(arg)
CK_NZ(arg)

■Usage

IF SW(1)&SW(2)|CK_Z(A) THEN : PRINT "OK" : END_IF

■Function

Zero test and non-zero test

■Explanation

Returns 1 if the CK_Z(arg) argument value is 0, and 0 if not 0.
Returns 0 if the CK_NZ(arg) argument value is 0, and 1 if not 0.

CLOSE

USB — Command

■Format

CLOSE [USB]

■Usage

CLOSE
CLOSE USB1

■Function

Closing USB port(s).

■Explanation

Closes USB port(s) opened by APPEND or OPEN.
If there is no argument, all open USB ports are closed.
If there is an argument (CLOSE USB for example), only the specified port is closed.

CLRPOS

Pulse generation — Command

■Format

CLRPOS [AXIS],[-1]

■Usage

CLRPOS
CLRPOS X_A
CLRPOS -1
CLRPOS X_A -1

■Function

Clearing the position counter and the encoder counter.

■Explanation

If there is no argument, current position is set to all 0.
If there is an axis specification constant, the target axis is set to 0. If -1 is given, the encoder counter is set to 0.
In the case of CLSPOS X_A-1, the X-axis encoder counter is cleared.

CLR_OUTP

IO Command

■Format

CLR_OUT arg

■Usage

CLR_OUTP 1|8
CLR_OUTP 15

■Function

I/O area initialization

■Explanation

CLR_OUTP [n]
n=1: Real output port
2: CUNET
4: MBK
8: Memory IO

Being a bit parameter, it is executed by setting the bits corresponding to the necessary initialization area to ON.
CLS_OUTP 15 initializes all.

CMP_C

Pulse generation Function

■Format

CMP_C(axis)
CMP_C(port,axis)

■Usage

WAIT CMP_C(X_A)==2
A=CMP_C(16,X_C)

■Function

Referring to the results of comparing the counter and COMP+/-.
If the results of comparing the counter and COMP+/- has changed, a specified port is set to ON.

■Explanation

MPG-2314 has COMP+ and COMP- registers and can compare the counter and the COMP registers in real time. The comparison result is referred to with CMP_C function.

```
CMP_C() = [BIT0 <= CMP+ ,BIT1 <= CMP-]
CMP+
1:  Counter value >= COMP+ register
0:  Counter value < COMP+ register
CMP-
1:  Counter value < COMP- register
0:  Counter value >= COMP- register
```

In addition, as the comparison counter value, the current position counter or the encoder counter can be chosen. If a setting of INSET X_1 CMP_PLS is made, the result of comparing the pulse position and the COMP registers can be found with CMP_C(X_A).
In the case of INSET X_A CMP_CNT, a comparison is made with the encoder counter.

COMP registers can be set with RANGE command.

```
RANGE X_A COMP+ COMP-
```

As in the example program, when described as CMP_C(port,X_A), after waiting for a change in the comparison flag, the specified port is set to ON, and the program is exited. In this case, if either bit of CMP+ or CMP- changes, a change is detected.

```
40     ACCEL  30000
50     CLRPOS
60     INSET  CMP_PLS
65    P_DET=500
70     RANGE  X_A P_DET P_DET
80     RMVC  X_A 1
100     DO
110    A=CMP_C(16,X_A)
120      INC  P_DET 500
130      OFF  16
135      RANGE  X_A P_DET P_DET
140     LOOP
```

CMP_CNT

Pulse generation — Reserved constant

■Format

CMP_CNT

■Function

Counter comparison

■Explanation

Applicable boards: MPG-2314

Compares the encoder counter and COMP+.

See also INTA_ON

```
INSET X_A CMP_CNT|PHASE1
```

CMP_P

Pulse generation — Function

■Format

CMP_P([axs,],v)

■Usage

CMP_P(n)

CMP_P(axs,n)

■Function

Comparison between the current position and point data

■Explanation

Compares the current position and specified point data. If there is no axis specification, values of all axes X, Y, Z, and U are compared, and returns 1 if they are all the same, and 0 if even one axis has a different value.

If an axis specification is given, comparison is made only for the specified axis.

```
10     PG  0
15     CLRPOS
20     ACCEL  8000
30     SETP  7000 10000 20000 30000 40000
40     MOVS  P(7000)
50     DO
60      IF  CMP_P(7000) THEN : PRINT  "Arrived" : BREAK  : END_IF
```

```
70       TIME   100
80       LOOP
90       RMVS   Z_A 100
100       WAIT   RR(Z_A)==0
110       PRINT   CMP_P(7000)
120       PRINT   CMP_P(VOID_Z,7000)
#run
 Arrived
 0
 1
#
```

CMP_PLS

Pulse generation — Reserved constant

■Format

CMP_PLS

■Function

Counter comparison

■Explanation

Applicable boards: MPG-2314
Comparison between the current pulse counter and COMP+
See also INTA_ON

```
INSET X_A CMP_PLS
```

CNFG#

Communication — Command

■Format

CNFG# COMn [RS485] "setting"

■Usage

CNFG# 1 "38400b8pns1NONE"
CNFG# 5 RS485 "38400b8pns1NONE"

■Function

RS-232C port initialization

■Explanation

COMn is the RS-CH number to be initialized. The character string contains the baud rate and character format.
COMn = 1 MPC-2000/2100 USER ch1
COMn = 2 MPC-2100 USER ch2 (Missing in MPC2000)
COMn = 3 MRS(DSW == 6) J4
COMn = 4 MRS(DSW == 6) J5 (RS422/485 shared)
COMn = 5 MRS(DSW == 6) J6 (RS422/485 shared)

A baud rate should be chosen from 4800, 9600, 19200, and 38400.
b8: 8-bit character
b7: 7-bit character
pn: No parity
pe: Even parity
po: Odd parity
s1 1stop bit
s2 2 stop bit

NONE: No XON/XOFF control (Not compatible with XON/XOFF control)

If RS485 is added as an argument, RS485 communication is enabled through the RS422/485 shared port.
The example program is for the case where An RS485 thermometer/hygrometer manufactured by CHINO is connected.
Data are written to the USB memory every minute.
* USB memory write and RS485 are required to be updated to MRS-MCOM (20081107).

```
CNFG#   5 RS485 "9600b7pes1NONE"
 FORMAT   "data00.txt"
f=0
 APPEND   STR$(f)
flag=1
 GOTO  *start
 DO
  WAIT   (&h00FF&TIME(0))==0
*start
  PRINT#   5 CHR$(5) "01" CHR$(2) "RPV01" CHR$(3) "¥n¥r"
  INPUT#   5 a$
  PRINT   VAL(a$) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0)
  PRINT   o=VAL(10) VAL(0) h=VAL(10)
  FORMAT   ""
  PRINT#   20 HEX$(TIME(0)) " B " page " ondo=" o " hum=" h "¥n¥r"
  PRINT#   5 CHR$(5) "02" CHR$(2) "RPV01" CHR$(3) " ¥n¥r"
  INPUT#   5 a$
  PRINT   VAL(a$) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0) VAL(0)
  PRINT   o=VAL(10) VAL(0) h=VAL(10)
  FORMAT   "T000"
 l$=STR$(o)
  FORMAT   "H000"
 l$=l$+STR$(h)
  PR_LCD   l$
  FORMAT   ""
  PRINT#   20 HEX$(TIME(0)) " A " page " ondo=" o " hum=" h "¥n¥r"
  IF   &hFF00&TIME(0)==0 THEN
   FORMAT   "data00.txt"
   CLOSE   :  f=f+1 : APPEND   STR$(f)
  END_IF
  WAIT   (&h00FF&TIME(0))!=0
 LOOP

==data00.TXT==
0173700 B 6 ondo=226 hum=366
00173700 A 6 ondo=226 hum=376
00173800 B 7 ondo=225 hum=368
00173800 A 7 ondo=226 hum=380
00173900 B 8 ondo=225 hum=365
00173900 A 8 ondo=226 hum=381
00174000 B 9 ondo=225 hum=369
00174000 A 9 ondo=226 hum=377
00174100 B 10 ondo=224 hum=353
00174100 A 10 ondo=225 hum=377
00174200 B 11 ondo=225 hum=357
00174200 A 11 ondo=224 hum=377
00174300 B 12 ondo=225 hum=356
00174300 A 12 ondo=224 hum=373
00174400 B 13 ondo=224 hum=358
00174400 A 13 ondo=224 hum=377
00174500 B 14 ondo=224 hum=358
00174500 A 14 ondo=224 hum=378
```

COMPOWAY

Character string Command

■Format

COMPOWAY n m l str1$ str2$
COMPOWAY str1$ v1 v2 v3 str2$

■Usage

COMPOWAY 1 2 0 cmnd$ buff$
COMPOWAY buff$ nod adr id rcv$

■Function

Generation and decomposition of character strings in the OMRON COMPOWAY format

■Explanation

OMRON COMPOWAY employs the following command format.
Sending: ADR+SADR+ID+CMND character string
Receiving: ADR+SADR+END+RES character string
COMPOWAY command efficiently generates and analyzes both character strings.
Generation: Specified address, subaddress, and command character string (cmnd$) are stored in buff$.

```
COMPOWAY 120 cmnd$ buff$
```

Decomposition: A received character string buff$ is stored in nod adr id, and the response character string in res$.

```
COMPOWAY buff$ nod adr id res$
```

Nod, adr, and id are numerical value data contained in the regular format of COMPOWAY. Because res$ has different responses depending on the command, read-out detection is made appropriately based on the specification of the connected equipment.
BCC error is reflected on rse_ after executing input# COMPOWAY command.
(0 indicates normal, and 4 BCC error.)

```
*RS-485_SEND_READ
 _VAR data_len
 cmnd_txt$=mrc_src$+hensu_shu$+str_adr$+bit_ichi$+yoso_su$+setteichi$
 COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
 PRINT# 5 COMPOWAY snd$
 INPUT# 5 COMPOWAY rcv$
 IF rse_!= THEN
// WHEN rse_ is 4 , BCC error happend , OTHER cases indicates RS-232c errors
   PRINT "communication error"
 END_IF
 COMPOWAY rcv$ node_no sub_adr end_code res$
 ptr_=res$+4
 res_code=HEX(PTR$(4))
 ptr_=res$+8
 res_data$=PTR$(data_len)
 RETURN
```

CONST

Operation Command

■Format

CONST var val

■Usage

CONST A_P 123

■Function
Converting a variable into a constant

■Explanation
Converts a variable into a constant so that it cannot be changed.

CONT

Multitasking Command

■Format
CONT arg

■Usage
CONT 8

■Function
Resuming a SLEEPING task

■Explanation
Resumes a task which is paused with PAUSE command.

COS

Floating point Command

■Format
cos deg r var [sf]

■Usage
cos 450000 100000 a
cos 4500000 100000 a 100000

■Function
COS operation

■Explanation
Performs a floating-point COS operation.
var = r×cos(deg/sf)
Rem) If sf is omitted, sf is set to 10000.

```
1) COS 600000 10000 a
#pr a
5000
#
```

This is an operation of cos(600000/10000) which is cos(60 degrees). Although the result is 0.5, because 0.5 is multiplied by 10000, 5000 is output.

```
#cos 450000 100000 a
#pr a
 70711
#
#cos 4500000 100000 a 100000
#pr a
 70711
#
```

CP

Pulse generation Command

■Format

CP

■Function

Displaying the current position

■Explanation

Displays the current position in the coordinate control of an MPG board

CSW

IO Function

■Format

CSW(arg)

■Usage

A=CSW(0)
IF A==1 THEN : GOSUB *A :ELSE : GOSUB *B

■Function

Waiting until a specified input port changes and returns the value after the change.

■Explanation

CSW(n) function itself contains a wait (polling). It continues to wait until the input status changes.

```
10          FORK  1 *task1
20          END
30         *task1
40          DO
50         A=CSW(-1)
60           PRINT  A
70          LOOP
#run

#on -1
#off -1
#0
on -1
#1
off -1
#0
```

CTRL_A

Maintenance Command

■Format

CTRL_A [val]

■Usage

CTRL_A 1
CTRL_A 0

■Function

Setting the CTRL_A function

■Explanation

Specifies whether a program may start after receiving SOH (CTRL_A) at the program port.
CTRL_A 0 : If the status of J1-5 and 6 is open, the program is restarted.
(Standard state)
CTRL_A 1 : Regardless of the status of J1-5 and 6, the program is not restarted.

CUNET

CUnet Command

■Format

CUNET arg1 arg2 arg3

■Usage

CUNET 0 8 31
CUNET 8 8 15

■Function

CUNET initialization

■Explanation

CUNET sa own end
Sa is the starting block number for securing an area: 0~63
own is the number of area blocks: 1~32
end is the number of blocks shared by the whole: 2~63 (Blocks are expressed as SA0~SA63.)
CUnet has 64 blocks of 8 bytes each. (512 bytes)
CUnet board is initialized by determining the area of those blocks to secure.
After the initialization, CUnet memory area becomes I/O addresses of 2000 or higher.

For example, CUNET 0 1 32 secures SA0 only. Thereby, in that MPC
OUT n SA0_B+m (m = 0~7) allows writing.
ON/OFF can be performed by ON SA0+m (m = 0~63).

In other stations, only IN/SW is enabled, using the same number for reading.
SA0 and SA0_B are reserved constants, prepared for each block.
If CUNET 8 8 32 is issued,
ON/OFF is performed from SA8 with 8*8*8 = 512 bit control.
OUT is performed from SA8_B with 8*8 = 64 byte control.

```
'dsiplay io
 CUNET   0 8 31
  DO
   OUT   IN(SA8_B) 2
   OUT   IN(SA8_B+1) 3
  LOOP

'scan IO
  CUNET   8 8 31
  DO
   FOR   i=0 TO 15
    ON   SA8+i
    WAIT   SW(SA8+i)
    TIME   5
    OFF   SA8+i
    WAIT   SW(SA8+i)==0
   NEXT   i
   OUT   0 SA_B8 : OUT   0 SA_B8+1
  LOOP
```

CU_POST

CUnet Command

■Format

CU_POST [n]|[VOID]

■Usage

CU_POST
CU_POST 28

■Function

CUnet mail server

■Explanation

A command to start the CUnet mail server.
Reads CUnet mails sent automatically, and stores data in P(n) and MBK(n) according to a transfer command. In addition, transfers self data based on a request (POST –n XXX command).

If CU_POST command is executed without any argument, it automatically searches for an empty task and starts the mail server.
The assigned task number is reflected on a global variable CUM_TASK.
In addition, if an argument is given in the range of 1~31, the mail server is started with that task number.
If VOID is given as an argument, or if OR between a specified task number of VOID is given, the execution status is displayed. The mail server is stopped by CTRL_A.

The operation status of the mail server can be monitored through the following global variables.
CUM_ERR (errors) is OR updated, CUM_CNT (mail counter) is incremented, and others are updated at every reception.

CUM_TASK: Task number used by CU_POST server.
CUM_SRC: Address of the sender of a received mail.
CUM_PNT: Category of a received mail, 1: P(n), 2: MBK (n)
CUM_NUM: The value of n in P(n) or MBK(n) of a received mail.
CUM_CNT: Incremented at every received mail.
CUM_ERR: Individual error bits are as follows:
BIT7: MAIL SEND ERROR
BIT6: There is no response to a transfer request.
BIT5: Communication stopped.
BIT4: Sending time out invalid (Normally 0).
BIT3: Sending block invalid (Normally 0).
BIT2: Sending time out occurred.
BIT1: Sending partner not present.
BIT0: Sending partner not standing by.

By using this command, MPC-side point data and MBK data can be rewritten or referred to from a PC. For this function, refer to USB-CUnet documents.

[Reference materials]
Mail transfer unit is as follows:
P(n): 15 long type*4
MBK(n): 120 Word type

Mail packet is 256 bytes in size and has the following constitution to use the first 16 bytes as a system area in partitions as follows:

256buffer= {Num(word)][Ary(byte)][Cmd(byte)][12byte reserved]P(n)~P(n+14]}

Num indicates the first place n of P(n), MBK(n), and IN(n).
Ary specifies either P(n), Mbk(n), or IN(n), where 1 is for P(n), 2 Mbk(n), and 3 IN(n).
However, in the case of IN(n), only 1 byte is dealt with at a time.
If 33 is specified, a bulk transfer occurs, obtaining all real I/O information through one-time communication. (This is enabled only with USB-CUnet.)
In IN(n), if a negative value is specified as n, the memory I/O area is referred to.

Cmd distinguishes between delivery or request, wherein 1 is for request and 2 for delivery.
* Because specifying 33 as Ary and 2 as Cmd instructs a batch I/O setting, its usage requires a caution.

buffer = {
00 64 01 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 03 E8 00 00 03 E8 00 00 03 E8 00 00 03 E8
00 00 27 10 00 00 27 10 00 00 27 10 00 00 03 E8
} Total 256 bytes

If Cmd==2,
CU_POST write data in its own area according to the values of Ary and Num in a mail sent to it.
If Cmd==1,
CU_POST returns data in its own area to the requester according to the values of Ary and Num in an mail sent to it.

[Concerning the example program]
SA2 and SA4 are loaded into each individual MPC-2000 system and executed.
If *xf is executed in the SA2 side, point data of SA2 are copied to SA4. (About 30 seconds for 5000 points)
If *rcv is executed in the SA2 side, point data of SA4 are copied to SA2. (About 30 seconds for 5000 points)
If arguments of CU_POST are omitted, the execution display will disappear.

```
==SA2==
10 CUNET 2 2 32
20 TIME 5
60 CU_POST VOID|25
65 PRINT CUM_TASK
70 END
80 *xf
90 FOR i=1 TO 5000
100 SETP i i i i i
110 NEXT
120 FOR i=1 TO 5000 STEP 15
130 POST 4 P(i)
140 NEXT
150 END
160 *rcv
170 FOR i=1 TO 5000 STEP 15
180 POST -4 P(i)
190 PRINT i
200 NEXT
#
==SA4==
10 CUNET 4 2 32
20 TIME 5
60 CU_POST
8-31
```

```
65 PRINT CUM_TASK
70 END
80 *xf
90 FOR i=1 TO 5000
100 SETP i i i i i
110 NEXT
120 FOR i=1 TO 5000 STEP 15
130 CUM_ERR=0
140 POST 2 P(i)
150 IF CUM_ERR!=0 THEN : PRINT "X_ERR" : END : END_IF
160 NEXT
170 END
180 *rcv
190 FOR i=1 TO 5000 STEP 15
200 POST -2 P(i)
210 PRINT i
220 NEXT
#
```

CW

Pulse generation — Reserved constant

■Format

CW

■Function

Origin return search direction specification
Circular interpolation specification

■Explanation

Applicable boards: MPG-2314
In SHOM the Z-phase search direction for origin search is specified.
In MOVT the rotation direction for circular interpolation is specified.

```
SHOM X_A|Y_A IN0_ON|CW    /* set HOME condition. CW movement until the sensor turns on
MOVT X_A|Y_A P(102) CW    /* continuous interpolation. CW revolution.
RMVC X_A CW               /* infinite pulse generation. CW movement.
```

C_LESS

Pulse generation — Reserved constant

■Format

C_LESS

■Function

Counter comparison

■Explanation

Applicable boards: MPG-2314
Interrupt if counter < COMP+
See also INTA_ON

C_MORE

Pulse generation — Reserved constant

■Format

C_MORE

■Function
Counter comparison

■Explanation
Applicable boards: MPG-2314
Interrupt if counter >= COMP+
See also INTA_ON

DA

AD_DA Command

■Format
DA val [ch]

■Usage
DA 1000 1
DA 2000

■Function
Acquiring MPC-AD12 data

■Explanation
Sets the DA output of MPC-AD12. A value in the range of 0~4095 is specified. A value in the range of 0~4095 is specified. The standard setting is 1 mV / 1 digit.
Set value is reflected on the DA output within 2 msec.
The DA output of MPC-AD12 has 4CHs, and 0~3 can be specified.
If one more MPC-AD12 is added, 4~7 are assigned as the DA output CH numbers.
Although CH is specified as the second argument, if it is omitted, CH0 is set.

DATE

Time control Function

■Format
DATE(0)
DATE(255)
DATE(VOID)

■Usage
```
IF  DATE(0)==&H20070731 THEN
  PRINT "HAPPY BIRTHDAY"
END_IF
```

■Function
Acquiring year, month, and day

■Explanation
The date value is obtained in the hexadecimal representation. If an argument is input, the logical product with the argument is returned.
If VOID is set as the argument, the date value is returned in the decimal format. Setting year, month, and day is performed with SET_RTC command.

```
IF  DATE(0)==&H20070731  THEN
    GOTO  *Thisday
END_IF
    PRX  DATE(0)
```

DATE$

Character string — Function

■Format
DATE$(n)

■Usage
a$=DATE$(1)+" "+TIME$(1)

■Function
Acquiring the date character string.

■Explanation
The date character string is obtained.
DATE$(0)-> 20090529
DATE$(1)-> 5/29/2009
DATE$(2)-> 5.29.2009
DATE$(3)-> 2009-05-29

```
a$=DATE$(1)+" "+TIME$(1)+": CNT="+STR$(i)
```

DEG

Floating point — Function

■Format
DEG(v)

■Usage
FLOAT A=DEG(ATAN(SQR(2))

■Function
Angle unit conversion.

■Explanation
An angle value in radians is converted into the value in degrees.

```
#
 FLOAT A=DEG(ATAN(1))
#pr A
 45
#
```

DELETE

Editing — Command

■Format
DELETE arg1 [arg2]

■Usage
DELETE n
DELETE n m
DEL *Label

■Function
Deleting a specified line
Deleting a specified SECTION.

■Explanation
A line or a range in a program is specified and deleted.

If a label is specified, the area specified with SECTION is deleted.
It is compatible with the merge function of FTMW.

DIM

Operation Command

■Format

DIM label(val)
DIM label(val1,val2)
DIM label1(val) label2(val) label3(val) ...

■Usage

DIM A(100)
DIM array(100,100)
DIM A(100) B(100) C(5)

■Function

Declaration of array elements

■Explanation

Either a one-dimensional or two-dimensional array can be freely declared within the range of 20000 data in total.
Up to 64 labels within 15 characters each can be used. Once the number of arrays exceeds 64, label control cannot be performed afterwards. Therefore, the program should be corrected and reloaded.

DIMCPY

Operation Command

■Format

DIMCPY arg1 arg2 count

■Usage

DIMCPY 1000 U(3) 60
DIMCPY X(1) aho(10) 10
DIMCPY MBK(1) Y(4) 50
DIMCPY X(3) MBK(200) 60
DIMCPY X(3) MBK(200~Lng) 60

■Function

Data transfer among MBK(), X(), Y(), Z(), U(), and defined array elements.

■Explanation

Data transfer is performed among MBK(), X(), Y(), Z(), U(), and defined array elements.
As in DIMCPY MBK(1) MBK(100) 50, even within the same element, if the areas do not overlap, a transfer is allowed. MBK data are treated only as the word type by DIMCPY.
If MBK data should be treated as the long type, ~Lng should be added.
However, in the case of DIMCPY MBK(1) MBK(100~Lng) 50 for example, both the source and destination are treated as the long type.

```
DIM   aho(300)
 DIM   baka(300)
 DIMCPY   1010 aho(3) 25
 FOR   i=1 TO 30
  PRINT   i aho(i)
 NEXT
 S_MBK   100 50 30
```

```
 DIMCPY  12345 MBK(52) 10
PR "MBK"
 FOR  i=50 TO 65
  PRINT  i MBK(i)
 NEXT
*test2
 FOR  i=1 TO 50
aho(i)=i*-1000
 NEXT  i
 DIMCPY  aho(1) baka(100) 30
PR "BAKA()"
 FOR  i=90 TO 135
  PRINT  i baka(i)
 NEXT
NEWP
 DIMCPY  baka(100) X(10) 20
 DIMCPY  baka(100) Y(11) 20
 DIMCPY  baka(100) Z(12) 20
 DIMCPY  baka(100) U(13) 20
PR "X()"
 FOR  i=5 TO 30
  PRINT  i P(i)
 NEXT
 DIMCPY  X(10) MBK(52) 20
PR "X->MBK"
 FOR  i=50 TO 65
  PRINT  i MBK(i)
 NEXT
 DIMCPY  MBK(52) baka(70) 20
PR "X->MBK"
 FOR  i=65 TO 95
  PRINT  i baka(i)
 NEXT
```

DIR

USB Command

■Format

DIR [USBn] [n]

■Usage

DIR
DIR 100
DIR USB1 1000

■Function

Acquisition of the file list of USB memory

■Explanation

When issued as DIR, USB memory directory is displayed.
If a number is given as an argument, there is no display but file names of the following types are copied to the MBK area in the format of 8 characters + .???. (Because each consists of 12 characters, 6 data at a time.)

**.P??
**.C??
**.T??
**.F??
MBK$(nn+4,12) File name 1
MBK$(nn+16,2) File name 2

Then, the acquired data are stored in the following places.

MBK(nn) --> Number of files
MBK(nn+1) --> Total number of files
MBK(nn) --> Total number of directories
MBK(nn) --> USB capacity used (Mbytes) (only for the root directory)

If a USB number is specified, USB memories on MRS-MCOM other than DSW==6 can be referred to.

DSW==6 --> USB (If the number is omitted, MRS-MCOM of DSW=6 is accessed.)
DSW==7 --> USB1
DSW==5 --> USB2

Rem) Direct acquisition of the remaining capacity of USB memory requires count-up processing of the actual empty blocks, which requires a fairly long time for a USB memory of 2G or higher. Because it takes about one minute for 8G, it is not practical.
As a substitute method, total number of bytes of files in the root directory can be detected, and the consumed number of bytes can be calculated from the total capacity. The total capacity can be obtained by functions such as USB(1,USB) after executing DIR command.

DO-LOOP

Control statement Stateme

■Format

DO
LOOP

■Usage

```
DO
ON 0 : TIME 1 : OFF 0
LOOP
```

■Function

Endless iterative execution

■Explanation

Endless iteration from DO to LOOP. To stop the iteration, a BREAK statement is used.

```
IF SW(0)==1 THEN : BREAK : END_IF
LOOP
```

DS_DACL

Pulse generation Command

■Format

DS_DACL [axs]

■Usage

```
DS_DACL
DS_DACL X_A
```

■Function

Deceleration disabling setting

■Explanation

Automatic deceleration is disabled. Used in continuous interpolation.

DS_SEC

Time control — Command

■Format
DS_SEC n

■Usage
DS_SEC 5

■Function
Stopping a one-second counter

■Explanation
Stops a one-second counter SEC(n) specified with n.

DUMP

Maintenance — Command

■Format
DUMP arg1
DUMP str_var

■Usage
DUMP &h200000

■Function
Displaying a memory area (including an I/O area). Character strings are dump displayed.

■Explanation
&h6000 is an I/O area. The following refers to when the register state of MPG-2541 is referred to.

```
#dump &h600100
00600100: 37 00 00 00 37 00 00 00|37 00 00 00 37 00 00 00
00600110: 41 41 41 41 41 41 41 41|41 41 41 41 41 41 41 41
00600120: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600130: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600140: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600150: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600160: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
00600170: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF
```

When a character string is specified as an argument, alphanumeric characters are displayed as they are, and control codes are displayed in hexadecimal.

```
#a$="12345"+chr$(13)+"abcde"
#pr a$
abcde5
#dump a$
12345[0D]abcde
#
```

EMG

Pulse generation — Reserved constant

■Format
EMG

■Function
Error bit setting

■Explanation
Applicable boards: MPG-2314
Emergency stop signal (DMGN) bit

```
IF LMT(X_A,EMG)!=0 THEN          /* confirming reason for stop
```

END

Control statement — Statement

■Format
END

■Usage
END

■Function
End of execution

■Explanation
End of a program. End of a multitasking program.
In the case of the task 0, the input prompt is displayed. In the case of a multitasking program, a task ends.

ENG

Maintenance — Command

■Format
ENG

■Function
Switching to the English mode.

■Explanation
After MPCINIT, the system is in the English mode. Error display becomes in English.

EN_DACL

Pulse generation — Command

■Format
EN_DACL [axs]

■Usage
EN_DACL
EN_DACL X_A

■Function
Deceleration enabling setting

■Explanation
Automatic deceleration is enabled.

EN_SEC

Time control — Command

■Format
EN_SEC n

■Usage
EN_SEC 1

■Function
Enabling the counting of a one-second counter

■Explanation
Setting a one-second counter SEC(n) specified with n into the counting mode.

EOL

Communication — Reserved constant

■Format
EOL

■Function
Receiving terminator setting

■Explanation
The receiving terminator is set.

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 EOL|10 a$            /* until receive LF(&HA=10)
30      PRINT   a$
RUN
 hello                                  /* send 'hello' from the terminal soft
```

ERASE

Editing — Command

■Format
ERASE

■Function
Erasing the FLASH ROM

■Explanation
FLASH ROM is erased. When the system is replaced, ERASE should be performed after MPCINIT.

ERR$

Maintenance — Function

■Format
ERR$(n)

■Usage
pr ERR$(err_)

■Function
Outputting the message corresponding to an error code

■Explanation

If an error interrupt is set using ON_ERROR, an error code and the corresponding statement number are stored in err_ variable. The upper 1 byte is the error code, and the lower 3 bytes are the statement number.
Err$() returns the error character string according to this code in the upper 1 byte.
Therefore, to refer to the error message manually, shift the value upwards by 24 bits as follows:

```
Print err$(1>>24)
```

FEED

Pulse generation — Command

■Format

FEED [axis] n
FEED fx fy fu fz

■Usage

FEED 10
FEED X_A 100

■Function

Speed setting

■Explanatio

A speed is set in 100 grades based on the maximum and minimum speeds set by ACCEL.
The numerical value is set in integer as a percentage of the maximum speed.
FEED X_A 100 sets the X-axis maximum speed.
FEED X_A 0 sets the X-axis minimum speed.
Intermediate numerical values are Fn = MIN + N*((MAX - MIN)/100).

If there is no axis parameter, specified parameters are interpreted as those for X, Y, U, and Z in that order. Therefore, FEED 100 specifies a speed only for the X axis.

FILL

Operation — Command

■Format

FILL array(N) Count [Val Inc]

■Usage

```
FILL   aho(0) 0 0
 FILL   aho(10) 10 -110 2
 FILL   X(6) 20 10000 100
 FILL   MBK(100) 10 500 -2
 FILL   MBK(200~Lng) 100000 10000
```

■Function

Data are continuously set to array elements. Also applicable to point data and MBK data.

■Explanation

This is a command to initialize array elements.
The first argument describes the top element of an array to be initialized.
The second argument is a count number to specify when to initialize.
If 0 is specified, the entire specified array is initialized.

For example,

```
FILL AHO(0) 0
```

sets all the content of AHO(0)~ within the array range to 0.

```
FILL AHO(5) 10
```

In this case, AHO(5)~AHO(14) are set to 0.

If the third argument is entered, a number other than 0 can be set.

```
FILL AHO(5) 10 100
```

This means that AHO(5)~AHO(14) are set to 100.
Furthermore, if the fourth argument is entered, the set value is automatically incremented.

```
FILL SYSDAT(1) 100 501~Lng 2
```

In this example, the value is set to SYSDAT(1)~SYSDAT(100) while it is incremented by 2 as
501~Lng
503~Lng
If a negative value is set, it is set while it is decremented.

Although array elements are battery backed up, the memory position changes due to a program change for example. Therefore, an appropriate initialization is always required.

```
DIM   aho(100)
 FILL   aho(0) 0 0
 FILL   aho(10) 10 -110 2
 FOR   i=8 TO 30
  PRINT   i aho(i)
 NEXT
 FILL   X(6) 20 10000 100
 FILL   MBK(100) 10 500 -2
 FILL   MBK(200~Lng) 100000 10000
```

FLIP_FLOP

IO Command

■Format

FLIP_FLOP o_port IN(port) [pat]

■Usage

FLIP_FLOP -1 IN(24)
FLIP_FLOP -1 IN(24) &H0F

■Function

Set/reset flip-flop

■Explanation

This is a set/reset-type flip-flop. It can be set for an I/O with an 8-bit bank unit.
As the execution content,

```
o_port |= IN(n) xor pat
```

If pat is omitted, pat is set to 0.
Therefore, if an input port becomes active, the corresponding output bit is set and retained. To clear it, issue OFF bit_port. Time required for setting is 1 msec.
If a negative logic is necessary, the corresponding bit of pat should be set to 1.

```
10      SETIO
20      FLIP_FLOP   -1 IN(24)
30      DO
```

```
40       PRX   IN(24) IN(-1)
50       TIME   500
60      LOOP
#RUN

00000000 00000000    ← SW(194)=0,SW(193)=0,SW(192)=0
00000001 00000001    ← SW(194)=0,SW(193)=0,SW(192)=1
00000000 00000001
00000002 00000003    ← SW(194)=0,SW(193)=1,SW(192)=0
00000000 00000003
00000004 00000007    ← SW(194)=1,SW(193)=0,SW(192)=0
00000000 00000007
```

FLOAT

Floating point Command

■Format

FLOAT equation1 equation2 ...

■Usage

FLOAT A=1/3*10000
FLOAT FP(1)=SIN(RAD(30))

■Function

Floating point operation

■Explanation

Operation in FLOAT command becomes a double-precision floating-point operation.
For the substitution of an integer variable in FLOAT command, the operation is performed with double precision, and conversion into an integer is made in substituting.
For the substitution of FP(n) in FLOAT command, the operation is performed with double precision, and the substitution is made in double precision.
Functions such as SIN, COS, TAN, ATAN, ACOS, SQR, RAD, DEG, and VAL in FLOAT command are used as double-precision functions.

```
' Get Pie
 FLOAT   FP(6)=ACOS(SQR(3)/2)*6
 FLOAT   FP(6)=(FP(6)-3)*10
 PRINT   "PIE=3." FP(10000,6)
' Get Napier
a=1
 FLOAT   FP(2)=1
 FOR   i=1 TO 100
a=a*i
  FLOAT   FP(2)=FP(2)+1/a
 NEXT
 FLOAT   FP(2)=(FP(2)-2)*10
 PRINT   "Napier=2." FP(10000,2)
'
 PRINT   "Second order equation X*X+4*X+3"
a=1 :  b=4 :  c=3
 FLOAT   FP(0)=(SQR(b*b-(4*a*c))-b)/2/a
 FLOAT   FP(1)=(SQR(b*b-(4*a*c))*-1-b)/2/a
 PRINT   FP(10000,0) FP(10000,1)
'
```

FLP

Pulse generation Command

■Format

FLP

■Usage

FLP

■Function

Flash ROM read

■Explanation

MPC-1000 dedicated command
Point data P(100)~P(299) are read in from the flash ROM.
Although this area is automatically read in at the time of power-on/reset, FLP command can also be used for reading them.
Writing onto the flash ROM is performed using FSP command.

```
10      FOR   I=100 TO 299
20       SETP   I I I+1 I+2 I+3
30      NEXT   I
40      FSP
50      NEWP
60      PRINT   "P(100)=" P(100) "P(299)=" P(299)
70      FLP
80      PRINT   "P(100)=" P(100) "P(299)=" P(299)
#RUN

 P(100)= 0 0 0 0 P(299)= 0 0 0 0
 P(100)= 100 101 102 103 P(299)= 299 300 301 302
```

FOR-NEXT

Control statement Statement

■Format

FOR var=arg1 TO arg2 [STEP arg3]

■Usage

FOR i=0 TO 15 STEP 2
ON i : TIME 100 :OFF i
NEXT

■Function

Increment or decrement iteration processing

■Explanation

This is a control statement used for an iterative processing for a determined number of times. Although no variable name is required to be entered in NEXT, if there is a variable name entered, its match with the variable name specified by FOR statement is checked.

FORK

Multitasking Command

■Format

FORK n *LABEL

■Usage

```
FORK 1 *LABEL
END
*LABEL
DO
LOOP
```

■Function

Starting a task

■Explanation

In multitasking, a program is executed from *LABEL. Task numbers which can be specified are 1~31. The started task can be ended using END or forcibly ended using QUIT.
If an already-FORKed task is FORKed, a duplicate-starting error occurs.
In this case, it should be restarted after issuing QUIT, or QUIT_FORK should be used.
Touch panel communication and CU_POST command occupy tasks. This command should be used so as not to interfere with these tasks.

FORMAT

Character string Command

■Format

FORMAT Strng

■Usage

```
FORMAT   " DatB=[s00.000]"
FORMAT   "D=S00000"
```

■Function

Defining the expansion format of STR$().

■Explanation

Unless defined by FORMAT command, STR$() expands character strings in the standard integer format.
STR$(1234) ->" 1234" STR$(-1234) ->"-1234"

FORMAT command can define the output format within 15 characters.

FORMAT " DatA=[S 0.000]" --> DatA=[- 8.000]
FORMAT " DatB=[s00.000]" --> DatB=[+02.000]
Numerical values are entered right-adjusted in the spaces or 0s of the character string specified by FORMAT.
S indicates the sign, wherein an upper-case S gives a space for a positive value, and an lower-case s gives a + sign for a positive value.
If neither S nor s is entered, no sign is added.

```
---MPC-XY03 example---
FORMAT “0000-00-00”      /* Setting the character string format
DT$=HEX$(DATE(0))        /* Acquiring the year, month, and day character string
FORMAT “00:00:00”        /* Setting the character string format
TM$=HEX$(TIME(0))        /* Acquiring the hour, minute, and second character string
PR "(1)" DT$ TM$         /* Displaying

* RUN result
(1) 2007-11-07 12:34:00
```

FP

Floating point — Reserved variable

■Format

FP(n)
FP(m,n)

■Usage

FP(0)=1000
STR(FP(100,1)

■Function

Floating-point variable array

■Explanation

There are FP(0)~FP(7), which can be used as double-precision floating-point variable in FLOAT command.
FLOAT FP(1)=1000
In this case, 1000 is stored in FP(1) as the floating-point double-precision type.
In addition, in combination with VAL, data in the exponential expression can be stored.
a$="Mx+9.7042e+002 "
FP(2)=val(a$) stores data as 9.704200E+02 in FP(2).
If FP(n) is used after conversion into integer, it should be described as FP(1,n).
If a magnification is needed, replacing 1 with a value in the range of 1~10000 applies the specified magnification and then integer conversion occurs.
To check the content, the print statement can be used.

```
#a$="Mx+9.7042e+002 My+6.3210e+002"
#fp(2)=val(a$) fp(3)=val(0)
#pr fp(2) fp(3)
 9.704200E+02 6.321000E+02
#
```

FREE

Editing — Command

■Format

FREE

■Function

Display of the remaining capacity

■Explanation

Remaining capacity is displayed by the number of bytes.

```
#free
176500
```

FREEZE

Editing — Command

■Format

FREEZE arg

■Usage

FREEZE 11
FREEZE 2007

■Function

Partial freezing of program and hiding the frozen area

■Explanation

The FREEZE command freezes and hides the part from the top of the program to the line which says FREEZE_END.

In the example program the command FREEZE_END appears on line 110. If “freeze n” is executed in this state (wherein n is a numerical value which becomes a password for releasing the FREEZE), the part from the program top to “FREEZE_END” is frozen. If the value of 1000 or more is given a numerical value as in the program example, the frozen area becomes hidden, allowing the program to be secret.

A secret program cannot be saved even by Program Save by FTM. Even if NEW command is executed, what is erased is the program portion after FREEZE_END. In the example program only lines 120-140 are erased. Even in Program Load by FTM, loading is performed from line 120, retaining the frozen area. Even if an attempt is made to forcibly edit the protected area, it is ignored. In order to release freezing, freeze n should be executed again. The same value as when it was frozen should be given to n. If a different value is given, the message “Already locked” is displayed.

Caution 1:
In Program Load and NEW with FREEZE performed, variable and array areas are not initialized. Therefore, if significantly different programs are reloaded, memory is wasted with variable and array areas left unused. Therefore, if a FREEZE is performed in the middle of program development, the FREEZE should be released at a certain point of settlement, resaving and reloading performed, and the FREEZE performed again.

Caution 2:
If a value of 1000 or smaller is set as the password, the program is frozen in a state in which LIST display is enabled.

```
LIST
10     'INITIALIZE SYSTEM
20       DIM   aho(100)
30       FOR   i=0 TO 99
40      aho(i)=i
50       NEXT   i
60       FOR   i=100 TO 1000
70        SETP   i i i i i
80       NEXT   i
90     '
100      PRINT   "init"
110      FREEZE_END
120    'USER_PROGRAM_START
130      ON   1
140      JUMP   P(1)
#freeze 2007
Locked!!
110
#list
120      ON   1
140      JUMP   P(1)

#freeze 2001
Allready locked!!

#freeze 2007
Unlocked!!
```

```
#list 0
10    'INITIALIZE SYSTEM
20     DIM   aho(100)
30     FOR   i=0 TO 99
40    aho(i)=i
50     NEXT  i
60     FOR   i=100 TO 1000
70     SETP  i i i i i
80     NEXT  i
90    '
100    PRINT  "init"
110    FREEZE_END
120    ON  1
140    JUMP  P(1)
#
```

FREEZE_END

Editing Command

■Format

FREEZE_END

■Usage

100 FREEZE_END

■Function

Specifying an area to be made secret

■Explanation

The FREEZE command makes a part of a program from the top to a line wherein FREEZE_END is stated to be secret
FREEZE_END is a dummy command for specifying that place.
FREEZE_END itself is ignored just like a comment line during command execution.

FSP

Pulse generation Command

■Format

FSP

■Usage

FSP

■Function

Writing to flash ROM

■Explanation

An MPC-1000 dedicated command. Point data P(100)~P(299) are written to the flash ROM. Although this area is written to the flash ROM together with the program after compiling, the FSP command is used for forcibly writing it during a program.

MPC-1000 has no battery backup function. P(100)~P(299) is used as an area where data (teaching points, backup variables, etc.) is retained even after the power supply is cut off. Reading from the flash ROM is performed by the FLP command.

```
10     FOR  I=100 TO 299
20     SETP  I I I+1 I+2 I+3
30     NEXT  I
40     FSP
```

```
50      NEWP
60      PRINT   "P(100)=" P(100) "P(299)=" P(299)
70      FLP
80      PRINT   "P(100)=" P(100) "P(299)=" P(299)
#RUN

 P(100)= 0 0 0 0 P(299)= 0 0 0 0
 P(100)= 100 101 102 103 P(299)= 299 300 301 302
```

GETDG

Floating point — Command

■Format

GETDG n m deg

■Usage

GETDG 1 3 deg

■Function

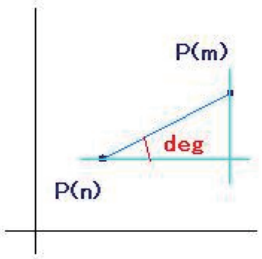
Angle calculation

■Explanation

The angle of the vector P(m)-P(n) relative to the X-axis is calculated.
The actual calculation is performed as follows:

deg = ATAN((Y(m)-Y(n))/(X(m)-X(n)))

The angle deg is returned to the variable as a value multiplied by 10000.
In the example program
X(2)-X(1)=> 17320508
Y(2)-Y(1)=1000000
Therefore, ATAN(100000/17320508) is obtained, which is 30 degrees, and because the result is multiplied by 10000, 300000 is obtained.




```
#setp 1 10000 20000  0 0
#setp 2 17330508  10020000   0 0
#getdg 1 2 a
#pr a
 300000
```

GET_AD

AD_DA — Command

■Format

GET_AD CH ARRAY() Cnt [ms]

■Usage

GET_AD 0 X(1090) 360 4
GET_AD 1 Z(1090) 100

■Function

Continuous acquisition of AD data

■Explanation

Real-time data acquisition from MPC-AD12, utilizing a 1 msec timer of the CPU side.
In the command line, the acquisition channel, data storage array, number of times of acquisition, and acquisition interval in the msec unit are specified.
This command ends without waiting for the data acquisition to be complete.
Checking data acquisition completion is performed using the updating of the final data

of the timer or data as a marker.
The following example acquires AD data at 4 msec intervals while rotating a stepping motor with MPC-1000. Point data, MBK array, or DIM declaration array can be used as the array.
In this example, data updating check is performed by updating the value of X(1449).

```
80     X(1449)=0
90      PGB   "P" 1800
95     SYSCLK=0
100     WAIT   SYSCLK=>360
110     GET_AD   0 X(1090) 360 4
120     WAIT   X(1449)!=0
125     PRINT   SYSCLK
130     WAIT   SW(RDY_PGB)==0
#
```

GET_VAL

Character string Command

■Format

GET_VAL strg_val arry(n) [FPn]

■Usage

GET_VAL a$ a(1)
GET_VAL a$ a(1) 100

■Function

Extracting numerical values from a character string and continuously substituting them into an array

■Explanation

Numerical values contained in a character string are batch substituted into an array.
Arguments are limited to a character string variable (XX$) and an array variable arry(n). (Substitution into mbk, x(n) is not allowed.)
If the third argument is omitted, a decimal point (dot) is also regarded as a delimiter.
If a numerical value such as 10, 100, and 1000 is set as the third argument, each number string containing a decimal point is substituted into an array as a numerical value after being multiplied by the specified magnification.

```
10      DIM   a(10)
20     a$="1111.12   -2222.13   3333.1   4444.5 345m-9730"
25      PRINT   a$
30      FILL   a(1) 99 777
50      GET_VAL   a$ a(1)
60      PRA   a(1)
65      PRINT   "FP"
70      FILL   a(0) 99 777
80      GET_VAL   a$ a(1) 100
90      PRA   a(1)
#run

 1111.12   -2222.13   3333.1   4444.5 345m-9730
a(1)=1111
a(2)=12
a(3)=-2222
a(4)=13
a(5)=3333
a(6)=1
a(7)=4444
a(8)=5
```

```
a(9)=345
 FP
a(1)=111112
a(2)=-222213
a(3)=333310
a(4)=444450
a(5)=345
a(6)=-9730
a(7)=777
a(8)=777
a(9)=777
#
```

GOSUB,GOSUB_NE

Control statement Statement

■Format

GOSUB *Label [arg1,arg2..]

■Usage

GOSUB *Label
GOSUB *Label arg1 arg2 ..

■Function

Subroutine call

■Explanation

Because subroutines use stack memory, a program called by GOSUB must also always return by RETURN.
If a program is made wherein the control returns to the original program from a subroutine using GOTO and the like instead of RETURN, a stack overflow error or underflow occurs, and the program halts.

GOSUB command of BL/1 allows adding arguments handed over to a subroutine after the destination label. These argument values are extracted by the _VAR command on the subroutine side. If a task-local variable is used as the _VAR argument, the subroutine becomes one of general-purpose.

If GOSUB_NE is used instead of GOSUB, even if the destination label does not exist, no compilation error occurs, and it is executable as is.
GOSUB_NE without a destination label is ignored at the time of execution.

```
10 GOSUB *CAL 300 400
20 _VAR RES
30 PR RES
40 END
50 *CAL
60 _VAR V_ W_
70 RETURN SQR(SQ(W_)+SQ(V_))
RUN

*
Compiling
---------------------
500
#
```

GOTO

Control statement — Statement

■Format

GOTO *Label

■Usage

```
IF A==1 THEN : GOTO *ERR : END_IF
GOTO *LOOP
```

■Function

Unconditional branching

■Explanation

Control is moved to a specified label.

HEX

Character string — Function

■Format

HEX(str)
HEX(arg)

■Usage

```
b$="ABC123 &H1234FJ  &HBCDEF1 "
  PRX   HEX(b$)
 SERCH   b$ "&H"
  PRX   HEX(5)
```

■Function

Reading out hexadecimal character strings.

■Explanation

Reading out hexagonal character strings in a character string.
Ordinarily, reading out is performed by specifying a numerical value (number of digits) after searching for the location using SERCH and the like.
After SERCH is used, a numerical value is entered without using a character string.
If the character consists of only hexadecimals such as in the case of a$=”ABC123”, HEX(a$) can also be used for reading it out.

```
LIST
10     b$="ABC123 &H1234FJ  &HBCDEF1 "
20      PRX   HEX(b$)
30      SERCH   b$ "ABC"
40      PRX   HEX(0)
50      SERCH   b$ "&H"
60      PRX   HEX(5)
#run

00ABC123
00000123
0001234F
#
```

HEX$

Character string — Function

■Format

HEX$(arg)

■Usage

a$=HEX$(100)
t$=HEX$(DATE(0))

■Function

Generating a character string of a numerical value in hexadecimal
Year, month, and date can be obtained by t$=HEX$(DATE(0)).

■Explanation

If there is no FORMAT specification, an 8-character hexadecimal expression is adopted.
If there is a FORMAT specification, it is followed. “S” and “s” of FORMAT are invalid.

```
10          FORMAT  ""
20          PRINT   HEX$(&H00ABCDEF)
40          FORMAT  "&H0000"
50          PRINT   HEX$(100000000)
#run
00ABCDEF
&HE100
#
```

HIN

IO — Function

■Format

HIN(arg)

■Usage

A=HIN(24)
A=HIN(24~Wrd)

■Function

If a type specification such as an 8-bit parallel input ~Lng or ~Wrd is entered, it can be read out as long or word parallel.

■Explanation

HIN() is a function having the same function as IN(). When read-out is specified to an input port of the I/O area, it reads out only once although ordinary IN() reads out twice for verification. (Read-out in an area which is not an actual input such as memory I/O is performed only once by both HIN and IN.)

HOME[MPG-2314]

Pulse generation — Command

■Format

HOME X Y U Z
HOME axs V

■Usage

HOME 10000 10000 1000 1000
HOME NEG_L NEG_L NEG_L NEG_L
HOME X_A -1000
HOME X_A|Y_A -1000

■Function

Origin-return command

■Explanation

The HOME sequence is executed while giving the stopping conditions set by SHOM.
Because HOME is effectively time-out enabled, it should be set appropriately.
If stopped by a time-out, the current value is not cleared. If X(0) and others are not 0, sn error stop is indicated.

Arguments of the HOME command are amounts of movement for a near-origin search. If near origin is detected during a movement, a deceleration stop is performed.
If IN1_ON/IN1_OFF is set by SHOM, after detecting anear origin, Z-phase search is performed at the lowest speed set by ACCEL.
The direction of movement at this time is determined by CW or CCW given by SHOM. The default is CCW. Program 1 is an example of an origin return at 10 seconds. HPT(XIN0), HPT(YIN0), and HPT(ZIN0) are monitors for the IN0 of each axis.
If located inside the near origin, a retreat movement is performed first, and then an origin return is performed.

When a large amount of movement for origin-return near origin is desired, POS_L and NEG_L should be used.
These are positive and negative maximum numbers of 3-type length.
As the amount of movement for near-origin detection, an axis-specified constant may also be used as in HOME X_A -1000.

```
----program1---
10          PG  1
20   ACCEL  40000
30   ACCEL  Z_A 20000
40  SHOM  X_A|Z_A|Y_A IN0_ON|IN1_ON|CW
50  IF  HPT(XIN0)==0 : RMVS  X_A 20000 : END_IF
60  IF  HPT(YIN0)==0 : RMVS  Y_A 20000 : END_IF
70   IF  HPT(ZIN0)==0 : RMVS  Z_A -20000 : END_IF
80   WAIT  RR(ALL_A)==0
85   TMOUT  10000
90   HOME  -100000 -100000 0 100000

---MPC-XY03 example---
*HOME1
 ACCEL X_A|Y_A 10000 100 100

 IF HPT(XIN0) != THEN
  RMVS X_A 10000
 END_IF
 IF HPT(YIN0) != THEN
  RMVS Y_A 10000
 END_IF
 WAIT RR(ALL_A)==0
 TIME 100

 SHOM X_A|Y_A IN0_ON
 HOME -100000 -100000 0 0
 WAIT RR(ALL_A)==0

 TIME 100
 RMVL 2000 2000 0 0
 WAIT RR(ALL_A)==0
 STPS 0 0 VOID VOID
 PRINT "XY HOME"
 TIME 100
 RETURN
```

HOME[MPG-2541]

Pulse generation Command

■Format

HOME X Y U Z
HOME axs V

■Usage

HOME -10000 -10000 0 -10000
HOME X_A -1000
HOME X_A|Y_A -1000

■Function

Origin return

■Explanation

The origin return of MPG-2541 is determined by the function of the built-in IC.
If the SD signal becomes inactive, a low speed (lowest speed determined by ACCEL command) is set, and if the ORG signal becomes active, it is stopped.
Even if the SD signal becomes active once, if it becomes negative later before ORG detection, the speed return to the maximum speed again, which requires caution.

What is given by the HOME command is the moving distance for an origin-return action.
HOME -1000 -1000 0 -10000
In the above example, the X, Y, and Z axes move by -10000 pulses in the CCW direction.
An axis specification constant may also be used as in HOME X_A -1000.

The speed at this time becomes either the maximum speed of ACCEL or a speed determined by FEED.
If ORG becomes active during that time, the coordinate value of the corresponding axis is set to 0, and when the actions of all the axes are complete, the HOME command is through.
After it is through, if the coordinate value is 0, it indicates that ORG was detected. *1)
Logics of SD and ORG are set by SHOM.

*1)
In HOME command, if ORG input is active after pulse generation stopped, the coordinate value is set to 0 assuming that origin return took place.
However, there occur cases wherein ORG is not active due to the influence of a slight overrun or servo accumulated error pulse after ORG became active once and the motor stopped as in a mechanical switch and the Z-phase of an encoder.
In such cases, current position remains without being set to 0 after origin return.
In a clear case wherein the coordinate value is not 0 even after origin return and stop occurred, current position is set to 0 by a command such as CLRPOS.

HOUT

Pulse generation Command

■Format

HOUT arg

■Usage

HOUT 1

■Function

Controlling the output port of the MPG board

■Explanation

This is a 4-bit batch setup of the MPG output port.
Caution) H_ON/H_OFF/HOUT cannot be used during the PG operation with MPC-2541, because turning the bit on/off and the PG command use the same register, which alters the each of their operating states.

HPT

Pulse generation Function

■Format

HPT(arg1)

■Usage

```
prx HPT(0)
WAIT HPT(XIN0)==1
IF   HPT(XIN0)==0 : RMVS   X_A 20000 : END_IF
```

■Function

Reading out the origin-return input port of the PG board.

■Explanation

[MPG-2314]
IN0~IN3, INPOS, and ALM input are read out through HPT(0) parallel.
In the parallel mode, there is a 32-bit parallel output for each 8 bit unit axis.

1) HPT(0)={U}{Z}{Y}{X}
{8bit}=ALM(bit7),INP(bit6),IN3(bit3)IN1(bit2,bit1)IN0(bit0)

IN1(bit2, bit1) indicates that IN1 input is internally shorted with IN1 and IN2 in MPG-2314. Therefore, when IN1 is turned on, IN2 is also turned on, in which case “hpt(0) -> 00000006” is realized.

2) The HPT(XIN0) specified port is read out.
Origin-return input:XIN0,XIN1,XIN2,XIN3~UIN0,UIN1,UIN2,UIN3
ALM input: XALM~UALM
INPOS input: XINP~UINP

If each specified port read out by HPT is ON, the value becomes 1.

[MPG-2541]
X_SD, ORG_X ~ Z_SD, Z_ORG are read in parallel.
If the corresponding bit is 1, it indicates that it is active.

HSW

IO Function

■Format

HSW(arg)

■Usage

```
A=HSW(192)
IF HSW(192)&HSW(200)&HSW(208) THEN
```

■Function

Bit reading of input port

■Explanation

SW() performs reading twice when an actual input port is specified as the input port. On the other hand, HSW() only perform reading once.
IF HSW(192)&HSW(200)&HSW(208) THEN
When taking logic of multiple SWs in this way, reading occurs twice with SW(), which slows down the execution speed. If HSW() is used as in the above example, logical operations are performed at a high speed.
This distinction does not exist in monitoring the output port, memory I/O, and MBK I/O area.

H_OFF

Pulse generation — Command

■Format

H_OFF arg

■Usage

H_OFF 2

■Function

Turning off the output port of MPG board

■Explanation

Turning off the bits is performed in the same manner as with an ordinary output port.
Caution) H_ON/H_OFF/HOUT cannot be used during PG operation with MPC-2541.
This is because bit turning on/off and PG command use the same register, which alters the each of their operation states.

H_ON

Pulse generation — Command

■Format

H_ON n

■Usage

H_ON 1

■Function

Turning on the output port of MPG board

■Explanation

Turning off the bits id performed in the same manner as with an ordinary output port.
Caution) H_ON/H_OFF/HOUT cannot be used during PG operation with MPC-2541.
This is because turning the bits on/off and the PG command use the same register, which alters each of the operating states.

IF-THEN-ELSE-END_IF

Control statement — Statement

■Format

IF arg THEN

■Usage

IF SW(0)==1 THEN : ON 0 : END_IF
IF SW(0)==1 THEN : ON 0 : ELSE : ON 1 : END_IF

■Function
Conditional branching

■Explanation
If arg is not 0, what follows THEN is executed. In the case of 0, either what follow ELSE is executed, or jumping to the place after END_IF.

IN

IO Function

■Format
IN(arg1)

■Usage
```
IF IN(0)==&HAA THEN
WAIT IN(1)==&H05
A=IN(0~Lng)
```

■Function
Parallel import at the input port (8 bits)

■Explanation
MPC-2000 input port is 24, 25.
The first MIO-1616 input port is 26, 27. Specifying a negative number indicates memory I/O. If ~Lng, ~Wrd, or ~Int is given as the address value, they indicate long read, 2-byte integer read, or signed 2-bye read, respectively.
For the touch panel mbk area, 70000 or larger should be specified.
ab takes a value range of 00~99.

IN(7ab00): Byte read
IN(7ab00~Ub): Hi-byte read
IN(7ab00~Wrd): Word read
IN(7ab00~Lng): Long read

```
---MPC-XY03 example---
DSW=IN(24)/16            /* GET DSW value and Shift down 4bits
```

IN0_OFF

Pulse generation Reserved constant

■Format
IN0_OFF

■Usage
SHOM X_A IN0_OFF

■Function
Stop input setup

■Explanation
Applicable boards: MPG-2314
XIN0~ZIN0 are enabled on the OFF signal. see also IN0_ON

```
SHOM X_A IN0_OFF
STOP X_A IN0_OFF
```

IN0_ON

Pulse generation Reserved constant

■Format

IN0_ON

■Usage

SHOM X_A|Y_A IN0_ON

■Function

Stop input setup

■Explanation

Applicable boards: MPG-2314

XIN0~ZIN0 are enabled on the ON signal.

```
100 SHOM X_A|Y_A IN0_ON              /* set HOME condition.
110 HOME -100000 -100000 0 0
120 WAIT RR(ALL_A)==0
-----
100 STOP X_A IN0_ON                  /* set stop condition. if XIN0 turn on then stop.
110 MOVL 5000 0 0 0
120 WAIT RR(X_A)=0                   /* wait for stop
130 IF HPT(XIN0)==1 THEN             /* confirming reason for stop
140   PRINT "IN0 stop"
150 ELSE
160   PRINT "normal stop"
170 END_IF
```

IN1_OFF

Pulse generation Reserved constant

■Format

IN1_OFF

■Usage

SHOM X_A IN1_OFF

■Function

Stop input setup

■Explanation

Applicable boards: MPG-2314

XIN1~ZIN1 are enabled on the OFF signal.

```
SHOM X_A IN1_OFF
STOP X_A IN1_OFF
```

IN1_ON

Pulse generation Reserved constant

■Format

IN1_ON

■Usage

SHOM X_A IN1_ON

■Function

Stop input setup

■Explanation
Applicable boards: MPG-2314
XIN1~ZIN1 are enabled on the ON signal.

```
SHOM X_A IN1_ON
STOP X_A IN1_ON
```

IN2_OFF

Pulse generation — Reserved constant

■Format
IN2_OFF

■Usage
SHOM X_A IN2_OFF

■Function
Stop input setup

■Explanation
Applicable boards: MPG-2314
XIN2~ZIN2 are enabled on the OFF signal.

```
SHOM X_A IN2_OFF
STOP X_A IN2_OFF
```

IN2_ON

Pulse generation — Reserved constant

■Format
IN2_ON

■Usage
SHOM X_A IN2_ON

■Function
Stop input setup

■Explanation
Applicable boards: MPG-2314
XIN2~ZIN2 are enabled on the ON signal.
See also IN0_ON

```
SHOM X_A IN2_ON
STOP X_A IN2_ON
```

IN3_OFF

Pulse generation — Reserved constant

■Format
IN3_OFF

■Usage
SHOM X_A IN3_OFF

■Function
Stop input setup

■Explanation

Applicable boards: MPG-2314
XIN3~ZIN3 are enabled on the OFF signal.

```
SHOM X_A IN3_OFF
STOP X_A IN3_OFF
```

IN3_ON

Pulse generation — Reserved constant

■Format

IN3_ON

■Usage

SHOM X_A IN3_ON

■Function

Stop input setup

■Explanation

Applicable boards: MPG-2314
XIN3~ZIN3 are enabled on the ON signal.
See also IN0_ON

```
SHOM X_A IN3_ON
STOP X_A IN3_ON
```

INC

Operation — Command

■Format

INC var [Val]

■Usage

INC A
INC A -10

■Function

Incrementing/decrementing a variable (Multitasking)

■Explanation

If incrementing/decrementing is performed using shared variables in multitasking, read and set may miss proper timing, preventing a task from correctly performing incrementing/decrementing variables.
Because the INC command completes read & set within a task, such a problem does not occur. If there is no argument, a simple increment of +1 is performed. If an argument is added, its value is added to the variable.

INCHK

Pulse generation — Command

■Format

INCHK

■Function

Monitoring the input status of PG board

■Explanation

If INCHK is entered, the status of the input port is displayed. Typing 'q' stops it.

```
inchk
 MPG-2314
 X=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
 Y=+LMT:off-LMT:off ALM:off INP:off IN0:on  IN1:off
 U=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
 Z=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off

MPG-2541
 X= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
 Y= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
 U= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
 Z= -EL:1 +EL:1 ORG:1 -SD:1 +SD:1 OTS:0
#
```

INPUT

Communication Command

■Format

INPUT [CH] [EOL|x] [CHR_C|x] [TMOUT|x] a$

■Usage

INPUT a$

■Function

Character string input

■Explanation

INPUT is a serial input command, which is INPUT# fixed to CH0 (Program port).
Because its usage is the same as INPUT#, the section of INPUT# should be referred to.

INPUT#

Communication Command

■Format

INPUT# [CH] [EOL|x] [CHR_C|x] [TMOUT|x] a$
INPUT# [CH] CLR_BUF

■Ulsage

INPUT# a$
INPUT# CH a$
INPUT# 5 EOL|10 c$
INPUT# 3 CHR_C|54 a$
INPUT# 3 TMOUT|10 a$
INPUT# 20 a$
INPUT# 2 CLR_BUF
INPUT# 5 COMPOWAY rcv$

■Function

Importing a character string through RS-232C port.
Reading one line of a USB memory file opened by OPEN command.

■Explanation

INPUT# imports a character string through a serial port. If CH number is omitted, CH1 is set. Although the terminator uses CR as default, it can be changed using the EOL|xx option. xx is the ASCII code.

In importing a character count, CHR_C|xx option should be used. xx is a numerical value of 255 or smaller.
If the count is specified, the terminator is ignored.
If timeout is needed, the TMOUT|xx option should be used. A time limit is entered in xx in the unit of second.
If the case of TMOUT|10, processing is cut off if reading cannot be completed within 10 seconds.
Whether a timeout occurred or not can be checked by referring to the rse_ variable. If rse_ is 1, it indicates that a timeout occurred.
If CLR_BUF is given as an argument, all character strings in the buffer are read and abandoned.
If COMPOWAY is given as an optional parameter, receiving is performed in the OMRON COMPOWAY format.
TMOUT option can also be used together. If a check-sum error occurs, rse_ becomes 4.
Characters received in the COMPOWAY format can be basic-decomposed by COMPOWAY command.

Numerical conversion I performed by character string processing commands such as VAL function and GET_VAL.

```
INPUT#  a$
```

a=VAL(a$) : b=VAL(0) See VAL function.
Port numbers 20~22 correspond to USB memory files on MRS-MCOM.
DSW==6 ->20 (If omitted, MRS-MCOM of DSW=6 is accessed.)
DSW==7 -> 21
DSW==5 -> 22

```
--Serial Communication--
CNFG#  3 "38400b8pns1NONE"
CNFG#  4 "38400b8pns1NONE"
CNFG#  5 "38400b8pns1NONE"
 a$="123456789012345678abcdefghijklmnopqrstuvwxyz$%&()01234"
'    GOTO  *RS422
  DO
   PRINT#  3 a$ "¥r"
   INPUT#  4 EOL|13 b$
   PRINT#  4 b$ "¥r"
   INPUT#  5 EOL|10 c$
   PRINT#  5 c$
   INPUT#  3 CHR_C|54 a$
   PRINT  a$
  LOOP
 *RS422
  DO
   PRINT#  4 a$ "¥r"
   INPUT#  5 b$
   PRINT  b$
   PRINT#  5 b$ "¥r"
   INPUT#  4 a$
  LOOP

--USB Memory Access--
     OPEN  USB "AUTO.F2K"
      DO
       INPUT#  USB a$
       IF  LOF(USB)==0 THEN  : BREAK  : END_IF
       PRINT  a$
      LOOP
     CLOSE  USB
```

INP_OFF

Pulse generation — Reserved constant

■Format
INP_OFF

■Usage
INSET X_A INP_OFF

■Function
'In position' setup

■Explanation
Applicable boards: MPG-2314
'In position' enabled on the OFF signal
If either INP_ON or INP_OFF is set, it is enabled. Otherwise, invalid.

```
INSET X_A INP_OFF              /* X-axis 'INPOSITION' enabled on signal 'OFF'
```

INP_ON

Pulse generation — Reserved constant

■Format
INP_ON

■Usage
INSET X_A INP_ON

■Function
'In position' setup

■Explanation
Applicable boards: MPG-2314
'In position`' enabled on the ON signal
If either INP_ON or INP_OFF is set, it is enabled. Otherwise, invalid.

```
INSET X_A INP_ON               /* X-axis 'INPOSITION' enabled on signal 'ON'
```

INSET

Pulse generation — Command

■Format
INSET [axs] Settings

■Usage
INSET PHASE4
INSET ALL_A ALM_ON|INP_OFF

■Function
MPG-2314 input setup command

■Explantion
Functions of the input port are set. The relationship between the functions and reserved constants is as follows.

INPOS = INP_ON,INP_OFF,INP_NO
ALARM = ALM_ON,ALM_OFF,ALM_NO
LMT = LMT_ON,LMT_OFF

Soft limit = SLMT_ON,SLMT_OFF
Encoder = UP_DWN,PAHSE1,PAHSE2,PAHSE4
PLS = MD_2PLS,MD_DPLS

INSET executed last becomes effective.
Settings other than those given with parameters are reset.

Example)

```
INSET X_A ALM_ON|INP_ON
```

Input setup for the X-axis. Alarm is enabled, wherein the ON state is set as alarm. In addition, 'In position' is enabled, which is enabled on the ON signal.

```
INSET ALL_A ALM_ON|INP_OFF
```

Input setting for all axes. Alarm is enabled on the ON signal, and INPOS is enabled on the OFF signal.

```
INSET   PHASE4
```

Encoder input is set to quadruple multiplication.

```
INSET ALL_A VOID
```

All settings are cleared. RANGE setting is also cleared.

INSPEC

Maintenance Command

■Format
INSPEC

■Function
Self-test

■Explanation
Currently only the write/read test of RAM is supported

```
#inspec
INSPECTION
1:Test Memory
PASSED
#
```

Int

Touch panel Reserved constant

■Format
Int

■Usage
IN(-1~Int)

■Function
Specifying the word type (signed)

■Explanation
Signed 16-bit read of S_MBK, MBK(), IN, or OUT is specified.

```
10      S_MBK   &H00008FFF 20~Wrd  /* WORD write
20      PRINT   MBK(20~Wrd)        /* unsigned WORD read
30      PRINT   MBK(20~Int)        /* signed WORD read
```

```
40      OUT   -1 -1~Wrd              /* WORD write
50      PRINT   IN(-1~Wrd)           /* unsigned WORD read
60      PRINT   IN(-1~Int)           /* signed WORD read
RUN

 36863                               /* unsigned
-28673                               /* signed
 65535                               /* unsigned
-1                                   /* signed
```

INTA_ON,INTB_ON

Pulse generation Command

■Format

INTA_ON portn (PG,axis)
INTB_ON portn (PG,axis)

■Usage

INTA_ON 16 (0,X_A)
INTB_OFF 17 (0,U_A)

■Function

Turning a port ON or OFF by an interrupt of MPG-2314

■Explanation

INTA_ON turns a port ON by a counter comparison detection interrupt of MPG-2314. In order to activate the interrupt, the following command setup is necessary.

- Comparison counter setup: INSET axis CMP_PLS (or CMP_CNT)
  CMP_PLS = Pulse position, CMP_CNT = Encoder/counter position
- Enabling an interrupt: STOP axis C_MORE (or C_LESS)
  C_MORE Pls >= COMP+ , C_LESS PLs < COMP+
- Setting a comparison value COMP+
  RANGE axis VAL1 dummy

If the above are set and the counter value exceeds VAL1 (in the case of C_MORE), an interrupt occurs, and the port specified by INTA_ON is turned ON. (In the case of OFF, INTA_OFF is used.)
After an interrupt occurred, the interrupt can be released by reading out RR3(axis) function.
However, before releasing it, the comparison value COMP+ needs to be changed to outside the condition by a RANGE setting.

```
Initialization
INSET axis CMP_CNT          /* Comparison counter selection
STOP  axis C_MORE           /* Setting an interrupt comparison condition
RANGE axis VAL1 dummy
a=RR3(axis)                 /* Clearing the interrupt in advance

As the order of execution,
RANGE axis VAL1 summy       /* Changing the condition in advance
a=RR3(axis)                 /* Clearing the interrupt
SWAP
INTA_ON port                /* Setting an interrupt port
WAIT SW(port)               /* Detecting the occurrence of an interrupt
```

* axis for RR3 is effective only for a single-axis specification such as X_A and Y_A.

Releasing an interrupt is performed by executing INTA_ON VOID or INTA_OFF VOID.
The interrupt may be INTB_ON or INTB_OFF, and up to two PG interrupts along with INTA_ are supported.
The example program generates an interrupt at every 500-pulse movement and outputs a timing trigger to the exterior. INTA_ON, _OFF, INTB_ON, and _OFF can precisely output position timing unlike a timing wait by software.

```
INTA_ON   VOID
 PG   0
 PG   0 1
 PG   0 3
 ACCEL   5000
 CLRPOS
 CLRPOS   -1
 INSET   X_A CMP_PLS
DET_P=500
 STOP   X_A C_MORE
 RANGE   X_A DET_P 0



 PG   0
 FORK   1 *MPG
 FORK   3 *MPG2
 END
*MPG2
 DO
  INTA_ON   0 (0,X_A)
  WAIT   SW(0)
  TIME   1
  OFF   0
  INC   DET_P 500
  RANGE   X_A DET_P DET_P
 A_=RR3(X_A)
  TIME   5
 LOOP
*MPG
 RMVC   X_A 1
 END
```

JMPZ

Pulse generation — Command

■Format

JMPZ Pnt

■Usage

JMPZ P(n)

■Function

JUMP without Z descending

■Explanation

This is a partial execution of the gate-motion command JUMP, which does not perform Z-axis descending.
JMPZ is a compound command wherein multiple actions are combined. Therefore, if PAUSE, STOP, or CONT is executed, an unexpected horizontal movement may occur.
In order to prevent this, JMPZ command has a built-in command to re-execute in case PAUSE is executed. To enable this function, the target task should be paused by PAUSE(STP_D,n).

For a task paused in this manner, the JMPZ command is re-executed by the CONT command.
After executing the CONT command, PAUSE(STP_D,n) should not be executed again for 0.1 seconds.

JPN

Maintenance Command

■Format
JPN

■Function
Switching to the Japanese mode

■Explanation
Switching to the Japanese mode. Errors will be displayed in Japanese. After MPCINIT the English mode is selected.

JUMP

Pulse generation Command

■Format
JUMP P(arg)
JUMP PL(pln;plm)
JUMP argx,argy,argu,argz

■Usage
JUMP P(1)
JUMP PL(0;5)
JUMP X Y U Z

■Function
Gate motion

■Explanation

JUMP P(n)	Gate-motion movement to Point n
JUMP PL(n;m)	Gate-motion movement to the m-th point of Pallet n
JUMP X Y U Z	Gate-motion movement to Coordinate point

JUMP is a compound command wherein multiple actions are combined. Therefore, if PAUSE, STOP, or CONT is executed, an unexpected horizontal movement or a descent may occur.
In order to prevent this, the JUMP command has a built-in command for re-execution if PAUSE is executed.
To enable this function, the target task should be paused by PAUSE(STP_D,n).
To the task paused in this manner, JUMP command is re-executed by the CONT command.
After executing the CONT command, PAUSE(STP_D,n) should not be executed again for 0.1 seconds.

The example program signifies
Line 1: Gate-motion movement of the Z-position of the 2nd position of Pallet 1 upwards by 500 pulses
Line 2: Waiting for the completion of pulse output
Line 3: Turning the mechanical chuck OFF, namely placing a work piece away.

```
JUMP PL(1;PT) AD_P(Z_A,500)
WAIT RR(ALL_A)==0
OFF 14
```

LABELS

Maintenance Command

■Format

LABELS

■Function

Label check

■Explanation

Checking the duplicate definitions of labels. If a duplicate label is found, the following is displayed.

```
The two same labels
12810   *bb
13400   *bb
```

LEN

Character string Function

■Format

LEN(string)

■Usage

print LEN(a$)
a=LEN(a$)

■Function

Counting the number of characters of a character string

■Explanation

The number of characters of a given character string.

LIFE_TIME

Time management Command

■Format

LIFE_TIME [val]

■Usage

LIFE_TIME 100

■Function

Time-slice time control

■Explanation

While the default time slice of MPC-2000 is 3 msec, this time may better be adjusted for some applications.
Using LIFT_TIME command, this time can be set between 500 µsec and 5 msec in the unit of 10 µsec. “LIFT_TIME 250” sets it to 2.5 msec.
Because the value is restored to the default by power-on reset, if change is necessary, it should be stated in the program.
In addition, if there is no argument, the current time-slice time is returned.

LIMZ

Pulse generation Command

■Format

LIMZ arg1 [arg2]

■Usage

LIMZ -5000
LIMZ -5000 100

■Function

Increasing the speed of JUMP (gate motion)

■Explanation

JUMP moves the XYU axes after a Z-axis ascent.
By default, it moves (ascends) until the Z value becomes 0, the device speed slows down.
This ascent ceiling can be determined by LIMZ.
In the case of LIMZ -1000, it ascends up to the position of -1000.
As to arg2, when arg2 msec has passed after the ascent of the Z-axis started, XYU movement is started.
Thereby, the movement of the starting-point side of the gate motion becomes arch-shaped.

LIST

Editing Command

■Format

LIST arg1 [arg2]

■Usage

LIST 10 3
LIST *AH0
LIST

■Function

Displaying a program list

■Explanation

The first argument is a statement number to display. The second argument is the number of lines to display.
When executed alone, LIST displays from the top. If LIST is executed again without any argument, the continuation is displayed.

LMT

Pulse generation Function

■Format

LMT(n)

■Usage

```
IF LMT(X_A,LMTp)!=0 THEN
 RMVS X_A -10000
 END_IF
```

■Function

Reading an error input

■Explanation

[MPG-2314]
LMT(0) allows referring to the error statuses of all XYZU axes.

byte= { EMG,ALM,LMTn,LMTp,SLMTn,SLMTp}
UbyteZbyteYbyteXbyte

Another method of reference is to give axis and bit parameters as in LMT(X_A,ALM).
In this case, axis specification(X_A) and referent bit determination(ALM) are performed.

[MPG-2541]
+X_LMT, -X_LMT ~ +Z_LMT, -Z_LMT are parallel-read.
If the corresponding bit is 1, the signal is active.

LMTn

Pulse generation Reserved constant

■Format

LMTn

■Usage

LMT(X_A,LMTn)

■Function

Error bit specification

■Explanation

Applicable boards: MPG-2314
Hard limit -bit

```
IF LMT(X_A,LMTn)!=0 THEN          /* confirming reason for stop
```

LMTp

Pulse generation Reserved constant

■Format

LMTp

■Usage

LMT(X_A,LMTp)

■Function

Error bit specification

■Explanation

Applicable boards: MPG-2314
Hard limit +bit

```
IF LMT(X_A,LMTp)!=0 THEN          /* confirming reason for stop
```

LMT_OFF

Pulse generation Reserved constant

■Format

LMT_OFF

■Usage

INSET ALL_A LMT_OFF

■Function
Setting the limit input

■Explanation
Applicable boards: MPG-2314
X-LMT~ZLMT enabled on the OFF signal.
Immediate stop when a limit is detected. Input cannot be disabled.

```
INSET ALL_A LMT_OFF              /* 'LIMIT' enabled on signal 'OFF'
```

LMT_ON

Pulse generation — Reserved constant

■Format
LMT_ON

■Usage
INSET ALL_A LMT_ON

■Function
Setting the limit input

■Explanation
Applicable boards: MPG-2314
X-LMT~ZLMT enabled on the ON signal.
Immediate stop when a limit is detected. Input cannot be disabled.

```
INSET ALL_A LMT_ON               /* 'LIMIT' enabled on signal 'ON'
```

Lng

Touch panel — Reserved constant

■Format
Lng

■Usage
MBK(20~Lng)

■Function
Long type (two words) specification

■Explanation
Specifying reading out the values of S_MBK, MBK(), IN, and OUT in the 32-bit long-type.

```
10      S_MBK   &H12345678 20~Lng        /* LONG write MBK data area 20,21
20      PRX   MBK(20~Lng)                /* LONG read MBK data area 20,21
30      PRX   MBK(21) MBK(20)            /* WORD read
40      OUT   &H87654321 -1~Lng          /* LONG write memory I/O area -1~-4
50      PRX   IN(-1~Lng)                 /* LONG read memory I/O area -1~-4
60      PRX   IN(-4) IN(-3) IN(-2) IN(-1)  /* BYTE read
RUN
12345678                                 /* LONG read
00001234 00005678                        /* WORD read
87654321                                 /* LONG read
00000087 00000065 00000043 00000021      /* BYTE read
```

LOF

Communication Function

■Format
LOF(ch)

■Usage
IF LOF(1)>10 THEN : input# 1 a$: END_IF

■Function
Returning the number of character strings in the buffer.

■Explanation
The returns the number of characters stored in the buffer of each RS-232C port. The argument CH corresponds to 0~11.
In addition, LOF(20) indicates the presence/absence of remaining characters in the USB memory, wherein 1 indicates the presence, and 0 indicates that the EOF has been reached.

LOG

Maintenance Command

■Format
LOG [arg]

■Usage
LOG
LOG 0
LOG 1

■Function
Log display

■Explanation
LOG is a record of characters output to the program board during execution.
LOG buffer is cleared by either NEW or LOG 0.
Program port output is displayed by LOG command while stopped or during execution.
Because 20 lines are displayed at a time, LOG command should be repeated to continue.
In order to display from the top, LOG 1 should be executed. To initialize LOG, LOG 0 should be executed.
When LOG command is executed, LOG stops. To resume, LOG 3 should be executed.
After monitoring the state of a device in operation by executing LOG, LOG 3 must be always executed to continue LOG.

LONG_PRG

Touch panel Reserved constant

■Format
LONG_PRG

■Usage
S_MBK LONG_PRG

■Function
Conversion of a program number into the long type

■Explanation

Conversion of the program numbers for the touch panel into the long type.
Ordinarily, the system sets the statement number of a program in execution in a word area MBK(7868)~MBK(7899).
When the program has become larger to have the number of 65535 or larger, word write is performed using this command. In this case, the program statement number is written in long integers in MBK(7836)~MBK(7899).

```
10 MEWNET 38400 1          /* RS-232C CH1 -> MBK-RS 38400bps
20 S_MBK LONG_PRG          /* upper MBK(7836) -> long numeric
```

MBK

Touch panel Function

■Format

MBK(arg)

■Usage

a=MBK(n)
a=MBK(n~Lng)
MBK(n)=a
b=MBK(n~Int)

■Function

Referring to and setting touch panel data

■Explanation

MBKMBK array is an array which is memory-shared when connected to a touch panel.
MBK(n) corresponds to DTn.

a=MBK(n) → Extracting touch panel data in the word type.

b=MBK(n~Int) → Extracting touch panel data in the signed word type. For example, if the value is &HFFF0, -16 is obtained.

a=MBK(n~Lng) → Extracting touch panel data in the long type. High word is filled with MBK(n+1).

MBK(n) = Formula → Substituting a word-type value into touch panel data

MBK(n~Lng) = Formula → Substituting a long-type value into touch panel data

MBK(n) has the following reserved areas.
1) Statement number
MBK(7868)~MBK(7899) is the program statement number in execution. It is in the word type. If the statement number exceeds 65535,

```
S_MBK LONG_PRG
```

should be executed. Afterwards
the statement number is stored in the long type in MBK(7836)~MBK(7899).

2) Version number
Stored in MBK(8053) is the version number.
If the firmware version is 1.12_60,

```
pr MBK(8053) -> 11260
```

3) MBK(7900)~MBK(7999) is treated as the R area of the touch panel side.
Banks 0~99 of the R area correspond to this area.

MBK$

Touch panel | Function

■Format
MBK$(adr,val)

■Usage
A$=MBK$(100,6)

■Function
Reading an MBK array as a character string

■Explanation
This is a function paired with S_MBK a$ adr c. It reads out a character string on an MBK array.

MBK_CMD

Touch panel | Reserved variable

■Format
MBK_CMD

■Usage
PRX MBK_CMD

■Function
Communication error character

■Explanation
This is a command which could not be processed in the MEWNET communication.
If 4142 is output by PRX MBK_CMD, it signifies AB.

MBK_ERR

Touch panel | Reserved variable

■Format
MBK_ERR

■Usage
PR MBK_ERR

■Function
Communication error counter

■Explanation
This is a variable holding the number of MEWNET communication errors.

MD_2PLS

Pulse generation | Reserved constant

■Format
MD_2PLS

■Usage
INSET ALL_A MD_2PLS

■Function
Setting the pulse output mode

■Explanation

Applicable boards: MPG-2314
Setting the pulse generator to the two-pulse mode (CW/CCW)

```
INSET ALL_A MD_2PLS          /* Set the pulse generator to '2 PULSE' mode
```

MD_DPLS

Pulse generation — Reserved constant

■Format

MD_DPLS

■Usage

INSET ALL_A MD_DPLS

■Function

Setting the pulse output mode

■Explanation

Applicable boards: MPG-2314
Setting the pulse generator to the one-pulse mode (direction instructed)

```
INSET ALL_A MD_DPLS          /* Set the pulse generator to 'DIR/PULSE' mode
```

MEWNET

Touch panel — Command

■Format

MEWNET arg1 [COMn] [mode]
MEWNET [COMn]

■Usage

MEWNET 9600
MEWNET 9600
MEWNET 38400
MEWNET 38400 5
MEWNET 9600 1 B7O
MEWNET 0

■Function

Setting the MEWNET protocol for the touch panel

■Explanation

A task is assigned to MEWNET (Panasonic FP Series computer link) communication, and data sharing is made between MBK() array and the touch panel.
(Sharing by WD, WC, RD, and RC protocols)
Which task is assigned is determined by the communication channel number according to the following rule.

Assigned task = 32 – ch number

Therefore, if the first user channel CH1 is used as MEWNET, Task 31 is occupied as the communication task.
The first CH number of MRS-MCOM is 3. In this case, 32 – 3 = 29 is assigned to the communication task.

Baud rate can be selected from 9600, 19200, or 38400.

The second argument is the RS channel number, wherein 1~5 can be specified. (Up to the first MRS-MCOM board)
The third argument is for setting the communication format. Although it is ordinarily omitted with 8-bit no-parity communication as default, when parity is needed or the number of bits should be changed, it is added.

B7O: 7-bit odd-parity
B7E: 7-bit even-parity
B8O: 8-bit odd-parity
B8E: 8-bit even-parity

Because MEWNET command includes the initialization of communication protocol, it should not be used in combination with CNFG# command.
Examples:
Connecting with MPC-2100 CH2, the occupied task is 30.

```
MEWNET 38400 2
```

Connecting with MRS-MCOM #1 CH5, the occupied task is 27 (for both RS-232 and RS-422, RS-485 is not supported).

```
MEWNET 38400 5
```

Connecting with MRC-2000 CH1, the occupied task is 31, 7-bit odd-parity. (Mitsubishi compact touch panel)

```
MEWNET 9600 1 B7O
```

B7E is for the case of 7-bit even-parity.

Once MEWNET command is executed, automatic start is enabled afterwards. Therefore, in order to change the CH, MEWNET [COMn] should be executed to delete the registration.
* In the case of MEWNET 1, MEWNET at COM1 is stopped.

In MPCINIT, all registrations are initialized.

---- Digital GP2400 setup example ----

Initial setup > I/O setup > Communication setup
Transmission speed 38400 (to be matched with MEWNET command)
Data length:8
Stop bit: 1
Parity bit: None
Control method: X-control
Communication method: RS-232C

Initial setup > I/O setup > Communication monitoring time setup
Communication timeout time (1-127) [10] sec → Shortened to [1] sec for example.

Date correspondence is as follows.
DT0~ : MBK(0)~

RD0~ : ON/OFF/SW/IN/OUT 7YYXX~ (Overlapped with DT7900~DT7999)
YY = Bank number (0~99), XX = Bit number (0~16)

Although DT area supports ordinary numbers, RD area takes values of 70000 or larger, wherein the lower two digits correspond to the bit number, and the middle two digits the bank number.
In the case of IN/OUT, the bit number of XX is set to 0.

MKY

CUnet Function

■Format

MKY(val)

■Usage

A=MKY(0)
PRX MKY(1)

■Function

Reading the control register of CUnet IC MKY

■Explanation

It read out the value of each register of MKY40 which is a CUnet chip.

MKY(0) SCR
MKY(1) BCR_SA (Upper two bits indicate Baud.)
MKY(2) BCR_OA (Upper two bits indicate LFS, CP)
MKY(3) CHIP_CD :“MKY4” is returned as a value. prx MKY(3) -> 4D4B5934
MKY(4) MES(Mail Error Status)
MKY(5) SSR(System Status Register)
MKY(6) MFR(Member Flag Register 0-31
MKY(7) MFR(Member Flag Register 32-63
MKY(8) MCR(Member Care Counter) Reading and clearing
MKY(9) LCR(Link Care Counter) Reading and clearing

The value of MKY(1) excluding the upper 2 bits becomes the values of DSW1 and DSW2 immediately after powering up.
Thereby, the start address can be set by the DSW value of CUnet.
MKY(3) allows checking the presence/absence of a board.
MKY(6) and MKY(7) allow checking the presence/absence of MKY (power ON/OFF) on the network.
If the values of MKY(8) and MKY(9) frequently increase (a nonzero value of 1 or larger is found at every reading), the communication quality is degraded due to external causes.

MON

Maintenance Command

■Format

MON [arg]

■Usage

MON
MON 1
MON 2

■Function

Checking/monitoring of the execution status

■Explanation

```
#mon
    *0  [-1]   *1  [980]   *2  [1240]  *3  [1320]
    *4  [1360]
```

Below is the description of monitoring the status of a task in real time with Task 0 in the stopped state (command-receivable).
When QUIT is issued from another task, it is quit with the statement number remaining,

and if it is stopped by END, the statement number becomes -1.

```
#mon 1
mon 1
    *1  RUNNING  [850]   *2   SLEEPING [1240]  *3   SLEEPING [1320]  *4   QUIT    [1360]
    *5  QUIT    [-1]
```

If there is a task consuming time among the tasks, a ! mark is displayed after the task number.

```
#mon
    *0  [-1]    *1  [820]   *2   [1240]   *3   [1320]
    *4  [1360]  *5! [1880]
#
```

If “MON 2” is executed, only the LOG data are written without displaying them.

MOVL

Pulse generation Command

■Format

MOVL P(n) [option]
MOVL PL(n;m) [option]
MOVL arg1,arg2,arg3,arg4 [option]

■Usage

MOVL P(1)
MOVL P(1) AD_P(X_A,100)
MOVL X Y U VOID
MOVL PL(1;1)
MOVL P(3) VOID_U

■Function

Linearly-interpolated move to a specified point or specified coordinate (Linearly-interpolated pulse generation by coordinate control)

■Explanation

MOVL is interpolated pulse generation with coordinate control.
As the arguments, direct coordinate values, point data, palette points, and the like can be given.
However, because the interpolation can only support up to three axes, a 4-axis movement causes an error.
As an option, interpolation functions and axis-specification constants such as AD_P, X_A|Y_A, and VOID_U can be added.
If X_A|Y_A allows interpolating a specified axis, and VOID_U ignoring a specified axis (not generating pulse) for example.

MOVS

Pulse generation Command

■Format

MOVS [axis] n
MOVS arg1 [arg2,arg3,arg4]

■Usage

MOVS x y u z
MOVS X_A n
MOVS x VOID u z

■Function

Pulse generation with coordinate control

■Explanation

Absolute-positioning pulse generation with no interpolation accompanying. MPC-2000 performs coordinate control.
MOVS takes the difference of the current position and a specified value and generates pulses by the amount of difference.
In the case of a short axis, the axis-specification constant can be used for specifying it.
In addition, if VOID is specified as an argument, that axis would not operate.

MOVT

Pulse generation — Command

■Format

MOVT axs Point [CCW|CW|0]

■Usage

MOVT X_A|Y_A P(101)
MOVT X_A|Y_A P(102) CCW
MOVT X_A|Y_A P(i) M(i)

■Function

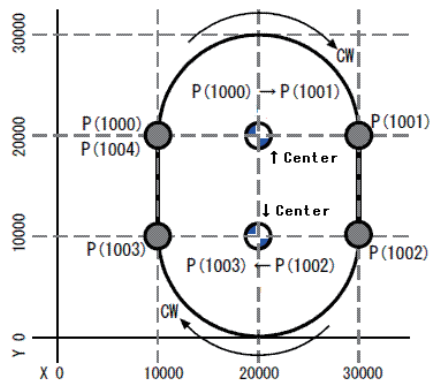
Continuous interpolated movement based on coordinate values

■Explanation

This is a continuous interpolation using point data. Although data are input in absolute values, movement is performed by converting them into relative coordinates from the starting point (P(100) in this case).
If CCW or CW is entered as the third argument, a circular interpolation is performed.
If there is no third argument or 0 is given, a linear interpolation is performed.

The example program is a general-purpose program using point data.
Setting coordinate data into P(1000)~ and instruction data into P(2000)~ allows various kinds of continuous movements.

```
PG  0
 ACCEL  8000
 CLRPOS
 GOSUB  *SET_POINT
axis=X_A|Y_A
 MOVL  axis P(1000)
 WAIT  RR(axis)==0
 FEED  axis Y(2000)
 DS_DACL
 FOR  pnt=1001 TO X(2000)
  MOVT  axis P(pnt) X(1000+pnt)
 NEXT  pnt
 EN_DACL
 WAIT  RR(axis)==0
 END
*SET_POINT
 SETP  1000 10000 20000 0 0 : 'Start
 SETP  1001 30000 20000 20000 20000 : 'Cir target and Center
 SETP  1002 30000 10000 0 0
 SETP  1003 10000 10000 20000 10000
 SETP  1004 10000 20000 0 0
 SETP  2000 1004 50 0 0 :    'End of Point and FEED value
 SETP  2001 CW 0 0 0 :      'P(1000) to P(1001) CW
 SETP  2002 0 0 0 0 :       'P(1001) to P(1002) linear
```



```
SETP   2003 CW 0 0 0 :       'P(1002) to P(1003) CW
SETP   2004 0 0 0 0 :        'P(1003) to P(1004) linear
RETURN
```

MPCINIT

Editing Command

■Format

MPCINIT

■Function

Setting MPC into the initial condition.

■Explanation

Clearing the program area.
Setting variable, point data, and array areas to 0. Setting all I/O areas to OFF into a clear state.

MPG

Pulse generation Command

■Format

MPG arg [taskn]
MPG

■Usage

MPG 1
MPG 1 4

■Function

Assigning an MPG board

■Explanation

Determining which MPG board to use. Specification can be made separately for individual tasks.
If taskn is not specified, MPG is specified with the executed task.
If specified, the specified task specifies its MPG. The result of specification can be listed by MPG.
Although there is PG as a similar command, it does not judge the absence/presence of the specified PG.
If a non-existing PG number is specified in MPG command, an error is displayed.
MPG 0~9 correspond to MPG-2314, supporting linear and circular interpolations.
MPG 10~17 correspond to MPG-2541, supporting a simple position determination without any interpolation function.
Although PG is a command having the same function, it does not give an error even if a non-existing PG number is specified.

M_SW

IO Function

■Format

M_SW([n,]n)

■Usage

M_SW(192)
M_SW(10,193)

■Function
SW function with filters

■Explanation
This is a SW() function used for input which tends to generate chattering signals such as mechanical switches and reflection sensors.
In M_SW(n), the n port is read three times at every 1 msec, and only when the same value is read all three times, the port value is returned.
In M_SW(t,n), t specifies the number of times of reading, and if the same value is read for t times (t msec), the port value is returned.
Therefore, when the input varies in a pulse form with 1 msec period, M_SW() function becomes suspended. As the port number n, only the on-board I/O can be specified, and memory I/O and the like cannot be used.

NEG_L

Pulse generation — Reserved constant

■Format
NEG_L

■Usage
HOME NEG_L NEG_L NEG_L NEG_L

■Function
A negative large number

■Explanation
A negative large number
If a large amount of movement is desired for the origin-return near-origin, POS_L or NEG_L should be used.
These are the positive and negative maximum numbers of 3-byte length.

```
#prx POS_L
007FFFF0
#prx NEG_L
FF80000F
```

```
HOME NEG_L NEG_L NEG_L NEG_L
```

NEW

Editing — Command

■Format
NEW

■Function
Erasing a program

■Explanation
Erasing a program and erasing variables except reserved variables.

NEWP

Pulse generation — Command

■Format
NEWP

■Function
Point data initialization

■Explanation
All point data are initialized to 0.

NOT

Operation Function

■Format
NOT(arg)

■Usage
A=NOT(1)

■Function
Bit inversion of an argument

■Explanation
Bit NOT in the long type

```
#prx NOT(&Hf)
FFFFFFF0
```

NO_PHASE

Pulse generation Reserved constant

■Format
NO_PHASE

■Usage
INSET NO_PHASE

■Function
Counter input setup

■Explanation
Applicable boards: MPG-2314
Disabling

```
INSET NO_PHASE                /* Counter disable
```

OFF

IO Command

■Format
OFF arg1 [arg2 arg3 arg4 ...]

■Usage
OFF 1 2 3 //MIO-1616etc
OFF A A+1
OFF -1 //Memory I/O area

■Function
Turning off output ports

■Explanation
Output ports are turned off. Open collector output goes into the floating state.

A negative value (-1~) indicates the bit turning off the memory I/O area.
A value of 2000 or larger (2000~) indicates the bit turning off of the CUnet area.
A value of 70000 or larger (7aabb) indicates the bit turning off of the MBK I/O area (RD area).
aa is a bank number (0~99), and bb is a bit number of 0~15.

ON

IO Command

■Format

ON arg1 [arg2 arg3 arg4 ...]

■Usage

```
ON 1 2 3   //MIO-1616 etc
ON A A+1
ON -1   //Memory I/O
ON 2000 // CUnet Area
ON  70000 // MBK I/O area
```

■Function

Turning on output ports

■Explanation

Output ports are turned off. Open collector output goes into the synchronizing state.
A negative value (-1~) indicates the bit turning on the memory I/O area.
A value of 2000 or larger (2000~) indicates the bit turning on the CUnet area.
A value of 70000 or larger (7aabb) indicates the bit turning on the MBK I/O area (RD area).
aa is a bank number (0~99), and bb is a bit number of 0~15.

ON

Multitasking Function

■Format

ON(n)

■Usage

```
WAIT   ON(-1)==0
 PRINT   "WATSHI HA " TASKn
OFF   -1
'
IF ON(-1)==0 THEN
 PRINT   "WATSHI HA " TASKn
OFF   -1
END_IF
```

■Function

Reading and setting the memory I/O (Semaphore)

■Explanation

ON(n) turns on the memory I/O or output port in the same manner as the ON command.
As a function value, it returns the value of the specified port immediately before turning it on.
If n is assigned to the memory I/O by ON(n), Port n becomes the semaphore.
In the same manner, n can be used as an ordinary output port number.

```
OFF   -1
 FOR   i=1 TO 10
  FORK   i *test
```

```
 NEXT
 END
*test
 WAIT  ON(-1)==0
 PRINT  "WATSHI HA " TASKn
 OFF  -1
 TIME  SYSCLK%1000
 GOTO  *test
```

ON_ERROR

Control statement Command

■Format

ON_ERROR arg

■Usage

ON_ERROR *USB
ON_ERROR VOID

■Function

Defining the destination of an error processing jump

■Explanation

When an error has occurred in a command or function for example, the program in execution ordinarily stops.
The ON_ERROR command does not stop the program but specifies an error processing program and has program execution continued. The method is as follows:
ON_ERROR *label defines the jump destination. To release the definition, ON_ERROR VOID should be executed.
Because ON_ERROR transfers the control to error processing at the occurrence of an error, the error processing program needs to appropriately classify processing according to the error code.

Ordinarily, most errors occurring in execution are fatal, making another attempt impossible. In this case, the error location and content are outside to be utilized for debugging.

However, when accessing USB memory, a runtime error may occur depending on the status of connected devices. In this case, the program can be continued by normalizing the devices through appropriate processes such as RST_USB.

For returning from an error processing routine by ON_ERROR to the normal processing program, GOTO or RESUME is used. In the case of GOTO, if the error location is inside a subroutine, caution should be exercised in specifying the location to which to return.
Because RESUME returns control to the location of error occurrence, RESUME should be stated to retry a command, and RESUME_NEXT to move on to the next processing without retrying.

The error code is reflected on a task variable, err_. A processing fit with the err_ value is described.
The upper 1 byte of err_ is the error code, and the lower 3 bytes is the program number.

```
err_>>24              --> Error code
ERR$(err_)            --> Error message
err_&&HFFFFFF         --> Program number
```

The error number is displayed at the end of the error message.
Listed below are error codes related to USB memory.

This USB is in use.
No USB memory is found.
No MRS-MCOM is found.
USB memory has an abnormal operation.

```
FORK   1 *case1
 TIME   500
 FORK   2 *case2
 END
*case1
 ON_ERROR  *err1
 DO
  S_MBK   1 9000
  PRINT   10
  PRINT   20
 LOOP
*err1
 PRINT   "case1=" TASKn  err_&&H00FFFFFF ERR$(err_)  err_>>24
 TIME   1000
 RESUME   _NEXT
 END
*case2
 ON_ERROR  *err2
 DO
  OUT   1 -10000
  PRINT   1
  PRINT   2
 LOOP
*err2
 PRINT   "case2=" TASKn  err_&&H00FFFFFF ERR$(err_)  err_>>24
 TIME   1000
 RESUME
 END
```

ON_USB,OFF_USB

USB Command

■Format

ON_USB
OFF_USB

■Function

Enabling/disabling the MPC-1000 USB port

■Explanation

The MPC-1000 USB port is made available by starting a USB file access system with Task 29. ON_USB performs necessary initialization and starts up Task 29.
Conversely, OFF_USB stops the port and releases Task 29.

OPEN

USB Command

■Format

OPEN [COM] str

■Usage

OPEN A$
OPEN USB1 "TXT.TXT"

■Function

Opening a USB file

■Explanation

OPEN file name allows reading text data by INPUT# USB A$ afterwards.
If there are no more characters to read, LOF(USB) becomes 0.
In addition, INPUT# starts returning empty character strings.
USB# are assigned as follows.

DSW==6 ->USB (If omitted, MRS-MCOM of DSW=6 is accessed.)
DSW==7 -> USB1
DSW==5 -> USB2

```
OPEN "TXT.TXT"
DO
IF LOF(USB)==0 THEN : END : END_IF
INPUT#  A$
PRINT A$
LOOP
```

OUT

IO Command

■Format

OUT val port
OUT val port1,port2..
OUT val port1 TO port2

■Usage

OUT &H55 2
OUT &HAA -1
OUT 0 1,2,5
OUT 0 -1 TO -10

■Function

Setting output ports and memory I/O to 8-bit parallel.

■Explanation

This is a command to set output ports as 1 byte, specifying them as banks.
The MPC-2000 I/O ports 0~7 become Bank 0, and 8~15 Bank 1.
First MIO-1616 is assigned Banks 2 and 3 in the same manner. If a bank is assigned a negative value, it becomes memory I/O.
~Lng given as the address value indicates long write, and ~Wrd or ~Int word indicates a 2-byte write.
There is no distinction between Wrd and Int in writing. For the mbk area of the touch panel a value of 70000 or larger should be specified.

ab assumes a value in the range of 00~99.
OUT data 7ab00) Byte write
OUT data 7ab00~Ub) Hi-byte write
OUT data 7ab00~Wrd Word write
OUT data 7ab00~Lng Long write

In addition, when setting multiple output ports to the same value, the output port numbers should be described in a row.
For setting a continuous range of ports to the same value, a description such as port1 TO port2 should be given.

P$

Character string Function

■Format

P$(val)

■Usage

a$=P$(100)

■Function

Conversion of point data into a character string

■Explanation

The point data area can be used as if it is a character string array using the “SETP n strngs” command.
This is a function for extracting data stored as a character string P$().

```
FORMAT  "Test s    "
 FOR   i=1 TO 10
a$="setp"+STR$(i-5)
 SETP   i a$
 NEXT
 FOR   i=1 TO 10
  PRINT   P$(i)
 NEXT
```

PALLET

Pulse generation Command

■Format

PALLET h P(i) P(j) P(k) [P(l)] m n * 0<=h <= 63 m,n ~32767
PALLET h P(i) P(j) m

■Usage

PALLET 1 P(1) P(2) P(3) P(4) 4 3
PALLET 1 P(1) P(2) P(3) 4 3
PALLET 1 P(11) P(12) 3

■Function

Defining a pallet

■Explanation

```
PALLET  1 P(1) P(2) P(3) P(4) 4 3
```

A4×3 pallet generated by points P(1)~P(4). If four points are specified, a distorted quadrangle is also possible.

```
PALLET  1 P(1) P(2) P(3)  4 3
```

A4×3 pallet generated by points P(1)~P(3). Specification of three points is regarded as a rectangle.

Points on a pallet are numbered as 1~, and the order in the example if the figure is P(1) → 1, 2, 3, 4, 5 (p2). If the specified number in the PL function is given as a positive number, 6 becomes the point next to P(1) toward the P(3) side. If the specified number is given as a negative number, 6 becomes the point above P(2), forming a zigzag order.

[Concerning one-row pallets]
In the case of a one-row pallet P(1) → P(2), the description becomes as follows: After 12_48,

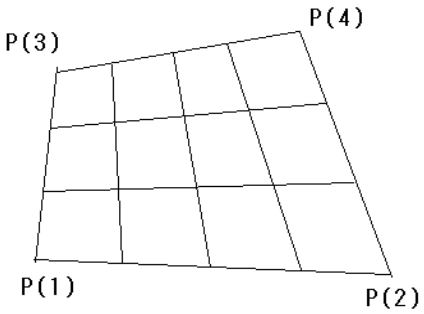
```
PALLET  1 P(1) P(2)   m
```

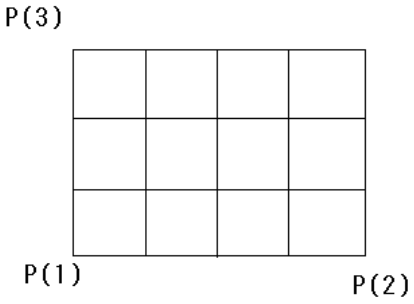
Before 12_48, the following description should be made:

```
PALLET  1 P(1) P(2) P(2)  m 2
```

The reason for specifying 2 is to avoid dividing by 0.

```
'4points teaching
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
SETP 3 0 20000 0 -5000
SETP 4 20000 20000 0 -5000
PALLET 1 P(1) P(2) P(3) P(4) 4 3
FOR   I_=1 TO 12
  JUMP   PL(1;I_)
  WAIT   RR(ALL_A)==0
NEXT
or
FOR   I_=-1 TO -12 STEP -1
  JUMP   PL(1;I_)
  WAIT   RR(ALL_A)==0
NEXT
----------
'3points teaching
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
SETP 3 0 20000 0 -5000
PALLET 1 P(1) P(2) P(3) 4 3
FOR   I_=1 TO 12
  JUMP   PL(1;I_)
  WAIT   RR(ALL_A)==0
NEXT
----------
'linear tray
SETP 1 0 0 0 -5000
SETP 2 20000 0 0 -5000
PALLET 1 P(1) P(2) 4
FOR I_=1 TO 4
 JUMP PL(1;I_)
 WAIT RR(ALL_A)==0
NEXT
```





PAUSE

Multitasking Command

■Format

PAUSE arg
PAUSE (STP_D,taskn)

■Usage

PAUSE n

■Function

Pausing a task
If the argument is (STP_D, taskn), the task is paused and the stop command is executed.

■Explanation

A task in execution is put into the SLEEP state (infinite timer stop), and is resumed by CONT.
If the argument is specified as in (STP_D, task), the target task is stopped, STOP STP_D is executed, and the re-execution flag is raised for the JUMP and JMPZ commands.
In this case, the task resumed by the CONT command is re-executed if the JUMP or JMPZ command is in execution.

PEEK

Character string — Function

■Format

PEEK(Str$+n)

■Usage

A=PEEK(b$+1)
B=PEEK(b$+LEN(b$)-1)

■Function

Obtaining a character string code

■Explanation

PEEK allows obtaining the code of a specified position of a specified character string. This is useful for computing the checksum for communication for example.

```
10     a$="123456789A"
20      PRX   PEEK(a$)
30      PRX   PEEK(a$+LEN(a$)-1)
#run

00000031
00000041
#
```

PG

Pulse generation — Command

■Format

PG arg1 [taskn]
PG

■Usage

PG 0
PG 1 2

■Function

Specifying an MPG board
MPG 0~9 for MPG-2314, high function, up to circular interpolation enabled
MPG 10~17 for MPG-2541, low price, no interpolation

■Explanation

Although the PG command has the same function as the MPG command, it does not check the presence of an MPG board (see the MPG command). Therefore, even if an uninstalled MPG is specified, no error is displayed.

PGA,PGB

Pulse generation — Command

■Format

PGA str$ val

■Usage

PGA "G" 1000
PGB "V"
pr V_PGB

■Function

PG control commands for MPC-1000

■Explanation

MPC-1000 has two simple PG functions, PGA and PGB.
Commands to control individual PGs are PGA and PGB, whose format and functions are as follows:
Although below are an examples of PGA, PGB can also be used in the same format.

```
PGA "G" pps  /* PPS-specified pulse generation(20~9000pps)
PGA "S" pps  /* Setting the pulse rate(20~9000pps)
PGA "W" duty/* PWM(40~970/1000)
PGA "P" pls  /* Pulse number specified pulse generation(+/-8000000)
PGA "A" pps  /* Acceleration/deceleration table generation(500~12000pps)
PGA "F" f    /* Speed selection(10~0)
PGA "R" pls  /* Acceleration/deceleration table generation, relative(+/-8000000)
PGA "M" pos /*Acceleration/deceleration table generation, coordinates(+/-8000000)
PGA "H" pos  /* Setting the current position(+/-8000000)
PGA "D" n    /* Pulse mode (0: Default 2PLS, 1: Direction instructed)
PGA "C"      /* Obtaining the current position
PGA "V"      /* Obtaining the version
```

The returned value after issuing the “PGA C” or “PGA V” is substituted for V_PGA.(V_PGB for PGB) Pulse generation can be stopped by OFF PGA or OFF PGB, respectively.

PGE

Pulse generation Function

■Format

```
PGE(0)
PGE(axs,val)
```

■Usage

```
IF PGE(X_A,ALM) THEN : GOTO *EMG_X_A : END_IF
IF PGE(0) THEN : GOTO *EMG : END_IF
IF PGE(X_A,CLR_ER|ALM) THEN : GOTO *EMG_X_A : END_IF
IF PGE(CLR_ER) THEN : GOTO *EMG : END_IF
```

■Function

Referring to the cause of a stop of MPG-2314

■Explanation

Pulse generation of MPG-2314 can be stopped by inputting EMG, ALM, LMT, or IN0~IN1.
After stopping, even if the cause input is released, the cause of the stop can be found by the PGE() function.
There are two ways of giving the arguments.
PGE(0) allows referring to the stop cause flags of all four axes.
PGE(0) = {Uaxs|Zaxs|Yaxs|Xaxs}, wherein all four bytes have meaning.
The bit construction of each byte is 8 bits of {EMG,ALM,LMTn,LMTp|IN3,IN2,IN1,IN0}.
In addition, if input specification is done with axes specification and constants as follows, individual checks can be performed.

```
PGE(X_A,LMTp)          Testing LMTp
PGE(X_A,(IN1|LMTp))    Testing LMTp or IN1 together
```

If only CLR_ER is set as the argument, the error status can be cleared at the same time as obtaining all the statuses.

In addition, in the case of axes specification, if CLR_ER is ORed to the bit condition, only the corresponding axes are read & cleared.

```
LIST
10     'XXXX=CLR_ER
20     'XXXX=(X_A,CLR_ER|IN0)
50      PG  1
60      ACCEL  4000
70      STOP  ALL_A IN0_ON
80      OFF  0 1
90      CLRPOS
100      MOVS  1000000 1000000 100000 100000
110      TIME  1000
120      ON  0 1
130      WAIT  RR(X_A)==0
140      PRX  PGE(0)
150      PRX  PGE(XXXX)
160      PRX  PGE(0)
#run

00000101
00000001
00000100
#prx XXXX
0001F100
#
```

PG_TASK0

Pulse generation — Reserved variable

■Format

PG_TASK0

■Usage

print PG_TASK0

■Function

Obtaining the PG number

■Explanation

This is a variable which returns the PG number assigned to Task 0. If the PG does not exist, the value becomes -1.

```
/* USE MPG-2314 #0 and MPG-2541 #0
10      PG  0
20      PRINT  PG_TASK0
30      PG  10
40      PRINT  PG_TASK0
50      PG  1
60      PRINT  PG_TASK0
#run

 0
 10
-1
```

PHASE1

Pulse generation | Reserved constant

■Format

PHASE1

■Usage

INSET PHASE1

■Function

Setting the counter input

■Explanation

Applicable boards: MPG-2314

The counter is set to the encoder input mode, and the count magnification to none.

```
INSET PHASE1                /* multiplier: 1 time
```

PHASE2

Pulse generation | Reserved constant

■Format

PHASE2

■Usage

INSET PHASE2

■Function

Setting the counter input

■Explanation

Applicable boards: MPG-2314

The counter is set to the encoder input mode, and the count magnification to 2 times.

```
INSET PHASE2                /* multiplier: twice
```

PHASE4

Pulse generation | Reserved constant

■Format

PHASE4

■Usage

INSET PHASE4

■Function

Setting the counter input

■Explanation

Applicable boards: MPG-2314

The counter is set to the encoder input mode, and the count magnification to 4 times.

```
INSET PHASE4                /* multiplier: 4 times
```

PL

Pulse generation — Function

■Format

PL(n;m)

■Usage

MOVS PL(1;10)
JUMP PL(2;100)

■Function

Pallet points are computed, and the point data handed over by a move command such as MOVS.

■Explanation

```
PALLET  1 P(1) P(2) P(3) P(4) 4 3
JUMP  PL(1;I)
```

This is used after the Pallet command is executed. 0~63 pallets can be specified.
Attention should be paid to the fact that the delimiter between the pallet number and the pallet point is ";".
If "," is used as the delimiter, the pallet cannot be correctly selected.
If a negative argument is used, the ZIG_ZAG order is taken.

PLIST

Pulse generation — Command

■Format

PLIST arg1

■Usage

PLIST
PLIST 10

■Function

Displaying point data

■Explanation

Point data are continuously displayed. 20 points are listed at a time and then waits for a key to be pressed.
Pressing the 'q' key ends the process, and pressing any other key continues it.

```
#plist
P(1)   X= 200  Y= 0  U= 0  Z= 0
P(2)   X= 0  Y= 0  U= 0  Z= 0
P(3)   X= 0  Y= 0  U= 0  Z= 0
P(4)   X= 0  Y= 0  U= 0  Z= 0
P(5)   X= 0  Y= 0  U= 0  Z= 0
P(6)   X= 0  Y= 0  U= 0  Z= 0
P(7)   X= 0  Y= 0  U= 0  Z= 0
P(8)   X= 0  Y= 0  U= 0  Z= 0
P(9)   X= 0  Y= 0  U= 0  Z= 0
```

POKE

Character string — Command

■Format

POKE arg1 arg2 .. (str$+n)

■Usage

POKE &H03 (a$+0)
POKE &H41 42 (a$+5)

■Function

Modifying character string data

■Explanation

This replaces codes in a character string with specified codes.
As a specified code, NULL and other binary codes can also be input, making it simple to set binary data such as CRSC
As the arguments, up to 8 can be set, wherein the last argument is used for specifying the character string and character position.
Specifying the character string and the character position is performed by closing with () as in (a$+n). In this case, the n-th character of a$ becomes the starting point.

```
LIST
10      a$="1234567890"
20       POKE   &h0041 &h0042 &h0050 &h0051 (a$+3)
30       PRINT   a$
#run

 123ABPQ890
#
```

POST

CUnet Command

■Format

POST dst ary

■Usage

POST 2 P(100)
POST 5 MBK(20)
POST -2 MBK(100)

■Function

Data transfer via CUnet

■Explanation

Data are transferred to a partner wherein CU_POST is started.
The transfer unit of one POST command is 240 bytes.
For point data, 15 points (240 bytes / 16 bytes).
For MBK data, 120 units (240 bytes / 2 bytes).

[Examples]

```
POST 2 P(100)
```

Data of P(100)~P(114) are transferred to a station of SA=2.

```
POST 3 MBK(20)
```

Data of MBK(20)~MBK(139) are transferred to a station of SA=3.

In addition, if dst is set to a negative number, data transfer is requested. (Rem) A request to SA0 is made by setting 64 instead of 0.
In this case, CU_POST must be started on the self side. This command stands by until a response comes back. If there is no response within two seconds, BIT6 of CUM_ERR is set.

[Example]

```
POST -3 MBK(20)
```

Data of MBK(20)~MBK(139) are requested to a station of SA=-3, and they are written into the same area of the self.

By this command, point data and MBK data can be shared between MPCs equipped with CUnet. However, the response speed is 0.1~0.5 sec. Because there is no real-time nature, high-speed sharing should be performed via the memory I/O of CUnet.

Whether a transmission is complete normally or not should be checked by referring to CUM_ERR.
If a communication error occurs, BIT7 becomes 1, and the details are reflected on BIT0~BIT3.

CUM_ERR

- BIT7: MAIL SEND ERROR
- BIT6: There is no response to a transfer request.
- BIT5: Communication stopped.
- BIT4: Transmission timeout is invalid (Usually 0).
- BIT3: Transmission block is invalid (Usually 0).
- BIT2: Transmission timeout occurred.
- BIT1: Transmission partner does not exist.
- BIT0: Transmission partner is not standing by for reception.

```
POST 3 MBK(20)
IF  CUM_ERR!=0 THEN  : PRINT  "X_ERR" CUM_ERR : END  : END_IF
```

Data of MBK(20)~MBK(139) are transferred to a station of SA=3, and whether the transmission was completed normally or not is checked.

If no data are specified as in “POST n”, whether CU_POST is started on the partner and self sides can be checked.
If normal, “Ok” is displayed.

[Example program]
A system is assumed, wherein MPC-A and MPC-B are connected via CU-net, a touch panel is connected only to the A side. At this time, MBK(1000)~ are assigned as the operation screen of MPC-A, and MBK(2000)~ the operation screen of MPC-B.

```
//MPC-A
        CUNET 2 2 32
        MEWNET 38400 2
        CU_POST
        FORK 1 *SHARE_MBK
        END
*SHARE_MBK
        DO
        POST 4 MBK(2000) //  MPC-B OUT AREA
        POST -4 MBK(2200) // MPC-B IN AREA
        TIME 100
        LOOP

//MPC-B
        CUNET 4 2 32
        CU_POST
        END
```

POS_L

Pulse generation — Reserved constant

■Format

POS_L

■Usage

HOME POS_L POS_L POS_L POS_L

■Function

A positive large number

■Explanation

Applicable boards: MPG-2314

If a large amount of origin-return near-origin movement is desired, POS_L or NEG_L should be used. These are positive or negative large number of 3-byte length.

```
#prx POS_L
007FFFF0
#prx NEG_L
FF80000F
```

PRA

Maintenance — Command

■Format

PRA array(n)

■Usage

PRA AHO(10)
PRA FOOL(10,1)

■Function

Displaying the values of an array

■Explanation

This is a tool to display the entire content of an array.

Array elements are displayed 20 at a time. It is also applicable to a two-dimensional array.

PRINT

Maintenance — Command

■Format

PRINT [val,str]

■Usage

PRINT "res=" a$ a cc bb$ a a a$ "123abc"

■Function

For display debugging of numerical character strings

■Explanation

This is a command to display variables and character strings, and is inserted in a program and used for monitoring the status.

```
10      a$="123"
15      bb$="koatae"
20      a=456 : cc=1096
```

```
30          PRINT   "res=" a$ a cc bb$ a a a$ "123abc"
#run

res= 123 456 1096 koatae 456 456 123 123abc
#
```

PRINT#

Communication Command

■Format

PRINT# [COM#] [Options] arg1 arg2 ...

■Usage

PRINT# 1 a$ "123\n"
PRINT# 5 COMPOWAY snd$
PRINT# 3 STR_LEN|32 a$

■Function

Outputting to a communication port

■Explanation

PRINT# performs output to the serial port.
If the first argument is a numerical value, that value specifies the RS-CH number.
As output arguments, character strings, character string variables, and variables can be used.

```
PRINT#   "    count=" i_ " " i_*i_
```

There is no space inserted between arguments in PRINT#.
In addition, although character strings as arguments cannot be connected using +, character strings can be connected in output by listing arguments in the following manner.

```
PRINT# CHR$(1) "DATA" CHR$(3)
```

Therefore, the same results are given as follows:

```
b$=CHR$(1)+"DATA"+CHR$(3)
PRINT# b$
```

▪Fixed-length output option STR_LEN

A character string output is usually terminated with a NULL. However, there are cases requiring a fixed-length character string output containing binary codes. The STR_LEN option is used in such cases.

```
a$="1234567" : b$="abcdfge"
print# STR_LEN|4 a$ b$
```

In this case, what is output is 1234abcd.

▪ Outputting a character string containing a NULL code

The NULL code, which is an ASCII code 0, is usually regarded as a terminator of a character string and is not output in the normal method.

1) In order to output a code of 0~4 in a simple manner, 0~4 should be described inside a character string constant.
   PRINT# "ABC0DEF" --> ABC~00DEF Code 00 is output between ABC and DEF.

2) Checksum output method 1
   For example, the following procedure is taken to output a 16-bit checksum.

```
HI=CHK_SUM>>8
LO=CHK_SUM&255
PRINT# STR_LEN|2 CHR$(HI) CHR$(LO)
```

3) Checksum output method 2

This is a method which directly embeds a binary code in a fixed character string packet.

```
CMND$="CMNDEXE"
SUM=0
FOR   i=0 TO LEN(CMND$)-1 :  SUM=SUM+PEEK(CMND$+i) : NEXT
 HI=SUM>>8
 LO=SUM&255
POKE   0 HI LO (CMND$+7)
PRINT#   STR_LEN|10 CMND$
```

* POKE command writes data directly into memory. If there is an error in the description, it will cause a malfunction and/or program destruction. Caution must be exercised in using it.

[Concerning options]
COMPOWAY:
If the constant COMPOWAY is given, the character string is output in the OMRON COMPOWAY format.
The character string to be transferred should be converted into packets by COMPOWAY in advance.

STR_LEN:
When OR is taken between the constant STR_LEN and the number of transferred characters (e.g., STR_LEN|32), in outputting the character string the NULL terminator is ignored and the specified number of transferred characters are output. This is used in a transfer containing a NULL code.
To create a character string containing a NULL, the ADD_STR command is used.

```
--- Examples---
 PRINT# 1 "ABC\r"      /* Xmit  "ABC[CR]" through CH1
 PRINT# 1 "ABC\n"      /* Xmit  "ABC[LF]" through CH1
 PRINT# 1 "ABC\r\n"    /* Xmit  "ABC[CR][LF]" through CH1
 PRINT# 1 "ABC\tDEF"   /* Xmit  "ABC[TAB]DEF" through CH1

 \r=[CR]=&H0D
 \n=[LF]=&H0A
 \t=[TAB]=&H09

--- An example of COMPOWAY---
  COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
  PRINT# 5 COMPOWAY snd$
```

PRX

Maintenance Command

■Format

PRX val

■Usage

PRX A

■Function

Hexadecimal-format display

■Explanation

A numerical value is displayed in the hexadecimal format. It is a command for debugging. If a hexadecimal expression is necessary as a character string in a program, HEX$() should be used.

```
A=100:B=1000:C=10000
prx A B C
00000064 000003E8 00002710
#
```

PR_CHK

Pulse generation — Reserved constant

■Format

PR_CHK

■Usage

RANGE PR_CHK|X_A 10000 -10000

■Function

Checking the move destination

■Explanation

Applicable boards: MPG-2314

IF PR_CHK is specified beforehand, whether a limit value is exceeded or not is pre-checked, and if it is exceeded, an error stop occurs before any operation. In the case of a software limit specification without PR_CHK, because a slow-down stop occurs when the limit is exceeded, overshooting occurs by the amount of the deceleration distance.

```
RANGE PR_CHK|X_A 10000 -10000
RANGE PR_CHK|Y_A 11000 -10000
RANGE PR_CHK|Z_A 12000 -10000
```

PR_LCD

Character string — Command

■Format

PR_LCD string

■Usage

PR_LCD DD$
PR_LCD "ERR"

■Function

Displaying a character string on the LCD

■Explanation

This displays eight characters of a given character string on the LCD. Characters displayable on the LCD are 0~9, A~Z, and some codes. Lower-case characters and complex characters cannot be displayed.

PR_LCD_DATE

Time management — Command

■Format

PR_LCD_DATE

■Function

Displaying the date on the LCD

■Explanation

Date data are extracted from the built-in calendar and displayed on the LCD. In the example program, the date and time are alternately displayed on the LCD.

```
10          DO
20           PR_LCD_TIME
30           TIME  1000
40           PR_LCD_DATE
50           TIME  1000
60          LOOP
```

PR_LCD_TIME

Time management Command

■Format

PR_LCD_TIME

■Function

Displaying the current time on the LCD

■Explanation

Time data are extracted from the built-in calendar and displayed on the LCD.
In the example program, the date and time are alternately displayed on the LCD.

```
10          DO
20           PR_LCD_TIME
30           TIME  1000
40           PR_LCD_DATE
50           TIME  1000
60          LOOP
#
```

PTR$

Character string Function

■Format

PTR$(m)

■Usage

```
ptr_=a$
ptr_=ptr_+10
k$=PTR$(5)
```

■Function

m characters from the pointer position

■Explanation

A character string of m characters from the pointer position is extracted.
A necessary character string can be easily cut out from a character string.
Because the point position is reflected on ptr_, manipulating this value allows adjusting the cut-out position of the character string.
ptr_ is initialized by ptr_=a$ or the SERCH command.

Example 1) Usage of PTR$() when the location and number of characters are cleared in advance.

```
---MPC-XY03 example---
FORMAT ""                   /* Clearing the character string format setting
TT$=HEX$(TIME(0))           /* Obtaining the hour, minute, and second
ptr_=TT$                    /* Obtaining the character string position
ptr_=ptr_+2                 /* Setting the point again
HH$=PTR$(2)                 /* Extracting two characters from the pointer position
ptr_=ptr_+2
```

```
MM$=PTR$(2)
ptr_=ptr_+2
SS$=PTR$(2)
CL$=HH$+":"+MM$+":"+SS$    /* Connecting character strings
PR "(1)" TT$ "->" CL$      /* TT$: original character string, CLS: connected character string

#RUN
(1) 00123400 -> 12:34:00
#
```

Example 2) How to decompose a character string consisting of two numerical expressions delimited by a space into independent character strings, respectively. Attention should be paid to the fact that the difference in the pointer positions is used as the length of the character string.

```
 C41$="Mx+9.7042e+002 My+6.3210e+002"
' Serching the space position
 a_=C41$
 l_=LEN(C41$)
  SERCH   C41$ " "
 b_=ptr_
'b_ is the space position.
 a_=ptr_-a_
 ptr_=C41$
 C1$=PTR$(a_)
 ptr_=b_
 C2$=PTR$(l_-a_)
  PRINT   C1$
  PRINT   C2$
#RUN
 Mx+9.7042e+002
 My+6.3210e+002
#
```

ptr_

Character string — Reserved variable

■Format

ptr_

■Usage

ptr_=a$

■Function

Character string pointer

■Explanation

Task variable. A task points to a position inside a character string.

```
10      a$=HEX$(DATE(0))
20       PRINT   a$
30      ptr_=a$                 /* set the pointer position
40      y$=PTR$(4)              /* copy 4 characters
50      ptr_=ptr_+4             /* re-set the pointer position
60      m$=PTR$(2)
70      ptr_=ptr_+2
80      d$=PTR$(2)
90       PRINT   y$ m$ d$
RUN

 20081117                   /* a$
 2008 11 17                 /* y$ m$ d$
```

PULSE_OUT

IO Command

■Format

PULSE_OUT port# interval [count]

■Usage

PULSE_OUT 0 10 10
PULSE_OUT 0 10
PULSE_OUT VOID
PULSE_OUT 32767

■Function

Automatic ON/OFF of an output port

■Explanation

An output port is automatically turned ON/OFF.
If count is specified, it is turned off after the specified number of turning ON/OFF, wherein the interval is set in units of 0.1 second.
To stop a port while turning ON/OFF, interval should be set to 0.

PULSE_OUT 0 10 turns Port 0 ON/OFF.
PULSE_OUT 0 0 stops turning ON/OFF.
PULSE_OUT VOID cancels all PULSE_OUT settings.
PULSE_OUT 32767 synchronizes the actions of all PULSE_OUT settings.

PWM

IO Command

■Format

PWM portn k

■Usage

PWM 15 A

■Function

PWM pulse generation

■Explanation

PWM of a specified port is turned on/off.
PWM period is 50 msec. The port is turned on only for the given k msec.
Used for controlling the electric power of a heating element or Peltier element
* PWM is an abbreviated expression of Pulse Width Modulation.

QUIT

Multitasking Command

■Format

QUIT arg1 arg2 arg3..

■Usage

QUIT 1
FOR I=1 TO 4 : QUIT I : NEXT

■Function

Halting a task

■Explanation
Multitasking program started by FORK is halted.

QUIT_FORK

Multitasking Command

■Format
QUIT_FORK n *LABEL

■Usage
QUIT_FORK 1 *LABEL

■Function
Starting a task

■Explanation
Although the same function as FORK, no error occurs even if the target task is already started.

RAD

Floating point Function

■Format
RAD(v)

■Usage
FP(0)=SIN(RAD(45))

■Function
Radian conversion

■Explanation
An angle is converted from degrees to radians. It can also be used as RAD(180) to obtain π.

```
#FP(0)=RAD(180)
#FP(1)=TAN(RAD(45))
#pr FP(0) FP(1)
 3.141593E+00 1.000000E+00
#
```

RANGE

Pulse generation Command

■Format
RANGE axis pos_limit neg_lmit

■Usage
RANGE X_A 10000 -10000
RANGE X_A|Y_A 20000 0
RANGE X_A|PR_CHK 1000 -1000
RANGE VRING|X_A 1000

■Function
Setting an operable range

■Explanation

RANGE sets a limit value which becomes a software limit for each axis.
The RANGE command only sets a value to an internal register.
To enable the software limit by this value, SLMT_ON should be added to an argument of the INSET command.
Example:INSET X_A XXXXX|SLMT_ON

If OR is taken between the axis specification and a constant PR_CHK, a move destination coordinate check is performed for the PtoP control commands (such as RMVS, MOVL, and JUMP). If the move destination is not within the specified range, an error is displayed, and the system halts.
The move destination check is disabled for commands such as RMVC and RMVT. SLMT_ON should be used together.

If OR is taken between the axis specification and a constant VRING, the internal position counter becomes a ring counter.
The ring counter is used for managing the position of a rotation axis among others.
If VRING is specified, the software limit is disabled.

```
10 PG 1
20 RANGE X_A|Y_A 200000 -1000
30 RANGE Z_A|PR_CHK 1000 -1000
40 INSET X_A|Y_A LMT_ON|SLMT_ON
```

RCV

CUnet Function

■Format

RCV(arg)

■Usage

A=RCV(A$)
A=RCV(P(100))
A=RCV(DAT(10))

■Function

Receiving mails

■Explanation

The RCV function is a mail-receiving function used paired with the XMT function.
It cannot be used along with CU_POST or POST.
P(n), X(n)~Z(n), MBK(n), an array, or a character string can be specified as the argument, and received 256-byte data are automatically stored in a specified location.
If no mail is received within a specified time (the default is 10 seconds), -3 is returned.
Changing the timeout time is performed by setting a specified time (in units of 0.1 sec) to timer_ and executing RCV().
A returned value of -2 indicates that CUM_ERR contains an error code, and -1 indicates that the argument specification is incorrect.
If reception is normal, the number of the origin which sent the received mail is returned.

```
--MPC A side--
LIST
10      CUNET   0 4 31
20      DIM   a(100)
30      FILL   a(0) 0
40      TIME   100
50     CUM_ERR=0
```

```
60      a$="1234567890"
70      IF   XMT(8,a$)!=0 THEN  : END   : END_IF
80      IF   RCV(a(1))!=8 THEN  : END   : END_IF
90      PRINT   a(1) a(2) a(3) a(63) a(64)
#run

 10 20 30 630 640
#
--MPC B side--
LIST
10      CUNET   8 4 31
20      DIM   b(100)
30      TIME   100
40      CUM_ERR=0
50      IF   RCV(b$)!=0 THEN  : END   : END_IF
60      PRINT   b$
70      FOR   i=1 TO 64 :  b(i)=i*10 : NEXT
80      IF   XMT(0,b(1))!=0 THEN  : END   : END_IF
90      END
#run

 1234567890
#
```

RENUM

Editing Command

■Format

RENUM [n]

■Usage

RENUM
RENUM 5

■Function

Renumbering of statement numbers

■Explanation

Statement numbers are renumbered with intervals of 10. If a number is specified, renumbering is made with that number as the interval.

RESUME

Control statement Command

■Format

RESUME [arg]

■Usage

RESUME
RESUME _NEXT

■Function

Returning from an error processing

■Explanation

This is a return processing from ON_ERROR.
Because RESUME returns the control to the location of occurrence, RESUME should be stated when retrying a command, and RESUME_NEXT when moving onto the next processing without retrying. See ON_ERROR.

RETURN

Control statement — Statement

■Format

RETURN [arg1,arg2..]

■Usage

```
GOSUB *LABEL

*LABEL
 RETRUN
*LABEL
 RETURN aho
```

■Function

Returning from a subroutine. In addition, arguments can be returned to the side which executed GOSUB.

■Explanation

Returning from a subroutine to the program which called GOSUB.
The program called by GOSUB must return using RETURN.
In addition, if an argument is given to RETURN, results can be returned to the parent program.

```
10 GOSUB *CAL 300 400
20 _VAR RES
30 PR RES
40 END
50 *CAL
60 _VAR V_ W_
70 RETURN SQR(SQ(W_)+SQ(V_))
RUN

*
Compiling
---------------------
500
#
```

RMVC

Pulse generation — Command

■Format

RMVC axis arg

■Usage

```
RMVC X_A CW
RMVC Y_A CCW
```

■Function

Pulse generation without specifying any quantity. CW and CCW specify the direction. Alternatively, +1 and -1 can be used.

■Explanation

CW-direction pulse is generated if a positive number is specified as arg, and CCW-direction pulse if a negative number is specified.

RMVL

Pulse generation Command

■Format

RMVL arg1 [arg2,arg3,arg4]

■Usage

RMVL x y 0 0
RMVL x y 0 z
RMVL 0 y u z

■Function

Pulse generating by linear interpolation.

■Explanation

This generates a pulse by linear interpolation for up to three axes. If four axes are specified, an error occurs.
As the speed in the interpolation, the speed of the enabled axis is used in the order of X>Y>Z>U.

RMVL 0 y u z

specifies linear interpolation of yuz, and the y-axis speed is used.

RMVS

Pulse generation Command

■Format

RMVS [axis] n
RMVS X [Y,U,Z]

■Usage

RMVS X_A n
RMVS x y u z

■Function

A specified amount of pulse is generated.

■Explanation

This is a relative pulse generation command with acceleration/deceleration. A positive value indicates the CW direction. A negative value indicates the CCW direction.

RMVT

Pulse generation Command

■Format

RMVT axs arg1 arg2 [CCW|CW|0 cent1 cent2]

■Usage

RMVT X_A|Z_A 20000 0
RMVT X_A|Z_A 0 20000 CCW 0 10000

■Function

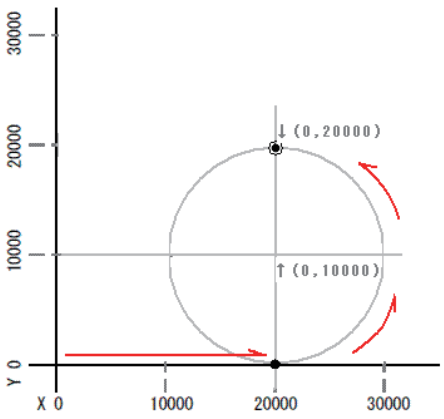
Continuous interpolation move

■Explanation

This is a continuous interpolation command by the relative coordinates. If CCW or CW is given as the third parameter, circular interpolation is used.
In this case, a parameter to specify the center is required.

All the coordinate values become relative coordinates from the position where the command is executed.

If there is no third parameter or 0 is given, linear interpolation is used.
In the example, circular interpolations of X and Y are performed.
EN_DACK and DS_DACL enable or disable deceleration.
The figure is a schematic view of executing RMVT X_A|Y_A 0 20000 CCW 0 10000.




```
PG  0
 ACCEL  8000
 CLRPOS
 DS_DACL
 RMVT  X_A|Y_A 20000 0
 RMVT  X_A|Y_A 0 20000 CCW 0 10000
 RMVT  X_A|Y_A 0 -20000 CCW 0 -10000
 RMVT  X_A|Y_A 10000 0
 EN_DACL
```

RR

Pulse generation Function

■Format

RR(arg1)

■Usage

WAIT RR(X_A)==0
WAIT RR(ALL_A)==0
IF RR(X_E)!=0 THEN

■Function

Monitoring the operation status of MPG

■Explanation

RR(arg1)
returns the AND value between the status and arg1. (arg1 & status)
However, if arg1 is 0, no AND is taken but the operation status is read as it is and returned. Ordinarily, stalling of operating axes is monitored as in

WAIT RR(X_A|Y_A)==1.

In reading status for MPG-2314, the upper 4 bits become the error statuses of the axes. Reserved constants X_E~U_E and ALL_E correspond to them.
* Status denotes the PRO register of MCX-314As.

RR3

Pulse generation Function

■Format

RR3(axis)

■Usage

A=RR3(X_A)

■Function

Reading and releasing an interrupt flag of MPG-2314

■Explanation

If an interrupt is set in MPG-2314, and INT interrupt occurs according to the content of an interrupt occurrence register RR3.
Releasing an interrupt requires releasing the interrupt condition and reading the RR3 register.
Because the RR3 register is independently prepared for each axis, its reading requires specifying the axis.
Therefore, a description such as RR3(ALL_A) cannot be made.
Reading must be performed by specifying a single axis as in RR3(X_A) and RR3(U_A).
The significance of the obtained value (1 byte) I as follows:
bit7:D_END bit6:C_STA bit5:C_END bit4:P>=COMP+ bit3:P
MPC-2000 only supports bit4 and bit 3, and no other interrupts occur.

RS

Maintenance | Command

■Format

RS ch

■Usage

RS 1

■Function

Displaying the receiving buffer

■Explanation

If this command is executed, the content of the 256-byte receiving buffer is displayed as follows.
The last corresponds to the most recently received character.
“>” indicates the current unread character position. INPUT starts reading from here.

```
#rs 1

The '>' shows the 1st chr-byte
 5   -CR [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 [D0 ~
 >1  1  1  1  3  3  3  n  m  n  m  n  m  CR ~
#
```

rse_

Communication | Reserved variable

■Format

rse_

■Usage

pr rse_

■Function

Communication error status

■Explanation

This is a task variable.

It expresses the content of an error at the time of communication.

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 TMOUT|5 a$           /* timeout 5 sec
35      IF   rse_==1 THEN               /* check timeout
36       PRINT   "timeout"
37       ELSE
38       PRINT   a$
40      END_IF
#RUN

 timeout                        /* fail
#RUN

 asdfg                          /* success
#
```

RUN

Control statement — Statement

■Format

RUN arg1

■Usage

RUN
RUN *LABEL
RUN 900

■Function

Program execution

■Explanation

After being compiled, a program is stored in a flash ROM and executed.
If the program is already compiled, it is immediately executed.

SA

CUnet — Function

■Format

SA(val)

■Usage

ON SA(5)+0

■Function

Obtaining the ON/OFF/SW number corresponding to a CUnet SA.

■Explanation

This is a function which relates a CUnet station address and its ON/OFF number.
The first ON/OFF of SA5 becomes SA(5).

SA0_B~SA63_B

CUnet — Reserved constant

■Format

SA0_B~SA63_B

■Usage
IN(SA0_B)

■Function
CUnet SA numbers

■Explanation
These are the I/O bank numbers corresponding to CUnet station addresses.

SA0~SA63

CUnet — Reserved constant

■Format
SA0~SA63

■Usage
ON SA0+5

■Function
CUnet SA numbers

■Explanation
These are the I/O numbers corresponding to CUnet station addresses.

SA_B

CUnet — Function

■Format
SA_B(val)

■Usage
OUT &H55 SA_B(5)

■Function
Obtaining the CUnet IN/OUT bank numbers

■Explanation
This is a function which relates a CUnet station address to the IN/OUT command bank number.
The first bank of SA5 becomes SA(5).

SEC

Time management — Command

■Format
SEC MBK(n)
SEC n h m s

■Usage
```
SEC   MBK(7000)
 SEC   7 17 20 2
 SEC   8 0
```

■Function
Initial setup of one-second counters

■Explanation
This performs the setup of one-second counters SEC(0)~SEC(15).

```
SEC  8 0
```
clears a one-second counter SEC(8).

```
SEC  7 17 20 2
```
sets a one-second counter SEC(7) to 17 hours 20 minutes 2 seconds.

```
SEC  MBK(7000)
```
determines the copy location of the counter value to MBK and enables copying.

```
SEC  MBK(7000)
 SEC  5 10 58 40
 SEC  6 16 10 1
 SEC  7 17 20 2
 SEC  8 0
 FOR  i=5 TO 8
  EN_SEC  i
 NEXT  i
 FORK  11 *mon
 END
*mon
 DO
  TIME  1000
  FOR  i=5 TO 8
   SEC  i
   PRINT  MBK(7000+(i*3)) MBK(7001+(i*3)) MBK(7002+(i*3))
  NEXT  i
  PRINT  "next"
 LOOP
```

SEC

Time management — Function

■Format

SEC(n)

■Usage

IF SEC(0)>SEC(1) THEN : print "TIME_OVER" : END_IF

■Function

One-second counters

■Explanation

There are 15 one-second counters, SEC(0)~SEC(15), prepared.
SEC(n) has its count stopped after a power-on rest.
The count is resumed by EN_SEC n.
The initialization of a counter is performed by the SEC command.
SEC n 0 clears the counter. SEC n 10 9 8 sets it to 10 hours 9 minutes 8 seconds.
Data of SEC(n) are in the 4-byte format of hours (2 bytes), minutes (byte), and seconds (byte), and the values cannot be directly referred to. To do so requires the following operations.

```
print SEC(0)/65536 -->Hours
print SEC(0)/256&255 -->Minutes
print SEC(0)&255 --> Seconds
```

To use this as a time alarm,

```
SEC 10 11 12 15
IF SEC(9)>SEC(10) THEN
```

In addition, the value of SEC(n) can be copied to three words in the MBK area in real time.

To enable this, SEC MBK(7800) needs to be executed for example, and either EN_SEC or DS_SEC be executed.
SEC(n) with neither EN_SEC nor DS_SEC executed is not copied.
The copying area of SEC(0) is determined thereby, and afterwards different counter values are copied every three words.

SEC(0) -> MBK(7800) MBK(7801) MBK(7802)
SEC(1) -> MBK(7803) MBK(7804) MBK(7803)

```
SEC  MBK(7000)
 SEC  5 10 58 40
 SEC  6 16 10 1
 SEC  7 17 20 2
 SEC  8 0
 FOR  i=5 TO 8
  EN_SEC  i
 NEXT  i
 FORK  11 *mon
 END
*mon
 DO
  TIME  1000
  FOR  i=5 TO 8
   SEC  i
   PRINT  MBK(7000+(i*3)) MBK(7001+(i*3)) MBK(7002+(i*3))
  NEXT  i
  PRINT  "next"
 LOOP
```

SEC

Time management — Reserved variable

■Format
SEC

■Usage
pr SEC

■Function
One-second counter

■Explanation
This is a counter which counts up at every one second.

```
10     SEC=0
20      PRX  TIME(0)
30      WAIT  SEC>10
40      PRX  TIME(0)
#RUN

00022538
00022548
```

SECTION ～ END_SECTION

Control statement — Statement

■Format
SECTION *Label ~ END_SECTION

■Usage

```
SECTION *AAA
 ON 1
END_SECTION
```

■Function

Constituting a group of programs which can be batch erased.

■Explanation

SECTION *Label ~ END_SECTION has the same function as *Label ~ RETURN.
Therefore, it is executed as a subroutine by GOSUB *Label or GOSUB_NE *Label.
The difference from normal subroutines is that it can be batch erased by DEL *Label.
The DEL command searches for SECTION ~ END_SECTION of a specified *Label and batch erases it.

SELECT_CASE

Control statement — Statement

■Format

```
SELECT_CASE arg
```

■Usage

```
SELECT_CASE IN(0)&&HF
 CASE 1 : GOSUB *A
 CASE 2 : GOSUB *B
 CASE_ELSE GOSUB *C
END_SELECT

SELECT_CASE VOID
 CASE SW(1) : GOSUB *A
 CASE SW(2) : GOSUB *B
 CASE_ELSE GOSUB *C
END_SELECT
```

■Function

Classified branching according to a numerical value in the CASE statement
Classified branching according to a logical formula in the CASE statement

■Explanation

SELECT_CASE is for exclusive classification control, wherein a given argument and the argument of each CASE are compared and only the part following a coinciding CASE statement is executed. [EXAM 1]

If the argument of SELECT_CASE is set to VOID, a logical formula proprietary to the CASE statement is evaluated for execution.
Evaluation of CASE statements is performed in order from the top.
In addition, logical conjunctions such as AND and OR can be used inside the logical formula in the CASE statement.

If two CASE statements are put alongside each other as in CASE 1: CASE 2, OR of the logic of the CASE statements is taken. [EXAM 3]

```
[EXAM 1]
SELECT_CASE  a
  CASE  1 :   PRINT  1
  PRINT  111
  CASE  2 :   PRINT  3
  PRINT  123
```

```
 CASE_ELSE :   PRINT  4
 PRINT  456
END_SELECT

[EXAM 2]
 SELECT_CASE  VOID
  CASE  SW(192)==1 : PRINT  192 : WAIT  SW(192)==0
  CASE  SW(193)==1 : PRINT  193 : WAIT  SW(193)==0
  CASE  SW(194)==1 : PRINT  194 : WAIT  SW(194)==0
  CASE_ELSE
  END_SELECT

[EXAM 3]
 SELECT_CASE  A
  CASE  0
  CASE  1 : PRINT  1
  CASE  2 : PRINT  2
  CASE  5 : PRINT  5
  CASE_ELSE  : PRINT  3
 END_SELECT

 SELECT_CASE  VOID
  CASE  A==0
  CASE  A==1 : PRINT  1
  CASE  A==2 : PRINT  2
  CASE  A==5 : PRINT  5
  CASE_ELSE  : PRINT  3
 END_SELECT
```

SENSE_ON,SENSE_OFF

IO Command

■Format

SENSE_ON port sw

■Usage

SENSE_ON 16 -1

■Function

Real-time on/off

■Explanation

If the specified input becomes 1, a specified port is turned on in real time (within 1 msec). (SENSE_OFF for turning off)

```
SENSE_ON 16  192
```

If SW(192) turns on, 16 is turned on.

```
SENSE_OFF 16 192
```

If SW(192) turns on, 16 is turned off. Once there is a response, the setting is released. In addition, forced release can be performed by SENSE_ON VOID.

SERCH

Character string Command

■Format

SERCH src$ f$

■Usage

SERCH A$ "C="

■Function

Searching for a character string

■Explanation

A character string is searched for, and the result is reflected on a point ptr_.
In the following example, it is used in combination with PTR$().

```
120     a$="adhjkashdjkas123_chuchu_tako_"
130      SERCH   a$ "123"
140     c$=PTR$(8)
150      PRINT   c$
#run

_chuchu_
#
```

SERCH$

Character string Function

■Format

SERCH$(str)

■Usage

```
ptr_=d$
 ptr_=ptr_+20
 a=SERCH$("we") :  j$=PTR$(2)
```

■Function

Searching for a specified character string and moving the pointer to the position after the character string.

■Explanation

A character string is searched for and the pointer is moved to the position after the character string.
SERCH$() has no character specified string to be searched for. Therefore, ptr_ needs to be determined beforehand.

```
ptr_=a$

a$="A=100 B=100"
ptr_=a$
ptr_=SERCH$("B=")-2
b$=PTR$(5)
```

b$ becomes B=100.

In practice the first search of a character string is performed by using the SERCH command wherein the character string can be specified, and the SERCH$ function is used for continued searching.
In the example program, a character string is cut out by also employing ptr_ and PTR$()

```
10      a$="1234567890abcdefgABCDEFG"
30       SERCH   a$ "a"
35      s=ptr_-1 :  e=SERCH$("A") :  c=e-s-1
40      ptr_=s
50      c$=PTR$(c)
60       PRINT   c$
#run
 abcdefg
#
```

SET

Pulse generation Command

■Format

SET n x y u z

■Usage

SET 0 1 1 1 1
SET 1 5 5 5 1

■Function

Setting the amount of inching in the TEACH command.

■Explanation

The TEACH command can perform inching with the xyuz keys, and the amount is set by this command.
There are four areas which can be specified by SET, for which 0~3 are specified to set respective values.
If a corresponding key of '0'~'3' is pressed in the TEACH command, the set amount of inching is called up.

SET_MCX

Pulse generation Command

■Format

SET_MCX axs Cmd WR6+WR7

■Usage

SET_MCX Z_A &h0006 400

■Function

Direct setting of the MCX314 command

■Explanation

When an action trigger is defined by the SYNC command and a certain amount of pulse is generated by the trigger, the command and pulse value are directly set to MCX-314 by the SET_MCX command.
In this example, when the X-axis count has reached 100, 50 Z-axis pulses are output.
SET_MCX Z_A &h0006 50 is a command for the Z axis, which specifies the command 06 for specifying the amount of movement and the amount of movement 50.
For the specification of this command, see the data sheet of MCX-314.

```
ACCEL   Z_A|OUTSL 1000000 10000 1000000
 ACCEL   X_A|OUTSL 3000
 INSET   X_A CMP_CNT|PHASE2

 SYNC   X_A &H00004001 0
 SET_MCX   Z_A &h0006 50
 SYNC   Z_A 0 1
 CLRPOS Z_A
 RANGE   X_A 100 0
'
 WAIT   CMP_C(Z_A)!=0
 WAIT   RR(Z_A)==0
```

SETP

Pulse generation Command

■Format

SETP n arg1 arg2 arg3 arg4
SETP n P(m)
SETP n PL(m;l)
SETP n strng

■Usage

SETP 1 100 100 20 3
SETP 2 X(0) Y(0) U(0) z(0)
SETP 13 P(3)
SETP 100 "abcdef"

■Function

Setting values to point data.
Specifying 0 as n indicates the current point.
Specifying -1 as n indicates the encoder counter, and using a character string as an argument indicates character string storage.

■Explanation

A point data editing command. Because P(n) or PL(m;n) can be specified as an argument, point data can be copied or generated using this command.
In addition, character strings can also be stored in the point data area.
Only single terms can be used as character string arguments. ('+' junction is not allowed in a$, " ", or str$().)

```
FOR   i=1 TO 10
 SETP   i STR$(i-5)
NEXT
FOR   i=1 TO 10
 PRINT   P$(i)
NEXT
FOR   i=1 TO 10
a$=STR$(i-5)+" Volt"
 SETP   i a$
NEXT
```

SET_AD

AD_DA Command

■Format

SET_AD [args]

■Usage

SET_AD 10 10 10
SET_AD AD1 10 10 10
SET_AD AD7890_10 30 30 30 30
SET_AD AD1 AD7890_10

■Function

AD setup

■Explanation

The SET_AD command specifies the type of AD converter or the number of samples for average value sampling.
Although reading in AD(1,ch) is set to the average value of 8 samples at 1 msec intervals by default, the sampling number can be specified within a range of 2~127.

For example, in the following command line:

```
SET_AD 10 10 10 10 20 20 20 30
```

the number of average values is set as follows:

```
CH0 10,CH1 10,CH2 10,CH3 10 ,CH4 20,CH5 20, CH6 20,CH7 30
```

Setting for the second MPC-AD12 board is performed as follows:

```
SET_AD AD1 10 10 10 10 20 20 20 30
```

If the AD converter is replaced with AD7890-10, the following should be executed:

```
SET_AD AD7890_10
```

This is a correction for assigning a value of 2048~4095 to a negative voltage by AD7890-10.
If the second MPC-AD12 board is replaced with AD7890-10, the following should be executed:

```
SET_AD AD1 AD7890_10
```

Here, AD1AD7890_10 is a reserved constant having a value of -10.

```
FORK  1 *disp
 END
*disp
 dd=0
 M=400
  SET_AD  AD7890_10
  SET_AD  40
  FORMAT  "S000"
  DO
 t=AD(1,0) :  t2=AD(1,2)
 d=(M-t)*2 : IF  d>35 THEN  :  d=25 : END_IF
  IF  d
  PWM  0 d
 a$=STR$(t)
 b$=STR$(t2)
 c$=a$+b$
  IF  dd%10==0 THEN
   PR_LCD  c$
  END_IF
  INC  dd
  TIME  100
 LOOP
```

SET_RTC

Time management — Command

■Format

set_rtc arg
set_rtc arg1 arg2 [arg3]

■Usage

SET_RTC &H20000119
SET_RTC &H00113000
SET_RTC 2007 12 19
SET_RTC 10 29 40

■Function

Setting the time of RTC

■Explanation

Date and time are set. Three arguments are input in the decimal format.

```
SET_RTC  2007 12 19
```

```
SET_RTC   10 29 40
```

In the hexadecimal format, there is only one argument in the following format:

```
#set_rtc &h20070731
```

Setting to July 31, 2007

```
#set_rtc &h182200
```

Setting to 18 hours 22 minutes 00 seconds.

Set time can be referred to by DATE(0) or TIME(0).
SET_RTC command cannot be executed in a protected/secret state with the FREEZE command executed. What can be executed is SET_RTC in a program protected by FREEZE.
In addition, if the calendar IC detects that the battery is low, it is preset to 2130/01/01.

```
SET_RTC   &H20000119
 SET_RTC   &H00113000
 PRX   DATE(0) TIME(0)
 SET_RTC   2007 12 19
 SET_RTC   10 29 40
 PRX   DATE(0) TIME(0)
 S_MBK   DATE(0) 1000
 S_MBK   TIME(0) 1003
 PRINT   MBK(1002) MBK(1001) MBK(1000)
 PRINT   MBK(1005) MBK(1004) MBK(1003)
```

SFTL

Arithmetic operation Command

■Format

SFTL arry(val)
SFTR MBK(n) TO MBK(m)

■Usage

SFTL ary(5)
SFTL MBK(5) TO MBK(14)

■Function

Left-shifting of an array

■Explanation

SFTL arry(5)
Left shifting in arry(0)~arry(5)
arry(5) -> arry(4) : arry(4) -> arry(3) .....

SFTL MBK(5) TO MBK(14)
Left shifting in MBK(5)~MBK(14)
MBK(14) -> MBK(13) : MBK(13) -> MBK(12) .....

```
130          FOR   i=0 TO 9
140          ary(i)=i*1000
150          NEXT   i
160          FOR   i=0 TO 9
170          PRINT   i ary(i)
180          NEXT
190          PRINT   "SHOW SFTL"
200          SFTL   ary(5)   --> 5->4 4->3 ~ 0 -> 5
210          FOR   i=0 TO 9
220          PRINT   i ary(i)
230          NEXT
```

```
SHOW SFTL
0 1000
1 2000
2 3000
3 4000
4 5000
5 0
6 6000
7 7000
8 8000
9 9000
```

SFTR

Arithmetic operation Command

■Format

SFTR arry(val)
SFTR MBK(n) TO MBK(m)

■Usage

SFTR arry(5)
SFTR MBK(5) TO MBK(14)

■Function

Right-shifting of an array or MBK data

■Explanation

SFTR arry(5)
Right-shifting in arry(0)~arry(5)
arry(1) -> arry(2) : arry(2) -> arry(3) .....

SFTR MBK(5) TO MBK(14)
Right-shifting in MBK(5)~MBK(14)
MBK(5) -> MBK(6) : MBK(6) -> MBK(7) .....

```
10      DIM   ary(5)
20      FOR   i=0 TO 4
30     ary(i)=i*1000
40      NEXT
50      FOR   i=0 TO 4
60       PRINT   i ary(i)
70      NEXT
80      SFTR   ary(3) :    'rotate ary(0)~ary(3)
90      PRINT   "SHOW SFTR"
100      FOR   i=0 TO 4
110       PRINT   i ary(i)
120      NEXT
RUN

 0 0
 1 1000
 2 2000
 3 3000
 4 4000
SHOW SFTR
 0 3000
 1 0
 2 1000
 3 2000
 4 4000
```

SHOM[MPG-2314]

Pulse generation Command

■Format

SHOM axis patn
SHOM patx paty patu patz

■Usage

SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF
SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF|CW
SHOM X_A|Z_A|Y_A IN0_ON
SHOM IN0_ON 0 0 0

■Function

Determining the condition for origin return

■Explanation

MPG-2314 has two origin return detection sensors for each axis, which are distinguished as IN0 and IN1.
Example: Those of the Y-axis are named as YIN0 and YIN1 on J4 of MPG-2314.

IN0 is for near-origin, and IN1 is set upon necessity because the Z phase is assumed.
In addition, the setting of SHOM is not enabled unless the HOME command is executed.

SHOM X_A|Z_A|Y_A IN0_ON
In this case, only the origin return of near-origin is assumed. If near-origin is on-detected, a stop is performed.

SHOM X_A|Z_A|Y_A IN0_ON|IN1_OFF|CW
In this case, operations on the X, Y, and Z axes are regulated.
After a near-origin stop, Z-phase search is performed. The search direction is the CW direction.
Near-origin is set to on-detection, Z-phase off-detection. If CW/CCW is omitted, the CCW direction is assumed.

SHOM[MPG-2541]

Pulse generation Command

■Format

SHOM pat

■Usage

SHOM &HFF

■Function

Logical setting of SD and ORG of MPG-2541

■Explanation

Origin return of MPG-2541 is fixed by the function of an IC.
Ordinarily, SD is connected to near origin, and ORG to the Z phase or the origin sensor.
In the default state, each function is enabled at ON (each input grounded).
SD denotes SLOW_DOWN and is set by ACCEL to the lowest speed.
When ORG is detected, pulse output is stopped. When SD is released, the maximum speed is restored. When setting logics, 1 is set to each corresponding bit using the arguments of the SHOM command.
Examples:

SHOME 3

Only X_SD and X_ORG are shorted as negative, and enabled by opening.

```
SHOM &HFF
```

All SD and ORG are logically inverted, and enabled by opening.

SIN

Floating point Command

■Format

sin deg r var [sf]

■Usage

sin 450000 100000 a
sin 4500000 100000 a 100000

■Function

Sine function operation

■Explanation

A floating-point SIN operation is performed.
var = r×sin(deg/sf)
Rem) If sf is omitted, sf is set to 10000.

The following are examples of executing the SIN command.

```
#SIN 300000 10000 a
#pr a
5000
#
```

This is a sin(300000/10000) operation to calculate sin(30 degrees).
Although the result is 0.5, 10000(sf) × 0.5 gives 5000.

```
#sin 450000 100000 a
#pr a
 70711
#
sin 4500000 100000 a 100000
#pr a
 70711
```

SIN,COS,TAN

Floating point Function

■Format

SIN(rad),COS(rad),TAN(rad)

■Usage

FP(0)=SIN(FP(0))
FP(1)=TAN(RAD(30))

■Function

Trigonometric functions

■Explanation

These are double-precision trigonometric functions with arguments in radians. They have significance only in the FLOAT command.

```
FLOAT FP(1)=SQR(SQ(SIN(RAD(i)))+SQ(COS(RAD(i))))
```

SLMTn

Pulse generation — Reserved constant

■Format
SLMTn

■Usage
LMT(X_A,SLMTn)

■Function
Error bit specification

■Explanation
Applicable boards: MPG-2314
Software limit – bit

```
IF LMT(X_A,SLMTn)!=0 THEN          /* confirming reason for stop
```

SLMTp

Pulse generation — Reserved constant

■Format
SLMTp

■Usage
LMT(X_A,SLMTp)

■Function
Error bit specification

■Explanation
Applicable boards: MPG-2314
Software limit + bit

```
IF LMT(X_A,SLMTp)!=0 THEN          /* confirming reason for stop
```

SLMT_OFF

Pulse generation — Reserved constant

■Format
SLMT_OFF

■Usage
INSET X_A|Y_A SLMT_OFF

■Function
Software limit setup

■Explanation
Applicable boards: MPG-2314
A software limit is disabled.

```
INSET X_A|Y_A SLMT_OFF             /* 'SOFT LIMIT' disable
```

SLMT_ON

Pulse generation — Reserved constant

■Format
SLMT_ON

■Usage
INSET X_A|Y_A SLMT_ON

■Function
Software limit setup

■Explanation
Applicable boards: MPG-2314
A software limit is enabled.

```
10 PG 1
20 RANGE X_A|Y_A 200000 -1000      /* XY axes operative restriction set
30 INSET X_A|Y_A SLMT_ON           /* 'SOFT LIMIT' enabled
```

SLOW_RUN

Maintenance — Command

■Format
SLOW_RUN taskn [timer]
SLOW_RUN TMOUT [n]

■Usage
SLOW_RUN 1 100
SLOW_RUN TMOUT 1000

■Function
Delay-executing a specified task
Setting the down counter time

■Explanation
There are two ways of using SLOW_RUN according to the arguments as explained below:

Example 1) SLOW_RUN 1 1000
In this case, a 1000-msec wait is specified every execution step of Task 1.
The value can also be modified during program execution.
In the beginning of debugging, the program should be executed cautiously and slowly by this command and faster according to the debugging progress.
Once the safety of the program is confirmed, “SLOW_RUN 1” should be executed.
In this manner, if only a task number is specified as the argument, timer wait is released.

Example 2) SLOW_RUN TMOUT 1000
If a reserved constant “TMOUT” is added as an argument, a timeout down counter is set. Although the down counter usually counts down every 100 msec, if a value of 100 or larger is set as in this example, down counting is performed at intervals of the specified msec value. In this example, down counting is performed every 1000 msec (1 sec).

Although the setting of SLOW_RUN is released by a power-on reset, if it is described inside a program, a delay element may be inadvertently set. Therefore, it should be used as a command.

SPEED

Pulse generation Command

■Format

SPEED [axs] n

■Usage

SPEED n
SPEED X_A n

■Function

Setting the pps of pulse generation

■Explanation

Pulse generation can be specified with n pps up to the maximum speed specified by ACCEL.
This command can specify the driving speed in a finer manner than the FEED command. The resolution is (the maximum speed / 8192) pps.
As in the example program, this command is also effective for fine modification of the speed during pulse generation.

```
40          ACCEL   40000 1000
50          RMVC   U_A 1
60          DO
70           FOR   i=1 TO 10
80            SPEED   U_A i*4000
90            TIME   100
100           NEXT
110           FOR   i=10 TO 1 STEP -1
120            SPEED   U_A i*4000
130            TIME   100
140           NEXT
150          LOOP
```

SQR

Floating point Function

■Format

SQR(v)

■Usage

FP(3)=SQR(3)
A=SQR(3*3+4*4)

■Function

Square root

■Explanation

In the FLOAT command it becomes a function for obtaining a double-precision square root. In an integer operation it becomes integer square root extraction.

```
FP(0)=SQR(1+3+5+7)
```

STACKS

Maintenance Function

■Format

STACKS

■Function

Displaying the consumption state of the stack area

■Explanation

STACK FREE is the number of long words in the unused stack area.
POS is the current position of the stack pointer. It is displayed in terms of the long-word count number. If 0 is displayed, it indicates a task which has not started yet.

```
#stacks
 TASK0 STACK FREE=156 STACK POS =38
 TASK1 STACK FREE=200 STACK POS =0
 TASK2 STACK FREE=200 STACK POS =0
 TASK3 STACK FREE=200 STACK POS =0
 TASK4 STACK FREE=200 STACK POS =0
 TASK5 STACK FREE=200 STACK POS =0
 TASK6 STACK FREE=200 STACK POS =0
 TASK7 STACK FREE=200 STACK POS =0
 TASK8 STACK FREE=200 STACK POS =0
 TASK9 STACK FREE=200 STACK POS =0
 TASK10 STACK FREE=200 STACK POS =0
 TASK11 STACK FREE=200 STACK POS =0
 TASK12 STACK FREE=200 STACK POS =0
 TASK13 STACK FREE=200 STACK POS =0
 TASK14 STACK FREE=200 STACK POS =0
 TASK15 STACK FREE=200 STACK POS =0
#
```

STOP

Pulse generation Command

■Format

STOP axis arg1

■Usage

STOP X_A STP_D
STOP ALL_A IN1_ON
STOP X_A|Y_A VOID

■Function

Issuing an instruction to stop or setting a stop mode

■Explanation

STOP X_A STP_D
This type of command issues an instruction of decelerating stop or immediate stop to a target MPG. STP_D is for decelerating stop, and STP_I for immediate stop.
In the example program an input switch is used for stopping during an operation.

STOP ALL_A IN0_ON|IN1_OFF
This type of command determines the functions of the MPG-2314 input ports.
In the case of IN0_ON|IN1_OFF, stop occurs if IN0(50) becomes on and IN1(51) off.

Because each stop condition is retained after command execution, VOID should be given as an argument to release it.

Stopping by INn becomes decelerating stop if the driving speed > the initial speed set by the ACCEL command and immediate stop if the driving speed == the initial speed.

To release the stop condition, VOID should be specified.
STOP X_A VOID

```
EX1:
MOVL 10000 10000 0
 WHILE RR(ALL_A) : IF SW(192) THEN : STOP STP_D : END_IF : WEND

EX2:
STOP X_A IN0_OFF            /* setting the STOP condition
  RMVS X_A POS_L            /* Generating pulse
  WAIT RR(X_A)==0
  STOP X_A VOID             /* clear the STOP condition
  RMVS X_A 1000
```

STPS

Pulse generation — Command

■Format

STPS axis n
STPS argx [argy,argu,argz]

■Usage

STPS X_A 1000
STPS 100 200 300 400
STPS VOID 100 200
STPS X_A|Y_A 1000

■Function

Setting the current position

■Explanation

If the axes are specified, the same value is set to the corresponding axes.
If the arguments are listed, those values are set to the X, Y, U, and Z axes in that order.
If VOID is given or an argument is omitted, the corresponding axis is not set.

STP_D

Pulse generation — Reserved constant

■Format

STP_D

■Usage

STOP X_A STP_D

■Function

Selecting a method to stop

■Explanation

Applicable boards: MPG-2314/2541
Decelerating stop

```
STOP X_A STP_D              /* X-axis Stop with deceleration
STOP ALL_A STP_D            /* All-axes Stop with deceleration
```

STP_I

Pulse generation — Reserved constant

■Format
STP_I

■Usage
STOP X_A STP_I

■Function
Selecting a method to stop

■Explanation
Applicable boards: MPG-2314/2541
Immediate stop

```
STOP X_A STP_I              /* X-axis Stop without deceleration
STOP ALL_A STP_I            /* All-axes Stop without deceleration
```

STR$

Character string — Function

■Format
STR$(arg)

■Usage
a$="data=" +str$(A)

■Function
Converting a numerical value into a character string

■Explanation
A numerical value is converted into a character string.
For example, after the following execution A$ becomes a character string of “DATA= 1000”.

```
A=1000
A$="DATA="+STR$(A)
```

In the standard condition, a space is added to the top of a positive value, and a “-“ to a negative value.
The conversion mode is set by the FORMAT command.

STRCPY

Character string — Command

■Format
STRCPY src$ dst$ [m n]

■Usage
STRCPY src$ dst$
STRCPY src$ dst$ 6 3

■Function
Copying a character string

■Explanation
A character string is copied. Denoted by m is the initial position of the source character string, and n is the number of copied characters.
If neither m nor n is specified, all characters are copied.

a$="012345abc"
strcpy a$ c$ 6 3
The above execution makes c$ => “abc”.

```
a$="111111111011aaaaaaa123baka_aabbanbQERaho_b11111229we48r9"
 PRINT  a$
 PRINT  LEN(a$)
 FOR  i=0 TO 5
  STRCPY  a$ b$ i 10
  PRINT  b$
 NEXT  i
 FOR  i=20 TO 25
  STRCPY  a$ b$ i
  PRINT  b$
 NEXT  i
#run
1111111110
1111111101
1111111011
111111011a
11111011aa
1111011aaa
23baka_aabbanbQERaho_b11111229we48r9
3baka_aabbanbQERaho_b11111229we48r9
baka_aabbanbQERaho_b11111229we48r9
aka_aabbanbQERaho_b11111229we48r9
ka_aabbanbQERaho_b11111229we48r9
a_aabbanbQERaho_b11111229we48r9
#
```

SUBST

Character string Command

■Format

SUBST str

■Usage

```
b$="ABC123 &H1234FJ  &HBCDEF1 "
    SERCH  b$ "&H"
    ptr_=ptr_-2
   SUBST  " $"
```

■Function

Substituting a character string

■Explanation

SUBST overwrites a section starting at the character string pointer position with a given character string.

```
70     b$="ABC123 &H1234FJ  &HBCDEF1 "
80     SERCH  b$ "&H"
120     ptr_=ptr_-2
130     SUBST  " $"
140     ptr_=SERCH$("&H")
150     ptr_=ptr_-2
160     SUBST  " $"
170     PRINT  b$
#ABC123  $1234FJ  $BCDEF1
#
```

SW

IO Function

■Format
SW(arg)

■Usage
```
A=SW(192)                          /*Reading out an input port
IF SW(A)==0 THEN : ON 5 : END_IF   /*Conditional branching according to the input
WAIT SW(192)==1                    /*Waiting for a condition
```

■Function
Reading out an input port

■Explanation
If the input port is shorted to GND, 1 is returned. In a floating state, 0 is given.

SWAP

Multitasking Command

■Format
SWAP

■Function
Forcibly swapping a program during execution (executed task replacement)

■Explanation
When a task requiring a long processing time is being executed, it occupies the full time-slice time of a task, slowing down the execution of other tasks. In such a case, the execution privilege can be forcibly moved to another task by artificially using SWAP.

SYNC

Pulse generation Command

■Format
SYNC axs WR6 WR7

■Usage
```
SYNC  X_A &H00004001 0
SYNC  Z_A 0 4
```

■Function
Setting the MCX-314 register

■Explanation
MCX-314 built in MPG-2314 has the function of performing real-time processing by the hardware.
- If the X axis has exceeded a certain number of pulses, another axis is started up.
- If an input signal has been entered, the counter value at that point in time is latched.

By this command, these kinds of functions can be executed in real time using the hardware mechanism.
Concerning what kind of values should actually be set to WR6 or WR7, see the data sheet of MC-314.
In the program example, the Z axis is started up based on the X-axis count value (100), and when the value has reached 50 pulses, the output port (03) is turned on.

SYSCLK

Time management — Reserved variable

■Format

SYSCLK

■Usage

pr SYSCLK

■Function

System clock

■Explanation

This variable increments approximately every 1 msec after power-on (CPU clock standard).

```
10    SYSCLK=0              /* SYSCLK clear
20     TIME   100           /* delay 100msec
30    a=SYSCLK
40     PRINT   a            /* display
RUN

 101
```

S_MBK

Touch panel — Command

■Format

S_MBK arg1 arg2
S_MBK str adr c
S_MBK arg adr count

■Usage

S_MBK 1 10
S_MBK 2 11
S_MBK 1000000 20~Lng
S_MBK a$ 100 10
S_MBK 100 50 20
S_MBK 100~Lng
S_MBK DATE(0) 100

■Function

Setting the touch panel data

■Explanation

The S_MBK command sets a value in the MBK array.

1) Setting 1 to the 10th element. -> S_MBK 1 10
2) Setting 1000000 to the 20th element. -> S_MBK 1000000 20~Lng
   In this case, the lower word enters MBK(20), and the upper word MBK(21).
3) Character string substitution -> A character string (“abc” is also allowed) is set to the address 100 for 10 characters.
   The character string should be either a character string variable or a character string constant.
4) Batch setting -> S_MBK arg adr count
   A value arg is batch set to MBK(adr) ~ MBK(adr + count – 1).
5) Display -> S_MBK n displays the contents of MBK(n) ~.
   For the Lng display, S_MBK n~Lng should be executed.

In the case of S_MBK DATE(0) n / S_MBK TIME(0) n,
Date, seconds --> MBK(n)
Month, minutes --> MBK(n + 1)
Year, hours --> MBK(n + 2)
This value is automatically updated every 1 second. To stop it, 0 should be specified as in S_MBK DATE(0) 0 or S_MBK TIME(0) 0. In this case, although the specified address is 0, no writing is performed onto MBK(0)~MBK(2).

TAIL

Editing Command

■Format
TAIL

■Function
Displaying the maximum statement number

■Explanation
The maximum statement number is displayed. It is used when adding a program online.

TAN

Floating point Command

■Format
tan deg r var [sf]

■Usage
tan 300000 100000 a
tan 3000000 100000 a 100000

■Function
TAN operation

■Explanation
A floating-point TAN operation is performed.
var = r×tan(deg/sf)
Rem) If sf is omitted, sf is set to 10000.

Example 1) TAN 450000 10000 a

```
#pr a
10000
#
```

This is an operation of tan(450000/10000), signifying tan(45 degrees).
Although the result is 1, being magnified as 10000 × 1, it becomes 10000.

```
#tan 300000 100000 a
#pr a
 57735
#
#tan 3000000 100000 a 100000
#pr a
 57735
#
```

TASK

Multitasking Function

■Format

TASK(arg1)

■Usage

WAIT TASK(1)!=0

■Function

Referring to the status of a task

■Explanation

The argument is a task number, and the result is as follows:
255: The task is completed or quit.
1: The task made to stand-by according to a timer.
0: The task is in execution.
If -1 is entered as the argument, the self task number is returned.

TASKn

Multitasking Reserved variable

■Format

TASKn

■Function

Obtaining the self task number

■Explanation

The task number of a task in execution is obtained. Although TASKn is a global variable, every time the execution privilege is transferred to the task, the task number is written to TASKn by the task monitor.
Therefore, even if TASKn is mistakenly modified to another value, it is restored to a normal value every time tasks are switched.

```
10      FORK   10 *SUBTASK
20      PRINT   "main=" TASKn
30      END
40     *SUBTASK
50      TIME   500
60      PRINT   "sub=" TASKn
70      END
#run

 main= 0
# sub= 10
```

TEACH

Pulse generation Command

■Format

TEACH

■Usage

TEACH
T

■Function

Teaching point data by inching operations

■Explanation

Before executing the TEACH command, PG must be selected, and the ACCEL command executed. Once the TEACH command is executed, the current position and inching amount are displayed as listed below. Inching of each axis is performed using the following keys:

x,X
y,Y
u,U
z,Z

```
#t
PG=[1]  X=1200 Y=0 U=0 Z=0 dx=200 dy=200 du=200 dz=200  P3
```

After the P command is pressed, the system waits for the input of a point number. Once the number is input, the current position is set to the specified point data.
The inching amount set by the SET command can be selected using a key 0~3.

If the target PG is MPG-2314, errors are also displayed along with the position.
PG=[1] X=1200 Y=0! U=0 Z=0 dx=200 dy=200 du=200 dz=200
An ! mark is displayed after the value of an axis having an error.

TIME

Time management — Function

■Format

TIME(0)
TIME(255)
TIME(VOID)

■Usage

```
IF   TIME(0) < &H00182800 THEN
 GOTO  *aho
END_IF
```

■Function

Obtaining time data

■Explanation

The time value is obtained in the hexadecimal format.
If an argument is inserted, a logical product between the value and the argument is returned.
If VOID is set as the argument, the value is returned as a decimal number.
Setting time is performed by the SET_RTC command.

```
5          *aho
10           IF   TIME(0)< &H00182800 THEN
30           PRX   TIME(0)
40           WAIT   TIME(255)%16==0
```

TIME

Time management — Command

■Format

TIME arg

■Usage
TIME 100

■Function
Stopping a task for specified msec

■Explanation
TIME is a command for improving execution efficiency as well as adjusting timing.
While waiting by TIME, the task is in the SLEEP state, and CPU time resource can be assigned to other tasks.

TIME$

Character string Function

■Format
TIME$(n)

■Usage
a$=DATE$(1)+" "+TIME$(1)

■Function
Obtaining the time character string

■Explanation
The time character string is obtained.
TIME$(0)-> 00100957
TIME$(1)-> 10:09:57
TIME$(2)-> 10:09

```
a$=DATE$(1)+" "+TIME$(1)+": CNT="+STR$(i)
```

TIMEOUT

Time management Function

■Format
TIMEOUT(n)

■Usage
WAIT SW(1)&SW(2) OR TIMEOUT(0)

■Function
Detecting the timeout of timer_

■Explanation
If n = 0, it has the same significance as (timer_==0).
Its significance becomes clear by the description of WAIT SW(1)&SW(2) OR TIMEOUT(0).
The TIMEOUT() function can also evaluate timer_ of other tasks.
n: -1 0 detection of timer_ of Task 0
n: 1~31 0 detection of timer_ of Task n

```
timer_=10
PRINT TIMEOUT(0)
WAIT SW(1)&SW(2) OR TIMEOUT(0)
PRINT TIMEOUT(0)
```

TIMER

Time management — Function

■Format

TIMER(arg)

■Usage

a=TIMER(3)
a=TIMER(VOID|3)
a=TIMER(1,1)

■Function

See timer_.

■Explanation

Because timer_ is a task variable, its value cannot be referred to or set by other tasks.
If a task number is specified as the argument of TIMER(n), the timer_ value of that task can be obtained.
Rem) The unit is 0.1 second.
In addition, if the logical sum of a task number and VOID is taken, timer_ of that task can be set to 0. Because the timeout managed by the TMOUT command uses this timer_ variable, in order to force a timeout by another task, this function is used to set timer_ to 0.
If arguments are given as TIMER(1,n), the set TIMOUT value of Task n can be called up.

timer_

Time management — Reserved variable

■Format

timer_

■Usage

IF timer_==0 THEN

■Function

Down counter

■Explanation

This is a task variable, which is down-counted every 0.1 second and stops at 0.

```
10      timer_=100              /* set 10Sec -> 0.1Sec count down
20       PRX   TIME(0)          /* display current time
30       WAIT   timer_==0       /* wait 10sec
40       PRX   TIME(0)          /* display current time
#run

00014841
00014851

----------

10       FORK   3 *JOB
20      SYSCLK=0                /* SYSCLK init
30       TIME   100
40       WAIT   TIMER(3)==0     /* wait "timer_" of the TASK3 == 0
50       PRINT   SYSCLK "mSec"
60       END
70     *JOB                     /* TASK3
80      timer_=100              /* set 10Sec -> 0.1Sec count down
90       DO   : SWAP   : LOOP
#RUN

 9995 mSec
```

TMOUT

Time management — Command

■Format

TMOUT n [taskn]

■Usage

TMOUT 100
TMOUT 100 n
TMOUT VOID

■Function

Setting the timeout time

■Explanation

The timeout time is set in the units of msec. The minimum time which can be set is 10 msec. (Applicable to WS0(), WS1(), and HOME.)
By default 13 days (20000 seconds) is set.
The second argument is a task specification. If it is omitted, the self task is specified.
If VOID is specified as the argument, the initial value of 20000 seconds is set.
The value set by TMOUT is set to timer_ in WS0, WS1, and HOME.
Therefore, manipulating the value of timer_ immediately before WS0, WS1, or HOME is invalid.
Issuing the TMOUT command without any argument displays the current set values.

```
#TMOUT
TMOUTs
0 10000
1 10000
2 2000000
3 2000000
4 2000000
5 2000000
6 2000000
7 2000000
8 2000000
9 2000000
10 2000000
11 2000000
12 2000000
13 2000000
14 2000000
15 2000000
#
```

TMOUT

Communication — Reserved constant

■Format

TMOUT

■Usage

INPUT# 1 TMOUT|5 a$

■Function

Setting the reception wait timeout

■Explanation

Although there is no time limit for the INPUT# command reception wait, a time limit can

be set using the TMOUT option. If the time limit is exceeded, the occurrence of TMOUT is reflected on the rse_ variable.

```
10      CNFG#   1 "9600b8pns1NONE"
20      INPUT#   1 TMOUT|5 a$           /* timeout 5 sec
35      IF   rse_==1 THEN               /* check timeout
36       PRINT   "timeout"
37       ELSE
38       PRINT   a$
40      END_IF
#RUN

 timeout                        /* fail
#RUN

 asdfg                          /* success
#
```

UIN0

Pulse generation — Reserved constant

■Format

UIN0

■Usage

HPT(UIN0)

■Function

HPT input specification

■Explanation

Applicable boards: MPG-2314
UIN0 is specified to the HPT input port.
Related: HOME
See also XIN0

UIN1

Pulse generation — Reserved constant

■Format

UIN1

■Usage

HPT(UIN1)

■Function

HPT input specification

■Explanation

Applicable boards: MPG-2314
UIN1 is specified to the HPT input port.
Related: HOME
see also XIN1

UP_DWN

Pulse generation | Reserved constant

■Format

UP_DWN

■Usage

INSET UP_DWN

■Function

Setting the counter input

■Explanation

Applicable boards: MPG-2314

The counter is set to an up/down counter.

```
INSET UP_DWN                /* Set the counter to 'UP/DOWN' mode
```

USB

USB | Function

■Format

USB(arg1)

■Usage

IF USB(USB)!=1 THEN : GOTO *NOUSB : END_IF

PR USB(1,USB)-MBK(1000+3)

■Function

Presence/absence of USB memory

■Explanation

The presence or absence of the USB memory can be obtained.

The USB(USB) function returns 1 if the USB memory is correctly installed and 0 if it is absent. Further, if MRS-COM itself does not exist, -2 is returned.

If the version of MRS-MCOM is not supported, -1 is returned.

In addition, if 1 is entered in the upper word as follows, the total capacity of the USB memory is returned (in Mbyte).

USB(1,USB)

This value becomes valid either immediately after the DIR command or when the USB memory is installed at the time of power-on.

```
FILE$="TEST"
*RETRY
 DO
  IF  USB(USB)!=1 THEN  : GOTO  *NO_USB : END_IF
  USB_WRITE  "TEST_WRITING¥r¥n"
  TIME  100
  DIR  1000
  IF  USB(1,USB) <100 THEN  : GOTO  *NO_SPACE : END_IF
  TIME  1000
 LOOP
*NO_USB
 WAIT  USB(USB)==1
 GOTO  *RETRY
*NO_SPACE
 PRINT  "CHANGE_USB_MEMORY"
```

USB_DEL {UDL}

USB Command

■Format
USB_DEL [USB#] Str

■Usage
USB_DEL "aaa.p2k"
UDL USB1 "aaa.f2k"

■Function
Deleting a USB memory file

■Explanation
A specified file on the USB memory is deleted.
Even if it is executed when the file does not exist, no error occurs.

USB_LOAD {UL}

USB Command

■Format
USB_LOAD [USB#] strg

■Usage
USB_LOAD "DEMO.F2K"
USB_LOAD USB2 "DEMO.F2K"

■Function
Loading a program from the USB memory

■Explanation
Program data SAVEd by FTMW are loaded.
Although FTMW can load a program containing FTM comments (comment statements with a "/*" mark), USB_LOAD has no such function.
The COM ports are
DSW==6 -> USB (If omitted, MRS-MCOM of DSW=6 is accessed.)
DSW==7 -> USB1
DSW==5 -> USB2

USB_PLOAD {UPL}

USB Command

■Format
USB_PLOAD [USB#] str

■Usage
USB_PLOAD "PL3.P2K"
USB_PLOAD USB1 "PL3.P2K"

■Function
Loading point data

■Explanation
P2K and P68 type data saved by FTMW are loaded from USB memory.
Because the operation is one of overwriting, if loading them as new data, NEWP should be executed before the execution.
In addition, USB_PLOAD also supports MBK data.

```
= Data example =
SETP 1 -200 9280 0 -12680
SETP 2 30880 9280 0 -13280
SETP 3 400 29400 0 -13280
SETP 4 31080 29280 0 -13680
SETP 101 8000 0 0 0
SETP 102 8000 8000 8000 4000
SETP 103 0 8000 0 0
SETP 104 0 0 0 4000
s_mbk 100 1
s_mbk 200 2
s_mbk 300 3
s_mbk 400 4
s_mbk 500 5
s_mbk 600 6
s_mbk 700 7
s_mbk 800 8
s_mbk 900 9
s_mbk 1000 10
s_mbk 30 7868
```

USB_PSAVE {UPS}

USB Command

■Format

USB_PASVE [USB#] P(n) cnt Str
USB_PASVE [USB#] MBK(n) cnt Str

■Usage

USB_PSAVE P(1) 5000 "aa.p2k"
USB_PSAVE MBK(10) 1000 "aa.p2k"

■Function

Saving point data or MBK data

■Explanation

The cnt pieces of data from the n-th piece of point data or MBK data are saved in the USB.
Because saving is performed as APPEND saving, if the file already exists, data are added. If saving as new data, the file should be deleted beforehand by USB_DEL (remove).

```
RM "AUTO.P2K"
USB_PASVE P(1) 2000 "AUTO.P2K"
USB_PASVE MBK(10) 1000 "AUTO.P2K"
```

In this case, because point data and MBK data are saved in AUTO.P2K, it can be used as recovery data.

USB_READ {URD}

USB Command

■Format

USB_READ String

■Usage

USB_READ a$
USB_READ -1

■Function

Reading in one line of a USB memory file

■Explanation

One line of a USB memory file is read out. The file name is specified by FILE$.
By executing this in sequence, read-out can be performed line by line.
When the end of file is encountered, a function EOF(0) returns 1, and read-out stops there.
If URD is continuously executed by ignoring it, read-out starts from the top of file again.
In order to stop read-out in the middle, USB_READ -1 should be executed.
By this processing, the file will be closed.

USB_WRITE {UWR}

USB Command

■Format

USB_WRITE [USB#] Strng

■Usage

USB_WRITE "123.456"
USB_WRITE USB1 STR$(n)

■Function

Append-writing to the USB memory (opening/closing every time)

■Explanation

The USB memory is appended to.
The file name is specified using a reserved character string:
As the character string array argument, a character string array to write should be set.
Because each file is opened/closed every time, even if the power supply is cut off in the middle, data written last would remain in the USB memory.

```
LIST
10      FOR   k=1 TO 100
20       FORMAT   "uwr_00.txt"
30     FILE$=STR$(k)
40     SEC=0
50       FOR   SUM=1 TO 100
60        FORMAT   "TEST_CNT=0000¥r¥n"
70     A$=STR$(SUM)
80        FORMAT   "APND_CNT=0000¥r¥n"
90     B$=STR$(SUM+1000)
100       USB_WRITE   A$+B$
110      NEXT
120      PRINT   k SEC
130     NEXT
#
```

U_A

Pulse generation Reserved constant

■Format

U_A

■Usage

RMVS U_A 1000

■Function

U-axis specification

■Explanation
Applicable boards: MPG-2314/2541
This is a command for axis specification in the PG commands such as RMVS.
see also X_A

U_C

Pulse generation — Reserved constant

■Format
U_C

■Usage
stps U_C 1000

■Function
Counter specification

■Explanation
Applicable boards: MPG-2314
The U-counter is specified.
see also X_C

U_E

Pulse generation — Reserved constant

■Format
U_E

■Usage
RR(U_E)

■Function
U-axis error specification

■Explanation
This is used as an argument of the RR() function for examining the presence/absence of any U-axis error after a movement.
A non-zero value indicates that a specific cause of error has occurred.
The details of the error should be investigated using the LMT function or the PGE function.
Applicable boards: MPG-2314
see also X_E

VAL

Character string — Function

■Format
VAL(str)
VAL(arg)

■Usage
```
a$="a=1000 b=-1000 c=100"
a=VAL(a$) : b=VAL(0) : c=VAL(0)
 a$="x=1000.123 y=-2120.1256 "
  SERCH   a$ "x="
  PRINT   VAL(1000)
```

■Function

Extracting numeric strings from a character string and obtaining their values.

■Explanation

A character string is searched for numeric strings, which are converted into numerical values.

If more than one numerical value is contained in the character string, continuously executing VAL(0) allows extraction of numeric strings in order and their conversion into numerical values.

If a value in the range of 10~100000 is entered in arg, decimals are multiplied by arg.

In the case of X=123.4567,

if it is read by VAL(10000), an integer value of 1234567 is obtained.

```
10      a$="x=1000.123 y=-2120.1256 "
20       PRINT   VAL(a$)
30       SERCH   a$ "x="
40       PRINT   VAL(1000)
50      ptr_=SERCH$("y=")
60       PRINT   VAL(10000)
```

A case wherein decimals are contained starting at the top of a character string:

```
10 A$="123.22 B=456.12 C=789.34"
80 ptr_=A$
90 A=VAL(100)
100 B=VAL(100)
110 C=VAL(100)
120 PRINT A B C
```

VAL

Floating point — Function

■Format

VAL(str)

VAL(0)

■Usage

FP(0)=VAL(A$)

■Function

Obtaining floating-point values

■Explanation

Numeric strings are obtained as floating-point variables in the FLOAT command.

Internally, each of the numeric strings above a decimal point, below a decimal point, and an E specification is handled as a double-precision integer.

Therefore, if any of the numeric strings exceed the range of double-precision integers (within 9 digits), an error occurs.

```
A$="Mx+9.7042e+002 My+6.3210e+002"
#FP(0)=VAL(A$)
#FP(1)=VAL(0)
#pr fp(0) fp(1)
 9.704200E+02 6.321000E+02
#
```

VARS

Maintenance Command

■Format
VARS [arg]

■Usage
VARS
VARS VOID
VARS 0

■Function
Listing variables

■Explanation
1) With no argument
Variables of four characters or more having only one different character are listed. This has the objective of reducing confusion caused by displaying variables which can be easily mistaken for another.
2) VOID
This is executed after RUN. Variables for which no substitution is performed during execution are displayed. This is convenient for discovering labels with incomplete initializations or which were mistakenly used.
3) Value "VARS 0"
If a value is specified, variables having that value are displayed.

VER

Maintenance Command

■Format
VER

■Usage
#VER
MPC-1000(SH7030) BL/I 1.12_92 2012/02/20
All Rights reserved. ACCEL Corp. .T32
#

■Function
Displaying the version

■Explanation
The VER command also stops all tasks as well as displaying the version.

VER$

Character string Reserved variable

■Format
VER$

■Usage
pr VER$

■Function
Obtaining the version data

■Explanation

This is a character string variable containing version data.
The version number can also be obtained using MBK(8053).

```
10      DIM   a(10)
20      FILL   a(0) 0
30     a$=VER$
40      PRINT   "MPC_Version" a$
50      GET_VAL   a$ a(0)
60      PRA   a(0)
70      PRINT   "MBK_8053=" MBK(8053)
#RUN

 MPC_Version 1.11_29 2009/01/22

a(0)=1
a(1)=11
a(2)=29
a(3)=2009
a(4)=1
a(5)=22
a(6)=-2147483648
a(7)=-2147483648
a(8)=-2147483648
a(9)=-2147483648
 MBK_8053= 11129
#
```

VOID

Pulse generation — Reserved constant

■Format

VOID

■Usage

MOVL VOID 1000 2000 VOID

■Function

Disabling an input
Releasing a setting

■Explanation

Applicable boards: MPG-2314/2541 and others
This is used for releasing the settings of pulse generation commands and I/O commands and SELECT_CASE and the like.

```
INSET ALL_A VOID          /* INSET conditions clear
MOVL 1000 0 0 VOID        /* Z axis disable
STOP ALL_A VOID           /* STOP conditions clear
INTA_ON VOID              /* INTA_ON disable
INTA_OFF VOID             /* INTA_OFF disable
INTB_ON VOID              /* INTB_ON disable
INTB_OFF VOID             /* INTB_OFF disable
----
SELECT_CASE VOID          /* SELECT_CASE condition
TMOUT VOID                /* TMOUT disable
TIMER(VOID|3)             /* TASK3 timer_=0
PULSE_OUT VOID            /* PULSE_OUT disable
SENSE_ON VOID             /* SENSE_ON disable
SENSE_OFF VOID            /* SENSE_OFF disable
CU_POST VOID              /* CU_POST monitor
```

VOID_U

Pulse generation — Reserved constant

■Format
VOID_U

■Usage
movl P(1) VOID_U

■Function
Specifying a disabled axis

■Explanation
Applicable boards: MPG-2314/2541
Axes excluding the U-axis are specified
Same as X_A|Y_A|Z_A

```
MOVL P(1) VOID_X          /* X axis doesn't move
MOVL P(2) VOID_Z          /* Z axis doesn't move
JUMP P(3) VOID_Z          /* It stops right above the P(3)
```

VOID_X

Pulse generation — Reserved constant

■Format
VOID_X

■Usage
movl P(1) VOID_X

■Function
Specifying a disabled axis

■Explanation
Applicable boards: MPG-2314/2541
Axes excluding the X-axis are specified
Same as Y_A|U_A|Z_A
see also VOID_U

VOID_Y

Pulse generation — Reserved constant

■Format
VOID_Y

■Usage
movl P(1) VOID_Y

■Function
Specifying a disabled axis

■Explanation
Applicable boards: MPG-2314/2541
Axes excluding the Y-axis are specified
Same as X_A|U_A|Z_A
see also VOID_U

VOID_Z

Pulse generation — Reserved constant

■Format
VOID_Z

■Usage
movl P(1) VOID_Z

■Function
Specifying a disabled axis

■Explanation
Applicable boards: MPG-2314/2541
Axes excluding the Z-axis are specified
Same as X_A|Y_A|U_A
see also VOID_U

VRING

Pulse generation — Reserved constant

■Format
VRING

■Usage
RANGE VRING|X_A 999

■Function
Setting the variable ring of MCX-314

■Explanation
The current position counter of MPG-2314 is specified as the RING counter.
For example, when specified as in the example below, it increments from 0 to 999 and returns to 0 when 999 is exceeded.
This is a useful function for a turret mechanism and the like.

```
RANGE VRING|X_A 999
```

WAIT

Control statement — Statement

■Format
WAIT logical_eqations

■Usage
WAIT SW(0)==1
WAIT SW(-2)

■Function
Conditional waiting

■Explanation
This waits for a conditional formula to become 1.

WAIT SW(0)==0 waits for SW(0) to become 0.
WAIT SW(-2) waits for SW(-2) to become 1.
WAIT A==100 waits for a logical formula A==100 to become 1. In other words, wait for A to become 100.

In addition, conjunctions such as AND and OR can be used in a conditional formula. If a timeout is necessary, as in
WAIT SW(1)&SW(2) OR TIMEOUT(0)
adding OR TIMEOUT(0) allows a timeout processing.

Adding “UNTIL” to WAIT allows a real-time support.
For example, in the description of WAIT UNTIL HSW(192)==1
the task which executed this command goes into a dormant state, and instead the OS executes the conditional formula.
If the conditional formula holds true at the timing of task switching, the dormant task is restored to an execution state, and the execution right is handed over to that task.
Thereby the reaction speed does not depend on the number of started tasks and becomes within the time-slice time (3 msec).
However, the WAIT statement with the UNTIL specification has a heavy load on the OS, and if many tasks wait for conditions using WAIT UNTIL, the processing may slow down instead. WAIT UNTIL should be used for detecting highly urgent conditions.

To speed up the response time, the time-slice time should be reduced by the LIFE_TIME command.

```
10     timer_=10
20      WAIT  SW(1)&SW(2) OR TIMEOUT(0)
30      IF  TIMEOUT(0) THEN  : GOTO  *TIME_OUT : END_IF
```

WHILE-WEND

Control statement Statement

■Format
WHILE logical formula ~ WEND

■Usage
```
WHILE SW(0)==1
ON 0 : TIME 1 : OFF 0
WEND
```

■Function
Executing a conditional infinite loop

■Explanation
This is used for repeated conditional executions.
As long as the value of a logical formula is 1, the program portion between WHILE and WEND is repeatedly executed.

Wrd

Touch panel Reserved constant

■Format
Wrd

■Usage
IN(-1~Wrd)

■Function
Word-type specification

■Explanation
This specifies the unsigned 16-bit read-out for S_MBK, MBK(), IN, and OUT.

```
10      S_MBK   &H00008FFF 20~Wrd    /* WORD write
20      PRINT   MBK(20~Wrd)         /* unsigned WORD read
30      PRINT   MBK(20~Int)         /* signed WORD read
40      OUT   -1 -1~Wrd            /* WORD write
50      PRINT   IN(-1~Wrd)          /* unsigned WORD read
60      PRINT   IN(-1~Int)          /* signed WORD read
RUN

 36863                      /* unsigned
-28673                      /* signed
 65535                      /* unsigned
-1                        /* signed
```

WS0,WS1

IO Function

■Format

WS0(arg1)

■Usage

IF WS0(0)==1 THEN : GOTO *TMOUT : END_IF

■Function

I/O-waiting function with timeout

■Explanation

WS0(n) waits for SW(n) to become 0, and if the wait time set by TMOUT is exceeded, the value of 1 is returned. If it becomes 0 within the time, 0 is returned.

WS1(n) waits for SW(n) to become 1, and if the wait time set by TMOUT is exceeded, the value of 1 is returned. If it becomes 1 within the time, 0 is returned.

Rem) WS0 and WS1 utilize timer_. Therefore, if a time-up monitoring using timer_ is performed by higher-level processing which executes WS0 or WS1, a copy of timer_ should be made inside WS0 or WS1, and timer_ be restored when exiting WS0 or WS1. Therefore, although the system operates with no inconsistency, an error of about 1 digit (0.1 second) occurs every time of exiting WS0.

X Y Z U

Pulse generation Function

■Format

X(arg1)
Y(arg1)
U(arg1)
Z(arg1)

■Usage

MOVS X(1)+A VOID U(1)+B VOID
setp 1 x(0) y(0) u(0) z(0)

■Function

Returning the coordinates of the current position and point data

■Explanation

When arg1 is 0, the current position is returned.
With a value other than 0, the coordinate values of a point whose number is specified are returned.

XIN0

Pulse generation — Reserved constant

■Format

XIN0

■Usage

HPT(XIN0)

■Function

HPT input specification

■Explanation

Applicable boards: MPG-2314
XIN0 is specified to the HPT input port.
Related: HOME

```
100  IF HPT(XIN0) != THEN          /* If IN0(near-org) is on
110    RMVS X_A 10000              /* Moving to opposite direction to HOME
120  END_IF
130  WAIT RR(X_A)==0
```

XIN1

Pulse generation — Reserved constant

■Format

XIN1

■Usage

HPT(XIN1)

■Function

HPT input specification

■Explanation

Applicable boards: MPG-2314
XIN1 is specified to the HPT input port.
Related: HOME

```
*HOME
  IF   HPT(XIN0)==1 THEN  : RMVS   X_A 5000 : END_IF
  IF   HPT(YIN0)==1 THEN  : RMVS   Y_A 5000 : END_IF
  IF   HPT(ZIN0)==1 THEN  : RMVS   Z_A -5000 : END_IF
WAIT   RR(ALL_A)==0
  SHOM   X_A|Z_A|Y_A IN0_ON
  HOME   -100000 -100000 0 100000
  WAIT   RR(ALL_A)==0
```

XMT

CUnet — Function

■Format

XMT(dst,arg)

■Usage

A=XMT(8,A$)
A=XMT(8,P(100))
A=XMT(16,DAT(10))

■Function

Sending mails

■Explanation

The XMT function is a mail-sending function which is used paired with the RCV function.
It cannot be used with CU_POST or POST.
P(n), X(n)~Z(n), MBK(n), an array, or a character string can be specified as the argument, and the specified data 256 bytes are sent to dst.
If sending is completed normally, 0 is returned.
Before executing XMT, RCV needs to be executed at the partner station.
If a value other than 0 is returned, sending failure is due to one of the following causes.

1: BIT0 RCV was not executed by the sending destination.
2: BIT1 Destination does not exist.
4: BIT2 Poor communication quality in sending mail.

* See the section of RCV() for a sample program.

X_A

Pulse generation Reserved constant

■Format

X_A

■Usage

RMVS X_A 1000

■Function

X-axis specification

■Explanation

Applicable boards: MPG-2314/2541
This is a command for axis specification in PG commands such as RMVS.

```
ACCEL X_A 30000 1000 500            /* Acceleration/deceleration setting
FEED X_A 100                    /* Speed setting
INSET X_A MD_2PLS|ALM_OFF|LMT_OFF     /* In port set
SHOM X_A IN0_ON                   /* Setting Return to the Origin
MOVS X_A 1000                    /* Absolute coordinate movement
RMVS X_A 1000                    /* Relative coordinate movement
STOP X_A STP_D                   /* Moving stop with deceleration
WAIT RR(X_A)==0                  /* Wait until moving complete
IF LMT(X_A,LMTp)|LMT(X_A,LMTn) != THEN  /* Confirming reason for stop
   etc
```

X_C

Pulse generation Reserved constant

■Format

X_C

■Usage

stps X_C 1000

■Function

Counter specification

■Explanation

Applicable boards: MPG-2314
The X counter is specified.

```
10      PG  0
20      STPS  X_C 1234          /* set the X counter
30      PRINT  X(-1)          /* display the X-counter value
#RUN

 1234
-----
a=CMP_C(16,X_C)                /* compare the COMP+ register to X counter
```

X_E

Pulse generation — Reserved constant

■Format

X_E

■Usage

RR(X_E)

■Function

X-axis error specification

■Explanation

This is used as an argument of the RR() fuction to examine the presence/absence of an X-axis error after a move.
If the value is not 0, it indicates a specific cause of error has occurred.
Error details can be investigated using the LMT function or the PGE function.
Applicable boards: MPG-2314

```
100  MOVS X_A 10000
110  WAIT RR(X_A)=0
120  IF RR(X_E) != THEN          /* Confirming error status
130    PRINT "ERROR STOP"
140  ELSE
150    PRINT "NORMAL STOP"
160  END_IF
170  PRX RR(X_E)
```

YIN0

Pulse generation — Reserved constant

■Format

YIN0

■Usage

HPT(YIN0)

■Function

HPT input specification

■Explanation

Applicable boards: MPG-2314
YIN0 is specified to the HPT input port.
Related: HOME
see also XIN0

YIN1

Pulse generation | Reserved constant

■Format
YIN1

■Usage
HPT(YIN1)

■Function
HPT input specification

■Explanation
Applicable boards: MPG-2314
YIN1 is specified to the HPT input port.
Related: HOME
see also XIN1

Y_A

Pulse generation | Reserved constant

■Format
Y_A

■Usage
RMVS Y_A 1000

■Function
Y-axis specification

■Explanation
Applicable boards: MPG-2314/2541
This is a command for axis specification in the PG commands such as RMVS.
see also X_A

Y_C

Pulse generation | Reserved constant

■Format
Y_C

■Usage
stps Y_C 1000

■Function
Counter specification

■Explanation
Applicable boards: MPG-2314
The Y counter is specified.
see also X_C

Y_E

Pulse generation — Reserved constant

■Format
Y_E

■Usage
RR(Y_E)

■Function
Y-axis error specification

■Explanation
This is used as an argument of the RR() function to examine the presence/absence of a Y-axis error after a move.
If the value is not 0, it indicates a specific cause of error has occurred.
The details of the error can be investigated using the LMT function or the PGE function.
Applicable boards: MPG-2314
see also X_E

ZIN0

Pulse generation — Reserved constant

■Format
ZIN0

■Usage
HPT(ZIN0)

■Function
HPT input specification

■Explanation
Applicable boards: MPG-2314
ZIN0 is specified to the HPT input port.
Related: HOME
see also XIN0

ZIN1

Pulse generation — Reserved constant

■Format
ZIN1

■Usage
HPT(ZIN1)

■Function
HPT input specification

■Explanation
Applicable boards: MPG-2314
ZIN1 is specified to the HPT input port.
Related: HOME
see also XIN1

Z_A

Pulse generation — Reserved constant

■Format
Z_A

■Usage
RMVS Z_A 1000

■Function
Z-axis specification

■Explanation
Applicable boards: MPG-2314/2541
This is a command for axis specification in the PG commands such as RMVS.
see also X_A

Z_C

Pulse generation — Reserved constant

■Format
Z_C

■Usage
stps Z_C 1000

■Function
Counter specification

■Explanation
Applicable boards: MPG-2314
The Z counter is specified.
see also X_C

Z_E

Pulse generation — Reserved constant

■Format
Z_E

■Usage
RR(Z_E)

■Function
Z-axis error specification

■Explanation
This is used as an RR() function argument to examine the presence/absence of a Z-axis error after a move.
A value of other than 0 indicates that a specific cause of error has occurred.
Error details can be investigated using the LMT function or the PGE function.
Applicable boards: MPG-2314
see also X_E

_VAR

Arithmetic operation Command

■Format

_VAR arg1 [arg2 ..]

■Usage

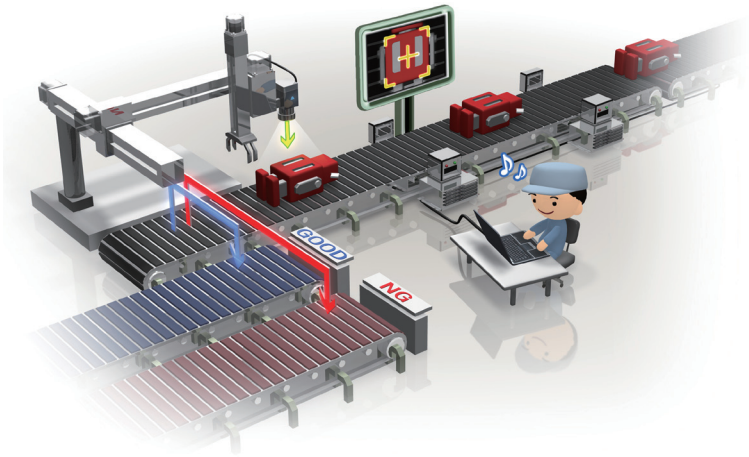
*TASK
VAR vala valb_

■Function

Extracting arguments given by GOSUB or RETURN.

■Explanation

The GOSUB statement can have a subroutine executed with arguments given.
_VAR extracts those arguments and have them substitute for specified variables.
_VAR can also extract arguments of the RETURN statement.


GOOD
NG