Chapter 4: Basics of Control Programs

In BL/1 various kinds of commands are prepared, dealing with each of the control actuators, and learning them all at once would be a huge undertaking. Here, several representative commands are introduced according to the control contents. It is hoped that a rough programming overview can be understood.

* Concerning the actual usage of commands, Command Reference should be referred to. As to practical application examples and writing programs, an MPC-2000 Tutorial is provided as a separate volume. Also available is a training kit XY03 according to its content.

The tutorial covers basic programs on XY control, touch panel interface, CUnet operation, and communication based on the training kit equipment.

4-1 I/O Control

ON/OFF

ON/OFF control in BL/1 is performed by the ON/OFF command. For example, ON/OFF repetition at 0.1-second intervals is described as follows. DO~LOOP is a control statement which indicates a return immediately after DO when LOOP is encountered.

```
DO
    ON 1
    TIME 100
    OFF 1
    TIME 100
LOOP
```

Sensor/input logic detection

Next, it is made to be a program which detects a sensor and turns ON/OFF once if detected.

```
DO
    WAIT SW(193)==1
    ON 1
    TIME 100
    OFF 1
    TIME 100
    WAIT SW(193)==0
LOOP
```

First WAIT SW(193)==1 means waiting for a sensor connected to an input port 193 to turn ON. WAIT SW(193)==0 confirms that the sensor turns OFF. Thereby, ON/OFF is repeated only for a change.

Condition/logical operations

In I/O controls, complicated logical operations may be performed. For example, although there were only conditions of SW(193) in the earlier example, a condition that SW(192) is also ON can be added as follows.
(SW(193)&SW(192)) is an AND operation of the values of SW(192) and SW(193). Therefore, unless both of those values are 1, the value of (SW(193)&SW(192)) does not become 1. This becomes a condition in which both of them are ON.

```
DO
    WAIT (SW(193)&SW(192))==1
    ON 1
    TIME 100
    OFF 1
    TIME 100
    WAIT SW(193)==0
  LOOP
```

Prepared for SW function is a separate @SW() function having an inverse value.

```
WAIT (SW(193)&@SW(192))==1
```

In this case, because @SW(192) has a reverse logic, it becomes 1 at OFF. Therefore, this example holds true in an AND condition wherein 193 is ON and 192 is OFF.

This kind of logical formula is used in IF and WHILE statements other than a WAIT statement, which has a positive logic when the values of all the formulae become 1. Therefore, WAIT SW(192)==1 and WAIT SW(192) wait for the same timing. Because a comparison operator == takes 1 when the compared results are equal and 0 when they are not, in SW(192)==1, the value of SW becomes 1, and comparison with 1 also takes the value 1. Thereby, complicated logical conditions of SW can be simply described.

```
IF SW(192)|SW(193)|SW(194)|flag THEN        Holds true when either 192, 193, or 194 is ON,
                                            or a variable flag becomes 1.
IF (SW(192)&SW(193))|SW(194) THEN           Both 192 and 193 are ON, or 194 is ON.
IF (SW(192)&SW(193))|@SW(194) THEN          Both 192 and 193 are ON, or 194 is OFF.
```

Time out processing

For time out processing, a timer (down-counting variable) is used. When a positive number is given to a timer, it decrements every 0.1 second and stops at 0. Time out processing containing a time out is described as follows.

```
timer_=1000
WAIT (SW(192)==1) | timer_==0
IF timer_==0 THEN : GOTO *TMOUT : END_IF
```

In order to refer to or modify timer_ variable from an external task, the TIMER() function is effective.

Character string processing

Main commands

string$	Attaching $ at the end makes it a character string variable.
FORMAT,STR$,HEX$,CHR$	Format, DEC → character string, HEX → character string, CODE → character string
VAL,ASC,HEX	Character string → numerical value conversion
STRCPY,PTR$	Copying
SERCH,SERCH$	Search
ptr_	Character string pointer

[Examples of command use]

1) Character string variables, combining

```
A$="" : B$="" : C$=""        /* Character string variable initialization
A$="2007/"                   /* Character string substitution
B$="11/15"                   /* Character string substitution
C$=A$+B$                     /* Character string combining
PRINT C$                     /* Display
```

```
*Result
2007/11/15
```

2) DEC → character string conversion (without format)

```
D=20071115                    /* Numerical value
FORMAT ""                     /* Character string format initialization
D$=STR$(D)                    /* Numerical value → character string conversion
PRINT D$                      /* Display

*Result
20071115
```

3) DEC→charactor string conversion (with format)

```
D=11152007                    /* Numerical value
  FORMAT "00/00th,0000"       /* Character string format specification
D$=STR$(D)                    /* Numerical value → character string conversion
PRINT D$                      /* Display

*Result
11/15th,2007
```

4) Character string conversion (with format)

```
D=&H20071115                  /* Numerical value (hexadecimal)
FORMAT "0000/00/00"           /* Character string format specification
D$=HEX$(D)                    /* Hexadecimal value → character string conversion
PRINT D$                      /* Display

*Result
2007/11/15
```

5) Example of reading the internal clock

```
FORMAT "0000/00/00  "         /* Set the character string format
DT$=HEX$(DATE(0))             /* Obtain a date character string
FORMAT "00:00:00"             /* Set a character string format
TM$=HEX$(TIME(0))             /* Obtain the time character string
PRINT DT$ TM$

*Result
2007/11/15  12:34:19
```

6) CODE→character conversion

```
A$=CHR$(&H41)+CHR$(&H43)+CHR$(&H43)+CHR$(&H45)+CHR$(&H4C)
PR A$

*Result
ACCEL
```

7) Character string → DEC conversion

```
A$="NOV15,2007"
A=VAL(A$)                     /* Obtain the first numerical character string.
PRINT A

*Result
15
```

8) Character string → CODE conversion

```
A$="NOV15,2007"
A=ASC(A$)                     /* Obtain the code for the first character.
PRX A
```

```
*Result
0000004E                          /* &H4E='N'
```

9) Character string → HEX conversion

```
 A$="E07F"                        /* Character string readable as a hexadecimal number
 A=HEX(A$)                        /* Convert into a numerical value.
 PRX A                            /* Hexadecimal display
 PRINT A                          /* Decimal display

*Result
0000E07F
57471
```

10) Character string copying (copying as is)

```
 A$="NOV15,2007"
 B$=A$                            /* Copy A$ to B$
 PR B$

* Result
NOV15,2007
```

11) Character string partial copying

```
 A$="NOV15,2007"
 STRCPY A$ B$ 3       /* Copy character No. 3 and later of A$ to B$ (counting the first character of A$ as 0.)
 PR B$

*Result
15,2007
```

12) Partial copying using a pointer

```
 FORMAT ""                        /*Clear the character string format setting.
 TT$=HEX$(TIME(0))                /* Obtain the current time.
 ptr_=TT$                         /* Obtain the character string position.
 ptr_=ptr_+2                      /* Advance the pointer by 2.
 HH$=PTR$(2)                      /* Cut out two characters at the pointer position and substitute it for HH$.
 ptr_=ptr_+2
 MM$=PTR$(2)                      /* Cut out two characters at the pointer position and substitute it for MM$.
 ptr_=ptr_+2
 SS$=PTR$(2)                      /* Cut out two characters at the pointer position and substitute it for SS$.
 CL$=HH$+":"+MM$+":"+SS$          /* Combine character strings.
 PR TT$ "->" CL$                  /*  TT$: Original character string, CL$: Synthesized character string.

*Result
00090835 -> 09:08:35
```

13) Search and partial copying

```
 a$="DATA X=AB0.4 Y=CD45 TEMP=DE55" /* Original character string
 SERCH a$ "X="                    /* Search for “X=” in a$.  The result enters to the pointer ptr_.
 b$=PTR$(5)                       /* Copy five characters starting at the ptr_ position to b$.
 ptr_=SERCH$("Y=")                /* Search for “Y=” frm the ptr_ position and enter the result to ptr_.
 c$=PTR$(5)                       /* Copy five characters starting at the ptr_ position to c$.
 ptr_=SERCH$("TEMP=")             /* Search for “TEMP=” frm the ptr_ position and enter the result to ptr_.
 d$=PTR$(4)                       /* Copy four characters starting at the ptr_ position to d$.
 PRINT b$ c$ d$

*Result
AB0.4 CD45 DE55
```

4-2 Touch Panel Connection

MEWNET protocol

In MPC-2000, a touch panel or display compatible with MEWNET can be connected to each serial port.

Although MPC-2000 provides only an RS-232 serial port, expanded serial board MRS-MCOM also provides an RS-422. The protocol is MEWNET only. MEWNET is a memory link protocol for FA of Matsushita Electric Works.

Although MEWNET protocol provides a very large number of procedures corresponding to the complicated contact functions of PLC, MPC-2000 assumes the memory area as the DT attribute and the I/O area as the R attribute and deals with only protocols related to these two. Because panel programs containing the other attributes and other company's panels claimed to be compatible with MEWNET may not be connectable, checking the connection in advance is desirable. To date, the following touch panels are confirmed to be connectable.

Panasonic Electric Works	GT series (such as AIGT0030 and AIGT2032)
Digital	GP-2000/3000 series (such as AGP-3300)
Mitsubishi	GOT series (GT-10XX)
Keyence	VT3 series (VT3-Q5M, VT3-W4T)
SAMKOON	SA series (SA-3.5A)

In order to start MEWNET, the following one line should be added to the top of a program. Once this command is executed, a touch panel is linked regardless of the program execution state.

Once linked, data are shared, and displaying data on the touch panel or setting data from the touch panel can be performed without being conscious of communication.

```
MEWNET 38400 1
```

The first argument 38400 indicates the baud rate. From the respect of reaction speed, 38400 is recommended.

The next argument is the CH number of the serial port used. (Character format is 8 bit nonparity.)

Touch panel communication is assigned one task, determined by the serial port CH number. In addition, some touch panels have parity fixed to odd, in which case one of the following constants specifying the character format is added.

bit7 odd parity
bit7 even parity
bit8 odd parity
bit8 even parity

Below is a case of 3800 bps bit7 odd parity.

```
MEWNET 38400 1  B7O
```

Task number used becomes 32 – CH number.

Therefore, if CH1 is specified (CH1 provided by MPC-2000 and 2100), Task 31 is assigned to touch panel communication. In this case, if Task 31 is used or carelessly quit in a program, touch panel communication is damaged.

Memory allocation

Memory sharing with a touch panel is performed by the MPC side using a reserved array MBK(). MBK is a word-type array, and 8192 of them are secured. Among them, 0~7835 are used as word data. In 7836~7899 the system constantly writes program numbers executed by each task.

The area of MBK(0)~MBK(7899) corresponds to DT0~DT7899 in the touch panel.

[Usage example]

```
IF MBK(100)==10 THEN
MBK(200)=1000
```

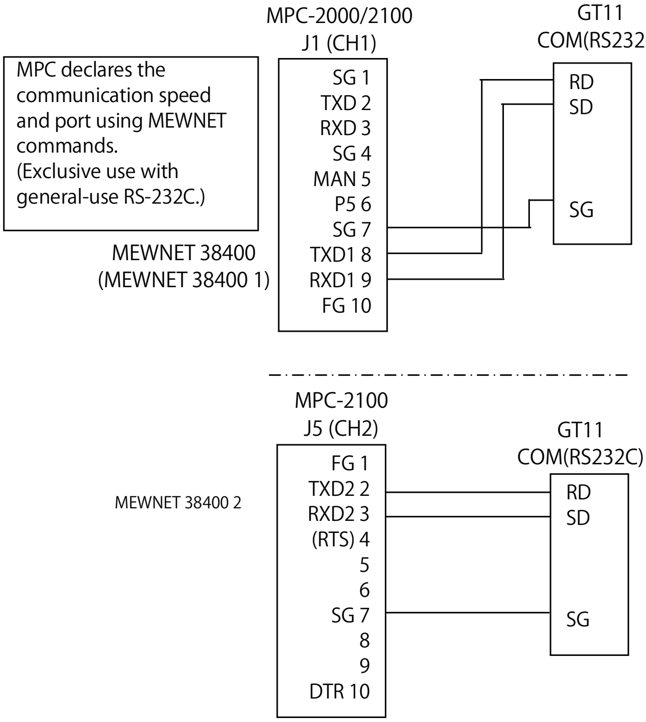
7900~7999 is made an I/O area, which can be operated with ON/OFF command.
ON 70000 → Turn Bank 0 Port 0 ON.
OFF 70115 → Turn Bank 0 Port 15 OFF.
In the touch panel this area becomes an I/O area specified with “R”.

[MBK area]

0 ～ 7835	Word data (corresponding to DT)	In general, 0~9 is used as a system area. Modifying: S-MBK n m or MBK(m)=n Referring: MK(m)
7868 ～ 7899(Wrd) 7836 ～ 7899(Lng)	Program number	Program numbers are updated in real time, and can be referred to and displayed in the touch panel side. Rem) Executing S_MBK LONG_PRG converts into the Lng type.
7900 ～ 7999	I/O area (Corresponding to R)	Area of 16 bits each (XX) in 0~100 bank (YY). ON 7YYXX,OFF 7YYXX OUT 1000 7YY00~Lng SW(7YYXX),IN(7YY00)

Touch panel connection examples

■ Example of connecting with Panasonic Electric Work GT11



MPC declares the communication speed and port using MEWNET commands.
(Exclusive use with general-use RS-232C.)

- PLC model
  “Panasonic Electric Work MEWNET-FP series”

- Example of communication setting
  COM Port

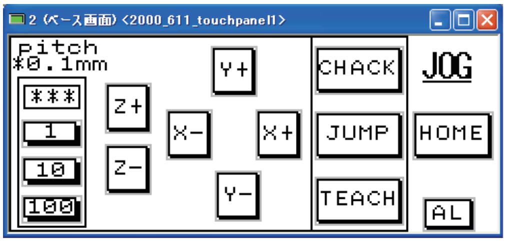
Transfer speed:	38400
Data length:	8
Stop bit:	1
Parity:	Disabled

Communication error processing
Retries: 0 times 0 seconds
Display error code: Yes (not retained)

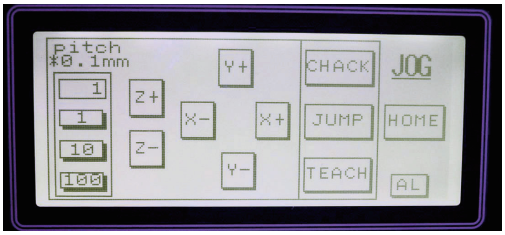
Send delay time: 0 ms

■ Example of screen design in Panasonic Electric Works GT11

[Schematic view]



[Actual photograph]



* Colors in the photo are altered.

[Details of arranged parts]

Display	Part	Basic setting	MPC command example	
X+	Switch part	Momentary R200	SW(72000)	IN (72000~Wrd) (2 byte read)
X-	Switch part	Momentary R201	SW(72001)	
Y+	Switch part	Momentary R202	SW(72002)	
Y-	Switch part	Momentary R203	SW(72003)	
Z+	Switch part	Momentary R204	SW(72004)	
Z-	Switch part	Momentary R205	SW(72005)	
HOME	Switch part	Momentary R206	SW(72006)	
CHACK	Switch part	Momentary R207	SW(72007)	
JUMP	Switch part	Momentary R208	SW(72008)	
TEACH	Switch part	Momentary R209	SW(72009)	
AL	Switch part	Momentary R20A	SW(72010)	
1	Functional switch part	Value set output destination DT101 value 1		
10	Functional switch part	Value set output destination DT101 value 10		
100	Functional switch part	Value set output destination DT101 value 100		
***	Data part	Reference device DT101	MBK(101)	

[Example program]

```
MEWNET 38400                                        /* Use RS-232C CH1

DO                                                  /* Loop to wait for SW to be pressed
  GT=IN(72000~Wrd)                                  /* Read in 2 byte
  IF GT<>0 THEN : BREAK : END_IF                    /* Exit the loop when any key is pressed.
  SWAP
LOOP

SELECT_CASE GT
  CASE &H01 : AX=X_A : MD=1  : GOSUB  *JOG_MV       /* X+ SW
  CASE &H02 : AX=X_A : MD=-1 : GOSUB  *JOG_MV       /* X- SW
  CASE &H04 : AX=Y_A : MD=1  : GOSUB  *JOG_MV       /* Y+ SW
  CASE &H08 : AX=Y_A : MD=-1 : GOSUB  *JOG_MV       /* Y- SW
  CASE &H10 : AX=Z_A : MD=1  : GOSUB  *JOG_MV       /* Z+ SW
  CASE &H20 : AX=Z_A : MD=-1 : GOSUB  *JOG_MV       /* Z- SW
  CASE &H40 : GOSUB  *JOG_HOME                      /* HOME SW
  CASE &H80 : GOSUB  *JOG_CHACK                     /* CHACK SW
  CASE &H100 : GOSUB  *JOG_JUMP                     /* JUMP SW
  CASE &H200 : GOSUB  *JOG_TEACH                    /* TEACH SW
  CASE &H400 : GOSUB  *ALIGN                        /* AL SW
  CASE_ELSE : PRINT "?"
END_SELECT
WAIT IN(72000~Wrd)==0
```

4-3 Time Management

MPC-2000,2100 and 2200 have a built-in RTC, which provides the date and time. The built-in RTC is RTC-7301 manufactured by Epson Toyocom and has a monthly error of about 1 minute. MPC-1000 and N816 do not have this function.

Setting

Using the calendar IC requires an initial setting. SET_RTC command is used for the setting.

```
SET_RTC 2009 4 1       ...Set to April 1, 2009
SET_RTC 12 2 0         ...Set to 12 hours 2 minutes 0 second.
```

Checking the set date is performed with date(0) and time(0) functions.

```
#prx date(0)
20090401
#prx time(0)
00120204
#
```

Time detection

In order to detect a specified date and time, numerical value comparisons are performed as follows. They are specified as hexadecimal constants.

```
IF   TIME(0)==&H130500 THEN    (13 hours 5 minutes and 0 second)
IF   DATE(0)==&H20090401 THEN             (April 1, 2009)
```

In the following example, 5 seconds and 15 seconds are detected every minute. By enabling only necessary digits, complicated time detection such as every hour and every day are possible.

```
DO
     WAIT   &HFF&TIME(0)==&h0005
   PRINT   "time_05"
   WAIT   &HFF&TIME(0)==&h0015
   PRINT   "time_15"
LOOP
```

Date and time character strings

As data and time character strings, DATE$() and TIME$() are used. Numerical values 0~2 specify different formats.

```
10      FORMAT   "00000000"
20     a$=DATE$(0)
30      FORMAT   "000000"
40     a$=a$+TIME$(0)
60      PRINT   a$
#run

 2009090900141849
#pr time$(1)
 14:19:02
#pr date$(1)
  9/ 9/2009
#
```

4-4 Axis Control

Pulse-generation boards, MPG-2541 and MPG-2314 are available. MPG-2541 is for simple positioning, not including interpolation or stopping. On the other hand, MPG-2314 can deal with complicated processes such as linear/circular interpolation and sensor stop. Up to 10 MPG-2314 boards and up to 8 MPG-2541 boards can be used in expansion, and the software can accommodate 18 boards × 4 axes. (Because the rack has up to 16 slots, the number of slots is limited.)

PG assignment

Which PG to use is set with PG command. MPG-2314 deals with DSW values of 0~9.
MPG-2541 uses DSW values with 10 added. (PG 10 for example.)

Acceleration and speed

ACCEL, FEED, and SPEED commands are available.
ACCEL determines the maximum speed, minimum speed, and acceleration. Presence/ absence of an S-curve acceleration/deceleration is also specified here. FEED command specifies a speed by providing an argument of 1~100 (%) in terms of m % of the maximum speed. On the other hand, SPEED command specifies it in pps. It specifies a speed as m pps withint the range of the maximum speed determined by ACCEL. (The resolution becomes 1/8192 pps of the maximum speed.)

Pulse generation commands

The following commands are available for actual pulse generation. They are selectively used according to the purpose.

Command	Purpose	Description
MOVS	Positioning	Acceleration/deceleration rate pulse generation, absolute position specification, without interpolation.
RMVS	Positioning	Acceleration/deceleration rate pulse generation, relative position specification, without interpolation.
MOVL	XY stage, etc.	Acceleration/deceleration rate pulse generation, absolute position specification, with linear interpolation.
RMVL	XY stage, etc.	Acceleration/deceleration rate pulse generation, absolute position specification, with linear interpolation.
MOVT	Painting robot NC	Track control continuous pulse generation, absolute position specification, circular/linear interpolation
RMVT	Painting robot NC	Track control continuous pulse generation, absolute position specification, circular/linear interpolation
RMVC	Spindle, etc.	Infinite pulse generation
STOP	General use	Command to stop pulse.
HOME	Auxiliary command	Origin-return macro command

Setting and errors

Positioning provides an interlock which is necessary for detecting various kinds of abnormal states and safety. Although MPG-2541 provides only limit input, MPG-2314 provides servo driver error input and detection stop input other than limit input.

Command/function	Purpose	Description
INCHK	Maintenance	MPG inputs, display
INSET	Error input setting	LMT logic setting, ALM
STOP	Stop condition input	Defining special origin return and stop condition
PGE( )	Stop cause – read out	After PG stop:EMG,ALM,LMTn,LMTp|IN3,IN2,IN1,IN0
LMT( )	Error cause read out	Constant reference:EMG,ALM,LMTp,LMTn,SLMTp,SLMTn
HPT( )	IN0~3 input	Reading out origin input, etc.
RR( )	Operation state	Detecting if PG is in operation.

Initial setting * Sample program below were created by MPG-2314.

Major commands

PG	Select PG
ACCEL,FEED	Set the speed
INSET	Set the input

Pulses cannot be normally generated by simply mounting MPG-2314. Initial setting is required.First, MPG is assigned to a task using PG command. Next, initial setting is made with ACCEL, etc.Although direct commands may also be used, they should eventually be reflected onto the program.

[Setting example]

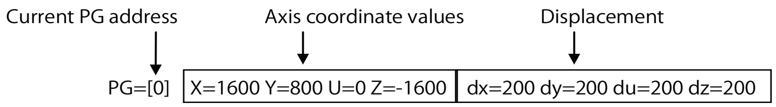
```
PG 0                           /* MPG-2314 board selection.  MPG-2314 is address-set using DSW1.
ACCEL ALL_A 30000              /* Setting the maximum speed and acceleration/deceleration.
FEED ALL_A 100                 /* Setting the speed used as 100 %.
INSET ALL_A ALM_ON|INP_OFF     /* Setting the input function. Alarm is enabled at ON, and INPOS is enabled at OFF.
CLRPOS                         /* Current point is 0-cleared.
```

Operation check in the teaching mode

Major commands:

PG	PG selection
T(TEACH)	Teaching mode
PLS	List the point data

The easiest way to check pulse output is the teaching mode. Teaching mode can be entered by typing T<Enter> in FTMW screen.



```
PG=[0] X=1600 Y=800 U=0 Z=-1600 dx=200 dy=200 du=200 dz=200
```

Displacement (number of pulse outputs at a time) is switched using 0~3 keys. This value can be changed with SET command.

Initial value 0: 200 pulses / 1: 400 pulses / 2: 600 pulses / 3: 800 pulses

Individual axes operate with X, x, Y, y, U, u, Z, and z keys. P key is used for inputting a point number. The point number to teach should be input. Pressing Q key exits the teaching mode.

Setting the maximum speed and acceleration/deceleration

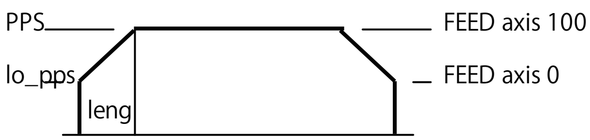
Major commands:

ACCEL	Set the maximum speed, acceleration/deceleration, and minimum speed
FEED	Specify the speed

[Format]

```
ACCEL [axis] PPS [leng lo_pps]
axis: Axis selection reserved constant
PPS: Maximum speed
Leng: Number of pulses in the acceleration/deceleration region
Lo_pps: Startup speed (Minimum speed)

FEED [axis] n
[axis]: Axis specification reserved constant
N: Speed specification 100 (Maximum speed) ~ 0 (Minimum speed)
```

Relationship between ACCEL and FEED



MPG-2314 input check

The input port of MPG-2314 can be checked using the INCHK command.

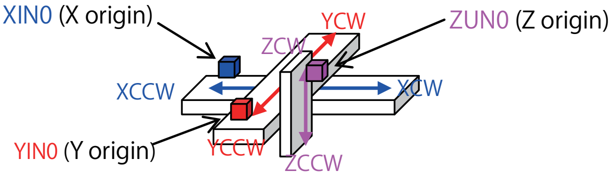
```
#PG 0                                                /*PG 0 assignment (described later)
#INCHK                                               /* MPG input check
 MPG-2314
 X=+LMT:off-LMT:off ALM:off INP:off IN0:on  IN1:off  /* IN0= Origin LS is ON
 Y=+LMT:off-LMT:off ALM:off INP:off IN0:on  IN1:off  /* IN0= Origin LS is ON
 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
#                                                    /* Stop scanning with any key.
```

Origin return

Major commands:

SHOM	Origin input setting
HOME	Origin return operation
HPT	Read in the origin input state

Each axis of XY03 has one limit switch installed, which is connected to the origin input of MPG-2314.



[Subroutine examples]

1) Example of Z single axis origin return subroutine

```
*Z_HOME
 PG 0
 ACCEL Z_A 10000 100 100
   /* Speed setting.  Maximu speed 10 KPPS, acceleration/deceleration region 100 pulses, minimum speed 100 PPS
 IF HPT(ZIN0)<>0 THEN          /* If XIN0 is ON, move backwards.
   RMVS Z_A -5000              /* Move by 1000 pulses in the CCW direction.
   WAIT RR(Z_A)==0             /* Wait for the operation to be complete.
 END_IF
 SHOM Z_A IN0_ON               /* Origin return setting.  Move until ZIN0 turns ON.
 TMOUT 10000                   /* Time out at 10 seconds.
 HOME 0 0 0 50000              /* 50K pulses in the Z-axis CW direction.
 WAIT RR(Z_A)==0               /* Wait for the operation to be complete.
 IF Z(0)<>0 THEN               /* If the coordinate is not 0 after the operation, time out.
   PRINT "Z TIME OUT"
 ELSE                          /* If the coordinate is 0 after the operation, HOME completion.
   PRINT "Z HOME"
 END_IF
 RETURN
```

2) Example of XY 2-axis simultaneous origin return subroutine

```
*XY_HOME
 PG 0
 ACCEL X_A|Y_A 10000 100 100/* Speed
 FEED X_A|Y_A 100
 RMVL 5000 5000 0 0                /*  X, Y forced retreat to CW (LS check omitted)
 WAIT RR(X_A|Y_A)==0               /* Wait for the operation to be complete.
 SHOM X_A|Y_A IN0_ON               /* Operate X and Y axes until their IN0 turn ON.
 TMOUT 10000                       /* Time out at 10 seconds
 HOME -100000 -100000 0 0          /* XY-axis simultaneous operation
 WAIT RR(X_A|Y_A)==0               /* Wait for the operation to be complete.
 RMVL 2000 2000 0 0                /* Offset according to necessity (Electric origin).
 WAIT RR(X_A|Y_A)==0
 STPS X_A|Y_A 0                    /* Current positions of X and Y axes are set to ‘0’.
 PRINT "XY HOME"
 RETURN
```

3) Main routine calling a subroutine

```
GOSUB *Z_HOME   /* Z axis is returned to the origin (lifted) first to prevent interference between hand and work.
GOSUB *XY_HOME
END
```

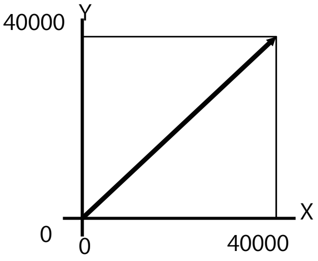
* These subroutines are also used in samples described later.

Absolute coordinate movement

Major commands

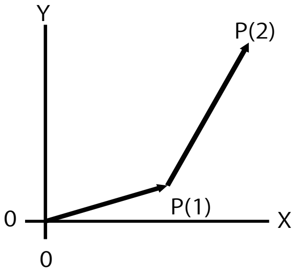
MOVL	Linear-interpolation move
MOVS	Single-axis move

1) Movement is performed with coordinates specified with constants or variables. MOVL performs linear interpolation.



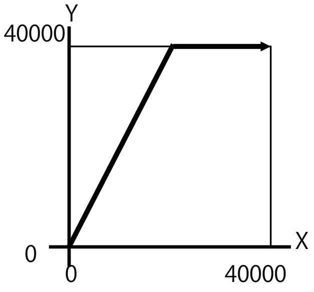
```
GOSUB *Z_HOME        /* Origin return subroutine described earlier
GOSUB *XY_HOME

ACCEL ALL_A 30000 3000 1000    /* Set speed and acceleration/deceleration.
FEED ALL_A 100                 /* Operate at the maximum speed.
MOVL 40000 40000 VOID VOID     /* XY-axis absolute coordinate movement
WAIT RR(ALL_A)==0              /* Wait for the operation to be complete.

END
```

2) Movement is performed with taught points specified. The points can be set in the teaching mode or a program. Point numbers can also be specified with variables.



```
GOSUB *Z_HOME
GOSUB *XY_HOME

ACCEL ALL_A 30000 3000 1000 /* Set speed and acceleration/deceleration.
FEED ALL_A 100              /* Operate at the maximum speed.
MOVL P(1)                   /* Linear-interpolation move to point P(1).
WAIT RR(ALL_A)==0           /* Wait for the operation to be complete.
PNO=2                       /* Variable specification
MOVL P(PNO)                 /* Linear-interpolation move to point P(2).
WAIT RR(ALL_A)==0

END
```

3) Although reached points are the same as in 1), MOVS does not perform linear interpolation. It can apply to preventing the vibration of a mechanism using a stepping motor and setting different speeds among different axes in a mechanism combining step and servo for example.



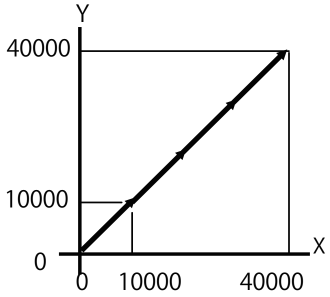
```
GOSUB *Z_HOME
GOSUB *XY_HOME

ACCEL X_A 15000 2000 1000    /* Set X-axis speed and acceleration/deceleration.
ACCEL Y_A 30000 3000 1000    /* Set Y-axis speed and acceleration/deceleration.
FEED ALL_A 100               /* Operate all axes at the maximum speed.
MOVS 40000 40000 VOID VOID /* Single-axis operation for X and Y.
WAIT RR(ALL_A)==0

END
```

Relative coordinate movement

Major commands

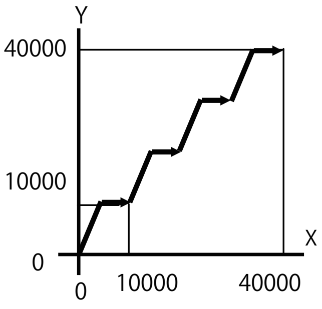
RMVL	Linear-interpolation move
RMVS	Single-axis move

1) Movement is performed with coordinates specified with constants or variables.
RMVL performs linear interpolation.



```
GOSUB *Z_HOME
GOSUB *XY_HOME

ALL_A 30000 3000 1000        /* Set speed and acceleration/deceleration.
FEED ACCEL ALL_A 100         /* Operate at the maximum speed.

FOR I=1 TO 4                 /* Repeat 4 times.
   RMVL 10000 10000 0 0      /* XY linear-interpolation move
   WAIT RR(ALL_A)==0
NEXT I

END
```

2) Although reached points are the same as in 1), RMVS does not perform linear interpolation.



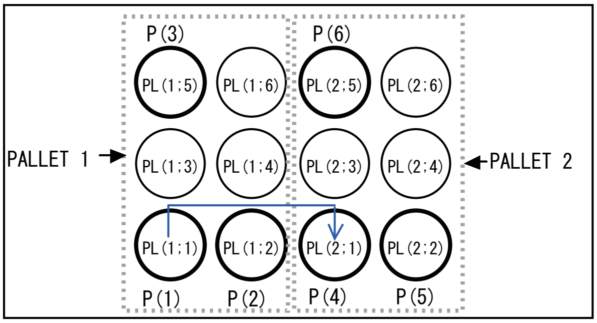
```
GOSUB *Z_HOME
GOSUB *XY_HOME

ACCEL X_A 15000 2000 1000    /* Set speed and acceleration/deceleration.
ACCEL Y_A 30000 3000 1000    /* Set speed and acceleration/deceleration.
FEED ALL_A 100               /* Operate at the maximum speed.

FOR I=1 TO 4                 /* Repeat 4 times.
   RMVS 10000 10000 0 0      /* XY simple move
   WAIT RR(ALL_A)==0
NEXT I

END
```

Palletization

Major commands

PALLET	Pallet declaration
PL	Work points

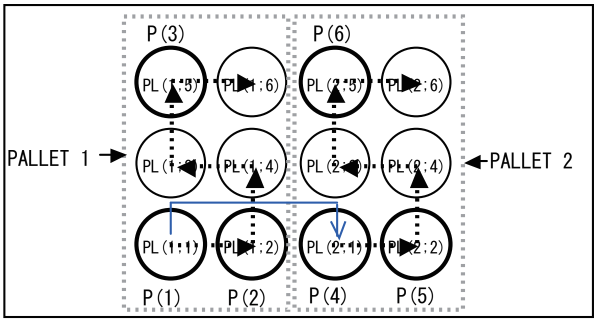
Used for moving between pallets. A work point PL on a pallet is calculated from three corner points and the numbers of rows and columns.



```
PALLET 1 P(1) P(2) P(3) 2 3          /* Pallet declaration
PALLET 2 P(4) P(5) P(6) 2 3

GOSUB *Z_HOME
GOSUB *XY_HOME
ACCEL ALL_A 30000 3000 1000          /* Set speed and acceleration/deceleration.
FEED ALL_A 100                       /* Operate at the maximum speed.

FOR M=1 TO 6                         /*Points in each PALLET*
 JUMP PL(1;M)                        /* Jump to point M of PALLET 1.
 WAIT RR(ALL_A)==0
 ON 14                               /* Close chuck.
 TIME 200
 JUMP PL(2;M)                        /* Jump to point M on PALLET 2.
 WAIT RR(ALL_A)==0
 OFF 14                              /* Open chuch.
NEXT M

END
```

When m of PL(n;m) is negative, ZIGZAG mode is entered. Moving distance between column becomes shorter.

```
FOR M=-1 TO -6 STEP -1      /* Line with * above.  Set the arguments negative.
```



* If four points are specified for PALLET, a distorted pallet can be dealt with.

Stopping

Major commands

STOP	Stop pulse.
INSET	Set MPG-2314 input.

■ Stopping by software

Input is monitored after a move is started, and once a switch turns on, STOP command is issued.

```
GOSUB *Z_HOME
GOSUB *XY_HOME
ACCEL ALL_A 30000 3000 1000        /* Set speed and acceleration/deceleration.
FEED ALL_A 100                     /* Operate at the maximum speed.

MOVL 40000 40000 VOID VOID         /* XY linear interpolation
WAIT SW(194)==1                    /* Wait for red SW to turn on.
STOP ALL_A STP_I                   /* Abrupt stop.  If STP_D, slow down to stop.
WAIT RR(ALL_A)==0

END
```

■ Stopping by hardware

Described below is stopping by utilizing an alarm input of MPG-2314. Stop conditions are set before moving. If X-axis alarm (J6 connector pin 13) or Y-axis alarm (pin 14 of the same) turns on while moving, both axes stop immediately.

```
GOSUB *Z_HOME
GOSUB *XY_HOME
ACCEL ALL_A 10000 3000 1000        /* Set speed and acceleration/deceleration.
FEED ALL_A 100                     /* Operate at the maximum speed.

INSET X_A|Y_A ALM_ON               /* Set alarm input.

MOVL 40000 40000 VOID VOID         /* XY linear interpolation
WAIT RR(ALL_A)==0

END
```

* INCHK command for checking the MPG-2314 input of ALM and the like.

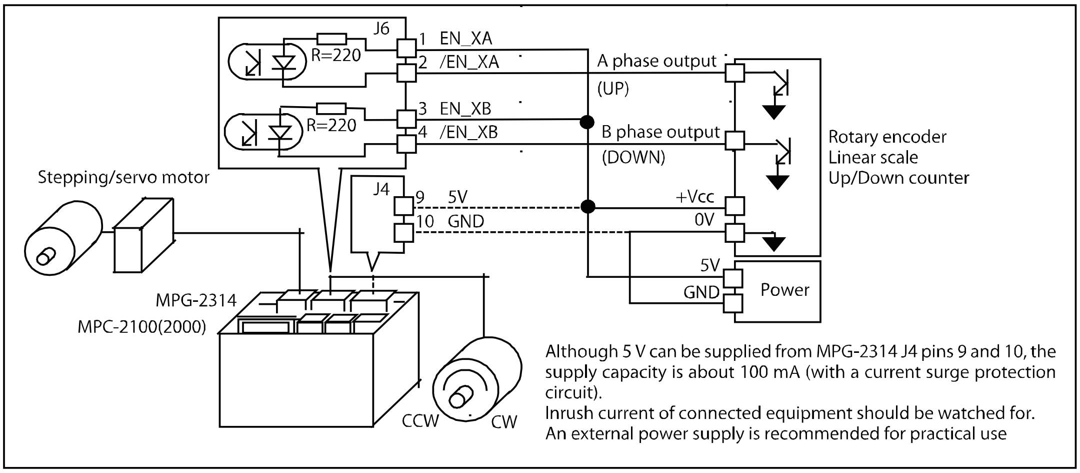
Encoder and counter input

MPG-2314 provides 2-axis encoder input by default. Two-phase or Up/Down can be selected by a command. (Two axes can be added as an option.)

Major commands

INSET X_A PHASE1	Two-phase encoder input, multiplication 1.
INSET X_A UP_DWN	Switch to Up/Down input. (Default is two-phase.)
INSET X_A CMP_CNT	Mode to compare COMP register and the count value.
CLRPOS –1	Clear X, Y, U, and Z counters.
STPS X_C n	Set X counter to n.
X(-1)	Return the X counter value.
X(-2,1)	Return the X counter value and clear the counter.
CMP_C(X_A)	Return the result of comparing COMP register and X counter.

[Connection example (X counter)]



Although 5 V can be supplied from MPG-2314 J4 pins 9 and 10, the supply capacity is about 100 mA (with a current surge protection circuit).
Inrush current of connected equipment should be watched for.
An external power supply is recommended for practical use

When the rotary encoder is operated, it turns the output on/off at every 100 counts.

```
PG 0
INSET PHASE1                  /* 1000 counts per 1 rotation with multiplication 1.
CLRPOS -1                     /* Clear the counter.
OUT 0 0
DO
  NOW_XC=X(-1)                /* Read in the X counter value.
  IF (NOW_XC%100)==0 THEN     /* Divide by 100.
    OUT @SW(0) 0              /* Output inversion on/off
    PRINT NOW_XC SW(0)        /* Display
    WAIT NOW_XC<>X(-1)
  END_IF
LOOP

Execution result
 0 1
 100 0
 200 1
 200 0
 100 1
 0 0
 -100 1
```

(For other sample programs, see Application Note an2k-009.)

Concerning MPC-1000 pulse generation function

MPC-1000 has two sub CPUs built-in, each of which can be used as a pulse generator.
* This PG function requires about 0.1~0.2 seconds for command communication.
* Because this PG function uses the internal oscillator of PIC, the speed specification has about +/-2 % errors.
* The pulse width of pulse generation with acceleration/deceleration is fixed to 15 μsec.

[Occupied ports]
The pulse generators are named as PGA and PGB and occupy the following ports.

	Output ports	Input ports
PGA	ON 12 (CW pulse) ON 13 (CCW pulse)	SW(192+12) READY SW(192+13) for communication
PGB	ON 14 (CW pulse) ON 15 (CCW pulse)	SW(192+14) READY SW(192+15) for communication

[Enabling PG]

In order to enable PGA or PGB, execute the following.

```
ON PGA
ON PGB
```

Also, for disabling, execute the following.

```
OFF PGA
OFF PGB
```

In a disabled state, I/O are not occupied and can be used as I/O for control. In addition, ON/OFF also function as software reset of PG. When stopping pulse generation, OFF PGA and OFF PGB should be executed for each and an OFF time of 10 msec or longer should be secured.

[PG commands] PGX stands for either PGA or PGB.

Function	Command	Range	Note	READY
Pulse method	PGX "D" n	0 or 1	0: Default 2 PLS 1: Direction indication	
PWM	PGX "W" n	40~970	Also usable as DA.	
PPS specified pulse generation	PGX "G" pps	20~9000		
Setting the pulse rate	PGX "S" pps	20~9000		
Pulse number specified pulse generation	PGX "P" count	-8000000~ 8000000		○
Acceleration/deceleration table generation	PGX "A" pps	500~12000		○
Speed selection	PGX "F" n	10~0	n*10 %	
Acceleration/deceleration pulse generation, relative	PGX "R" count	-8000000~ 8000000		○
Acceleration/deceleration pulse generation, coordinate	PGX "M" count	-8000000~ 8000000		○
Clearing the current position	PGX "H" count		Setting the current position.	
Obtaining the current position	PGX "C"		V_PGA for PGA V_PGB for PGB	
Obtaining the version	PGX "V"	20091105 or later		

* "○" mark in the READY column indicates a command which requires waiting for the execution completion such as specified number of pulse generation.

[Usage 1] As pulse generation

```
*PGAPGB
 TIME   300
 ON   PGA PGB                          Enabling PG
 WAIT   SW(192+12)==1                  Confirming the enabling
 PGA   "V" : PRINT   V_PGA             Obtaining and displaying the version
 DO
  FOR   i=20 TO 6020 STEP 1000         Pulse rate is changed from 20 to 6020.
   PGA   "G" i                         (CW)
 TIME 100
  NEXT
  FOR   i=6020 TO 20 STEP -1000        Pulse rate is changed from 6020 to 20.
   PGA   "G" 0-i                       (CCW with a negative value)
   TIME 100
  NEXT
  TIME   100
  PGA   "G" 0                          Stop G command.
  PGA   "S" 2000                       Set a pulse rate.
  PGA   "P" 1600                       Generating 1600 pulses, CW (no acceleration/deceleration)
  WAIT   SW(192+12)
  PGA   "P" -1600                      READY=0 during pulse generation
  WAIT   SW(192+12)                    Generating 1600 pulses, CCW (without acceleration/deceleration)
```

[Usage 2] As position-control pulse generation

```
      PGA   "A" 9000                        Set the acceleration/deceleration table to 9000 pps/s
      WAIT   SW(192+12)                     Wait for the completion of table generation
      PGA   "H" 0                           Specify the current position.
 ‘
      FOR   j_=5 TO 10
      PGA   "F" j_                          Specify the speed.
      FOR   i_=1 TO 10
       PGA   "R" 800                        Relative pulse generation
       WAIT   SW(192+12)
      NEXT
      PGA   "M" 0                           Move to the 0 position.  Coordinate move
     NEXT
```

[Command explanations] PGX stands for either PGA or PGB.

Pulse method	PGX “D” n	Two pulse method of CW and CCW or direction indication is specified. To change to the direction indication, PGX “D” 1 should be executed.
PWM	PGX “W” n	PWM pulse generation, generating pulses in the CW side only. By default, it defines n μsec ON-time at 1 kpps pulse rate. When changing the pulse rate, PGX “S” pps should be executed first, then PGX “W” n. To stop it, OFF PGA (PGB) or PGX “W” 0 should be executed.
PPS specified pulse generation	PGX “G” pps	Constant speed pulse generation, generating pulses at a specified rate. It becomes CW with a positive value, and CCW with a negative value. After it is started, the speed can be changed. To stop it, OFF PGA (PGB) or PGX “G” 0 should be executed.
Pulse rate setting	PGX “S” pps	Pulse rates of PWM and pulse number specified pulse generation are determined.
Pulse number specified pulse generation	PGX “P” count	Constant-speed, specified pulse generation, generating pulses in the CW direction with a positive value and the CCW direction with a negative value.
Acceleration/ deceleration table generation	PGX “A” pps	Generates the speed table for pulse generation with acceleration/ deceleration. Acceleration distance is fixed by 1/10 of the specified pulse rate. Acceleration/deceleration speed is fixed in flash ROM. The number of rewriting flash ROM is deemed to be within 100 thousand times, which should be watched for. (If the arguments are the same, no rewriting is performed.)
Speed selection	PGX “F” n	The speed specified with A command is divided into ten stages, and n/10 speed specification I performed. Although modifying the acceleration/deceleration table takes time, speed change which does not take time.
Acceleration/ deceleration pulse generation, relative	PGX “R” count	Pulse generation with acceleration/deceleration, which is coordinate controlled and can be used along with M command and C command.
Acceleration/ deceleration pulse generation, coordinate	PGX “M” count	Pulse generation with acceleration/deceleration, which generates the differential pulse between the current position and the specified position. It is coordinate controlled and can be used along with R command and C command.
Clearing the current position	PGX “H” count	Current position specification. Setting count to 0 makes it the origin.
Obtaining the current position	PGX “C”	Results are returned to reserved variables V_PGA and V_PGB for PGA and PGB, respectively.
Obtaining the version	PGX “V”	Results are returned to reserved variables V_PGA and V_PGB for PGA and PGB, respectively.

4-5 Data Communication

RS-232/RS-485

MPC-2000 can handle 10-CH serial communication. Although the CPU board alone can only handle RS-232C, if MRS-MCOM is used, RS-422 and RS-485 communication can also be handled. Sufficient receiving communication interrupt buffer of 256 bytes is provided for each CH. * MPC-1000 can use CH1 as RS-485.

1) Configuration
For configuration, CNFG command is used as CNFG# 1 “38400b8pns1NONE” for example. It is compatible with various kinds of formats from 1200 bps to 38400 bps. For conducting RS-485 communication, the following should be executed.

```
CNFG#  5 RS485 "38400b8pns1NONE"
```

By providing a reserved constant RS485 as an argument, the communication direction can be automatically switched.

2) Sending
PRINT# command is used. Character strings should basically be used in a PRINT# statement. Although variables can also be used, the format cannot be regulated. Although “\n” (LF), “\r” (CR), and “\t” (TAB) may be used in character strings, CHR$() should be used for other control characters.

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

3) Receiving
INPUT# command is used. INPUT# statement can take only character strings as its arguments. After receiving them as character strings, the content can be analyzed using VAL function, GET_VAL and SERCH commands, and the like to obtain data.

4) Options
In a INPUT# or PRINT# statement, the number of received characters, time-out time, delimiter, and code may also be specified. In addition, COMPOWAY and STR_LEN are provided as special options. Although COMPOWAY is the protocol of a basic procedure regulated by OMRON, automatic sending and receiving of this format are supported. SRT_LEN is used for sending a character string including a null code.

RS-232C device connection example

Major commands

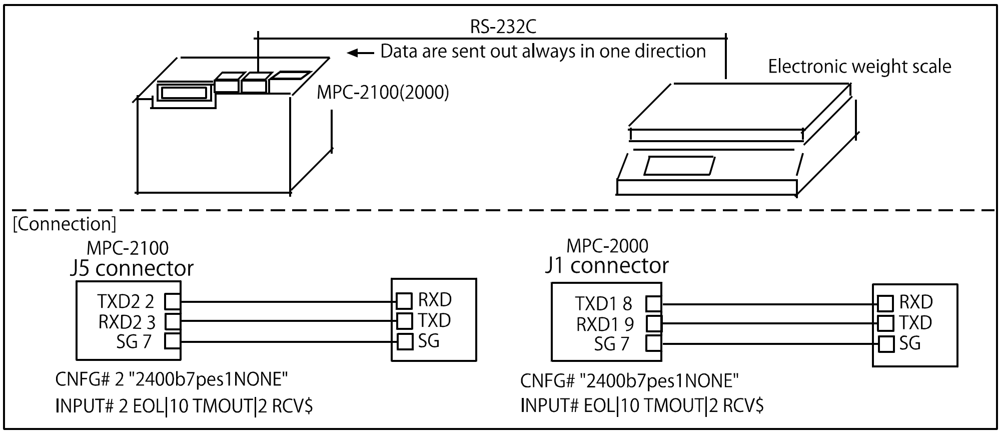
CNFG#	Communication configuration
PRINT#	Output
INPUT#	Input

[Example of extracting numerical value data from a received character string]

```
/* To execute this sample, TXD1 and RXD1 of CH1 should be short-circuited in a loop-back.
CNFG# 1 "9600b8pns1NONE"   /*Communication port initialization
FOR I=0 TO 20 STEP 2
  FORMAT "ABC0.0DEF\n"     /* Character string format
  SND$=STR$(I)             /* Creating a character string to be sent
  PRINT# 1 SND$            /*Sending
  INPUT# 1 RCV$            /* Receiving
  PRINT RCV$ VAL(RCV$) VAL(0)  /* Received character string, first numerical value, next numerical value.
NEXT I
```

```
*Result
ABC0.0DEF 0 0
ABC0.2DEF 0 2
ABC0.4DEF 0 4
```

[Example of connecting with an electronic weight scale]



```
CNFG# 2 "2400b7pes1NONE"
INPUT# 2 EOL|10 TMOUT|2 RCV$
```

```
CNFG# "2400b7pes1NONE"
INPUT# EOL|10 TMOUT|2 RCV$
```

- Character string data sent out from an electronic weight scale

Example) WT,+000000.0 g<CR><LF>
WT:Header character. WT = stable state, US = unstable state, OL = overloaded.
+:   Positive/negative sign. If negative, -.
000000.0:   Data. Fixed to 8 characters, the decimal point may change the position or be absent.
g: Unit.
Sending cycle: Asynchronous discharge at a little fewer than four times per second.
No control from the MPC side

- Sample program

```
 CNFG# 2 "2400b7pes1NONE"              /* Initialization
 FORMAT ""
 TOTAL_CNT=0                           /* Total count
 RETRY_CNT=0                           /* Retry count
 DO
*RETRY
  INPUT# 2 EOL|10 TMOUT|2 RCV$         /* Receive up to LF.  TMOUT 2 seconds.
  IF rse_<>0 THEN                 /* TMOUT processing.  rse_ is a reserved variable, always in lower case.
   RETRY_CNT=RETRY_CNT+1
   PRINT "tmout retry" RETRY_CNT rse_
   GOTO *RETRY
  END_IF
  'PR RCV$
  ptr_=RCV$ /* Pointer for character string RCV$.  ptr_ is a reserved variable, always in lower case.
  HEADER$=PTR$(2)                      /* Two characters at the top of RCV$ are copied to HEADER$.
  ptr_=RCV$+14                         /* Advance the pointer by 14 characters.
  UNIT$=PTR$(1)                        /*  One character from the pointer position is copied to UNIT$.
  SELECT_CASE HEADER$                  /* Check the header.
   CASE "WT" : RESULT$="○"
   CASE "US" : RESULT$="□"
   CASE "OL" : RESULT$="×"
   CASE_ELSE                           /* Unexpected case
    PRINT "invalid header"
    GOTO *RETRY
  END_SELECT
  TOTAL_CNT=TOTAL_CNT+1
  WEIGHT1$=STR$(VAL(RCV$))   /* First numerical value in character string RCV$ (an integer in this case)
  SERCH RCV$ "."          /*Decimal part may not exist depending on the weight scale setting (assumption).
  IF ptr_<>0 THEN                      /*If a searched character is found = If decimal part is found.
   WEIGHT2$=","+STR$(VAL(0)) /* Next numerical value in character string RCV$ (decimal part in this case)
  ELSE                                 /* ↑ Due to a convenience for LCD display, “.” is replaced with “,”.
   WEIGHT2$=""                         /* If there is no decimal part, it is left empty.
  END_IF
  PRINT TOTAL_CNT RETRY_CNT RESULT$ WEIGHT1$ WEIGHT2$ UNIT$   /* FTMW display
  BUF$=HEADER$+WEIGHT1$+WEIGHT2$+"G   "     /* Lower-case English character cannot be displayed
   on LCD.
  PR_LCD BUF$                               /* MPC-2100 LCD display, example “WT117,3G”
LOOP
```

```
Execution result (FTMW display)
 1 0 ○ 0 ,0 g
 2 0 ○ 0 ,0 g
 3 0 ○ 0 ,0 g
 4 0 ○ 0 ,0 g
 5 0 ○ 0 ,0 g
 6 0 ○ 0 ,0 g
 7 0 ○ 0 ,0 g
 8 0 ○ 0 ,0 g
 9 0 □ 51 ,3 g← An item is placed on the weight scale.
 10 0 □  111 ,9 g
 11 0 □ 117 ,0 g
 12 0 □ 117 ,2 g
 13 0 □ 117 ,3 g
 14 0 □ 117 ,3 g
 15 0 □ 117 ,3 g
 16 0 □ 117 ,3 g
 17 0 ○ 117 ,3 g
 18 0 ○ 117 ,3 g
```

(see also: Application Note an2k-005.)

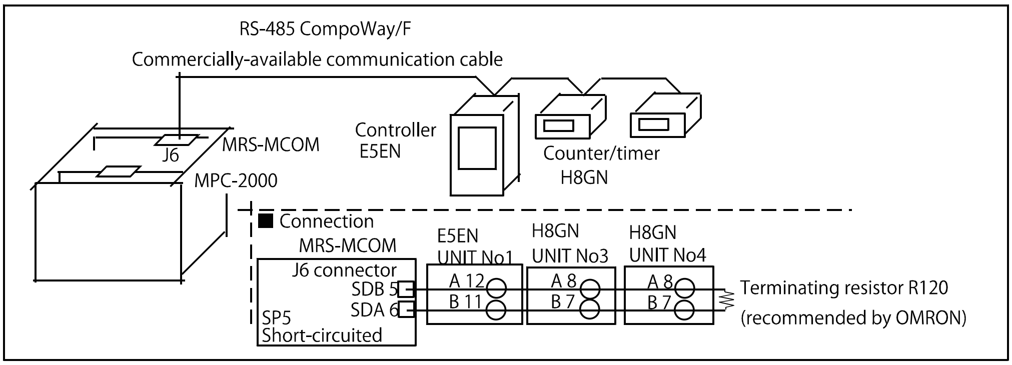
RS-485 device connection example

Major commands

CNFG#	Communication configuration (RS485 is specified as a parameter.)
PRINT#	Output
INPUT#	Input
COMPOWAY	OMRON CompoWay/F protocol macro command / Reserved constant

RS-485 is supported with J5 and J6 connectors of communication expansion board MRS-MCOM. Because MRS-MCOM has a fail-safe circuit built-in, there is no need of an externally-attached circuit other than a terminating resistor on the device side.

[Device connection example]
This is an example of multidrop-connecting OMRON digital controller E5EN and electronic counter/timer H8GN.



* CompoWay/F is a unified communication protocol in the general-use serial communication of OMRON Corp.

* Users should beware that the names of RS-485 signals A and B may be vice-versa depending on the manufacturer.

The current value (temperature) in the variable area of OMRON digital controller E5EN is read in. A character string is assembled according to the format of CompoWay/F protocol, BCC is calculated, sent, BCC is calculated from received data, and a necessary part is cut out. The character string processing is in the conventional (MPC-684) style.

```
CNFG# 5 RS485 "9600b7pes2NONE"     /*  MRS-MCOM Ch5 RS485 configuration
FORMAT ""                          /* No character string format
SEND$=CHR$(2)                      /* STX
SEND$=SEND$+"01"                   /* Note number
SEND$=SEND$+"000"                  /* Sub-address, SID
SEND$=SEND$+"0101"                 /* MRC,SRC
SEND$=SEND$+"C0"                   /* Variable type
SEND$=SEND$+"0000"                 /* Starting address
SEND$=SEND$+"00"                   /* Bit position
SEND$=SEND$+"0001"                 /* Number of elements
SEND$=SEND$+CHR$(3)                /* ETX
PUT_BCC=0                          /* BCC to be sent is calculated
FOR I=1 TO LEN(SEND$)-1
 STRCPY SEND$ BUF$ I 1
 PUT_BCC=PUT_BCC^ASC(BUF$)&&HFF    /* Exclusive logical sum
NEXT I
PRINT# 5 SEND$ CHR$(PUT_BCC)       /* Sending
DO
 INPUT# 5 CHR_C|1 BUF$             /*Receiving one character at a time
 IF ASC(BUF$)==&H02 THEN           /* Waiting for STX (top of data)
  BREAK
 END_IF
LOOP
GET_STR$=""                  /*  Received character variable (from the response frame STX to ETX)
DO
 INPUT# 5 CHR_C|1 BUF$             /* Receiving one character at a time
 GET_STR$=GET_STR$+BUF$
 IF ASC(BUF$)==&H03 THEN           /* If ETX is received, exit the LOOP.
  BREAK
 END_IF
LOOP
INPUT# 5 CHR_C|1 GET_BCC0$         /* Receiving one character (BCC data).
GET_BCC0=ASC(GET_BCC0$)            /* Received BCC data -> Numerical value
GET_BCC1=0
FOR I=0 TO LEN(GET_STR$)-1         /* BCC is calculated from the received character string.
 STRCPY GET_STR$ BUF$ I 1
 GET_BCC1=GET_BCC1^ASC(BUF$)&&HFF
NEXT I
IF GET_BCC0<>GET_BCC1 THEN
 PRINT "BCC ERROR"
 PRINT "Received BCC=" HEX$(GET_BCC0) " Calculated BCC=" HEX$(GET_BCC1)
 END
END_IF
STRCPY GET_STR$ NODE$ 0 2          /* Two characters from 0 constitute the node No.
STRCPY GET_STR$ GET_TMP$ 14 8      /* Eight character from 14 constitute the temperature.
```

Using CompoWay/F communication macro commands simplifies assembling character strings and eliminates the need of calculating BCC.

1) Sending procedure

- Construct a text sent by COMPOWAY command.
- If PRINT# command is given COMPOWAY option and executed, it sends a command frame with STX, ETX, and BCC added.

2) Receiving procedure

- If PRINT# command is given COMPOWAY option and executed, it receives a response form and calculates BCC.
- Elements are developed into variables from the response form by COMPOWAY command.

3) Example of communication by COMPOWAY macro command
(using a pointer in character string processing)

```
CNFG# 5 RS485 "9600b7pes2NONE"        /* Communication initialization
FORMAT ""                             /* No character string format
/* Elements of the text part of command frame are put in variables / character string variables.
node_no=1                             /* Node No.
sub_adr=0                             /* Sub-address
sid=0                                 /* SID

mrc_src$="0101"                       /* MRC,SRC
hensu_shu$="C0"                       /* Variable type
str_adr$="0000"                       /* Starting address
bit_ichi$="00"                        /* bit position
yoso_su$="0001"                       /* Number of element
setteichi$=""                         /* No set value
cmnd_txt$=mrc_src$+hensu_shu$+str_adr$+bit_ichi$+yoso_su$+setteichi$  /* Command text is created.

COMPOWAY node_no sub_adr sid cmnd_txt$ snd$    /* All from the node No. to command text are put
    together in snd$.
PRINT# 5 COMPOWAY snd$                /* Command frame is sent.

INPUT# 5 COMPOWAY TMOUT|2 rcv$        /* Response frame is received in rcv$.
COMPOWAY rcv$ node_no sub_adr end_code res$     /* Character string of command text enters in res$.

/* Four characters from the 4th character of res$ counted from 0 constitute a response code.
ptr_=res$+4  /* ptr_ is a pointer reserved variable.  It points to the 4th character of res$.
res_code=HEX(PTR$(4))                 /* Copy four characters from the position of ptr_.

/* Eight characters from the 8th character of res$ counted from 0 constitute next data.
ptr_=res$+8                           /* Pointer point to the 8th character of res$.
res_data$=PTR$(8)                     /*Copy eight characters from the position of ptr_.
PRINT res_code HEX(res_data$)         /* Display the temperature.

Execution example
0 58                                  /* Response code = 0, temperature 58°C
```

(See also: Application Note an2k-004.)

USB memory

MPC-1000 and MRS-MCOM have a port dedicated to USB memory built-in, so that point data and programs created by a PC can be read in. It can be applied for replacing data when switching models for example.

Major commands:

DIR	Obtaining the file list of USB memory
USB_LOAD	Reading a program from USB memory
USB_SAVE	Writing a program onto USB memory
USB_PLOAD	Reading point data from USB memory
USB_PSAVE	Writing point data to USB memory
USB_WRITE	Appending to a USB memory file
USB_READ	read one line from an usb file

■ Reading and writing a program

Applicable for maintenance such as updating and storing a program.

[Execution example]

```
#LIST 0                         /* Current MPC program
10      DO
20       FOR   I=0 TO 2
30        ON   I
40        TIME   100
50        OFF   I
60        TIME   100
70       NEXT   I
80      LOOP
#DIR                            /* USB memory content = empty
 Drive A has no volume label.

File not found.
      0 files          0 bytes
      0 directories
A:>
#USB_SAVE "TEST.F2K"            /* MPC program is written onto USB memory.
#DIR
 Drive A has no volume label.

2009/00/02  10:46             108 TEST.F2K         /* New file
      1 files  108 bytes
      0 directories
A:>
#NEW                            /*Current MPC program is deleted.
#LIST                           /* Confirmation display = empty

#USB_LOAD "TEST.F2K"            /* Try reading the file just written.
#LIST                           /* Confirmation display
10      DO
20       FOR   I=0 TO 2
30        ON   I
40        TIME   100
50        OFF   I
60        TIME   100
70       NEXT   I
80      LOOP
#
```

■ Reading/writing of point data

Applicable for switching models, storing work data, and the like.

[Execution example]

```
#DIR
 Drive A has no volume label.

File not found.
      0 files  0 bytes          /* Empty USB memory
      0 directories
A:>
#USB_PSAVE "TEST.P2K"           /* Current MPC point data are stored.
#DIR
 Drive A has no volume label.
```

```
2009/03/12  00:28     395 TEST.P2K        /*  New file
        1 files               395 bytes
         0 directories
A:>
#NEWP                                     /* Current MPC point data are deleted.
#PLS 0                                    /*Confirmation display
P(1)   X= 0  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
(Omitted)
#USB_PLOAD "TEST.P2K"                     /*Try reading the point data just stored in USB memory just now.
#PLS 0                                    /* Confirmation display
P(1)   X= 3440  Y= 17480  U= 0  Z= -19027
P(2)   X= 16420  Y= 18120  U= 0  Z= -18707
P(3)   X= 3200  Y= 43640  U= 0  Z= -19267
P(4)   X= 29100  Y= 17960  U= 0  Z= -18947
P(5)   X= 42020  Y= 17880  U= 0  Z= -18867
P(6)   X= 28920  Y= 43480  U= 0  Z= -19187
P(7)   X= 0  Y= 0  U= 0  Z= 0
 (Omitted)
#
```

■ Writing text data

USB_WRITE command successively performs APPEND OPEN, WRITE, and CLOSE and appends a character string to a specified file. This function can also be applied as a data logger.

[Execution example]

```
#LIST                                     /* Display the program currently in MPC.
10      FILE$="TEST.CSV"                  /* File name.  FILE$ is a reserved variable.
20       USB_DEL   FILE$                  /* Delete any preexisting file having the same.
30       DO
40        FORMAT   "00/00/00"
50      DT$=HEX$(DATE(0))
60        FORMAT   "00:00:00"
70      TM$=HEX$(TIME(0))
80        USB_WRITE   DT$+","+TM$+"\n"    /* USB write
90        TIME   1000
100       LOOP
#DIR                                      /* USB memory content = empty
Drive A has no volume label.

File not found.
         0 files  0 bytes
            0 directories
A:>
#RUN                                      /* Execute.

@None_file                                /*Message when there is no preexisting file (does not stop).
       *0    [90]                         /* Stop with Ctrl+A after a while.
#DIR                                      /* USB memory content
 Drive A has no volume label.

2009/00/02  11:02        133 TEST.CSV     /* New file
         1 files  133 bytes
            0 directories
```

```
A:>
#TYPE "TEST.CSV"                    /* Display the file content.
09/00/02,11:02:17
09/00/02,11:02:18
09/00/02,11:02:19
09/00/02,11:02:20
09/00/02,11:02:21
09/00/02,11:02:22
09/00/02,11:02:23

A:>
#
```

■ Reading text data

USB_READ reads out character strings in a file one line at a time. The file name is specified using FILE$.
In the following example, all content is read out and displayed. The EOF(n) function is a function for judging if the end of a file has been reached while reading. The value 1 indicates that the end of the file has been reached.
To stop reading the file mid-way, USB_READ -1 should be executed.

```
10    FILE$="AUTO.P2K"
20          DO
30            USB_READ   a$ : PRINT   EOF(0) a$
40        IF   EOF(0)==1 THEN  : END   : END_IF
50          LOOP
```

■ Difference of the USB function between MPC-1000/2200 and MRS-MCOM

Excluding USB_RST and USB(0) functions, there is no difference in the usage and specification of commands.

Command	Function	MRS-MCOM	MPC-1000/2200
USB_RST*	Resetting USB process	No USB memory ON/OFF	USB memory ON/OFF
USB(0)	Detecting USB presence	Invalid (Always 1)	Present:1, Absent: 0

* After USB_RST, MPC starts initialization communication with the USB memory. Therefore, for several seconds after USB_RST no operation can be made on the USB memory. Although the timing can be detected by the judgment of the USB() function in the MPC-1000/2200, going through the MRS-MCOM, a timer such as TIME 2000 is used.

■ Errors related to USB memory

	Number	Meaning
USB_INUSE	53	File already in use.
USB_NONE	54	USB memory not connected.
USB_HALT	56	USB memory operation halted.
USB_NORSP	68	USB memory process not responding.
NO_FILENAME	69	File name inappropriate.
NO_FILE	70	Specified file does not exist.

* In cases where an operation error has occurred (56 and 68), the USB_RST command is executed by the ON_ERROR process. By this command the USB memory and the USB memory processes are restored to their initial states, and the same process is repeated.

■ Precautions using USB memory

1) Only USB memory products which are made by established manufacturers and confirmed to function should be used. Among cheap or non brand name models there are some of poor-quality which cannot endure read/write and have no reliability from the beginning.

2) USB memory should be considered as being consumable. When continuous read and write are repeated for about one week, USB memory is damaged. The length of this period is due to the performance limitations of the flash memory built into the USB memory. Therefore, the USB memory should be replaced with a new one confirmed to function after a specified degree of use.

3) USB memory of the smallest possible size should be used (2GB or smaller is recommended). USB memory of 8GB class may be specialized for USB 3.0 or have larger number of sectors or sector size, which slows down the response speed when connected.

4) USB memory used in an MPC should be a dedicated one and formatted using a PC before use.

5) The number of files used in USB memory should be about 10~20. If the number of files becomes too large, the response slows down, and errors such as time-out occur.

6) File names usable in an MPC should only be in the format of “8 + 3” ASCII characters. Long file names or Japanese file names should not be placed in the USB memory used. In addition, no subdirectories should be created, as this can cause problems.

7) MPC supports only FAT and FAT32. FAT12 is not recognized.

CUnet

MPC-2000 also provides a network function, by which it can perform a more complex data communication at high speed. The network used, CUnet, is a network for FA developed and manufactured/sold by Step Technica, and allows sharing of a 512-byte memory image on the network. It supports up to 64 units of stations and is designed so that the shared memory synchronizes within 2.5 msec. In order to use CUnet, MPC-2000 side is required to have MPC-CUnet2, and the PC side USB-CUnet.

If the operation is limited to interlocks and exchanges of simple numerical value among MPC-2000 units, because the shared memory can be directly referred to and modified by IO commands such as IN, OUT, SW, ON, and OFF, high-speed dispersed control can easily be constructed.

In addition, MPC-2000 provides an information exchange function (CU_POST, POST) which utilizes the mail function of CUnet, and allows block transfer of point data and MBK data area and character string exchange between MPCs and between MPC and PC. The mail transfer unit is 15 (4 bytes*4*15) for P(n), and 120 (2 bytes*120) for MBK(n).

The software compatible with USB-CUnet for PCs can easily create applications such as VB using the dedicated DLLs (device drivers need to be set up).

■ Example of usage between MPCs

Major commands:

CUNET	MPC-CUnet initialization
SW,ON,OFFI	Bit 2000~6095 operation
N,OUT	Bank 2000~2511 operation
CU_POST	CUnet mail server task started

By appropriately initializing two MPCs as follows, their memories on CUnet can be mutually referred to and can be used as virtual I/O.

MPC A side	MPC B side
#cunet 0 8 31 #pr IN(SA_B(8)) 100 #on SA(0)+10	#cunet 8 8 31 #out 100 SA_B(8) #pr SW(SA(0)+10) 1 #

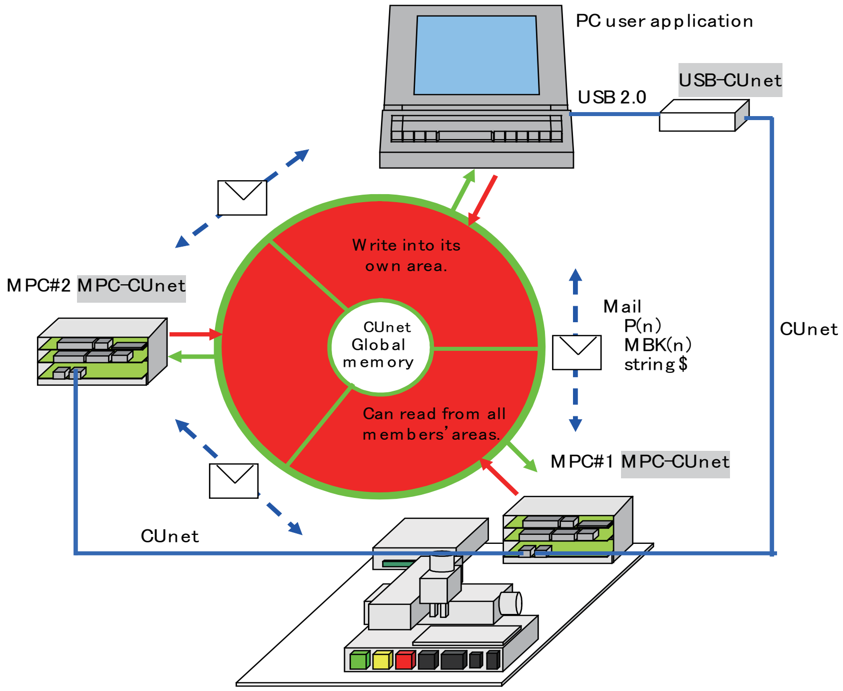
In addition, using CU_POST and POST commands, point data of MPC wherein CU_POST server on MPC is started can be rewritten.

In the following example, point data (for 15) are copied to the A side by POST command on the B side.

MPC A side	MPC B side
CU_POST #pls 1 P(1) X= 46217 Y= 46218 U= 1 Z= 2 P(2) X= 0 +Y= 0 U= 0 Z= 0 P(3) X= 0 Y= 0 U= 0 Z= 0 P(4) X= 111 Y= 112 U= 0 Z= 0 P(5) X= 104 Y= 105 U= 0 Z= 0 P(6) X= 120 Y= 121 U= 0 Z= 0	POST 0 P(1)

■ Information exchange with PC

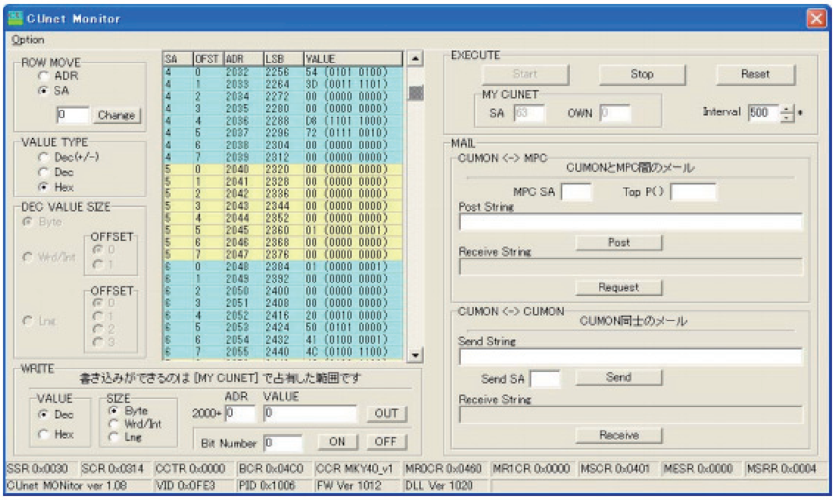
The following figure is a conceptual diagram of coordinating two MPC units and a PC. High-speed interlock between MPCs and exchanges of model data and operation information between PC and MPC become possible.



■ Monitoring tool

In order to construct such a network environment, a tool which refers to data status and modifies the data from a PC is required.

CUnet monitor (CUMON.EXE) is a tool to check read/write of global memory and sending/receiving of mails. In addition, it can also check the register status of MKY40 which is the main body chip of CUnet. It is used for checking the operation after a setup or while debugging. It is freely downloadable from our company's web site.



SA	OFST	ADR	LSB	VALUE
4	0	2032	2256	54 (0101 0100)
4	1	2033	2264	3D (0011 1101)
4	2	2034	2272	00 (0000 0000)
4	3	2035	2280	00 (0000 0000)
4	4	2036	2288	D8 (1101 1000)
4	5	2037	2296	72 (0111 0010)
4	6	2038	2304	00 (0000 0000)
4	7	2039	2312	00 (0000 0000)
5	0	2040	2320	00 (0000 0000)
5	1	2041	2328	00 (0000 0000)
5	2	2042	2336	00 (0000 0000)
5	3	2043	2344	00 (0000 0000)
5	4	2044	2352	00 (0000 0000)
5	5	2045	2360	01 (0000 0001)
5	6	2046	2368	00 (0000 0000)
5	7	2047	2376	00 (0000 0000)
6	0	2048	2384	01 (0000 0001)
6	1	2049	2392	00 (0000 0000)
6	2	2050	2400	00 (0000 0000)
6	3	2051	2408	00 (0000 0000)
6	4	2052	2416	20 (0010 0000)
6	5	2053	2424	50 (0101 0000)
6	6	2054	2432	41 (0100 0001)
6	7	2055	2440	4C (0100 1100)

ROW MOVE: 0; SA: 63; OWN: 0; Interval: 500; ADR: 0; VALUE: 0; Bit Number: 0

SSR 0x0030 SCR 0x0314 CCTR 0x0000 BCR 0x04C0 CCR MKY40_v1 MR0CR 0x0460 MR1CR 0x0000 MSCR 0x0401 MESR 0x0000 MSRR 0x0004

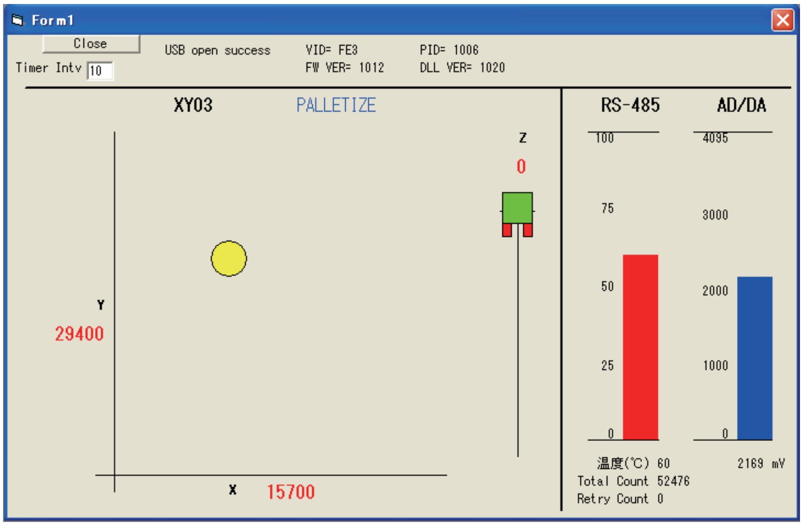
CUnet MONitor ver 1.08 VID 0x0FE3 PID 0x1006 FW Ver 1012 DLL Ver 1020

(see also: DOWNLOAD > TOOL > CUnetMonitor)

■ Visual Basic application

▪ Global memory read and write

An example of VB6. Operated by reading the XY03 coordinate values, RS-485 controller (temperature), AD voltage, and the like are written into global memory by MPC.



Timer Intv 10

USB open success VID= FE3 PID= 1006

FW VER= 1012 DLL VER= 1020

Z 0

Y 29400

X 15700

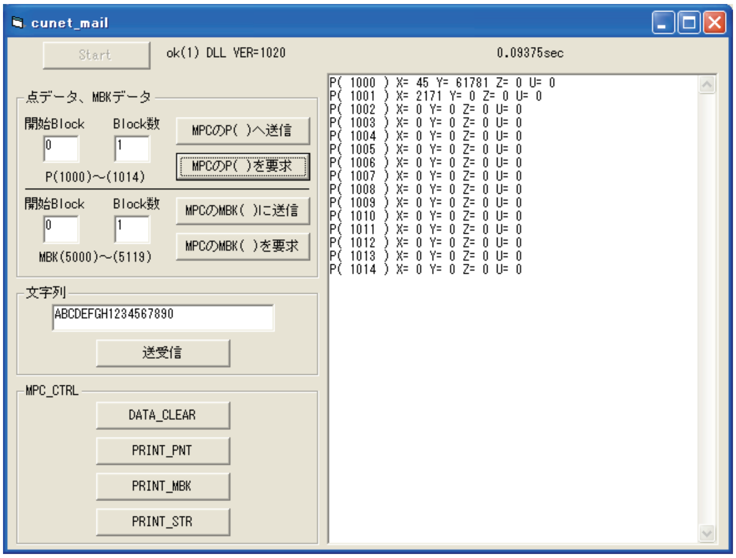
温度(℃) 60 2169 mV

Total Count 52476

Retry Count 0

■ CUnet Mail communication

Sample VB6. Performs block transfers of point data and MBK data and exchanges of character strings with MPC.



ok(1) DLL VER=1020　　　0.09375sec

開始Block 0　Block数 1　P(1000)~(1014)

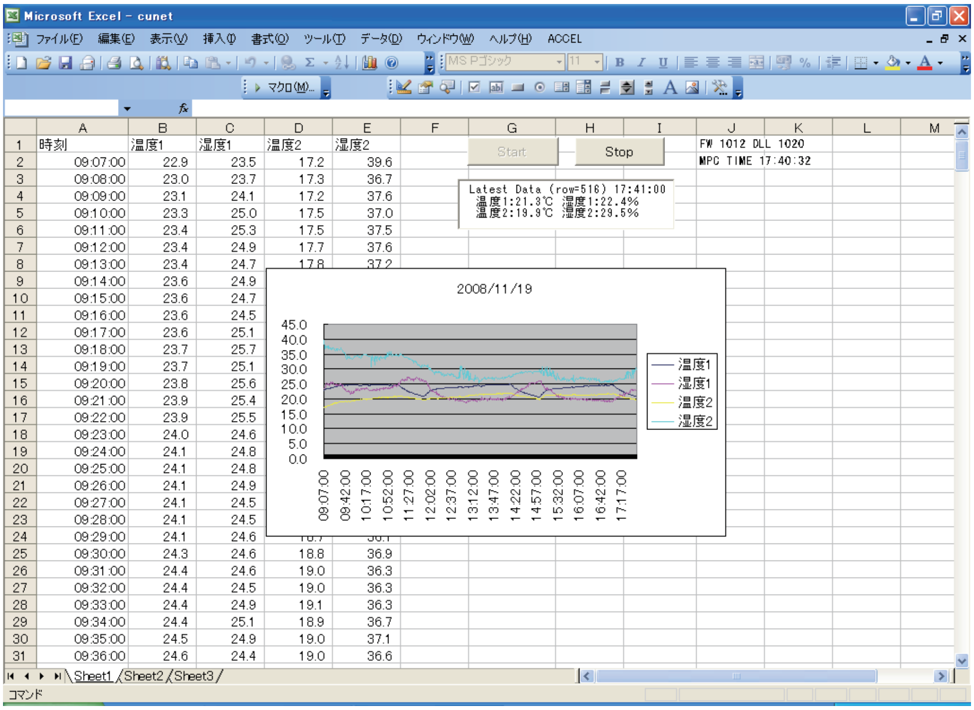
開始Block 0　Block数 1　MBK(5000)~(5119)

文字列: ABCDEFGH1234567890

```
P( 1000 ) X= 45 Y= 61781 Z= 0 U= 0
P( 1001 ) X= 2171 Y= 0 Z= 0 U= 0
P( 1002 ) X= 0 Y= 0 Z= 0 U= 0
P( 1003 ) X= 0 Y= 0 Z= 0 U= 0
P( 1004 ) X= 0 Y= 0 Z= 0 U= 0
P( 1005 ) X= 0 Y= 0 Z= 0 U= 0
P( 1006 ) X= 0 Y= 0 Z= 0 U= 0
P( 1007 ) X= 0 Y= 0 Z= 0 U= 0
P( 1008 ) X= 0 Y= 0 Z= 0 U= 0
P( 1009 ) X= 0 Y= 0 Z= 0 U= 0
P( 1010 ) X= 0 Y= 0 Z= 0 U= 0
P( 1011 ) X= 0 Y= 0 Z= 0 U= 0
P( 1012 ) X= 0 Y= 0 Z= 0 U= 0
P( 1013 ) X= 0 Y= 0 Z= 0 U= 0
P( 1014 ) X= 0 Y= 0 Z= 0 U= 0
```

■ MS-EXCEL

Sample MS Excel VBA. The two hygrothermometers below are RS-485 multidrop-connected; measured values are written onto a worksheet at constant intervals, and simultaneously plotted.



FW 1012 DLL 1020
MPC TIME 17:40:32

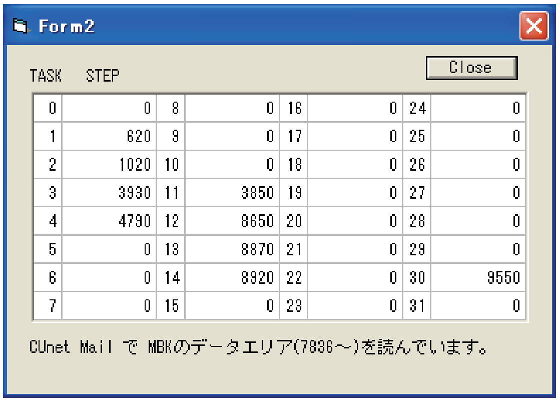
```
Latest Data (row=516) 17:41:00
 温度1:21.3℃ 湿度1:22.4%
 温度2:19.9℃ 湿度2:29.5%
```

	A	B	C	D	E
1	時刻	温度1	湿度1	温度2	湿度2
2	09:07:00	22.9	23.5	17.2	39.6
3	09:08:00	23.0	23.7	17.3	36.7
4	09:09:00	23.1	24.1	17.2	37.6
5	09:10:00	23.3	25.0	17.5	37.0
6	09:11:00	23.4	25.3	17.5	37.5
7	09:12:00	23.4	24.9	17.7	37.6
8	09:13:00	23.4	24.7	17.8	37.2
9	09:14:00	23.6	24.9		
10	09:15:00	23.6	24.7		
11	09:16:00	23.6	24.5		
12	09:17:00	23.6	25.1		
13	09:18:00	23.7	25.7		
14	09:19:00	23.7	25.1		
15	09:20:00	23.8	25.6		
16	09:21:00	23.9	25.4		
17	09:22:00	23.9	25.5		
18	09:23:00	24.0	24.6		
19	09:24:00	24.1	24.8		
20	09:25:00	24.1	24.8		
21	09:26:00	24.1	24.9		
22	09:27:00	24.1	24.5		
23	09:28:00	24.1	24.5		
24	09:29:00	24.1	24.6		
25	09:30:00	24.3	24.6	18.8	36.9
26	09:31:00	24.4	24.6	19.0	36.3
27	09:32:00	24.4	24.5	19.0	36.3
28	09:33:00	24.4	24.9	19.1	36.3
29	09:34:00	24.4	25.1	18.9	36.7
30	09:35:00	24.5	24.9	19.0	37.1
31	09:36:00	24.6	24.4	19.0	36.6

■ Examples of task monitor

Using USB-CUnet and MPC-CUnet, the execution statement number of each task is found in the same manner as in the “Touch panel connection”.

In the VB6 application example the MBK area of MPC is read and substituted for MSFlexGrid using the cunet_req_mbk function of CUnet Mail. Useful for debugging and maintenance.



TASK STEP

0	0	8	0	16	0	24	0
1	620	9	0	17	0	25	0
2	1020	10	0	18	0	26	0
3	3930	11	3850	19	0	27	0
4	4790	12	8650	20	0	28	0
5	0	13	8870	21	0	29	0
6	0	14	8920	22	0	30	9550
7	0	15	0	23	0	31	0

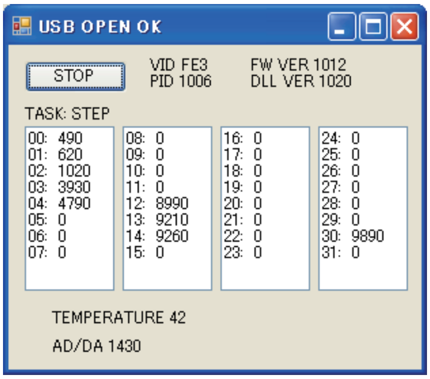
CUnet Mail で MBKのデータエリア(7836～)を読んでいます。

▪ VB6 program example (Periodic reading using Timer)

```
Private Sub Timer1_Timer()
Dim ar(0 To 119) As Long

 res = cunet_req_mbk(4, 7836, ar(0))
         ‘Reading the MBK area (120 words)  Parameters: Request SA, MBK() top, stored array
 i = 0

 For c = 1 To 8 Step 2
  For r = 0 To 7
   s = CStr(ar(i) + ar(i + 1) * &H10000)
            ‘Converted to 4-byte length because MPC has "S_MBK LONG_PRG" specified.
   MSFlexGrid1.TextMatrix(r, c) = s
   i = i + 2
  Next r
 Next c

End Sub
```

■ Visual Basic 2008 Express Edition

Example of creating a program using the VB2008 Express Edition. Task statement numbers are monitored by CUnet-Mail, and temperature and AD/DA voltage are displayed by reading global memory.



```
VID FE3      FW VER 1012
PID 1006     DLL VER 1020

TASK: STEP
00:  490     08:  0       16:  0       24:  0
01:  620     09:  0       17:  0       25:  0
02:  1020    10:  0       18:  0       26:  0
03:  3930    11:  0       19:  0       27:  0
04:  4790    12:  8990    20:  0       28:  0
05:  0       13:  9210    21:  0       29:  0
06:  0       14:  9260    22:  0       30:  9890
07:  0       15:  0       23:  0       31:  0

   TEMPERATURE 42

   AD/DA 1430
```

- VB2008 program example (Periodic reading using Timer)

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick
   Dim ar(0 To 119) As Integer
   Dim res, i, r, c As Integer
   Dim s As String

   res = cunet_req_mbk(4, 7836, ar(0))     ‘ MBK Area Read.  param= Request SA, MBK top addr, Storage array

   TextBox1.Clear()
   TextBox2.Clear()
   TextBox3.Clear()
   TextBox4.Clear()
   i = 0
   For c = 1 To 4
      For r = 0 To 7
         s = Format((i / 2), “00”) + “:  “ + CStr(ar(i) + ar(i + 1) * &H10000)
         If c = 1 Then TextBox1.SelectedText = s + Chr(13) + Chr(10)
         If c = 2 Then TextBox2.SelectedText = s + Chr(13) + Chr(10)
         If c = 3 Then TextBox3.SelectedText = s + Chr(13) + Chr(10)
         If c = 4 Then TextBox4.SelectedText = s + Chr(13) + Chr(10)
         i = i + 2
      Next r
   Next c

   Label5.Text = “TEMPERATURE “ + CStr(cunet_in(2064, Cu_Int))    ‘ Global Memory Read
   Label6.Text = “AD/DA “ + CStr(cunet_in(2080, Cu_Wrd))         ‘ Global Memory Read

End Sub              (For all sources of this example, see Application Note an2k-010.)
```

4-6 Analog Control

MPC-AD12 is used for analog control. Both AD and DA can be easily handled with commands. Up to two boards of MPC-AD12 can be mounted, and up to 16 CHs of AD input and 8 CHs of DA output can be provided.

AD conversion

Function AD( ) is used. A value within a range of 0~4095 is obtained in the standard state of MPC-AD12, wherein 1 digit corresponds to 1 mV. If A becomes 1000 by A = AD(0), it means that the input was 1000 mV, namely 1 V. AD function also has a mode to obtain an average value, wherein a value which was automatically averaged by MPC-AD12 can be obtained.

As the AD conversion IC, AD7890-4 manufactured by Analog Device is used, which is mounted on an IC socket. This IC has another type called AD7890-10 which has a different voltage range, and by changing to that type, +/-10 V can be handled. In this case, the resolution will become 10/2048 mV = 4.88 mV/digit. If AD7890-10 is necessary, it can be specified at the time of purchase.

In addition, MPC-AD12 (CEP-125F version) can handle synchronous input. It is a function to obtain data automatically for a pulse array, which enables handling AD conversion wherein real-time nature is important.

DA conversion

Command DA is used for DA output. Executing DA 1000 1 outputs 1000 mV, namely 1 V, to DA-CH1.

Various kinds of settings

SET_AD command is prepared for setting the number of samples of the average values of AD converter, configuration at the time of changing to AD7890-10, and the like.