Changes for the Better

MITSUBISHI ELECTRIC

CNC

MELDAS 60 Series
CUSTOM RELEASE (APLC)

LIBRARY MANUAL

BNP-B2219E(ENG)

Introduction

This manual describes the details of the various library functions that support the user's software development when using the M60 Series custom release system. Please read this manual thoroughly before starting development, and refer to it as necessary for the applications and purposes of the developed functions.
All functions of the M60 Series custom release system are described in this manual. However, there may be limits to the functions that can be used due to the CNC model or option configuration. Check the CNC specifications before starting.
The following manuals are available as reference. Refer to them as required.

- Custom Release (APLC) Programming Manual BNP-B2217
- PLC Programming Manual (Ladder) BNP-B2212
- PLC Programming Manual (Ladder with MELSEC tool) BNP-B2269
- PLC Interface Manual BNP-B2211
- DDB Interface Manual BNP-B2214

Precautions for using this manual

This manual is written for persons having an understanding of C language.
An effort has been made to describe special handling, however, if the item is not described in this manual, please interpret it as "not possible".

[List of Custom Release Library Functions]

The functions that can be used in the custom release program are as shown below.

1. Screen Display Function I/F

1.1 Custom Screen Control Functions

1.2 Display Request Functions

1.3 Graphic Display Request Functions

1.4 Screen Display Auxiliary Functions

2. DDB I/F

2.1 CNC Data Read/Write Functions

3. Machine Control I/F

3.1 PLC Device Access

3.2 PLC Device High-speed Access

4. File Release I/F

4.1 File Data Input/Output Functions

5. General Functions

5.1 Data Conversion Functions

Contents

1. Screen Display Release I/F

1.1 Custom Screen Control Functions

1.1.1 p_ope () Screen Transition and Screen Function Control Function

This is software created by Mitsubishi and has a function to control "M_OPE".
Mainly screen transition and screen functions are controlled.

(Note) "M_OPE" is software used by the user to create original screens, etc.

1.1.2 pcoini () Initialization Function

This function carries out initialization when the power is turned ON.
The user's initialization function "mopeini ()" is called.

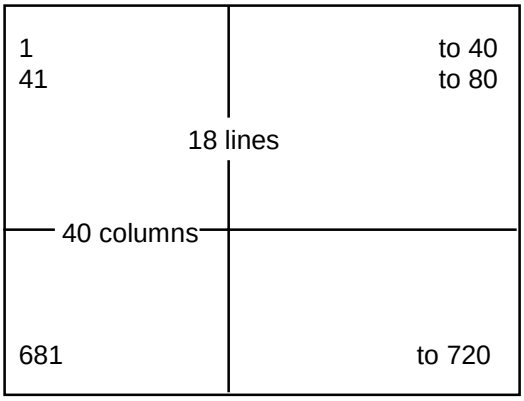
(Note) "mopeini ()" is an initialization function carried out by the user, and is called once.
• Use this to initialize the custom RAM area, etc.

1.2 Display Request Functions

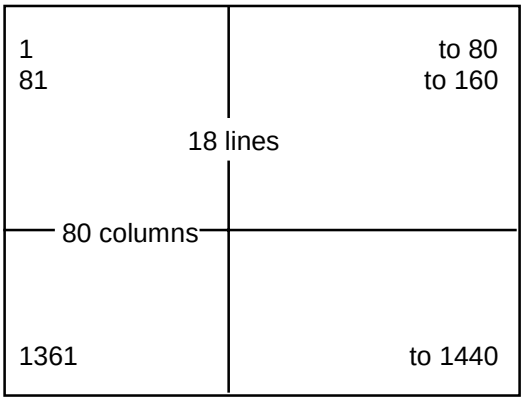
1.2.1 enquet () Character Display

Note that as shown below, the start position and end position of the data designated with the display request function count column at the upper left of the screen as 1.

40-character mode



80-character mode



When the cursor is displayed, the lower left will be 0 instead of 1.

enquet Function for requesting display

Function: Display of characters on the CRT is requested.
Use the command according to the type of characters to be displayed.
The character display commands are shown below.

Command list

Command name	Function
TXTYPE	Requests text/title data display
CTTYPE	Requests continuous text/title data display
NDTYPE	Requests numeric data
CNTYPE	Requests continuous numeric data
VRATYPE	Requests VRAM direct change (type A)
VRBTYPE	Requests VRAM direct change (type B)
CLTYPE	Requests deletion (line)
WCLTYPE	Requests deletion (matrix)
INDTYPE	Requests indirect display

Format: ret = enquet (id, type, flag, datptr, windid);

long id;	Possessory right ID	: Specify "pcoptb.ocb.scrnumb".
char type;	Command name	: Refer to the command list.
char flag;	Attribute change data	: Specify 0.
char *datpt;	Display data pointer	: Pointer for data to be displayed
long windid;	Window ID	: Specify 1.

Program example

```
TXDATA 1ot0[] =  {6,6,C_RK|NORMATR,4,0x00,0x00,(char **)d1ot00,
                  26,26,C_YK|NORMATR,16,0x00,0x00,(char **)d1ot01,
                  15,15,C_WK|NORMATR,144,0x00,0x00,(char **)d1ot02,
                  2,2,C_WK|NORMATR,245,0x00,0x00,(char **)d1ot03,
                  0 };
enquet (pcoptb.ocb.scrnumb,TXTYPE,O,1ot0,1L};
```

(Note)

For the actual screen display, display process in the CNC is carried out. The display request function carries out a process to set data in the display data table on display process in the CNC.
Thus, in the following cases, characters will not be displayed.

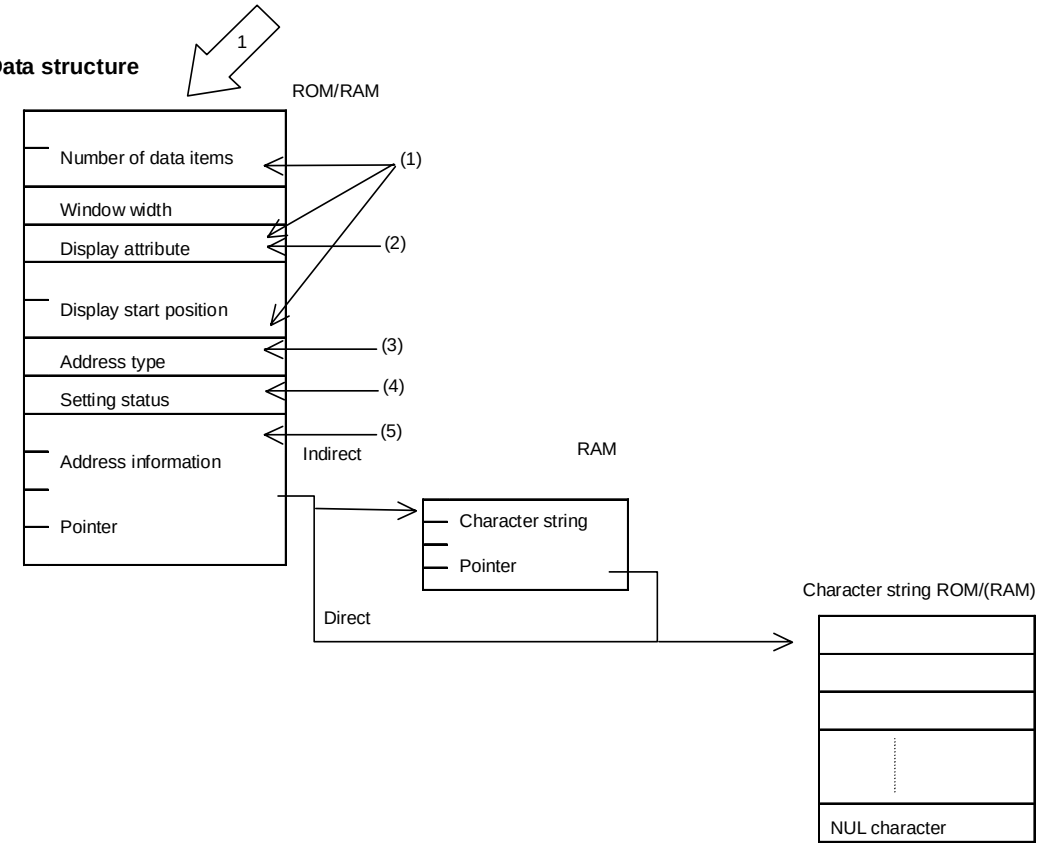
- When display data is created on a stack (internal variable).
  The characters may be displayed due to the processing timing, but the display data must be created on the ROM or on an RAM (global variable) for which the contents are guaranteed.

- When display request reset is carried out before the display is completed.
  When the display request is reset, the display data that has not been displayed will be invalid.
  When carrying out display request and display request reset in succession, reset the display request after confirming that the display has been completed.

TXTYPE

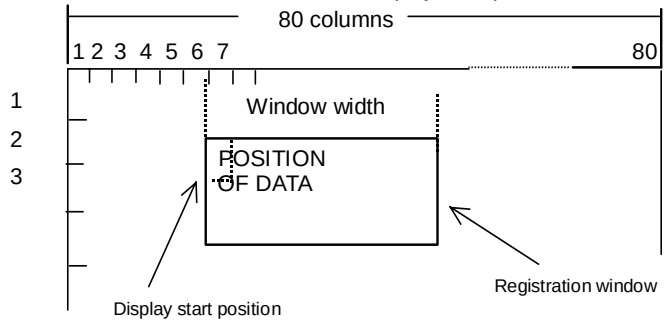
Function: Display of single tile/text data

Data structure



1 This structure name is defined as "TXDATA".

(1) Number of data items, Window width and Display start position



In the above case

Number of data items —— 16 (8 × 2)
Window width —— 8
Display start position —— 87

(2) Display attribute

Color CRT

Bit No. 7 6 5 4 3 2 1 0

B	G	R	B	G			

Enlargement designation ★
Highlight designation ☆
Background color designation
Character color designation

Character color	B	G	R
Black	0	0	0
Red	0	0	1
Green	0	1	0
Yellow	0	1	1
Blue	1	0	0
Magenta	1	0	1
Cyan	1	1	0
White	1	1	1

Background color	B	G
Black	0	0
Green	0	1
Blue	1	0
Cyan	1	1

Example) 0×60 (Character color) Yellow, (Background color) Black

☆ When highlight designation is set to 1, the color designated for the characters will be displayed as the background color, and the color designated for the background will be displayed as the character colors.

Example) 0×64 (Character color) Black, (Background color) Yellow

Mono-chrome CRT

Bit No. 7 6 5 4 3 2 1 0

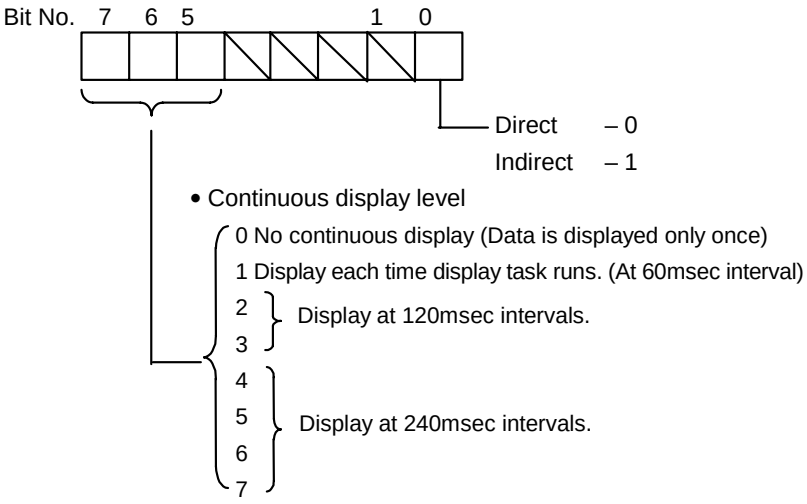
Enlargement designation ★
Highlight designation

Example) 0×05 Reverse and enlarge

★ Enlargement designation is valid for the following characters.
If data containing characters for which enlargement designation is not valid, nothing will be displayed.

1) Uppercase alphabet characters, numbers (A to Z, 0 to 9)
2) +, –, (,)

(3) Address type



For the title data (fixed), the data is fixed at 0X00 (direct, and) is not continuously displayed.

★ The following figure shows the meanings of the continuous display levels (1 to 7).

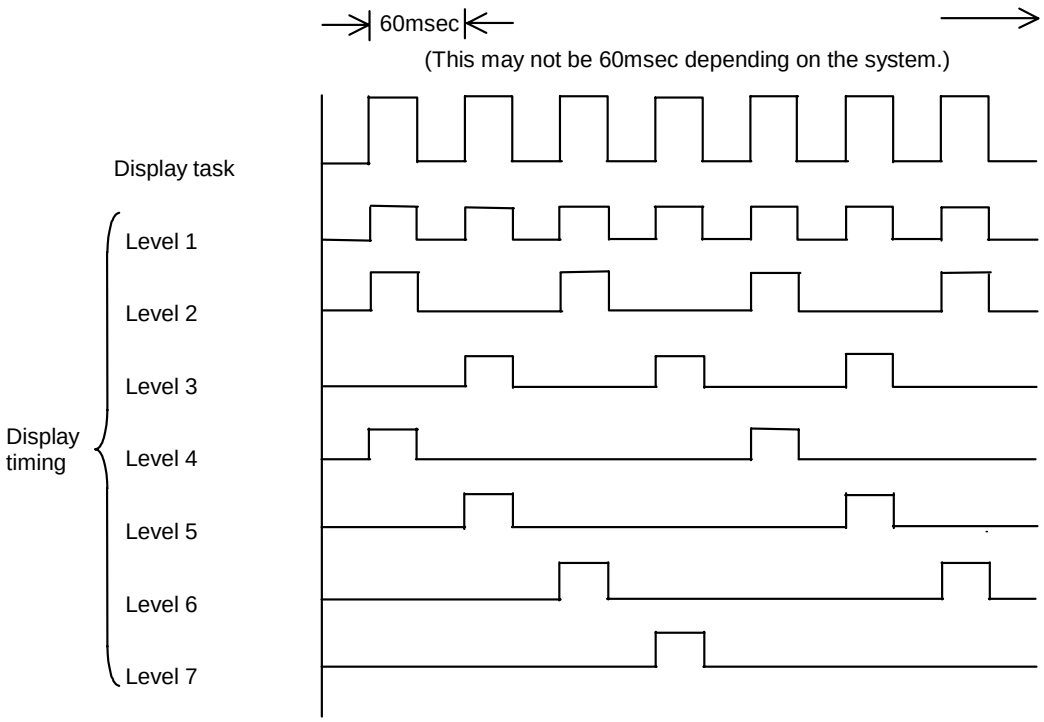


Image of display timing for continuous display level

When designating the continuous display level, the user should make the display task processing time be equal each time.

(4) Setting status
Set the setting status is fixed at 0x00.

(5) Address information pointer
The address is set in the address information pointer, so take care when creating the text display data.

Program example

```
char dkpa10[] = {"[Machining parameter]"};

TXDATA kpal[] = {12,12, C_YK|NORMATR,1,0x00,0x00,(char **)dkpa10,
                 1,1,C_YK|NORMATR,264,0x00,0x00,(char **)&paramet.common.kyoko,
                 0};
enquet (pcoptb.ocb.scrnumb,TXTYPE,0,kpal,1L);
```

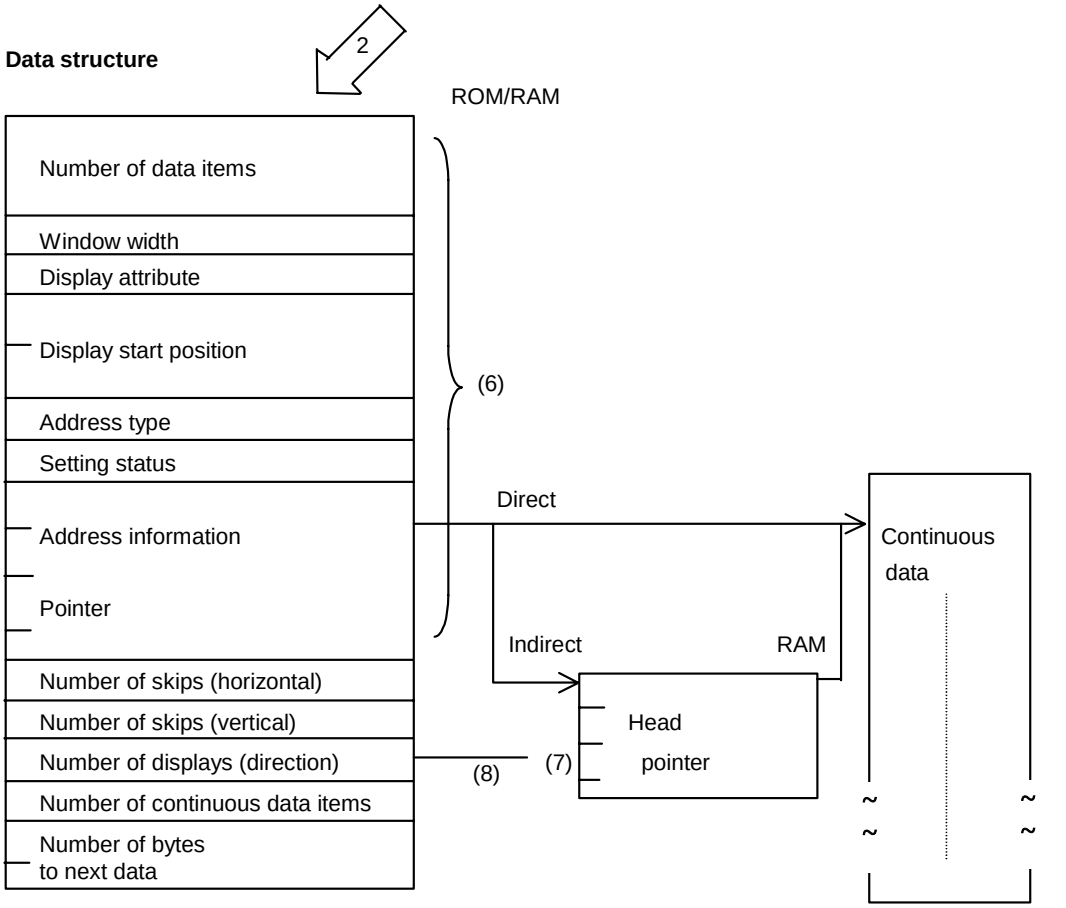
Single title

Single text

Structure's end code

CTTYPE

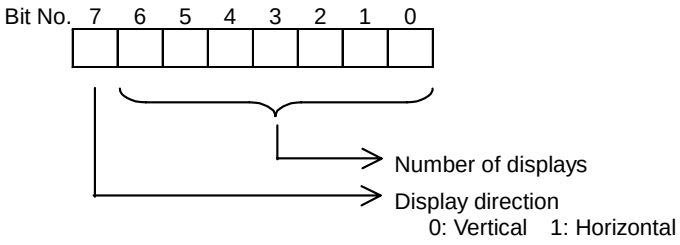
Function: Display of continuous title/text data



2 This structure name is defined as "CTXDAT".

(6) Refer to the single title/text data.

(7) Number of displays (direction)



The continuous title/text data is used when the following conditions are satisfied as shown below;
There are character string display areas on the CRT that has a set pattern (vertical skip, horizontal skip).
The attributes such as the character display are the same.
The address storing the display character string has a set interval from the top.

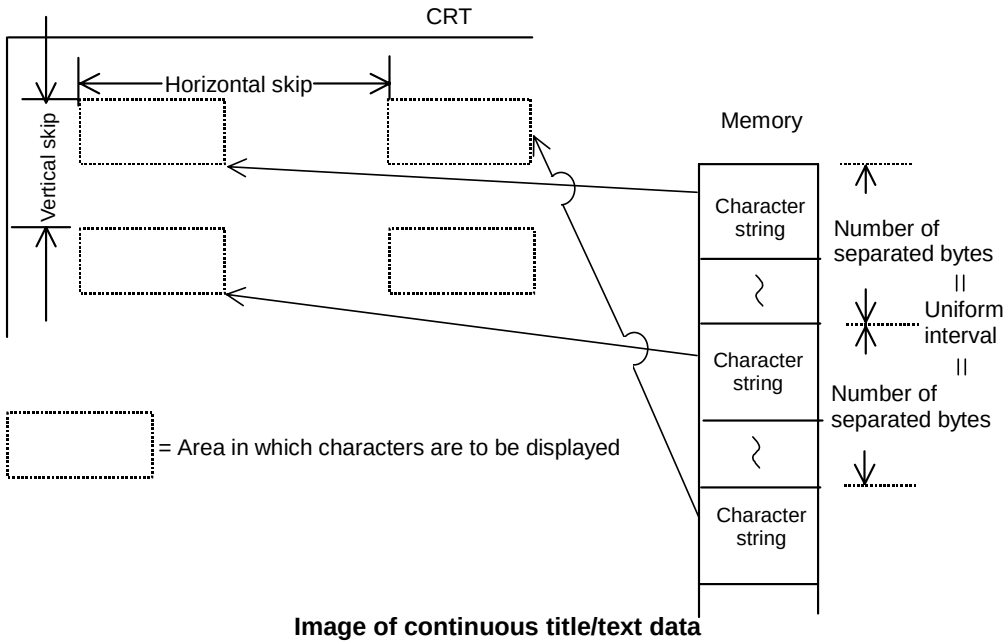
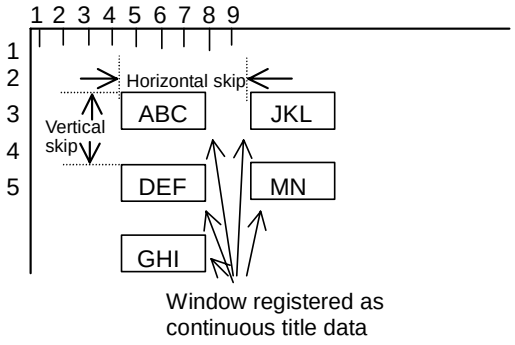
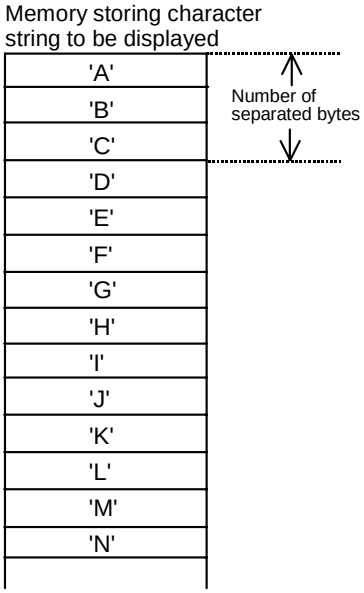


Image of continuous title/text data

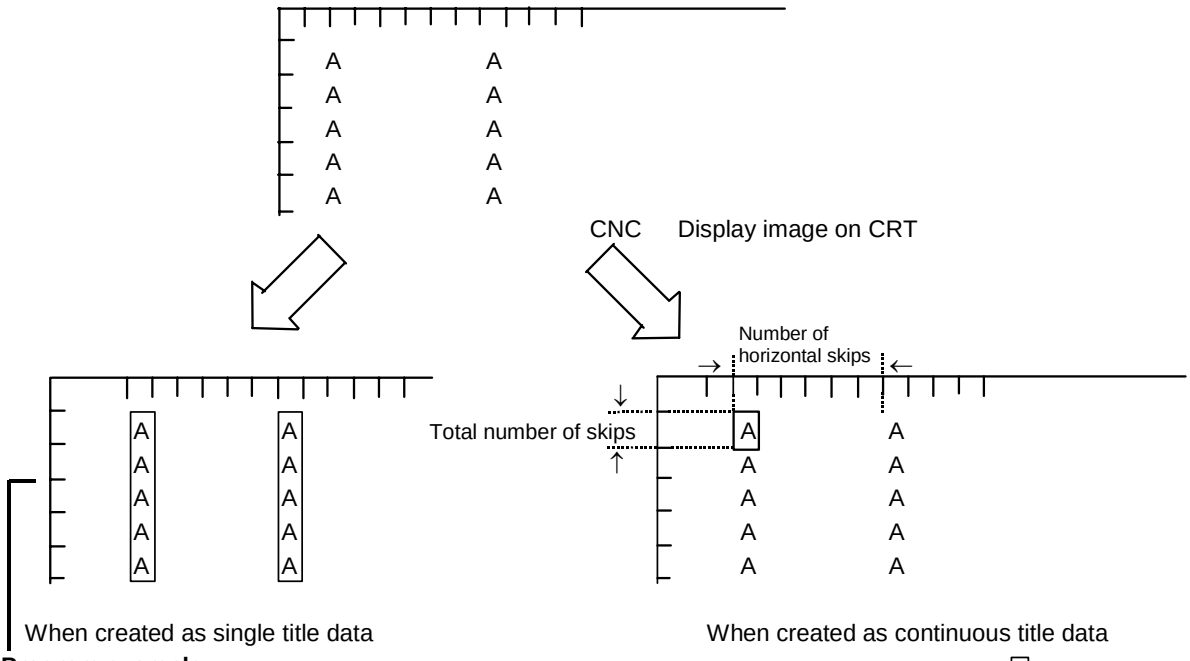




In the above case:

Number of horizontal skips	— 6
Number of vertical skips	— 2
Display direction	— Vertical
Number of displays	— 3
Number of data items	— 5
Number of bytes to next data	— 3
(Number of displays (direction)	— 3)

The following figure shows a comparison of single title data and continuous title data.



Program example

```
char   dsamp0[]  =  {“AAAAA”};
char   dsamp1[]  =  {“AAAAA”};
TXDATA   samp[]  =  {5,1,C_WK|NORMATR,83,0x00,0x00,(char   **)dsamp0,
                     5,1,C_WK|NORMATR,89,0x00,0x00,(char   **)dsamp1,
                     0};
```

Example of single title data

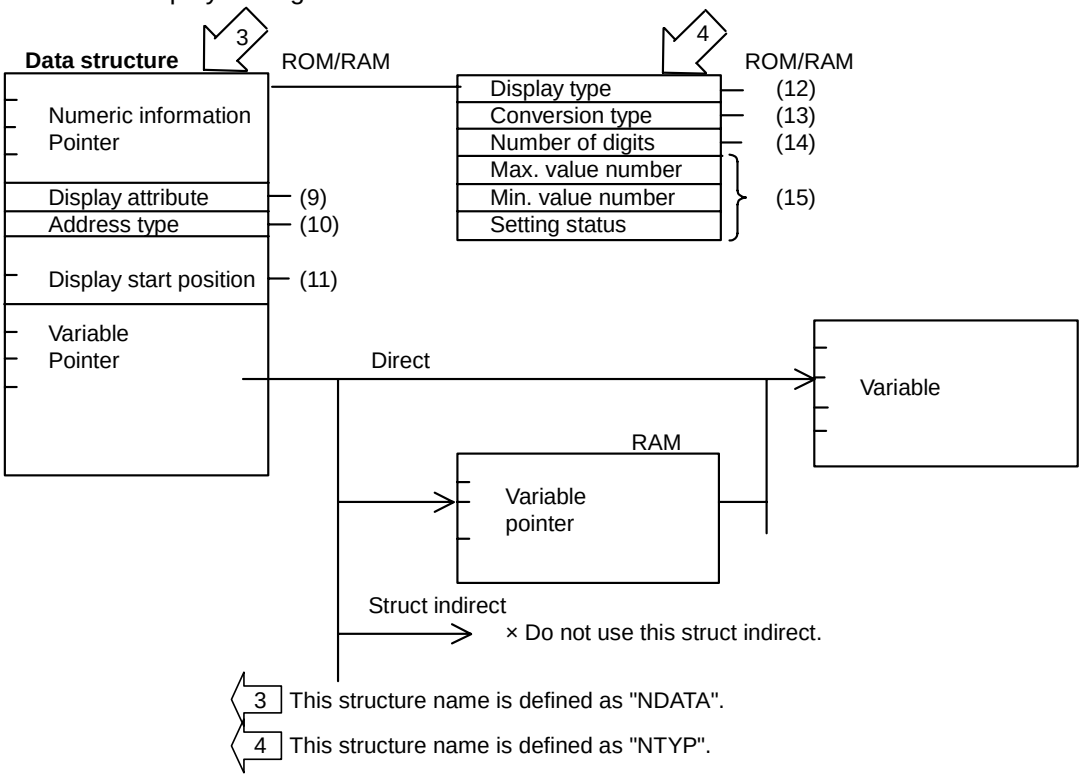
```
char   dsamp0[]  =  {“A”};
CTXDAT   asamp[]  =  {1,1,C_WK|NORMATR,83,0x00,0x00,(char   **)dsamp0,6,1,TATEDIR|5,10,0,
                      0};
```

Example of continuous title data

Comparison of single title data and continuous title data

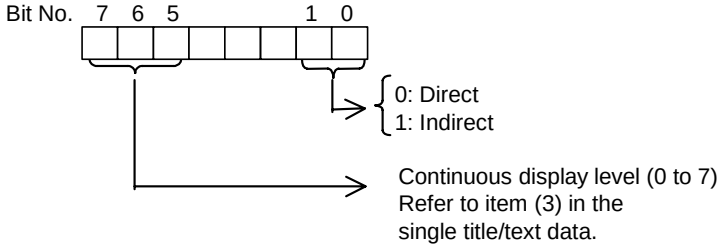
NDTYPE

Function: Display of single numeric data

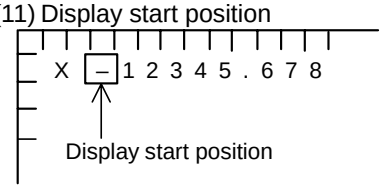


(9) Display attribute
Refer to the single title/text data.

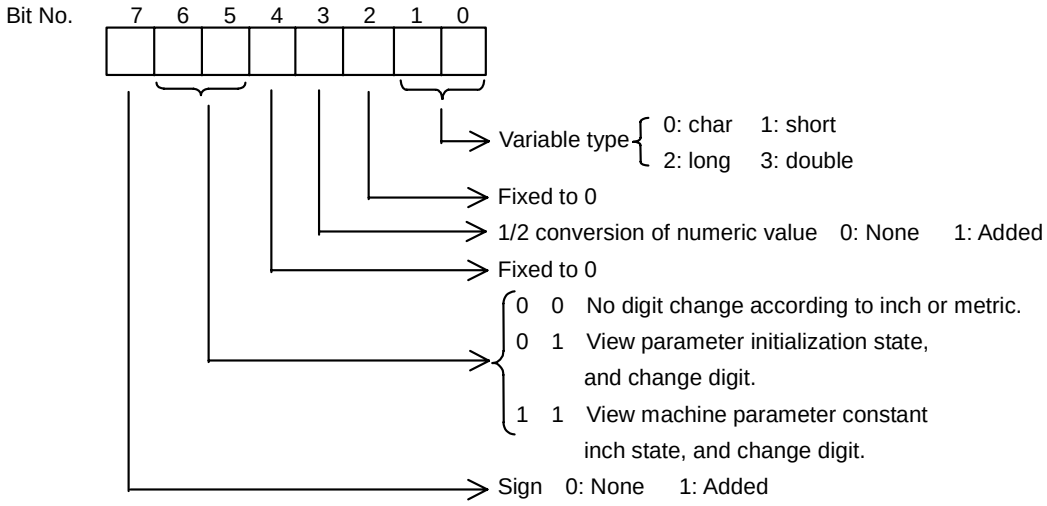
(10) Address type



(11) Display start position



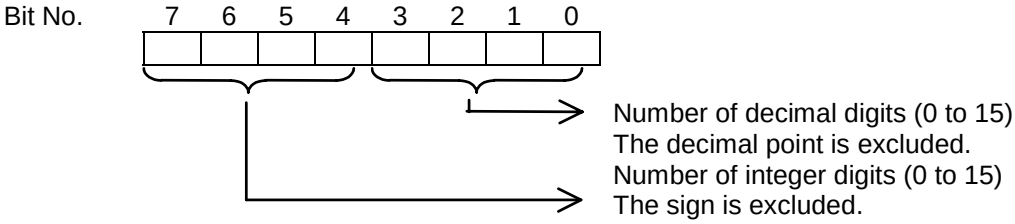
(12) Display type



☆ The digit change according to inch or metric indicates the following:
When metric is designated for the numeric value with 5 digits in the integer section and 3 digits in the decimal section, the value is converted for inches with 4 digits in the integer section and 4 digits in the decimal section according to the parameter state.
☆ Normally CNC axis data is numerically displayed as 0xAA.

(13) Conversion type
0: Binary display
1: Decimal display
2: Hexadecimal display
3: bit display
4: BCD display

(14) Number of digits

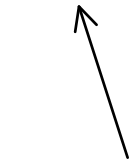


☆ If the conversion type is binary, hexadecimal, bit or BCD, the number of digits will be stored without being divided into the decimal section and integer section.

(15) Max. value number, min. value number and setting status
The max. value number and min. value number are fixed to 0.
The setting status is fixed to 0x00.

An example of the single numeric value data is shown below.

```
NTYP  tkpa12[]  =  {0x82,1,0x53,0,0,0x00};
NTYP  tkpa13[]  =  {0x22,1,0x53,0,0,0x00};
NTYP  tkpa14[]  =  {0x00,1,0x30,0,0,0x00};
NTYP  tkpa15[]  =  {0x22,1,0x53,0,0,0x00};
NTYP  tkpa16[]  =  {0x22,1,0x53,0,0,0x00};
NDATA nkpa1[]  =  {tkpa12,C_YKINORMATR,0x00,495,(char  *)&paramet.common.kkakudo,
                   tkpa13,C_YKINORMATR,0x00,736,(char  *)&paramet.common.scale,
                   tkpa14,C_YKINORMATR,0x00,982,(char  *)&paramet.common.ovride,
                   tkpa15,C_YKINORMATR,0x00,1056,(char  *)&paramet.common.angle,
                   tkpa16,C_YKINORMATR,0x00,1136,(char  *)&paramet.common.inn,
                   0};
```

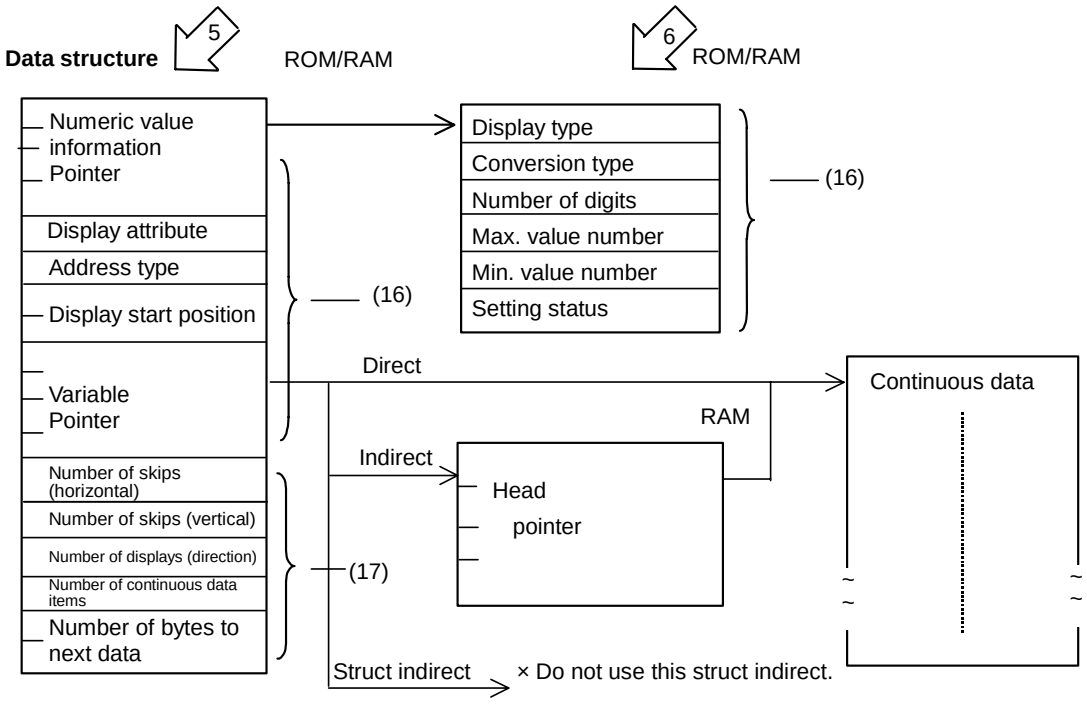


Variable pointer
Take care to "&"

Example of single numeric value display data

CNTYPE

Function: Display of continuous numeric data



5 This structure name is defined as "CNDATA".
6 This structure name is defined as "NTYP".

(16) Refer to the single numeric data.
(17) Refer to the continuous title/text data.

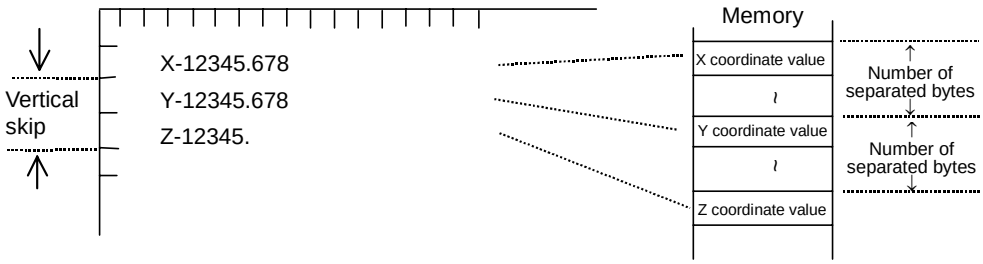


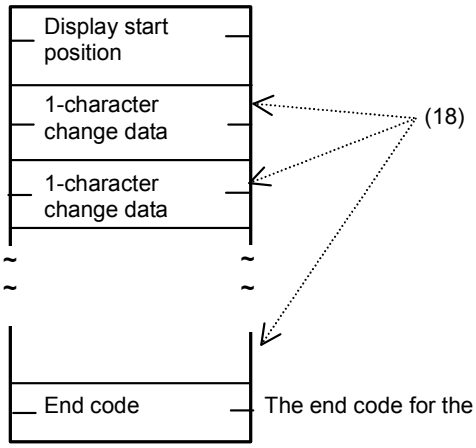
Image of continuous numeric data

The continuous numeric data is used when all the following conditions are satisfied as shown above:
The display position of the numeric data on the CRT has a set pattern (vertical skip, horizontal skip).
The attributes such as the display color are the same.
The address storing the numeric data has a set interval from the top.

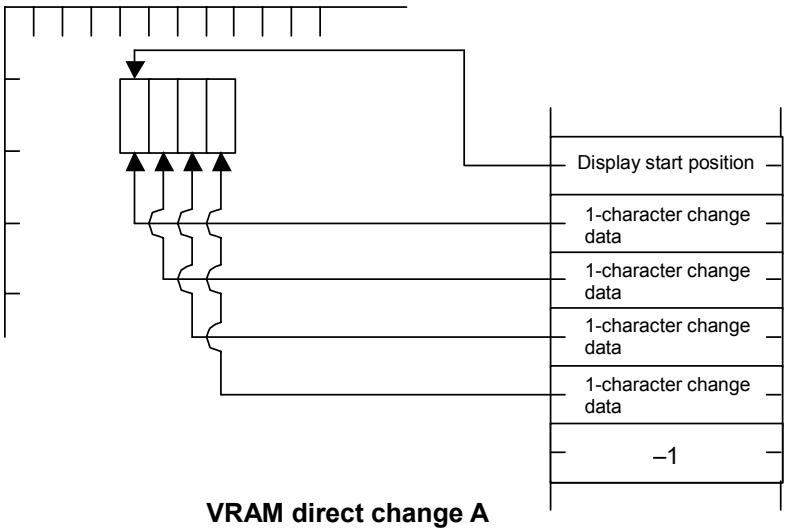
VRATYPE

Function: VRAM direct change A data

Data structure



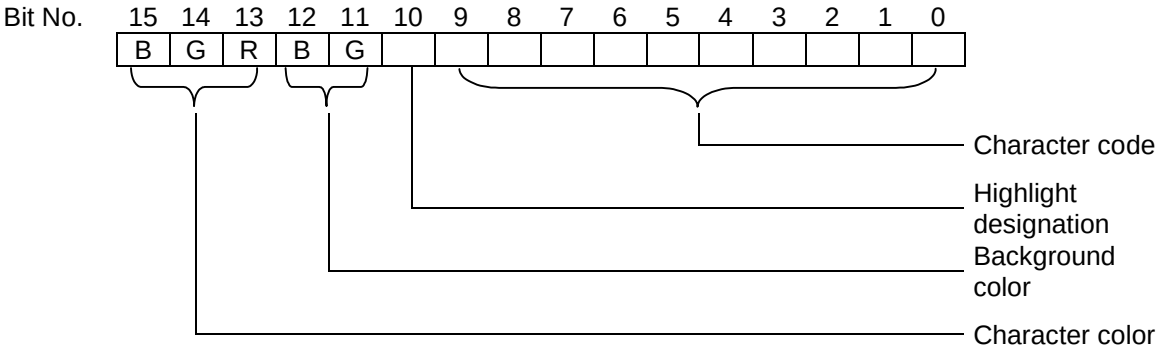
The VRAM direct change A data is used to rewrite the characters displayed on the CRT as shown below. The data is configured of the 1-character change data between the display start position and end code (–1).



VRAM direct change A

(18) 1 character change attribute

<Color CRT>

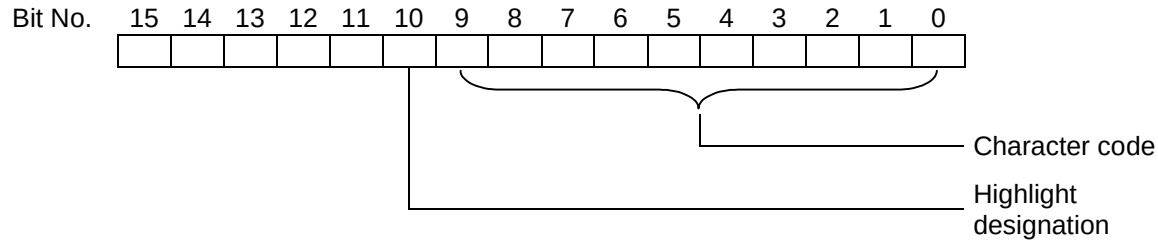


Character color	B	G	R
Black	0	0	0
Red	0	0	1
Green	0	1	0
Yellow	0	1	1
Blue	1	0	0
Magenta	1	0	1
Cyan	1	1	0
White	1	1	1

Background color	B	G
Black	0	0
Green	0	1
Blue	1	0
Cyan	1	1

☆ When highlight designation is set to 1, the color designated for the characters will be displayed as the background color, and the color designated for the background will be displayed as the character colors.

<Monochrome CRT>



☆ For the 1-character change data, if all bits corresponding to the character color and background color are 0, the display color will not be changed. If all bits corresponding to the character codes are 0, the character will not be changed.

(Note)

If the character color and background color is set to 0, the display color will not be changed.
If the character code is 00, the characters will not be changed.

An example of the VRAM direct change A data is shown below.

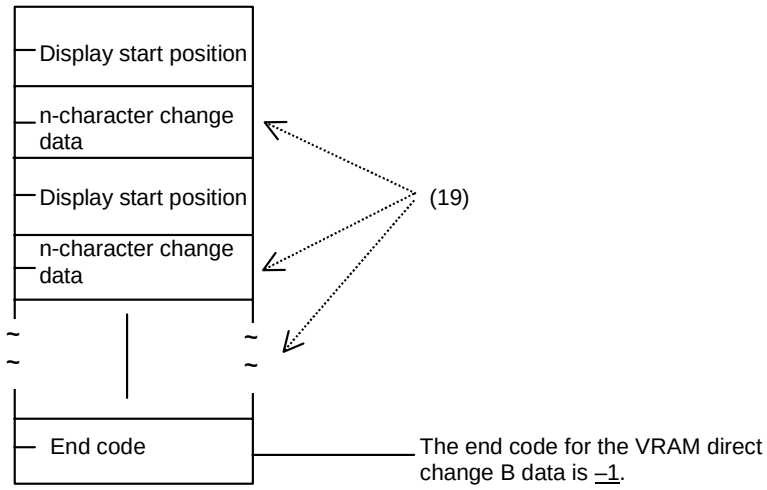
```
short vrama [] = {81,0xa041,0xa042,0xa043, -1};
```

End code

VRBTYPE

Function: VRAM direct change B data

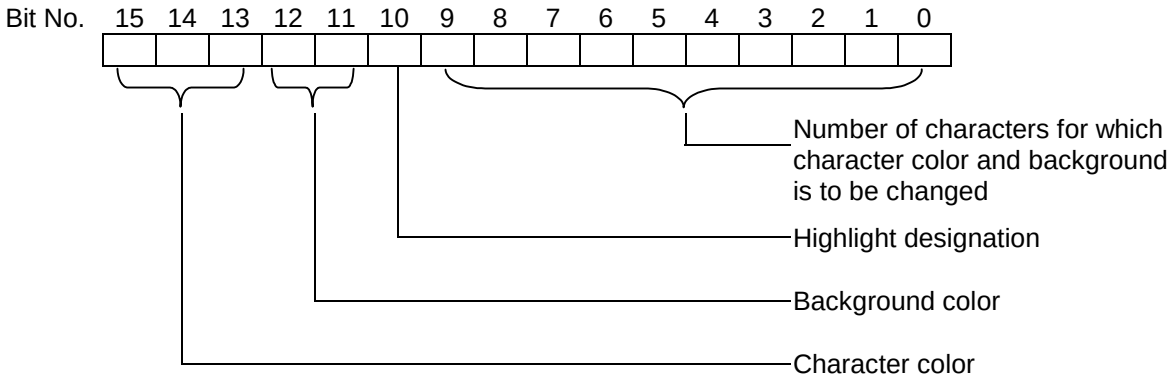
Data structure



The VRAM direct change B data is used to change the colors of the displayed characters and background on the CRT. The data is configured of the display start position and n-character change data up to the end code (–1).

(19) n-character change data

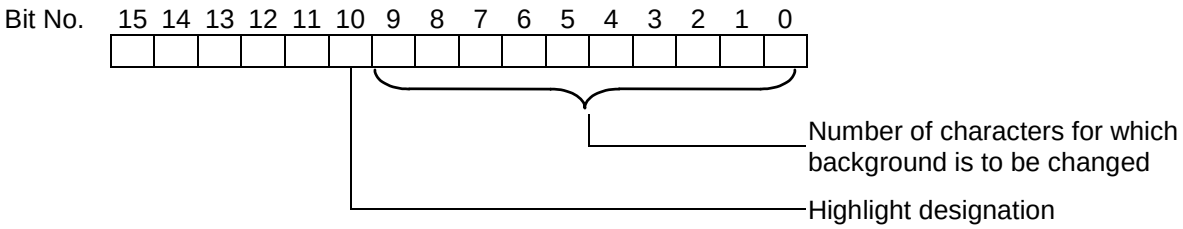
<Color CRT>



Character color	B	G	R
Black	0	0	0
Red	0	0	1
Green	0	1	0
Yellow	0	1	1
Blue	1	0	0
Magenta	1	0	1
Cyan	1	1	0
White	1	1	1

Background color	B	G
Black	0	0
Green	0	1
Blue	1	0
Cyan	1	1

<Monochrome CRT>



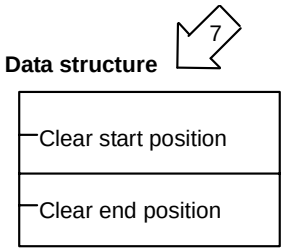
An example of the VRAM direct change B data is shown below.

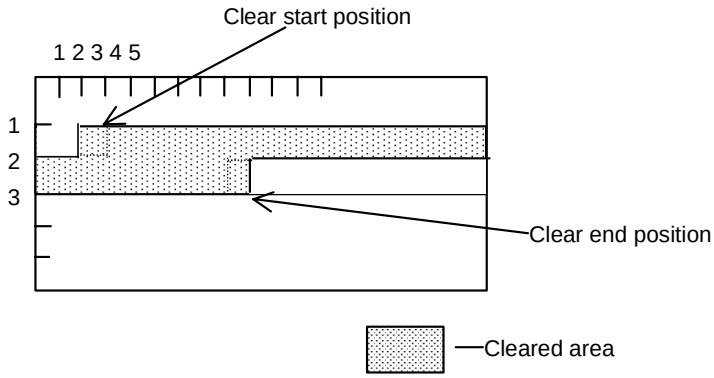
```
short vramb[] = {81,0xa005,161,0xa005,241,0xa005,-1};
```

End code

CLTYPE

Function: Clearing of data





7 This structure name is defined as "CDATA".

An example of the clear data is shown below.

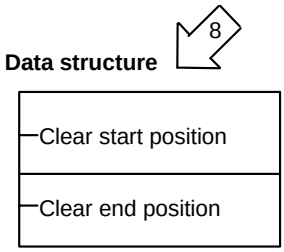
```
CDATA erase[ ] = {5, 105, 0} ;
```

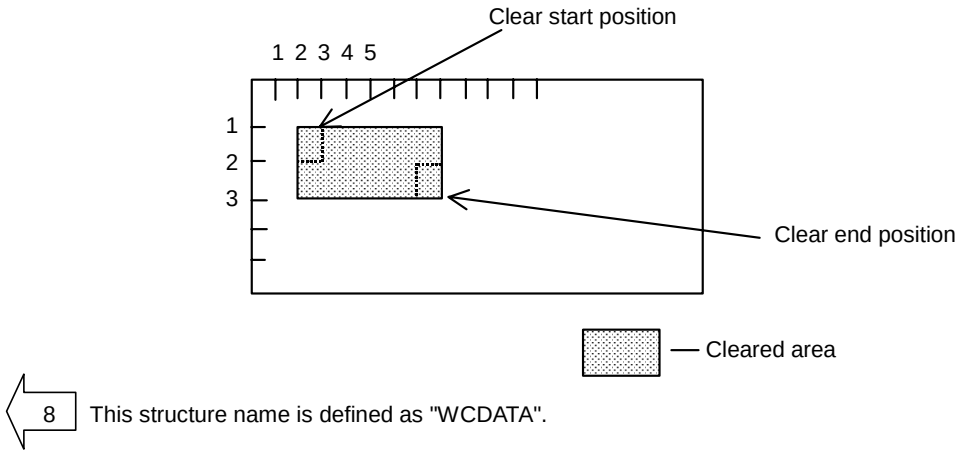
The above clear data can be written as follows.

```
short erase[ ] = {5, 105, 0} ;
```

WCLTYPE

Function: Clearing of data in window





An example of the clear window data is shown below.

```
WCDATA werase[] = {5, 105, 0} ;
```

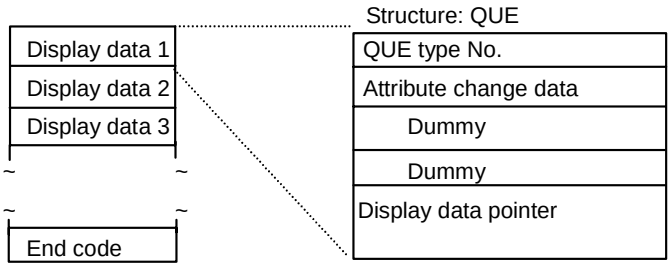
The above clear window data can be written as follows.

```
short werase[] = {5, 105, 0} ;
```

INDTYPE

Function: Indirect display request
When the display request function is called once, multiple display requests are carried out.

Data structure



QUE type No.
Attribute change data
Display data pointer

} Refer to the display request function (enquet) explanation for details.
Set 0 for all when they are the end code.

Example

```
static QUE inddat[] = {  NDTYPE, 0, 0, 0, ntork,
                         CTTYPE, 0, 0, 0, atork,
                         CNTYPE, 0, 0, 0, ctork,
                         TXTYPE, 0, 0, 0, tork,
                                 0, 0, 0, 0, 0};
enquet (pcoptb.ocb.scrnumb, INDTYPE, 0, inddat, 1L);
```

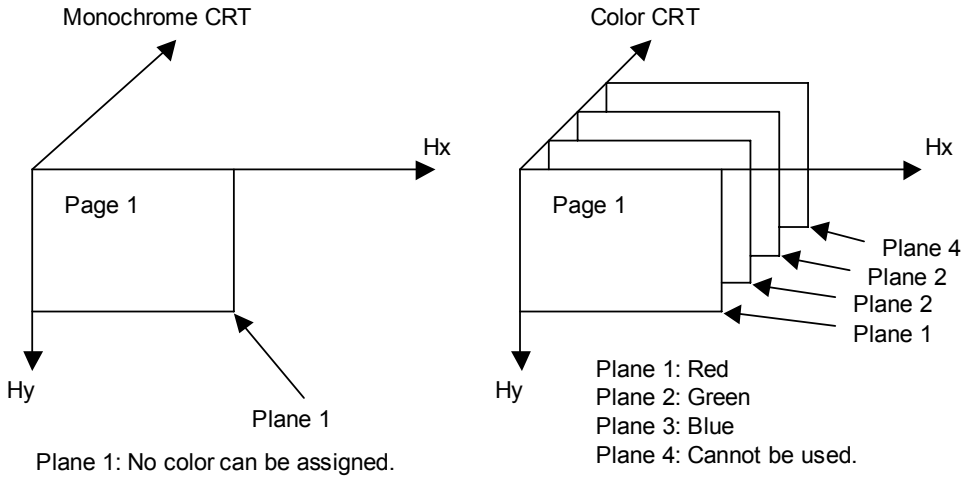
★ This command can also be used with the graphics drawing function (Chapter 1.3).

1.3 Graphics Drawing Functions

The specifications for drawing graphics with custom release functions are described in this chapter.

[Graphics environment]

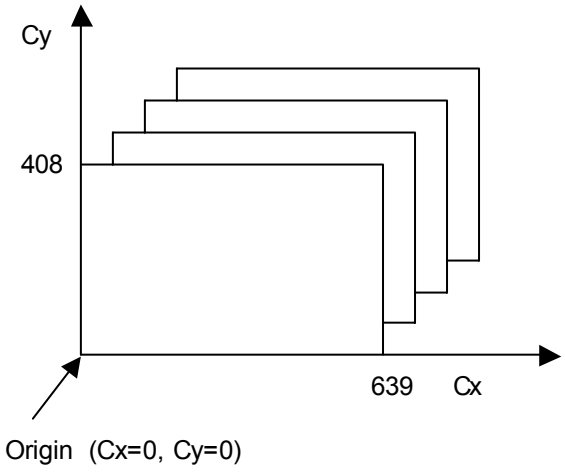
(1) Plane and page configuration



Plane 1: No color can be assigned.

(2) Graphics environment for custom release

The environment when drawing graphics with custom release is as follows.



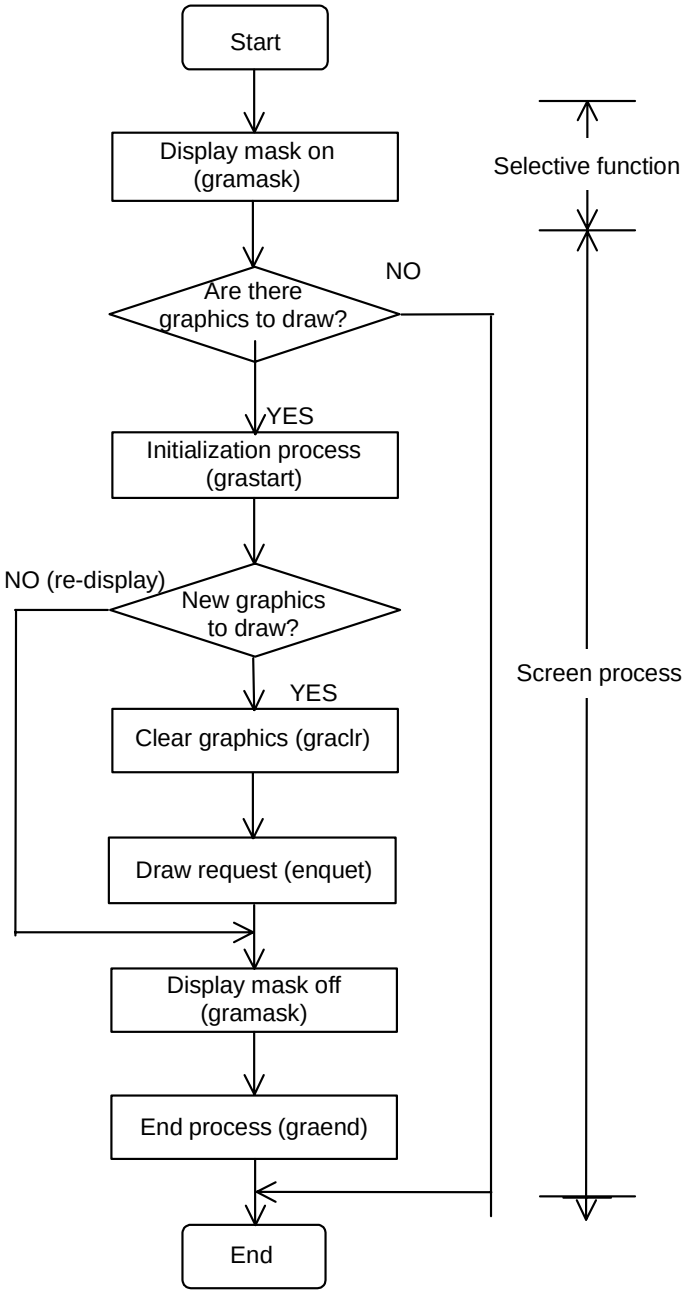
Item	Color	Monochrome
Origin setting	Lower left (Hx=0, Hy=1635)	
Color designation setting (During graphic drawing)	0: Black 1: Red 2: Green 3: Yellow 4: Blue 5: Magenta 6: Cyan 7: White	None

For color: Plane 1 to 3
For monochrome: Plane 1

(Note) The coordinate system shown in (1) is the hardware coordinate system (Hx, Hy), and the coordinate system actually drawn is the display coordinate system (Cx, Cy).

[Graphics Drawing Procedure]

Graphics are drawn with the following procedure.
(The flow is that from when the key is pressed to when the graphic is drawn.)



The functions used in each item are described on the following pages.

1.3.1 grastart ( ) Initialization of Environment for Graphics

Function: The graphic environment is initialized for custom release use.
Always call this function before drawing graphics with custom release.
The details set with this function are as follow.

Setting item	Setting details (common for color/monochrome)
Page setting	Page 1
Origin setting	Lower left of screen (Coordinate: Hx=0, Hy=1635)
Graphic valid area	Page 1 (one page)
Graphic mask	All page display/write masks off.

(Note) The planes targeted for masking are as follow.
For monochrome: Plane 1
For color: Planes 1 to 3

Calling sequence: grastart ( ) ;

Note:

(1) Do not call this function from the "M_OPE" initializing function "mopeini ()".
(2) Graphic drawing uses page 1 with the custom release.
If graphics are drawn without calling this function, the display may not be correct, and the CNC screen may be affected.

1.3.2 gramask ( ) Graphic Mask Control

Function: The mask is set by designating the mask type (display mask) and the plane.

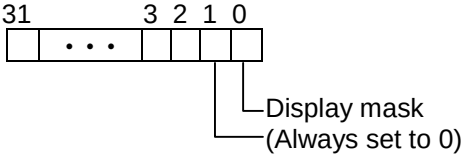
* Display mask: Displaying designated plane's graphics is prohibited.

Main applications: The display mask can be controlled to quit the display of the screen while holding the contents drawn in the graphic memory.
On screens that do not use graphics, the display of graphics can be prohibited by turning the display mask ON.

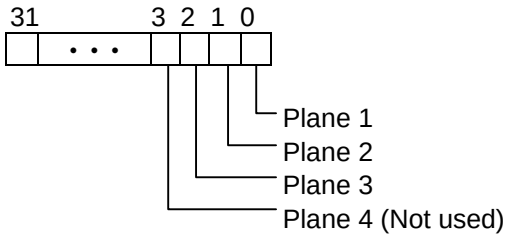
Calling sequence: ret = gramask (mode, plane) ;

long ret; Return status: 0 Normal completion
–1 Time out error
(The QUE is full so the request was invalidated.)

long mode; Mask type:



long plane; Plane designation:



The mask is turned ON/OFF with the plane designation.
The mask ON process is carried out to the planes for which the bit is ON. (Prohibition of display)
(In the plane for which the bit is OFF, the mask is turned OFF, and the display mask is enabled.)

Example

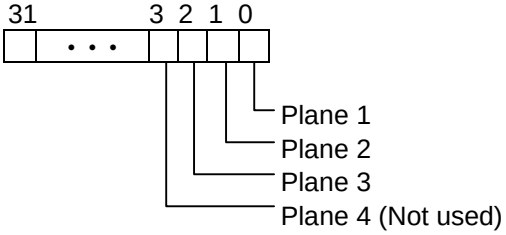
Example 1) To turn display mask for planes 1, 2 and 3 ON
ret = gramask (1L, 7L);
Example 2) To turn display mask OFF for only plane 1
ret = gramask (1L, 0L);

1.3.3 graclr ( ) Clearing of Graphics

Function: The graphics in the designated plane are cleared.

Calling sequence: ret = graclr (plane, startx, starty, lengx, lengy);

long plane; Plane designation:



long startx; Clear start X point coordinate : Clear start X point coordinate from origin
Set as "startx" = 0

long starty; Clear start Y point coordinate : Clear start Y point coordinate from origin
Set as "starty" = 0

long lengx; X direction clear width : X coordinate width from clear start point of clear area
Set as "lengx" = 640

long lengy; Y direction clear width : Y coordinate width from clear start point of clear area
Set as "lengy" = 409

1.3.4 enquet ( ) Request of Drawing Graphics

Function: The drawing of various graphics on the CRT is requested. Differing types of graphics can be drawn according to the type of command designated.
The graphic commands are shown below.

Command list

Command name	Function
GLBCP	Sets current pointer.
GLBSLS	Sets the line type and drawing plane.
GLBALIN	Absolute value line drawing
GLBRPLN	Multiple linked line drawing
GLBCRCL	Circle drawing
GLBAARC	Absolute value arc drawing
GLBAMLN	Multiple unlinked line drawing
GLBAMAC	Multiple unlinked arc drawing

Calling sequence: ret = enquet (id, type, flag, datptr, windid) ;

long id;	Possessory right ID	: Specify "pcoptb.ocb.scrnumb".
char type;	Command name	: Refer to the command list.
char flag;	Attribute change data	: Specify 0.
char *datptr;	Display data pointer	: Pointer for graphic data to be displayed
long windid;	Window ID	: Specify 1.

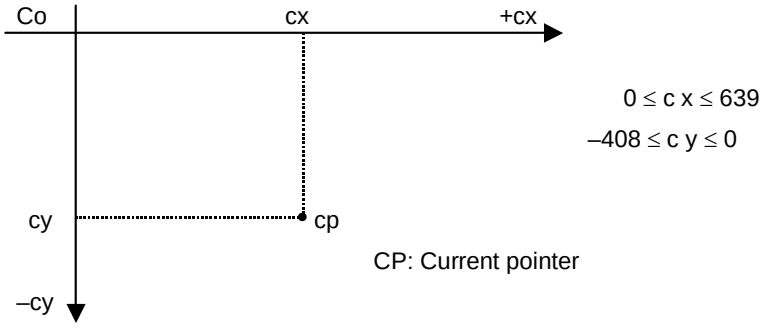
Program example

To draw a red solid line from coordinates (450, 300) to coordinates (550, 450).

```
long cptbl[] = {450,300};
long slstbl[] = {0,1};
long alintbl[] = {550,450};
enquet (pcoptb.ocb.scrnumb,GLBCP,0,cptbl,1L);      <-- Set drawing start point
enquet (pcoptb.ocb.scrnumb,GLBSLS,0,slstbl,1L);    <-- Designate drawing line type and color (plane)
enquet (pcoptb.ocb.scrnumb,GLBALIN,0,alintbl,1L);
```

GLBCP Setting of drawing start point

Function: Sets the drawing start point (Current pointer) at the point (cx, cy) on the display coordinate system.

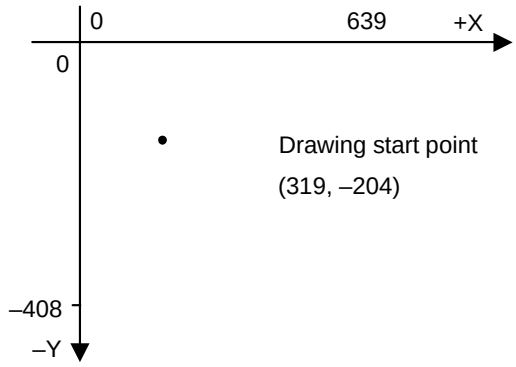


Data format: long table [] = {cx, cy} ;

cx: X axis drawing start point (display coordinate system)
cy: Y axis drawing start point (display coordinate system)

Program example

```
long cntlin [] = {319, –204} ;
enquet (pcoptb.ocb.scrnumb, GLBCP, 0, cntlin, 1L) ;
```



GLBSLS Designation of line type and color

Function: Designates the type of line and plane (color) to be drawn with the draw command.
The designated line type and color information are retained until commanded again with this command.

Data format: long table [] = {line, plane}

line	0 Solid line	· · · · ·	
	1 Broken line	· · ○○ · ·○○	(Every two dots)
	2 Dot-dashed	· · · · · · · · · ·○○· ·○○	
	3 Dotted line	· · · · · ·○○○○○○· · · · · ·	(Every six dots)
	4 Not used	Disappears from the background. Used to clear the line.	
	5 Broken line		
	6 Dot-dashed		
	7 Dotted line		

plane
0 Black
1 Red
2 Green
3 Yellow
4 Blue
5 Magenta
6 Cyan
7 White

☆ When using a monochrome CRT, the figure will be drawn with a line type no matter which plane (color) is designated.
(The color designation is invalid.)

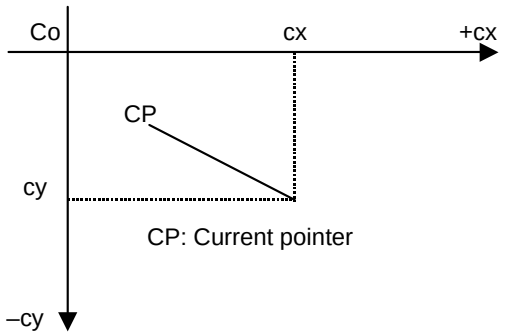
Program example

```
long pdada [] = {0, 3}
enquet (pcoptb.ocb.scrnumb, GLBSLS, 0, lpdata, 1L) ;
```

After the above program is executed, the line drawn will be a yellow solid line.

GLBALIN Draw absolute value line

Function: Draws a line from the current pointer to the end point (cx, cy) on the display coordinate system.



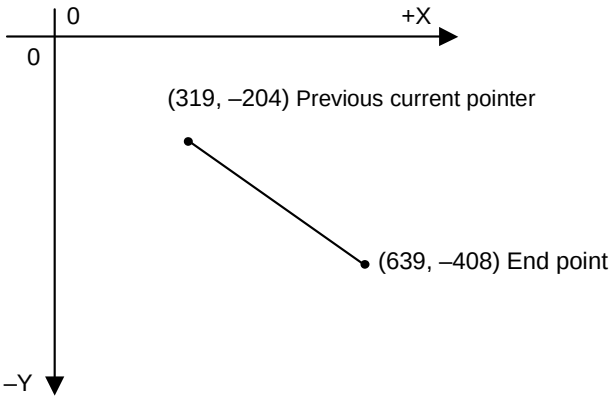
After drawing, the current pointer (CP) will move to the end point.
The line type and color are assigned with "GLBSLS".

Data format: long table [] = {cx, cy} ;

cx: X axis end point (display coordinate system)
cy: Y axis end point (display coordinate system)

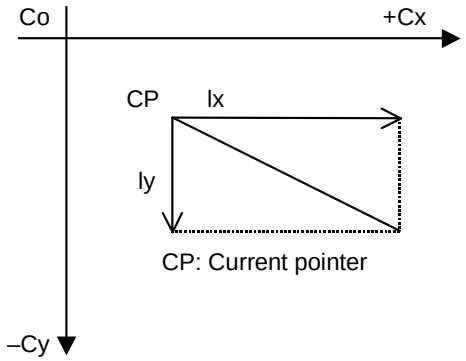
Program example

```
long cntlin [] = {639, -408} ;
enquet (pcoptb.ocb.scrnumb, GLBALIN, 0, cntlin, 1L) ;
```



GLBRLIN Draw relative value line

Function: Draws a line from the current pointer to a relative value on the display coordinate system.



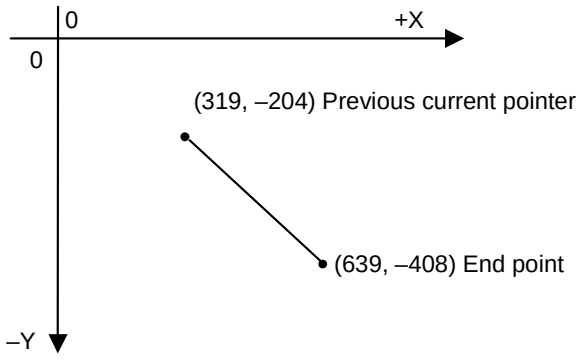
After drawing, the current pointer (CP) will move to the end point.
The line type and color are assigned with "GLBSLS".

Data format: long table [] = {lx, ly} ;

lx: X axis relative value from current pointer
ly: Y axis relative value from current pointer

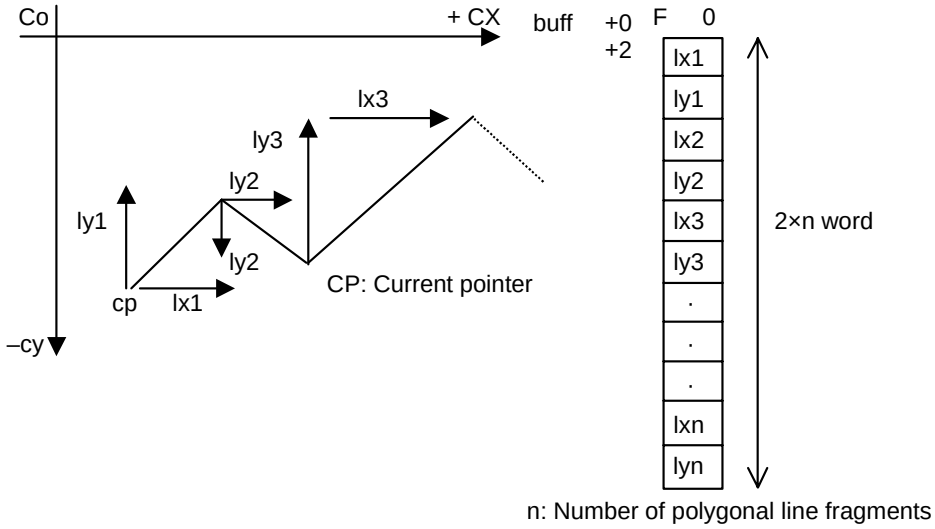
Program example

```
long lbrlin [] = {320, –204} ;
enquet (pcoptb.ocb.scrnumb, GLBRLIN, 0, lbrlin, 1L) ;
```



GLBRPLN Draw polygonal line

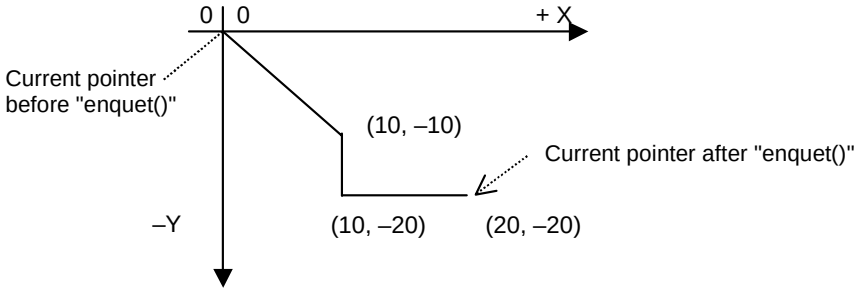
Function: Draws a polygonal line on the display coordinate system using the current pointer as a start point.
The number of polygonal line fragments and the shift amount designated with a relative value are assigned.



After drawing, the current pointer (CP) will move to the end point.
The line type and color are assigned with "GLBSLS".

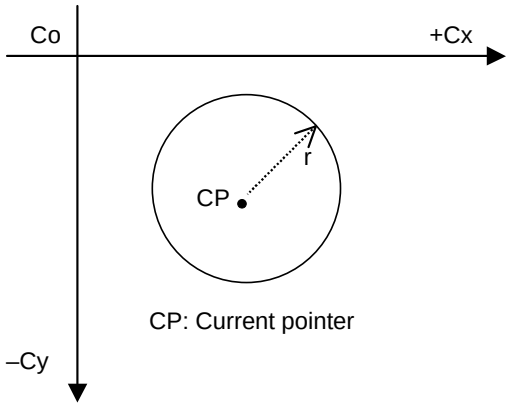
Data format:
```
typedef struct {
        long datan ;
        long *dataptr ;
    } NBUFF ;
    long buff [] = {lx1, ly1, lx2, ly2, ..., lxn, lyn} ;
    NBUFF table [] = {n, buff} ;
    n       : Number of polygonal line fragments
    buff    : Shift amount table pointer
```

Program example
```
typedef struct {
  long datan ;
  long *dataptr ;
} NBUFF ;
long zeroX [] = {10, –10,
                  0, –10,
                 10,   0} ;
NBUFF crossd [] = {3, zeroX} ;
enquet (pcoptb.ocb.scrnumb, GLBRPLN, 0, crossd, 1L) ;
```



GLBCRCL Draw circle

Function: Draws a circle (radius r) on the display coordinate system using the current pointer as the center.



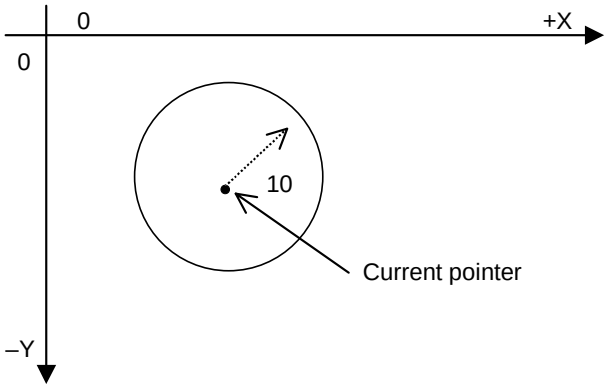
After drawing, the current pointer does not move.
The line type and color are assigned with "GLBSLS".
The circle is drawn in the clockwise direction.

Data format: long table [] = r ;

r: Radius

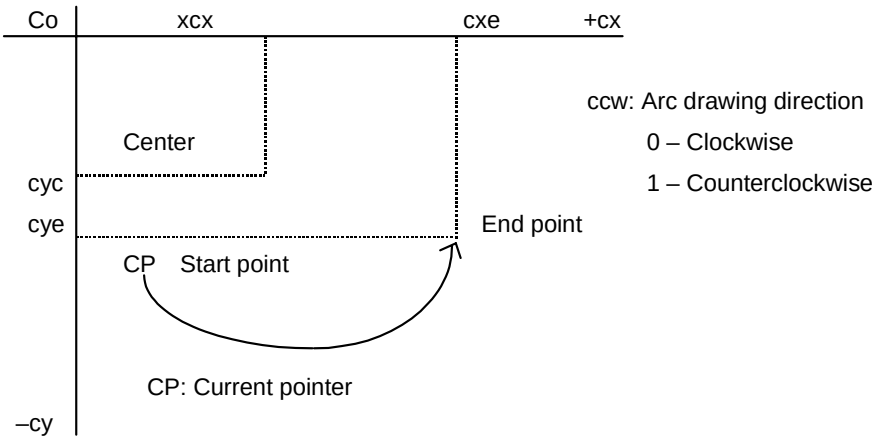
Program example

```
long zpntr [] = 10 ;
enquet (pcoptb.ocb.scrnumb, GLBCRL, 0, &zpntr, 1L) ;
```



GLBAARC Draw arc

Function: Draws an arc on the display coordinate system using the current point as the start point. The arc is drawn toward the end point (cxe, cye) in the direction designated with the ccw parameter. The center is (cxc, cyc)



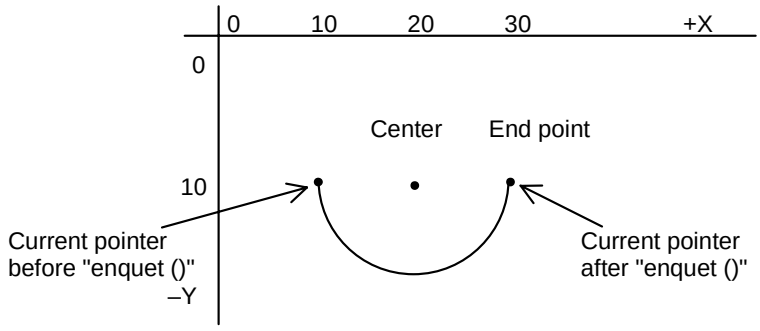
After drawing, the current pointer will move to the end point.
The line type and color are assigned with "GLBSLS".
If the current pointer and the end point are the same, a circle will be drawn.

Data format: long table [] = {cxc, cyc, cxe, cye, ccw} ;

cxc : X axis arc center value (CRT coordinate)
cyc : Y axis arc center value (CRT coordinate)
cxe : X axis arc end point (CRT coordinate)
cye : Y axis arc end point (CRT coordinate)
ccw : Arc drawing direction

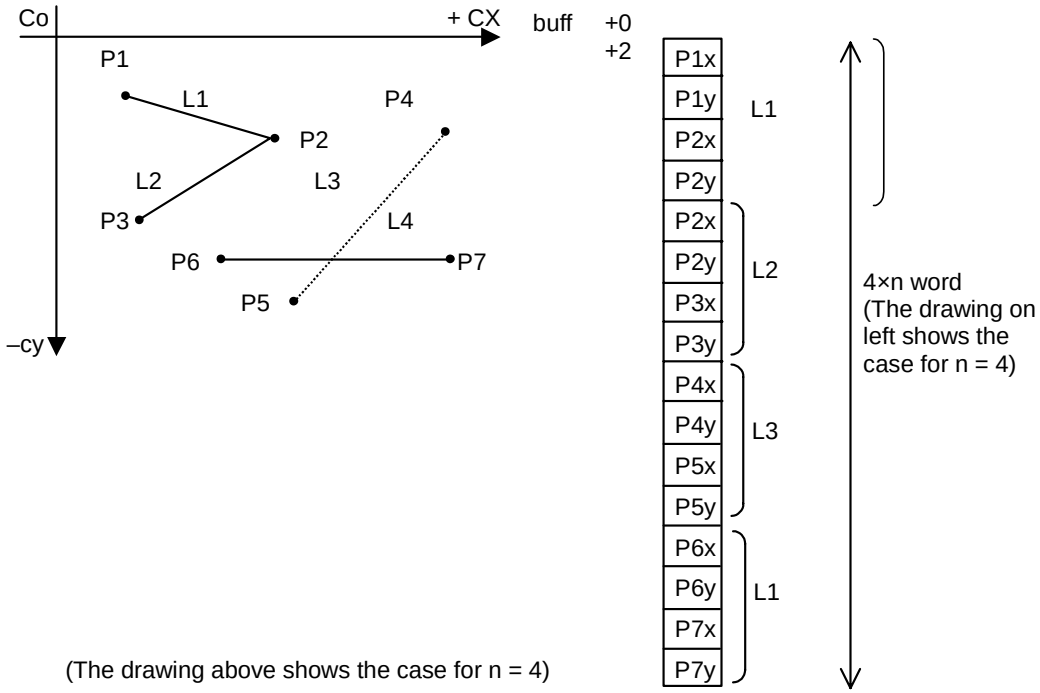
Program example

```
long clipof [] = {20, –10, 30, –10, 1} ;
enquet (pcoptb.ocb.scrnumb, GLBAARC, 0, cliplf, 1L) ;
```



GLBAMLN Draw unlinked line

Function: A line (segment) is drawn according to the start point and end point assigned with buff on the display coordinate system.
The number (n) of lines (segments) to be drawn is also assigned as drawing data.



(The drawing above shows the case for n = 4)

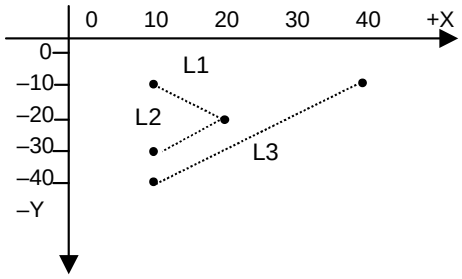
The coordinates are assigned with absolute values.
After drawing, the current pointer moves to the end point of the line (segment) drawn last.
The line type and color are assigned with "GLBSLS".

Data format:
```
typedef struct {
  long datan ;
  short *dataptr ;
} NBUFF2 ;
short buff [] = {p1x, p1y, p2x, p2y, ..., p7x, p7y} ;
NBUFF2 table [] = {n, buff} ;
```

n: Number of lines (segments)
buff: Coordinate value table pointer
p1x, p1y, ... p7x, p7y: Coordinate values (display coordinate system)

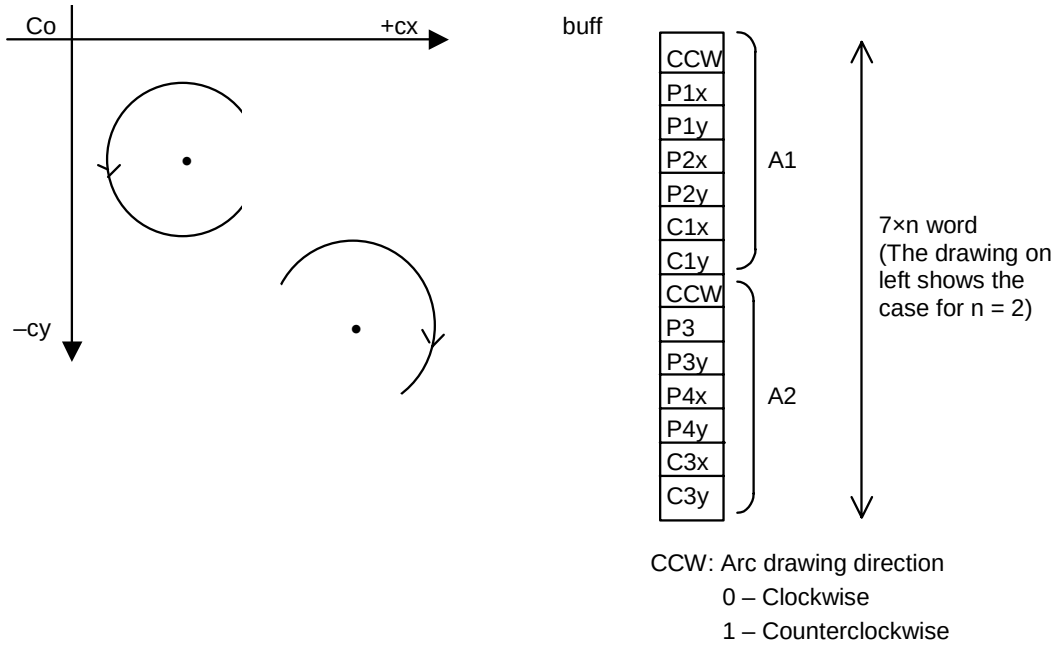
Program example

```
typedef struct {
  long datan ;
  short *dataptr ;
} NBUFF2 ;
short gidlin [] = {10, –10, 20, –20,   /* L1*/
                   20, –20, 10, –30,   /* L2*/
                   10, –40, 40, –10} ; /* L3*/
NBUFF2 gpttol [] = {3, gidlin} ;
enquet (pcoptb.ocb.scrnumb, GLBAMLN, 0, gpttol, 1L) ;
```



GLBAMAR Draw unlinked arcs

Function: An arc is drawn according to the arc rotation direction, start point, end point and center point values assigned with buff **on the display coordinate system**.
The number (n) of arcs to be drawn is also assigned as drawing data.
The coordinates are assigned with absolute values.



CCW: Arc drawing direction
0 – Clockwise
1 – Counterclockwise

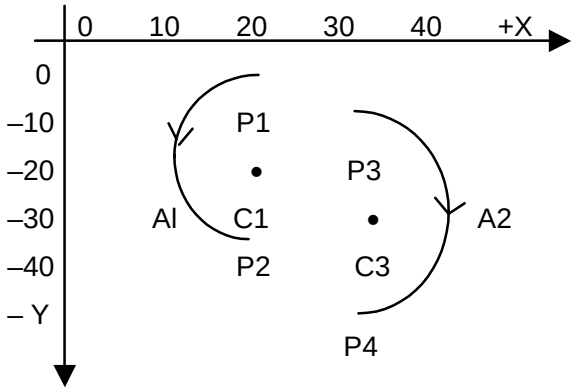
After drawing, the current pointer moves to the end point of the arc drawn last.
The line type and color are assigned with "GLBSLS".
If the arc's start point and end point are the same, a circle will be drawn.

Data format:
```
typedef struct {
   long datan ;
   short *dataptr ;
 } NBUFF2 ;
short buff [] = {ccw, p1x, p1y, p2x, p2y, c1x, c1y
                 , ccw, p3x, p3y, p4x, p4y, c3x, c3y} ;
NBUFF2 table [] = {n, buff} ;
```

n: Number of arcs to be drawn
buff: Coordinate value table pointer
ccw: Arc drawing direction 0 – Clockwise
1 – Counterclockwise
pKx, pKy (K = 1, 3, 5, ...) : Coordinates of arc draw start point
pLx, pLy (L = 2, 4, 6, ...) : Coordinates of arc draw end point
cMx, cMy (M = 1, 3, 5, ...) : Coordinates of arc center

Program example

```
typedef struct {
  long datan ;
  short *dataptr ;
} NBUFF2 ;
short mardata [] = {1, 20, –10, 20, –30, 20, –20,
                    0, 30, –15, 30, –45, 30, –30} ;
NBUFF2 lbamar [] = {2, mardata} ;
enquet (pcoptb.ocb.scrnumb, GLBAMAR, 0, lbamar, 1L) ;
```



1.3.5 graend ( ) End of Graphics

Function: Ends the graphics drawing request.
Always call this function when all graphics drawing requests have been completed.

Calling sequence: graend () ;

Program example:

```
gramask (1,0x0f);                                          Display mask ON
grastart ();                                               Initialize graphics
graclr (0x0f,sx,sy,lx,ly);                                 Clear graphics
enquet (pcoptb.ocb.scrnumb,GLBCP,0,cptbl,1L);
enquet (pcoptb.ocb.scrnumb,GLBSLS,0,slstbl,1L);            Drawing request
enquet (pcoptb.ocb.scrnumb,GLBALIN,0,alintbl,1L);
gramask (1,0x00);                                          Display mask OFF
graend ();                                                 End of graphics
```

1.4 Screen Display Auxiliary Functions

List of functions

1.	Check of display completion	dspend ()
2.	Menu highlight	smenhi ()
3.	Menu display (no language changeover)	smenud ()
4.	Reset of display request	squrst ()
5.	Clear text data from screen	texers ()
6.	Read setting area control data information	ikeyset ()
7.	Set cursor position data	ocurini ()
8.	Set setting area control data	omakkcb ()
9.	Clear setting area buffer	ostclr ()
10.	Display setting area title data	setdisp ()
11.	Setting area data control	skey ()
12.	Cursor control	cursor ()
13.	Set 40-character mode screen	scrst40 ()
14.	Set 80-character mode screen	scrst80 ()

1.4.1 dspend () Check of Display Completion

Format: dspend ( ) ;

Example:
```
enquet (pcoptb. ocb. scrnumb,  TXTYPE, 0,  ce02, 1L) ;
enquet (pcoptb. ocb. scrnumb,  CTTYPE, 0,  ace02, 1L) ;
enquet (pcoptb. ocb. scrnumb,  CNTYPE, 0,  cce02, 1L) ;
enquet (pcoptb. ocb. scrnumb,  CTTYPE, 0,  alin0, 1L) ;
dspend ( ) ;
```
Display request

Wait for display completion

When this function is called, the completion return will not occur until the display of all requested displays is completed.

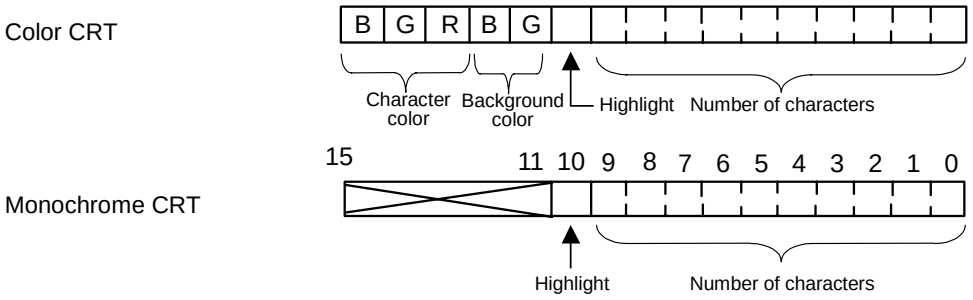
1.4.2 smenhi () Menu Highlight

Format:
```
short menatr1 [] = {start,
                    atrcnt,
        End code ──────► –1} ;
```
Always create the same number of menu highlight data items as the number of menus.

```
short menatr2 [] = {start,
                    atrcnt,
                          –1} ;
```
(This is the display data structure of VRAM direct change B)

```
.
.
.
short *table [] = {menatr1,
                   menatr2,
                      .
                      .
                      .      } ;
smenhi (table, menuno) ;
```

menuno : Menu number to be highlighted. (0 for menu 1, 4 for menu 5)
start : Highlight display start position
atrcnt : n-character change data



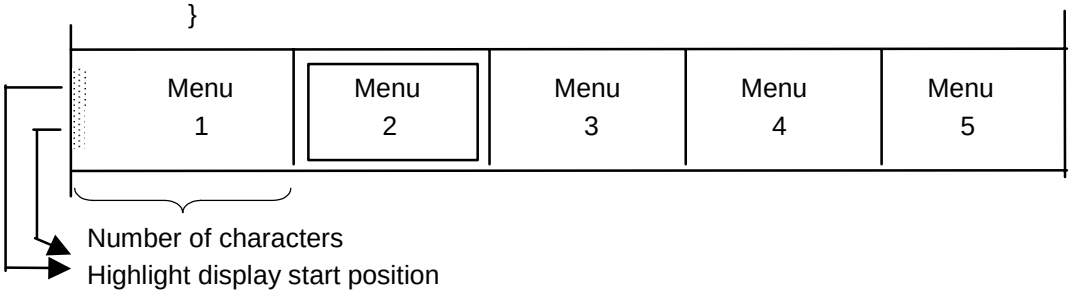
Example

Always create the same number of menu highlight data items as the number of menus.

```
short   menatr1[] = {1841, 0x0400 | 8,     /* menu 1 atr change data */
                     1921, 0x0400 | 8,–1};
short   menatr2[] = {1850, 0x0400 | 8,     /* menu 2 atr change data */
                     1930, 0x0400 | 8,–1};
short   menatr3[] = {1859, 0x0400 | 8,     /* menu 3 atr change data */
                     1939, 0x0400 | 8,–1};
short   menatr4[] = {1868, 0x0400 | 8,     /* menu 4 atr change data */
                     1948, 0x0400 | 8,–1};
short   menatr5[] = {1877, 0x0400 | 8,     /* menu 5 atr change data */
                     1957, 0x0400 | 8,–1};
```

Always register the same number of menu highlight data addresses in this table as the number of menus.

```
short   *menatrp[] = { menatr1,
                       menatr2,
                       menatr3,
                       menatr4,
                       menatr5 };

menuhi ()
{
    smenhi(menatrp, pcoptb.ocb.menuno);     /* menu reverse   */
    pcoptb.ocb.mncflag &= 0xFFEF;           /* bit4 OFF       */
}
```



Note: To return to the highlight display made last time, execute "smenud ()" (menu display) before "smenhi ()".

1.4.3 smenud () Menu Display

Format: TXDATA *menutx [] = {menutxa,
menutxb,
.
.
. } ;

short menuers [] = {start, end, 0} ;

smenud (menutx, mnuers, index) ;

menutx: Menu display data address table
menutxa:
menutxb: } Menu display data

meners : Menu area clear data
start : Menu area clear start position
end : Menu area clear end position

index: Index value of menu display data address table

Example

Menu display data / Always create as single text data.

```
TXDATA men1 [] = {16,16,C_WK|NORMATR,1282,0x00,0x00,(char **)dmen10,
                  0};
TXDATA men2 [] = {50,25,C_WK|NORMATR,1282,0x00,0x00,(char **)dmen20,
                  8,8,C_WK|NORMATR,1353,0x00,0x00,(char **)dmen21,
                  0};
TXDATA men3 [] = {16,16,C_WK|NORMATR,1344,0x00,0x00,(char **)dmen30,
                  17,17,C_WK|NORMATR,1281,0x00,0x00,(char **)dmen31,
                  0};
TXDATA men4 [] = {16,16,C_WK|NORMATR,1344,0x00,0x00,(char **)dmen40,
                  8,8,C_WK|NORMATR,1281,0x00,0x00,(char **)dmen41,
                  0};
```

Menu frame data

```
CTXDATA men0 [] = {3,1,C_WK|NORMATR,1209,0x00,0x00,(char **)dmen00,5,1,YOKODIR14,4,0,
                   8,8,C_WK|NORMATR,1201,0x00,0x00,(char **)dmen01,5,1,YOKODIR15,5,0,
                   0};
```

Menu display data address table

```
                                          /* menu TXDATA address table     */
TXDATA *menutx[]={men1,                   /* menu block 0 */
                  men2,                   /* menu block 1 */
                  men3,                   /* menu block 2 */
                  men4 };                 /* menu block 3 */
```

Menu area clear data

```
short meners [] = {80 * 15 + 1, 80 * 18, 0};          /* menu erase    */
```

```
mendsp ()
{
```

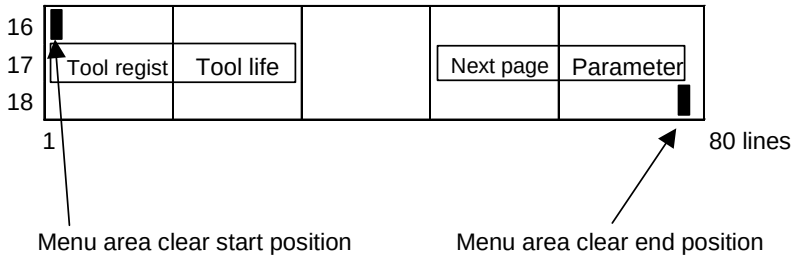
Menu display

```
    smenud(menutx, meners, pcoptb.ocb.mnblno);     /* menu display   */
```

Menu frame display

```
    enquet(pcoptb.ocb.scrnumb, CTTYPE, O, meno, 1L);
                                                    /* waku display     */
    pcoptb.ocb.mncflag &= 0xFFF7;                   /* bit3 OFF         */
}
```

Display example)



1.4.4 squrst () Reset of Display Request

- Resets the display request in the Continuous-display-Que table and once-display-Que table.

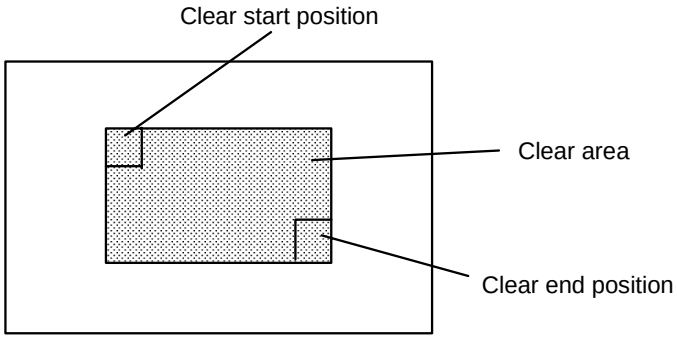
Format: squrst ( ) ;

Example: Program as follows to clear the display area from a continuous display area.

```
mscrcl ()
{
  static short  texclr[] = {1, 40, * 18, 0};    /* screen clear data      */

  squrst ();                                    /* display Que reset      */   } Resets all display requests.

  texers (texclr);                              /* screen clear           */   } Clears data from text screen.

  graclr (7L,0L,0L,640L,409L);                  /* graphic clear          */   } Clears data from graphic screen.

  cursor (pcoptb.ocb.scrnumb, 0xFE, 0, 0x8000, 1L);
                                                /* cursor OFF             */   } Clears cursor 1.

  pcoptb.ocb.mncflag &= 0xFFFD;                 /* bit1 OFF               */

}
```

1.4.5 texers () Clear Text Data from Screen

Format: short table [] = {start, end, 0} ;
texers (table) ;
start: Clear start position
end : Clear end position



Example: short scrclr [] = {1, 720, 0} ;
texers (scrclr) ;
Clear data from the entire screen (40-character mode).

1.4.6 ikeyset () Obtain number of setting area, start and end positions, and setting area buffer pointer where cursor 1 is currently displayed.

Syntax: numb = ikeyset (start, end, textpt);

long numb:	Setting area number	0: No setting area has been set. 1 to 32: Setting area number
long *start:		
long *end :	Setting area start position data pointer Setting area end position data pointer (Start, end positions: 0 to 719 (1439))	
char **textpt:	Setting area buffer pointer	

Example:

```
                    char *textpt;
                    long start;
                    long end;
register   long numb;
/* cursor is not in setting */
   if (!(numb = ikeyset (&start, &end, &textpt)))
      return;
```

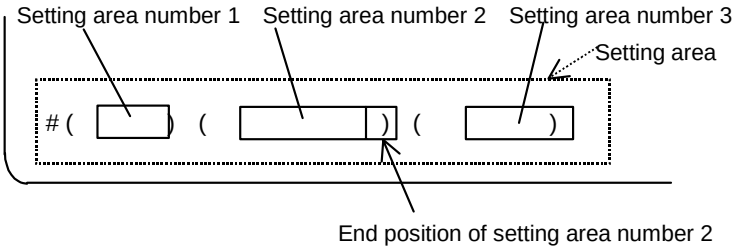
Note: Always create the setting area control information with "omakkcb ( )" before calling "ikeyset ( )".

1.4.7 ocurini () Set end position of designated setting area number at cursor 1 position (pcoptb.kcbtblr.cursor).

Format: ocurini (number) ;
long number ; Setting area number (1 to 32)

Example: ocurini (2L) ;

Set end position of setting area number 2 at "pcoptb.kcbtblr.cursor".



Note:
- If an illegal value (for example, 3L for screen with two text items, or 0L) is designated for the setting area number, the cursor will turn off.
  (In this case, the "pcoptb.kcbtblr.cursor" value will be 0 x FFFF.)
- Always create the setting area control information with "omakkcb ( )" before calling "ocurini ( )".

1.4.8 omakkcb () Create setting area control information kcbtblr from setting area information table kcbtbl.

Format: omakkcb (index);
long index : Index value of setting area information table having setting area information to be displayed.

Example:

dkcb.c table (Setting area title/text information)

```
KCBTBL kcbtbl [] = {7,–1,blif1,flif1,0L
,2,–1,btork,btork,0L ──► This setting area is displayed.
};
```

```
omakkcb  (1L) ;
setdisp    (  ) ;
```

Note: With a screen/function of a screen having a setting area, always use the support functions in the following order in the screen display phase.

```
omakkcb  (  ) ;
setdisp    (  ) ;
ocurini    (  ) ;
```

1.4.9 ostclr () Clear setting area buffer designated with setting area number.

Format: ostclr (numb) ;
long numb ; Setting area number (1 to 32)

Example: ostclr (1L) ;
Clear setting area buffer # 0.

Note: Always create the setting area control information with "omakkcb ( )" before calling "ostclr ( )".

1.4.10 setdisp () Display the setting area title with the setting area control information contents. The setting area text is displayed as a blank area. The setting area buffer contents are cleared.

Format: setdisp ( )

Example:

dkcb.c table (Setting area title/text information)

```
KCBTBL kcbtbl [] = {7,–1,blif1,flif1,0L
,2,–1,btork,btork,0L ──► This setting area is displayed.
};
```

```
omakkcb  (1L) ;
setdisp    (  ) ;
```

Note: Always create the setting area control information with "omakkcb ( )" before calling "setdisp ( )".

1.4.11 skey () Setting Area Processing Main

The following processes are carried out according to the pressed key.

(1) Setting area processing

Key type	Process
Input	The following processes will be carried out on the setting area containing the cursor. • All data in the setting area is right-justified, and a null character is added at the end. • If there is set data, the set character bit effective flag is set. • The cursor moves to the right end of the setting area where the cursor is located. • The add mode is entered.
Data	The following processes will be carried out on the setting area containing the cursor. • Add mode ... The data in the setting area buffer is shifted to the left, and then the key data is added. (1) When 2 is pressed: (1 2) • Insert mode ... The key data is added, and the data in the setting area buffer is shifted to the left. (1 2 3) When 4 is pressed: (1 4 2 3) • Replace mode ... The data in the setting area buffer is replaced with the key data. • If the cursor is in the right end of the setting area, the add mode will be entered. • If the cursor is not at the right end of the setting area, the cursor will shift to the right.
Cursor	When the cursor is in the setting area: The up/down keys will be invalid. <Left key> • If the cursor is not at the left end, it will shift one to the left, and then the replace mode will be entered. ( ) • If the cursor is at the left end, it will shift to the right end of the previous setting area, and then the add mode will be entered. ( ) ( ) <Right key> • If the cursor is not at the right end, it will shift one to the right, and then the replace mode will be entered. ( ) • If the cursor is at the right end, it will shift to the right end of the next setting area, and then the add mode will be entered. ( ) ( )

Key type	Process
Cancel	Data is cleared from all setting area buffers, and then the replace mode is entered.
Tab	When the cursor is in a setting area: Left tab key ..... The cursor shift to the right end of the previous column, and then the add mode is entered. Right tab key ..... The cursor shift to the right end of the next column, and then the add mode is entered. The position succeeding the end setting area is the head of the setting area. The position preceding the head setting area is the end setting area.
Delete	When the cursor is in a setting area, the character at the cursor position is deleted while right-justifying the remaining data. If the cursor is at the right end of the setting area, or the add mode is not entered, the replace mode will be entered.
Insert	When the cursor is in a setting area: • The cursor is shifted one to the left. (1 2 [3] 4 5) • The data before the cursor is shifted one to the left. ↙↙↙ • A blank is set at the cursor position. (1 2 [ ] 3 4 5) • The insert mode is entered. The above processes do not take place if the cursor is ([1] 2 3 4 5) at the left end or at the second position. (1 [2] 3 4 5)
Clear block	When the cursor is in a setting area: • All data is cleared from the designated setting area buffer. • The cursor moves to the right end of the setting area. • The add mode is entered.

(2) Display of data (text) in setting area buffer using "enquet ( )".

Display of current cursor position with "cursor ( )".

Format: skey ( ) ;

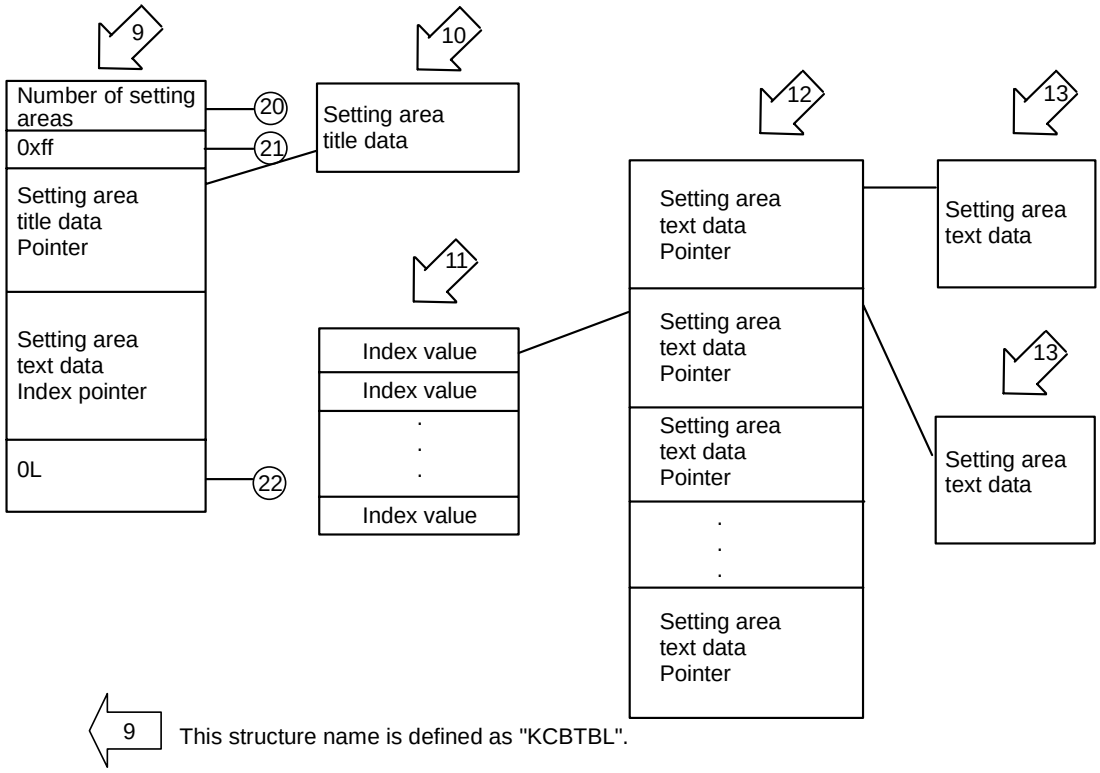
Example: skey ( ) ;

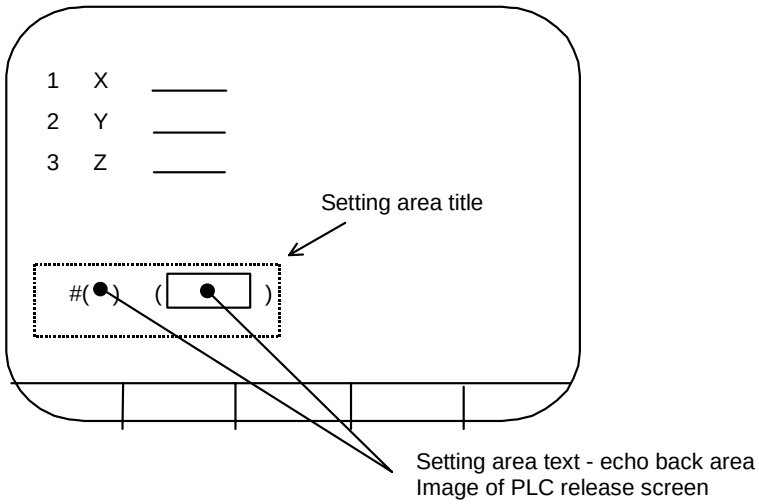
Note: Always create the setting area control information with "omakkcb ( )" before calling "skey ( )".

(3) KCB table data

The KCB table data is used to display the setting area.
The structure of this table is described in this section.
The following figure shows the setting area display data. This data contains the setting area title used to display fixed character string data (parentheses, etc.), and the setting area text used to echo-back the pressed key.

☆ When creating the setting area text data, the number of text data items must be 10 or less per screen. The size of the setting area text must be 1 column (vertical) × 31 columns (horizontal) or less.





(20) Number of setting area
This is the number of setting area texts displayed on the screen.

(21) 0xff (fixed).

(22) 0L (fixed).

10 The structure of the setting area title data is the same as the single title data structure.
Refer to page 5 "Display of single title/text data (TXDATA)".

11 This is a table for referring to the setting area texts displayed on one screen. In each screen, the number of areas in this table equals the number of setting areas.

12 This is configured of the pointers to the setting area texts of all release screens.

13 The structure of the setting area text data is the same as the single text data.
Refer to page 5 "Display of single title/text data (TXDATA)".

☆ The structure of the KCB table can be understood by actually referring to an example shown in the figure.

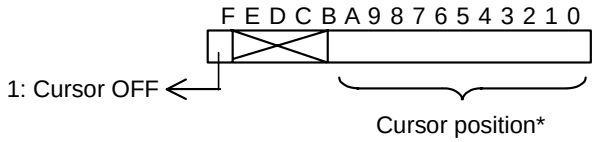
```
char hclpa25[] = {"# (__)"};
char hkpa110[] = {"# (__) DATA (___________)(___________)"};
char hwoff1[] = {"# (__) DATA (___________)"};
TXDATA bclpa[] = {5,5,C_SK|ORMATR,1281,0x00,0x00,(char **)hclpa25,
                  0};
TXDATA bkpal[] = {34,34,C_SK|ORMATR,1281,0x00,0x00,(char **)hkpa110,
                  0};
TXDATA bwoff[] = {21,21,C_SK|ORMATR,1281,0x00,0x00,(char **)hwoff1,
                  0};7
TXDATA xclpa26[] = {2,2,C_YK|NORMATR,1283,0x00,0x00,(char **)&pcoptb.settei.dvar0[0],
                  0};
TXDATA xkpal12[] = {10,10,C_YK|NORMATR,1292,0x00,0x00,(char **)&pcoptb.settei.dvar1[0],
                  0};
TXDATA xkpal13[] = {10,10,C_YK|NORMATR,1304,0x00,0x00,(char **)&pcoptb.settei.dvar2[0],
                  0};
TXDATA xwoff3[] = {10,10,C_YK|NORMATR,1291,0x00,0x00,(char **)&pcoptb.settei.dvar1[0],
                  0};
char fclpa[] = {0};
char fkpal[] = {0,1,2};
char fwoff[] = {0,3};
TXDATA *tset[] = {xclpa26,xkpa112,xkpa113
                  ,xwoff3
                  };
KCBTBL kcbtbl[] = {1,0xff,bclpa,fclpa,0L
                  ,3,0xff,bkpa1,fkpa1,0L
                  ,2,0xff,bwoff,fwoff,0L
                  };
```

Example of KCB table

1.4.12 cursor () Cursor Display Request

Format: cursor (id, type, (char)0, cursor1, 0) ;

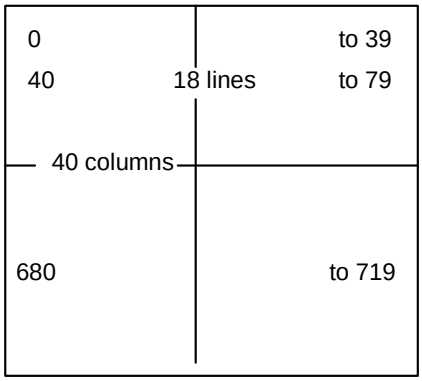
long id:	Possessory right ID (Designate as "pcoptb.ocb.scrnumb".)
char type:	que type number 0xFE: Cursor 1 is valid
short cursor1:	Cursor 1 position



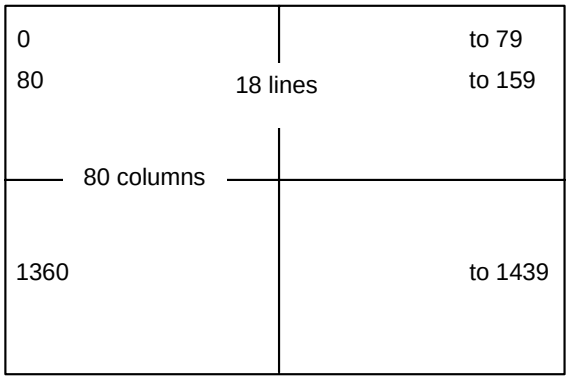
0 to 719 (40-character mode, 720 characters)
0 to 1439 (80-character mode, 1440 characters)

* The cursor position counts the top left end column as 0 as shown below.

40-character mode



80-character mode



Example: cursor (pcoptb. ocb. scrnumb, 0xFE, (char)0,
pcoptb. kcbtblr. cursor, 0) ;

Cursor 1 is displayed at the "pcoptb.kcbtblr.cursor" value.

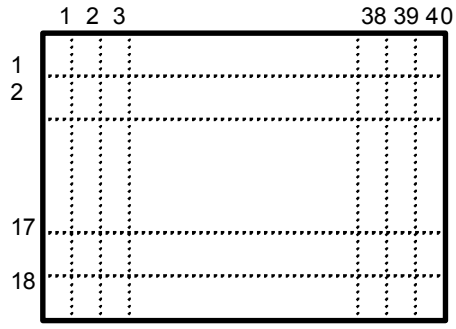
1.4.13 scrst40 () Set 40-character Mode Screen

Format: sts = scrst40 () ;

long sts; 0: Normal completion –1: Error completion

Function: Changes the screen display (character display) to the 40-character mode.

Example: scrst40 () ;



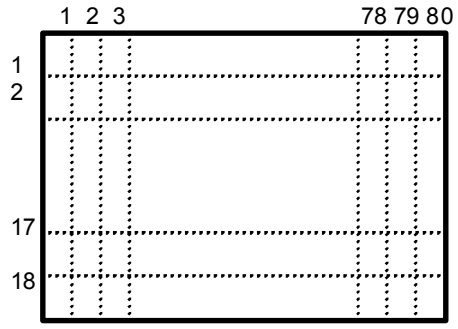
1.4.14 scrst80 () Set 80-character Mode Screen

Format: sts = scrst80 () ;

long sts; 0: Normal completion –1: Error completion

Function: Changes the screen display (character display) to the 80-character mode.

Example: scrst80 () ;



Example of program using scrst40() and scrst80()

<mseltb.c>

```
/************************************************************************************/
/*                                                          (M64 system)    */
/*  <NAME>          mseltb.c                                                */
/*                                                                          */
/*  <FUNCTION>      m_ope Selection function table                          */
/*                                                                          */
/*          <PROGRAM>  APLC                                                 */
/*                                                                          */
/*          COPYRIGHT (C) 1998 MITSUBISHI ELECTRIC CORPORATION  */
/*          ALL RIGHT RESERVED                                              */
/************************************************************************************/
/*
<Outline>
        The function table corresponds to the following bits
          Bit1: Display request reset
          Bit2: 40-column mode selection
          Bit3: 80-column mode selection
          Bit4: Clearing of screen display area

<In/Out>
   In    : –
   Out : –

<Machine>               M64             <OS>                    VxWorks
 <Type>                 NC              <Language>              C
 <CPU>                  R4000           <Compiler/Assembler>    multi
*/

/************************************************************************************/
/*      Include file                                                        */
/************************************************************************************/
/* ("define" for controlling the include file is defined here.)             */

/************************************************************************************/
/*      External reference                                                  */
/************************************************************************************/
/*---- Data declaration ------------------------------------------------------------*/

/*---- Function declaration --------------------------------------------------------*/

                                        /* User function        */
extern          void        mquerst() ;
extern          void        mode40() ;
extern          void        mode80() ;
extern          void        mscrcl() ;

/************************************************************************************/
/*    Data definition                   Only file defined here is used      */
/************************************************************************************/
/*---- Define declaration ----------------------------------------------------------*/

/*---- Data declaration ------------------------------------------------------------*/

/************************************************************************************/
/*    General-purpose function     Only file defined here is used          */
/************************************************************************************/

/************************************************************************************/
/*    Function body                                                         */
/************************************************************************************/
/**/
long (*mseltb[])() = {
            (long (*)())0L,                 /* BTON0:0x0001                                 */
            (long (*)())mquerst,            /* BTON1:0x0002: Display request reset          */
            (long (*)())mode40,             /* BTON2:0x0004: 40-column mode selection       */
            (long (*)())mode80,             /* BTON3:0x0008: 40-column mode selection       */
            (long (*)())mscrcl,             /* BTON4:0x0010: Clearing of screen display area */
            (long (*)())0L,                 /* BTON5:0x0020                                 */
            (long (*)())0L,                 /* BTON6:0x0040                                 */
            (long (*)())0L,                 /* BTON7:0x0080                                 */
            (long (*)())0L,                 /* BTON8:0x0100                                 */
```

```
        (long (*)())0L,                 /* BTON9:0x0200                 */
        (long (*)())0L,                 /* BTONA:0x0400                 */
        (long (*)())0L,                 /* BTONB:0x0800                 */
        (long (*)())0L,                 /* BTONC:0x1000                 */
        (long (*)())0L,                 /* BTOND:0x2000                 */
        (long (*)())0L,                 /* BTONE:0x4000                 */
        (long (*)())0L                  /* BTONF:0x8000                 */
} ;
```

<menublk.c>

```
/************************************************************************************/
/*                                                          (M64 system)    */
/*  <NAME>          menublk.c                                               */
/*                                                                          */
/*  <FUNCTION>      m_ope Screen decision table                             */
/*                                                                          */
/*          <PROGRAM>  APLC                                                 */
/*                                                                          */
/*          COPYRIGHT (C) 1998 MITSUBISHI ELECTRIC CORPORATION              */
/*          ALL RIGHT RESERVED                                              */
/************************************************************************************/
/*
<Outline>

                Menu 1    Menu 2    Menu 3    Menu 4    Menu 5

<In/Out>
   In    : –
   Out : –

<Machine>               M64             <OS>                    VxWorks
 <Type>                 NC              <Language>              C
 <CPU>                  R4000           <Compiler/Assembler>    multi
*/

/************************************************************************************/
/*    Include file                                                          */
/************************************************************************************/
/* ("define" for controlling the include file is defined here.)             */

                                        /* System                       */
#include "o_type.h"
#include "po_typ.h"

/************************************************************************************/
/*        External reference                                                */
/************************************************************************************/
/*---- Data declaration ------------------------------------------------------*/

/*---- Variable declaration --------------------------------------------------*/

                                        /* User function        */
extern          void        sc00(void) ;                /* Screen display      */

/************************************************************************************/
/*    Data definition                  Only file defined here is used       */
/************************************************************************************/
/*---- Define declaration ----------------------------------------------------*/

/*---- Data declaration ------------------------------------------------------*/

/************************************************************************************/
/*    General-purpose function     Only file defined here is used           */
/************************************************************************************/

/************************************************************************************/
/*    Function body                                                         */
/************************************************************************************/
/**/
/**/
MENUBLK menublk[] = {
/* mmcflag Control flag
bit0:
bit1:"mquerst" Display request reset
bit2:"mode40" 40-column mode selection
```

```
bit3:"mode80" 80-column mode selection
*/
/* Menu block 0 */
/* Control      Screen processing address   Next menu      Custom flag                          */
/* Flag                                     Block No.                                           */
/*mmcflag       mfuncp                      mnxtno         aftflag       Menu No.               */
 0x06,          (long (*)()) sc00,          –1,            0,            /* 0:[Screen display ] */
   0,           (long (*)()) –1,            –1,            0,            /* 1:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 2:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 3:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 4:[               ] */

/* Menu block 1 */
/*mmcflag       mfuncp                      mnxtno         aftflag       Menu No.               */
   0,           (long (*)()) –1,            –1,            0,            /* 0:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 1:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 2:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 3:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 4:[               ] */

/* Menu block 2 */
/*mmcflag       mfuncp                      mnxtno         aftflag       Menu No.               */
   0,           (long (*)()) –1,            –1,            0,            /* 0:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 1:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 2:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 3:[               ] */
   0,           (long (*)()) –1,            –1,            0,            /* 4:[               ] */


} ;

short           sbmbln    = (sizeof(menublk) / sizeof(MENUBLK)) ;          /* Number of menus set  */
```

<mode40.c>

```
/*****************************************************************************************/
/*                                                          (M64 system)                 */
/*  <NAME>          mseltb.c                                                             */
/*                                                                                       */
/*  <FUNCTION>      m_ope Selection function (bit2) 40-chracter mode selection           */
/*                                                                                       */
/*          <PROGRAM>  APLC                                                              */
/*                                                                                       */
/*          COPYRIGHT (C) 1998 MITSUBISHI ELECTRIC CORPORATION                           */
/*          ALL RIGHT RESERVED                                                           */
/*****************************************************************************************/
/*
<Outline>
        Set screen to 40-column mode
        Selective call flag OFF

<In/Out>
   In    : None
   Out : None

<Machine>               M64             <OS>                    VxWorks
 <Type>                 NC              <Language>              C
 <CPU>                  R4000           <Compiler/Assembler>    multi
*/

/*****************************************************************************************/
/*     Include file                                                                      */
/*****************************************************************************************/
/* ("define" for controlling the include file is defined here.)             */

                                        /* System                       */
#include "o_type.h"
#include "pcoptb.h"
#include "c_def.h"

/*****************************************************************************************/
/*        External reference                                                             */
/*****************************************************************************************/
/*---- Data declaration ------------------------------------------------------------*/
```

```
                                        /* Work                 */
extern    PCOPTB    pcoptb ;
/*---- Function declaration ----------------------------------------------------------*/


                                        /* System library       */
extern    void          scrst40() ;
/*****************************************************************************************/
/*    Data definition                  Only file defined here is used                    */
/*****************************************************************************************/
/*---- Define declaration ------------------------------------------------------------*/

/*---- Data declaration --------------------------------------------------------------*/

/*****************************************************************************************/
/*    General-purpose function     Only file defined here is used                        */
/*****************************************************************************************/

/*****************************************************************************************/
/*    Function body                                                                      */
/*****************************************************************************************/
/**/
void   mode40( void )
{
       scrst40() ;                              /* 40-column mode ON          */
       pcoptb.ocb.mncflag &= ~BTON2;            /* Selective call flag OFF (bit2)*/
}
```

2. DDB I/F

2.1 CNC Data Read/Write Functions

2.1.1 ddbrd (), ddbwt () CNC Data Read/Write Functions (Type 2)

1. Read function

sts = ddbrd (id1, id2, ddbsz, sysno, axisno, ground, &ddbbf) ;

2. Write function

sts = ddbwt (id1, id2, ddbsz, sysno, axisno, ground, &ddbbf) ;

long sts	Return value	0 Normal completion –1 No option –2 Size over –3 Axis number illegal –4 Section number illegal –5 Write protect –6 System number error
long id1 long id2	Section number Sub-section number	Refer to the "DDB Interface Manual" (BNP-B2214).
long ddbsz	Data size	1 – 1 byte 2 – 2 bytes 4 – 4 bytes
long sysno	System number	0 – 1st system 1 – 2nd system
long axisno	Axis number	0 – No axis 1 – 1st axis 2 – 2nd axis . . . 7 – 7th axis
long ground	Ground	0 – Fixed
long & ddbbf	Read/write data buffer address	

3. Usage example

Example) G53 offset

Section number	: 4
Sub-section number	: 272
Size	: 4 bytes
System	: 1st system
Axis	: 3rd axis
Ground	: 0

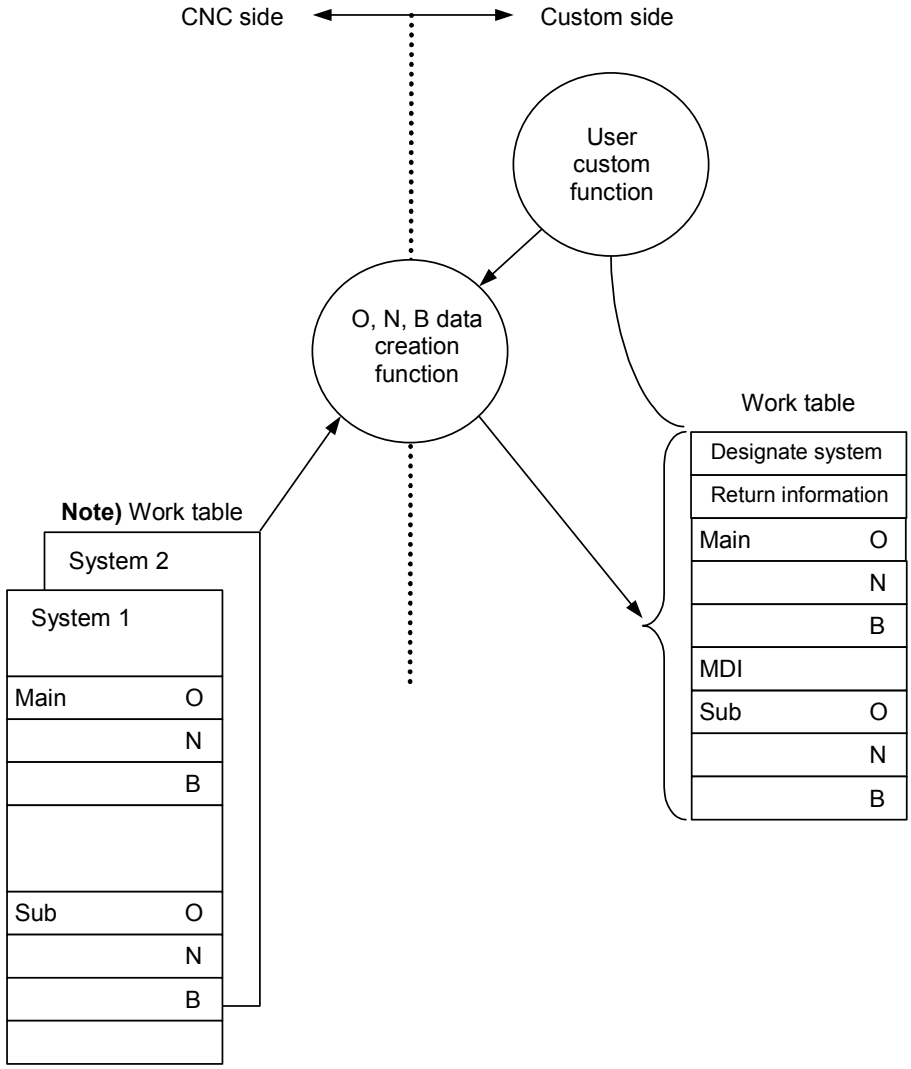
Set as follows to read the data.

```
long sts;     Return value
long ddbbf; Read buffer

sts = ddbrd (4, 272, 4, 0, 3, 0, &ddbbf) ;
```

2.1.2 smkonb () O, N, B Data Read

1. Outline

This is the custom release function that obtains the O (program number), N (sequence number) and B (block number) of the machining program being executed.
With this function, the O, N and B data of the main program being executed in one system, and the O, N and B data of the sub-program can be obtained.



Note) Even when using 2-system, only the data for 1-system will be read out.

2. Details

2-1 Function

The O, N and B data of the main program being executed in 1-system, and the O, N and B data of the sub-program is obtained.

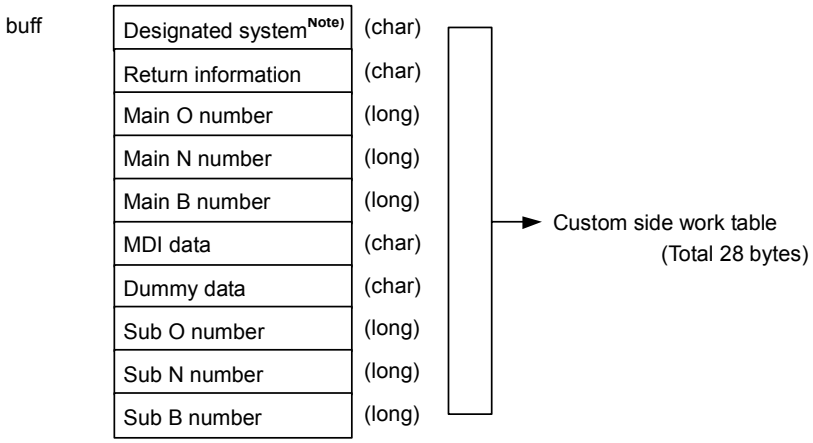
2-2 Calling sequence

(1) Function name

sts = smkonb (&buff) ;

(2) Argument

Top address of work table



Note) Set to 1.

(3) Calling order

1) Set the designated system in the work table. (Only 1-system)

2) Call the top address of the work table as the argument.
Example) smkonb (&buff) ;

3) The data will be set in the work table with the ONB data creation function.

(4) Contents of data

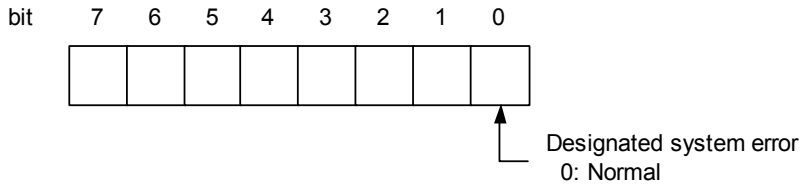
	Data type	Input/ output	Range	Others
Designated system	char	Input data	1 to 5	1: System 1 ... **Note)**
Return information	char	Output data	0 to 255	Refer to section (5) Return information.
Main O number	long	Output data	0 to 99999999	
N	long	Output data	0 to 99999	
B	long	Output data	0 to 99999	
Sub O number	long	Output data	0 to 99999999	This is the ONB data of the sub-program at the deepest nest level being executed. If a sub-program is not being executed, the O number will be 0.
N	long	Output data	0 to 99999	
B	long	Output data	0 to 99999	
MDI data	char	Output data	0 to 1	0: No MDI 1: Searching MDI

Notes:

1) If the 9000 address (9000 to 9999) sub-program is being executed when the custom program lock (edit lock C) is ON, 0 will be set in the ONB data for the sub-program.

2) Always set the designated system as 1. Nothing will be set in the main ONB data or sub ONB data if any other value is set.

(5) Return information



Designated system error

If the designated system does not exist, 1 will be set. If the designation is correct, 0 will be set. (If the designated system does not exist, nothing will be set in the main ONB data or sub ONB data.)

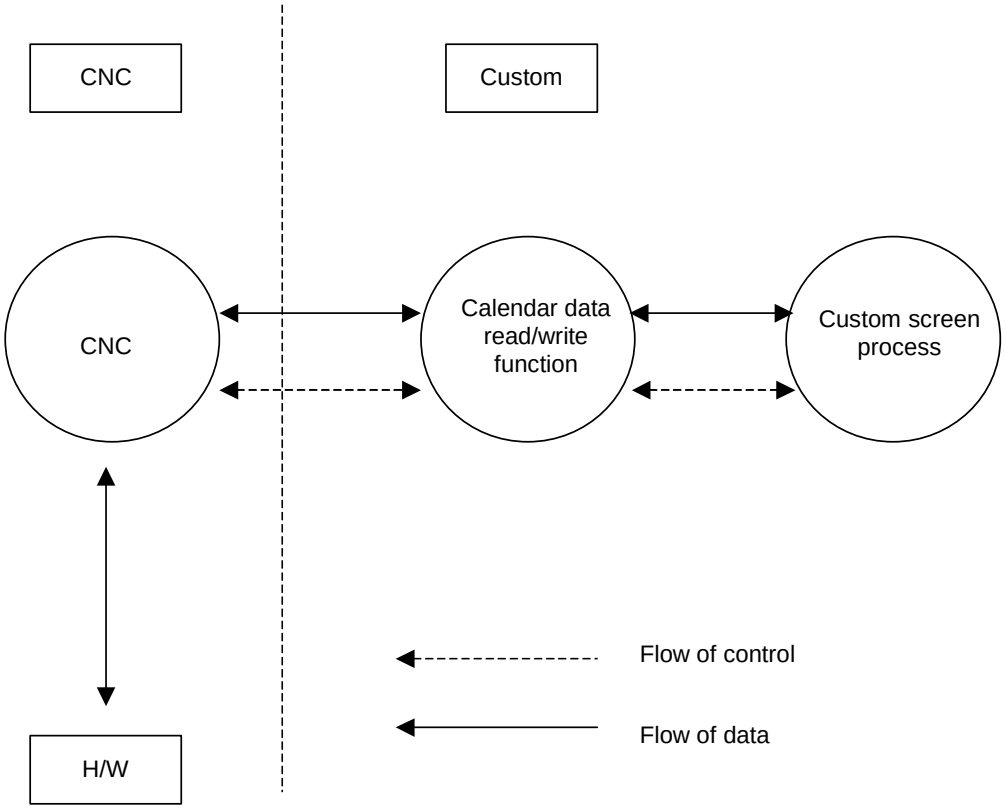
2-3 Example

```
typedef  struct  {
        char      systm;          Designated system (Only 1-system)
        char        info;         Returned information
        long      maino;          Main O number
        long      mainn;          Main N number
        long      mainb;          Main B number
        char        mdi;          MDI data
        char       dmyd;          Dummy data
        long       subo;          Sub O number
        long       subn;          Sub N number
        long       subb;          Sub B number
}  BUFF;.....Declares the structure type.

BUFF   buff [3] ;.....Secures the work table.

test ()
{
        buff [0].systm = 1 ; .....Sets system 1.
        smkonb (&buff [0]) ;.....Obtains system 1 ONB data.
}
```

2.1.3 scaldr () Calendar Function

1. Outline

This function is used to read/write the calendar data from the custom release screen process. The data flow is shown below.



2. Function

Explanation

By using the calendar data read/write function "scaldr ()", the current date can be read and written.

2-1 Using the functions

Calling sequence: sts = scaldr (&buff [0]) ;

Argument: &buff [0] Head address of buffer having a 20-byte size.

short buff [0] : Command	When 0, calendar data is written. When 1, calendar data is read.
short buff [1] : Year	(0 to 99)
short buff [2] : Month	(1 to 12)
short buff [3] : Date	(1 to 31)
short buff [4] : Hour	(0 to 23)
short buff [5] : Minute	(0 to 59)
short buff [6] : Second	(0 to 59)
short buff [7] : Not used	(Input 0.)
short buff [8] : Day	(0 to 6: 0 is Sunday, 6 is Saturday)
short buff [9] : Spare	(Input 0.)

Return value: sts Error status (short)
Refer to the section 2-2 "List of error details" for details.

Function

(1) Write of calendar data

Input 1 in "buff [0]", and input the current date in "buff [1]" to "buff [8]" following the format given above.
After inputting all of the data in "buff [0]" to "buff [9]", call this function.

(2) Read of calendar data

Input 0 in "buff [0]", and call this function.
The current date will be written into "buff [1]" to "buff [8]" following to the format given above.

2-2 List of error details

sts	Error details
–1	Command data illegal
–2	Year data setting range error
–3	Month data setting range error
–4	Date data setting range error
–5	Hour data setting range error
–6	Minute data setting range error
–7	Second data setting range error
–8	Not used
–9	Day data setting range error
–10	Spare

2.1.4 sgetmes () Message Function

Function name: sts = sgetmes (para1, para2, & buff[0]) ;

Include files
"mes_def.h".

Argument: long para1 : Message section
OPEMES : Operation message
SETERR : Setting error

long para2 : Message sub-section
Set a message number. (Refer to the next page.)

char & buff [0] : Head address of buffer having 20-byte size.

Returned value: long sts : 0 ..... Normal completion
–1 ..... Section illegal
–2 ..... Sub-section illegal
–100 ..... Other error

Function

This function is used to read the message data that the CNC has with the custom software.
First, designate the message type with "para1", and designate the message to be read out with "para2". Finally, transfer the head of a buffer having a size of 20 bytes as an argument to this function.
For example, to read "E01 SETTING ERROR", set "SETERR" in "para1" and "CE01" in "para2".
Refer to the next page for details on the sub-section.

Program example

```
char buff [20] ;
sgetmes (SETERR, CE01, &buff [0]) ;
```

	Error message
CE01	SETTING ERROR
CE02	DATA OVER
CE03	No. NOT FOUND
CE04	DEV. NOT READY
CE05	NOT ACCEPTABLE
CE06	NO SPEC
CE07	RESET END
CE08	PHYSICAL ERR
CE09	TIME OUT
CE10	MEMORY OVER
CE11	PROG. No. DUPLI
CE12	FILE ENTRY OVER
CE13	NB NOT FOUND
CE14	PROG. NOT FOUND
CE17	PARITY H ERR
CE18	PARITY V ERR
CE20	OVER RUN ERR
CE21	PROGRAM RUNNING
CE22	CODE CHANGE ERR

	Error message
CE23	NOT ADD I/O
CE25	DATA MEMORY ERR
CE35	COMPARE ERROR

	Error message
CE50	FILE ERR
CE51	FILE OPEN ERR
CE52	FILE CLOSE ERR
CE53	FILE SEEK ERR
CE54	FILE READ ERR
CE55	FILE DELETE ERR
CE57	FILE INSERT ERR
CE60	IOP ERR□□
CE64	PROGRAM No. ERR
CE65	PROG. No. DUPLI
CE66	NO PROG. NUMBER

	Error message
CE70	TOOL No. DUPLI
CE71	TOOL ENTRY OVER
CE80	TOP SEARCH ERR
CE81	PROGRAM ERROR
CE82	ALREADY RESERCH
CE83	COUNT OVERFLOW
CE84	CAN'T IN/OUT
CE85	LINE BUSY
CE86	INPUT DATA ERR

	Error message
CM01	SEARCH EXECUTION
CM02	SEARCH COMPLETE
CM03	RESEARCH EXECUTION
CM04	RESEARCH COMPLETE
CM05	BUFFER EDIT
CM06	MDI NO SETTING
CM07	MDI SETTING COMPLETE
CM08	MDI ENTRY COMPLETE
CM09	MDI RUNNING
CM10	PUSH KEY SERCH/PROG
CM11	EDITTING
CM12	PROGRAM RUNNING
CM13	VOICE OUTPUT
CM14	DATA IN EXECUTION
CM15	DATA WRITING
CM16	DATA IN COMPLETE
CM17	VARIABL CONVERT ERR
CM18	COMPARE EXECUTION
CM19	COMPARE COMPLETE
CM20	DATA OUT EXECUTION

	Error message
CM21	DATA OUT COMPLETE
CM22	ERASE EXECUTION
CM23	ERASE COMPLETE
CM24	COPY EXECUTION
CM25	COPY COMPLETE
CM26	CONDENSE EXECUTION
CM27	CONDENSE COMPLETE
CM28	MERGE EXECUTION
CM29	MERGE COMPLETE

2.1.5 oexsrch () Operation Search

Calling sequence

oexsrch (&srchinf) ;

SRCHINF srchinf ;

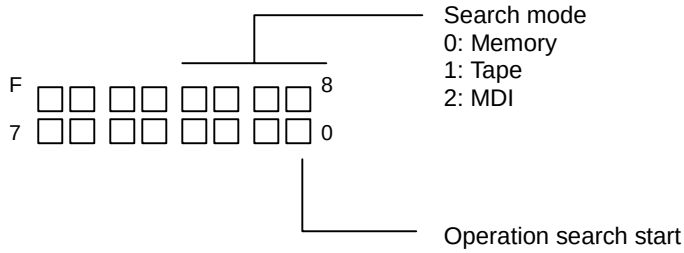
* Create the following table, and transfer the pointer as the argument.

```
typedef struct {  short command ; Command
                  short status   ; Status
                  long  progno   ; Program number
                  long  seqno    ; Sequence number
                  long  blkno    ; Block number
                  long  blkrep   ; Number of block appearances (Reserved for restart search)
                  long  sys      ; Designate system to be searched (0: Fixed to 1-system)
} SRCHINF ;
```

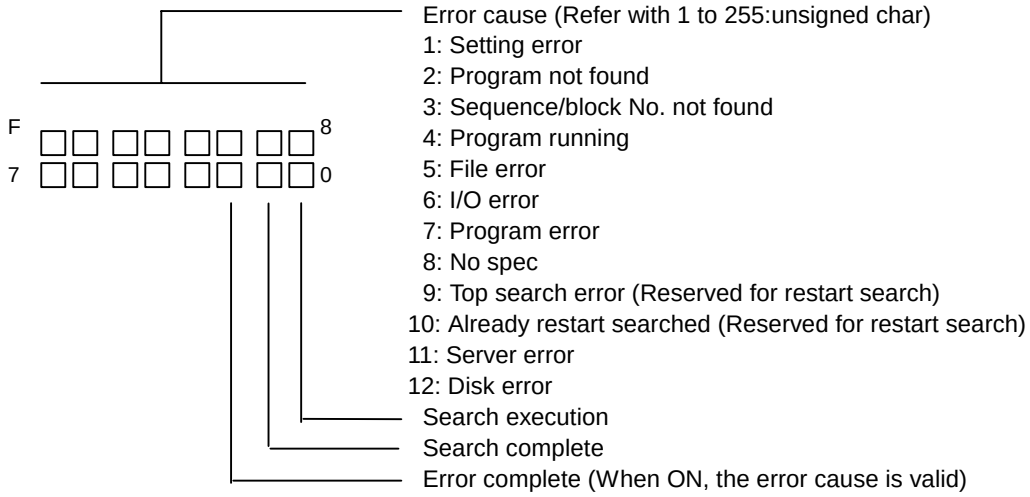
Function

When this function is called, 1-system operation search of the machining program or MDI program designated in the table is executed.
The table setting data is explained below.

(1) Command ... The mode is designated and the search is started.



(2) Status ... The search status is expressed. (Set by the NC.)



(3) Program number

- For memory/tape
  Set the number of the program to be searched with a binary.
  For restart search, the program number cannot be omitted (designated ()).
  0 to 99999999 (max. 8 digits)
- For MDI (only operation search)
  Normally set 0.
  Set 1 to reset the search.

(4) Sequence number

Set the number of the sequence to be searched with a binary.
0 to 99999 (max. 5 digits)

(5) Block number

Set the number of the block to be searched with a binary.
0 to 99999 (max. 5 digits)

(6) Number of block appearances (Reserved for restart search)

Set the number of times the block to be searched appears.
For example, when searching for a block in the sub-program, the block is executed the same times the program is called, so set which execution block to search.
When 0 is set, the first execution block will be searched.
0 to 9999 (max. 4 digits)

List of set patterns (operation search) * o, n, b indicate a number (other than 0) is designated.

Program No.	Sequence No.	Block No.	Search status
0 (other than MDI)	o	o	Setting error
0 (other than MDI)	n	o	Search designated sequence number in selected program
0 (other than MDI)	n	b	Search designated sequence block number in selected program
0 (other than MDI)	o	b	Setting error
p (including MDI)	o	o	Search head of designated program
p (including MDI)	n	o	Search designated sequence number in designated program
p (including MDI)	n	b	Search designated sequence block number in designated program
p (including MDI)	o	b	Search for a block that is the designated number of blocks ahead from the head of the designated program Note that if there is a sequence number if the blocks midway, the search cannot be carried out.

ex)

```
O100
 G28  XYZ ;          ← Search is possible with "O100 N0 B1"
 G00  X100. Y50. ;   ← Search is possible with "O100 N0 B2"
 G01  Z20. F1000 ;   ← Search is possible with "O100 N0 B3"
 N100 Y50. ;         ← Search is not possible with "O100 N0 B4" (This block is "O100 N100 B0")
```

Program example

To search block N2 in the machining program No. O100

```
srchinf.command = BITO ;
srchinf.progno = 100 ;
srchinf.seqno = 2 ;
srchinf.blkno = 0 ;
oexsrch(&srchinf) ;
if ( srchinf.status & BIT2 )
  Error process
```

BITO: 0x0001
BIT2: 0x0004
BIT9: 0x0200

To search for the head of the MDI program

```
srchinf.command = BITO | BIT9 ;
srchinf.progno = 0 ;
srchinf.seqno = 0 ;
srchinf.blkno = 0 ;
oexsrch (&srchinf) ;
if ( srchinf.status & BIT2 )
   Error process
```

To reset the MDI program search

```
srchinf.command = BITO | BIT9 ;
srchinf.progno = -1 ;
oexsrch (&srchinf) ;
if ( srchinf.status & BIT2 )
   Error process
```

2.1.6 ievarrd () CNC Variable Read Function (IEEE double type)

Function

Reads the CNC variables with the standard "double" format (IEEE double).

Format

sts=ievarrd (mode,varno,syst,ground,varptr);

long	sts;	Status	1: Blank variable retrieved 0: Normal completion –1: Mode error –2: Variable number error –3: System number error –4: Ground designation error –5: File unformatted error
long	mode;	Variable mode	0: Common variable 3: Local variable
long	varno;	Variable number	

Variable	Variable number		Target	
			Machining center	Lathe
Common variable	100	100 to 149, 500 to 549	○	○
	200	100 to 199, 500 to 599		
	300	100 to 199, 500 to 699		
	600	100 to 199, 500 to 999		
	700	100 to 199, 400 to 999		
Local variable	1 to 33 (32), 101 to 133 (132), 201 to 233 (232), 301 to 333 (332), 401 to 433 (432) * The 100th place is the level designation. 201 → # 1 in level 2.		○	○

Note: The maximum number in the local variables not shown in parentheses is for the machining center. The number for the lathe is shown in parentheses.

If a variable not provided in the options is designated, –2: Variable number error will occur.

long	syst;	System number	0: 1st system 1: 2nd system . . .	Common variables for each system valid only for (#100 to # 199) (Fixed to 0 in other cases.)
long	ground;	Ground designation	0: Fixed	
double	*varptr;	Variable data pointer		

Example

Read the common variable #510 with standard "double".

```
extern    long ievarrd ();

mvalrd ()
{
          long          sts;
          double        dbuff;
                          .
                          .
                          .
          sts = ievarrd (0L,510L,0L,0L,&dbuff);
          if (sts < 0)
          {
                    merror ();
          }
                          .
                          .
                          .
}
```

2.1.7 ievarwt () CNC Variable Write Function (IEEE double type)

Function

Writes the CNC variables with the standard "double" format (IEEE double).

Format

sts = ievarwt (mode,varno,syst,ground,varptr);

long	sts;	Status	0: Normal completion –1: Mode error –2: Variable number error –3: System number error –4: Ground designation error –5: File unformatted error
long	mode;	Variable mode	0: Common variable 3: Local variable
long	varno;	Variable number	(Refer to section 2.1.7.)
long	syst;	System number	0: 1st system 1: 2nd system . . .
long	ground;	Ground designation	0: Fixed
double	*varptr;	Variable data pointer	

Common variables for each system valid only for (#100 to #199) (Fixed to 0 in other cases.)

Program example

Write to the 1st system FORE ground common variable #120 with standard "double".

```
extern long ievarwt ();

mvalwt ()
{
        long            sts;
        double          dbuff;
                          .
                          .
                          .
        dbuff = 0.001;
        sts = ievarwt (0L,120L,0L,0L,&dbuff);
        if (sts != 0)
        {
                        merror ();
        }
                          .
                          .
                          .
}
```

2.1.8 ievarclear () CNC Variable Clear Function

Function

The CNC variable is "cleared".

Format

sts = ievarclear (mode,varno,syst,ground);

long	sts;	Status	0: Normal completion –1: Mode error –2: Variable number error –3: System number error –4: Ground designation error –5: File unformatted error
long	mode;	Variable mode	0: Common variable 3: Local variable
long	varno;	Variable number	(Refer to section 2.1.7.)
long	syst;	System number	0: 1st system 1: 2nd system . . .
long	ground;	Ground designation	0: Fixed

Common variables for each system valid only for (#100 to #199) (Fixed to 0 in all cases.)

Program example

Clear the 1st system FORE ground common variable #120.

```
extern long ievarclear ();

mvalclear ()
{
        long            sts;
                          .
                          .
                          .
        sts = ievarclear (0L,120L,0L,0L);
        if (sts != 0)
        {
                        merror ();
        }
                          .
                          .
                          .
}
```

3. Machine Control I/F

3.1 PLC Device Accessing Functions

[Function List]

1. Setting the bit device set_☐
2. Resetting the bit device rst_☐
3. Testing the bit device tst_☐
4. Setting the word device set_☐
5. Testing the word device tst_☐
6. Setting the long data lset_☐
7. Testing the long data ltst_☐

3.1.1 set_☐ Setting the Bit Device

Format

set_☐ (no);
☐ : Devices name For PLC 4B – X, Y, M, L, F, E, U, W, S, I, J, G
For GPPQ/W – X, Y, M, L, F
long no: Device number

Function

Turns ON the designated bit device.

Example

set_X (0x40) ; Turns ON device X40.

3.1.2 rst_☐ Resetting the Bit Device

Format

rst_☐ (no) ;
☐ : Devices name For PLC 4B – X, Y, M, L, F, E, U, W, S, I, J, G
For GPPQ/W – X, Y, M, L, F
long no: Device number

Function

Turns OFF the designated bit device.

Example

rst_X (0x40) ; Turns OFF device X40.

3.1.3 tst_☐ Testing the Bit Device

Format

fg = tst_☐ (no) ;
☐ : Devices name For PLC4B – X, Y, M, L, F, E, U, W, S, I, J, G, TO, CO, QO, BO
For GPPQ/W – X, Y, M, L, F, TO, CO
long fg: Results of bit device test
long no: Device number

Function

Returns the bit status (ON/OFF); device OFF: returns 0.
ON : returns numeric data other than 0.

Example

extern long tst_X () ; "extern" declaration to byte size.
fg = tst_X (0x40) Test the status of device X40.

(Note)

The "extern" declaration must be executed.

3.1.4 set_☐ Setting the Word Device

Format

set_☐ (no, setdt) ;
☐ : Devices name D, R, T, C
long no : Device number
long setdt: Data to be set

Function

Sets data in the designated word device.

Example

set_R (1900, 100) ; Sets 100 into device R1900.

(Note)

The set data must not exceed the word size range.
If the set data exceeds the word size, the data that is set will not be guaranteed.

3.1.5 tst_☐ Testing the Word Device

Format

tstdt = tst_☐ (no) ;
☐ : Devices name D, R, T, C
long tstdt : Results of word device test
long no : Device number

Function

Returns the data stored in the designated word device with a word size.

Example

extern long tst_R () ; "extern" declaration to word size
tstdt = tst_R (1900) ; Returns the data stored in device R1900.

(Note)

The "extern" declaration must be executed.

3.1.6 lset_☐ Setting the Long Data

Format

lset_☐ (no, setdt) ;
☐ : Devices name D, R
long no : Device number
long setdt: Long data to be set

Function

Sets the data in the designated word device with a long size.

Example

lset_R (1900, 0x12345678) ; Sets 0x12345678 in devices R1900 and R1901.

(Note)

The set data must not exceed the long size range.
If the set data exceeds the long size, the data that is set will not be guaranteed.
When using a word device with a long size, use two devices. (no and no +1).
Note that the high-order two bytes and low-order two bytes of the set data will be interchanged.
An actual example is given below.

Execute "lset_R (1900,0x12345678)"; →

Device R1900	0x5678
R1901	0x1234

3.1.7 ltst_☐ Testing the Long Data

Format

tstdt = ltst_☐ (no) ;
☐ : Devices name D, R
long tstdt : Results of the word device test
long no : Device number

Function

Returns the data stored in the designated word devices with a long size.

Example

extern long tst_R () ; "extern" declaration to long size.
tstdt = tst_R (1900); Returns the data in device R1900 with a long size.

(Note)

The "extern" declaration must be executed.
When using a word device with a long size, use two devices. (no and no +1).
Note that the high-order two bytes and low-order two bytes of the returned data will be interchanged.
An actual example is given below.

Device R1900	0x1234
R1901	0x5678

Execute "tstdt=ltst_R (1900)"; → tstdt:0x56781234

[Note]

Designate the device X and Y devices numbers as a hexadecimal, and the other device numbers as a decimal.

3.2 PLC Device High-speed Access Functions

The following functions are PLC device access functions used within the PLC function called with the CALL command in the ladder program.

[Function list]

1. Setting the bit device melplcBset☐
2. Resetting the bit device melplcBrst☐
3. Testing the bit device melplcBtst☐
4. Setting the word device melplcWset☐
5. Testing the word device melplcWtst☐
6. Setting the long data melplcLset☐
7. Testing the long data melplcLtst☐

(1) High-speed access to bit device function

- melplcBset☐( ) : Turns the designated bit device ON.
- melplcBrst☐( ) : Turns the designated bit device OFF.
- melplcBtst☐( ) : Checks the designated bit device.

(2) High-speed access to word device function

- melplcWset☐( ) : Sets the designated value as a word size in the designated word device.
- melplcWtst☐( ) : Checks the designated word device.
- melplcLset☐( ) : Sets the designated value as a double-word size in the designated word device.
- melplcLtst☐( ) : Checks the designated word device with a double-word size.

3.2.1 melplcBset □( ) Setting the Bit Device

Format

long melplcBset□ (□ : Devices name
For PLC 4B – X, Y, M, L, F, E, TI, TO, CI, CO, U, W, S, I, J, QI, QO, BI, BO, G
For GPPQ/W – X, Y, M, L, F, SM, TI, TO, CI, CO)

```
(
  long   devNo          Device number to be set
)
```

Explanation

Turns the designated bit device ON.

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.2 melplcBrst □( ) Resetting the Bit Device

Format

long melplcBrst□ (□ : Devices name
For PLC 4B – X, Y, M, L, F, E, TI, TO, CI, CO, U, W, S, I, J, QI, QO, BI, BO, G
For GPPQ/W – X, Y, M, L, F, SM, TI, TO, CI, CO)

```
(
  long   devNo          Device number to be reset
)
```

Explanation

Turns the designated bit device OFF.

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.3 melplcBtst □( ) Testing the Bit Device

Format

long melplcBtst□ (□ : Devices name
For PLC 4B – X, Y, M, L, F, E, TI, TO, CI, CO, U, W, S, I, J, QI, QO, BI, BO, G
For GPPQ/W – X, Y, M, L, F, SM, TI, TO, CI, CO)

```
(
  long   devNo          Device number to be tested
  char   *bValue        Address to store tested values
)
```

Explanation

Tests (reads) the designated bit device. The test results (ON: 1, OFF: 0) are set in bValue.

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.4 melplcWset □( ) Setting the Word Device

Format

long melplcWset□ (□ : Devices name: R, D)

```
(
  long   devNo      Device number to be set
  short  wValue     Word size value set in device
)
```

Explanation

Sets the designated value as a word size in the designated word device

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.5 melplcWtst □( ) Testing the Word Device

Format

long melplcWtst□ (□ : Devices name: R, D)

```
(
  long   devNo      Device number to be set
  short  *wValue    Address to store tested values
)
```

Explanation

Tests (reads) the designated word device as a word size.
The test results are set in "wValue".

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned.

3.2.6 melplcLset □( ) Setting the Long Data

Format

long melplcLset□ (□ : Devices name: R, D)

```
(
  long   devNo      Device number to be set
  long   lValue     Double-word size value set in device
)
```

Explanation

Sets the designated value as a double-word size in the designated word device (register).

Limit

Only even number device numbers can be designated.

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned. If the device number is odd, –2 is returned.

3.2.7 melplcLtst □( ) Testing the Long Data

Format

long melplcLtst□ (□ : Devices name: R, D)

```
(
  long   devNo      Device number to be set
  short  *lValue    Address to store tested values
)
```

Explanation

Tests (reads) the designated word device (register) as a double-word size.
The test results are set in "lValue".

Limit

Only even number device numbers can be designated.

Return value

0 is returned when completed normally.
If the designated device number is not found, –1 is returned. If the device number is odd, –2 is returned.

Precautions

The types of devices that can be handled differ between the PLC4B and GPPQ/W ladders. However, the library accessing the bit device automatically converts the assignment. For example, if "melplcBsetU(0)" is executed when using the GPPQ/W ladder, the same results as when "melplcBsetX (0x4C))" is executed will be attained. Refer to the "MELDAS 64 Series GPPQ/GPPW Interface Function Specifications (BNP-B3941-057)" for details on the assignment conversion.

4. File Release I/F

4.1 File Data Input/Output Functions

This is an interface used to read/write the machining programs, etc., in the CNC memory from the custom release system.

Function list

No.	Function outline	Function
1	Register new machining program No.	prmake ()
2	Change machining program No.	prrena ()
3	Delete machining program	prdele ()
4	Read machining program registration state	prdir ()
5	Write comment to machining program	prcmwt ()
6	Read comment from machining program	prcmrd ()
7	Write one block of machining program	plwrit ()
8	Read one block of machining program	plread ()
9	Insert one block in machining program	plinst ()
10	Delete one block of machining program	pldele ()
11	Read number of machining program blocks	plrcunt ()
12	Read machining program being executed	plrunblk ()
13	Read machining program information	prinfo ()
14	Check program being run	prunchk ()

[Explanation of functions]

Each function is explained with the following format.

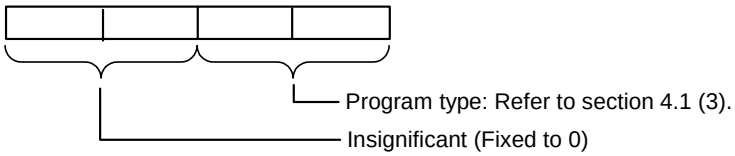
(1) prmake ( ) Register new machining program No. ——— Function name and features

Calling sequence

sts = prmake (mode, name) ——— How to call the function

unsigned long sts : Status number (Refer to section [Status]) —— Explanation of argument

unsigned long mode :



unsigned long name : Program number (1 to 99999999)

Function

The machining program designated with the argument "name" is registered into the NC memory.
——— Features of function and usage methods

The details of the registered program are only EOR (%).
With "prmake()", the machining programs for the number of systems are registered.
If a machining program with the same number exists, 10 will be set in the return value sts.
——— Example of C language description

Program example

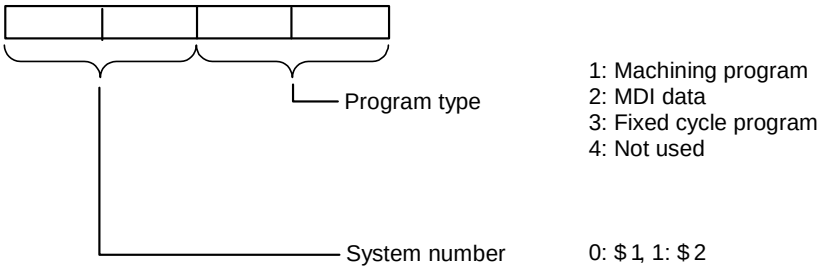
To register machining program number O100.
sts = prmake (1, 100) ;

(1) The program number setting range is as follows.
- Machining program : Range of 1 to 99999999
- MDI data : Insignificant (fixed to 0)
- Fixed cycle : (G code) * 10

The G code can be used in the range of G70 to G89.
(Example) Set 710 for the G71 program.

(2) Maximum number of registerable characters and number of programs in MDI and fixed cycle
MDI data : Number of registerable characters = 500
Fixed cycle: Number of registerable characters = 12000
Number of registerable programs = 24

(3) For the mode setting, the high-order bytes are the system number, and the low-order bytes are the program type.



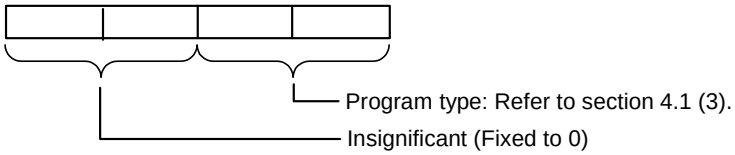
4.1.1 prmake () Register New Machining Program No.

Format

sts = prmake (mode, name)

unsigned long sts : Status number (Refer to section [Status])

unsigned long mode :



unsigned long name : Program number (1 to 99999999)

Function

The machining program designated with the argument "name" is registered into the NC memory. The details of the registered program are only EOR (%).
With "prmake()", the machining programs for the number of systems are registered.
If a machining program with the same number exists, 10 will be set in the return value sts.

Program example

To register machining program number O100.
sts = prmake (1, 100) ;

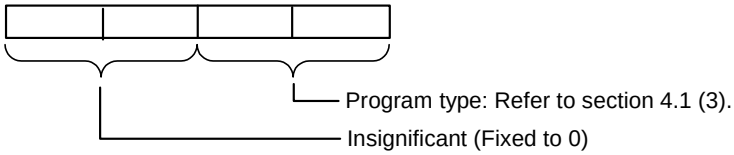
4.1.2 prrena () Change Machining Program No.

Format

sts = prrena (mode, name1, name2)

unsigned long sts : Status number (Refer to section [Status])

unsigned long mode :



unsigned long name1 : Old program number

unsigned long name2 : New program number

Function

The number of the machining program designated with argument "name1" is changed to the number designated with argument "name2".
If the machining program designated with argument "name1" is not registered in the NC memory, 11 will be set in the return value "sts". If the machining program designated in argument "name2" already exists, 10 will be set in the return value "sts".

Program example

To change machining program O100 to O200.

```
sts = prrena (1, 100, 200) ;
```

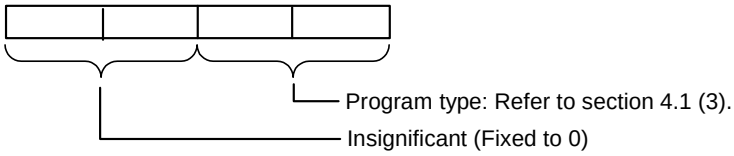
4.1.3 prdele () Delete Machining Program

Format

sts = prdele (mode, name)

unsigned long sts : Status number (Refer to section [Status])

unsigned long mode :



unsigned long name : Program number (1 to 99999999)

Function

The machining program designated with argument "name" is deleted.
With "prdele ()", the machining programs corresponding to the number of systems will be deleted.
If the machining program designated with argument "name" is not registered in the NC memory, 11 will be set in the return value "sts".

Program example

To delete the machining program O100.

```
sts = prdele (1, 100) ;
```

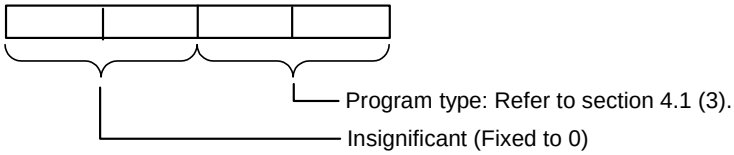
4.1.4 prdir () Read Machining Program Registration State

Format

sts = prdir (mode, start, size, &buff [0])

unsigned long sts : Status number (Refer to section [Status])

unsigned long mode :



unsigned long start : Index number of program registered in NC memory.
(0 ≤ start)

unsigned long size : Number of programs to be read out

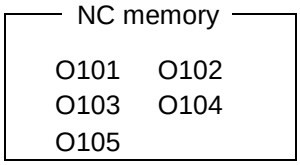
unsigned long buff [N] : Registered program number read buffer (N ≥ size)

Function

The numbers of the machining programs registered in the NC memory are read into the designated buffer.
If the reading of all program numbers in the NC memory is completed while setting the program numbers corresponding to the designated number of programs (size), –1 will be set in the corresponding buffer.

Program example

When there are five machining programs (O101 to O105) in the NC memory, and three are to be read out at a time.




```
long buff [3] ;
sts = prdir (1,0,3,&buff [0]);
                .
                .

sts = prdir (1,3,3,&buff [0]);
```

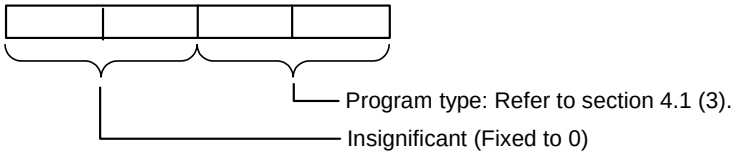
buff [0]	101
buff [1]	102
buff [2]	103

buff [0]	104
buff [1]	105
buff [2]	–1

4.1.5 prcmwt () Write Comment to Machining Program

Format

sts = prcmwt (mode, name, &comm [0])

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 

unsigned	long	name	: Program number (1 to 99999999)
unsigned	char	comm [18]	: Storage buffer for comment to be written

Function

The comment character strings in the buffer are written as the comment for the machining program designated with the argument "name".
For the comment, a character string containing up to 18 characters to the end code (NULL) is written.
If the machining program designated with argument "name" is not registered in the NC memory, 11 will be set in the return value "sts".

Program example

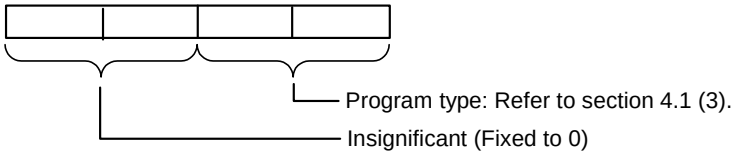
To set the comment for the machining program "O100" to "MITSUBISHI".

```
sts = prcmwt (1, 100, "MITSUBISHI") ;
```

4.1.6 prcmrd () Read Comment from Machining Program

Format

sts = prcmrd (mode, name, &comm [0])

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 
unsigned	long	name	: Program number (1 to 99999999)
unsigned	char	comm [18]	: Storage buffer for comment to be read

Function

The comment character string of the machining program designated with the argument "name" is read into the buffer.
The comment buffer must have enough space for 18 characters.

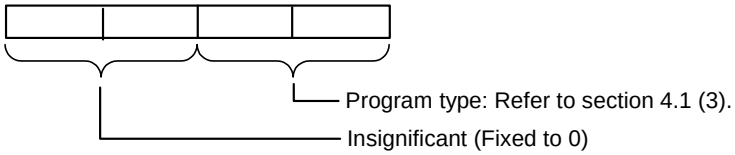
Program example

To read the comment for the machining program "O100".
sts = prcmrd (1, 100, &comm [0]) ;

4.1.7 plwrit () Write One Block of Machining Program

Format

sts = plwrit (mode, name, block, &buff [0])

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 
unsigned	long	name	: Program number (1 to 99999999)
unsigned	long	block	: Block number (block ≥ 1)
unsigned	char	buff [N]	: Storage buffer for one block of data to be written (0<N≤248)

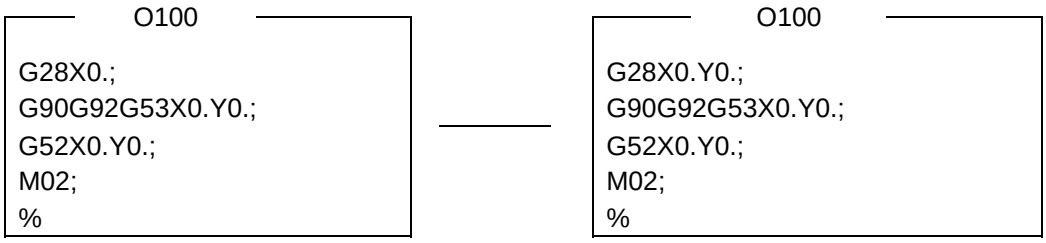
Function

One block of data in the buffer (buff [N]) is written into the designated machining program.
One block of the NC memory machining program has up to 248 bytes including EOB(;). The block number is counted from 1.
The zero (NULL) and space (' ') codes in the machining program are ignored.
If the designated machining program is not registered in the NC memory, 11 will be set in the return value "sts". If the block number does not exist, 12 will be set in the return value.

Program example

To set head block of machining program "O100" to "G28X0.Y0.;"

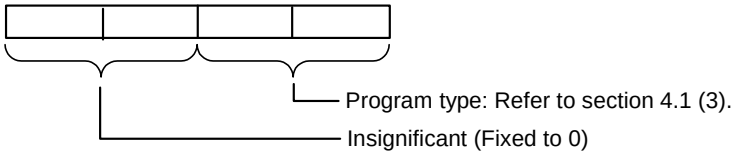
```
sts = plwrit (1, 100, 1, "G28X0.Y0.;") ;
```



4.1.8 plread () Read One Block of Machining Program

Format

sts = plread (mode, name, block, &buff [0])

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 
unsigned	long	name	: Program number (1 to 99999999)
unsigned	long	block	: Block number (block ≥ 1)
unsigned	char	buff [N]	: Storage buffer of one block of data to be read. (N = 248)

Function

One block of data in the designated machining program is read out the designated buffer (buff [N]). One block of the NC memory machining program has up to 248 bytes including EOB(;). The block number is counted from 1.
If the designated machining program is not registered in the NC memory, 11 will be set in the return value "sts". If the block number does not exist, 12 will be set in the return value.

Program example

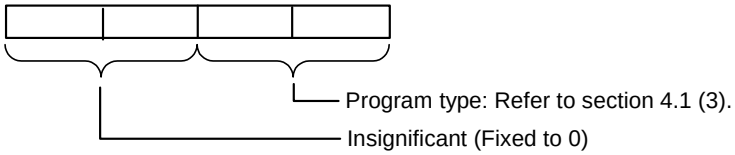
To read the head block of machining program "O100".

```
sts = plread (1, 100, 1, &buff [0]) ;
```

4.1.9 plinst () Insert One Block in Machining Program

Format

sts = plinst (mode, name, block, &buff [0])

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 

unsigned	long	name	: Program number (1 to 99999999)
unsigned	long	block	: Block number (0 ≤ block)
unsigned	char	buff [N]	: Storage buffer of one block of data to be inserted (N ≤ 248)

Function

One block of data in the designated buffer (buff [N]) is inserted into the designated machining program.
One block of the NC memory machining program has up to 248 bytes including EOB(;). The block number is counted from 1.
To insert the block data at the head of the machining program, designate 0 as the block number.
To add the block data at the end of the machining program, designate the number of blocks in the corresponding program as the block number.

Program example

To insert "G28X0.Y0.;" at the head block of the machining program "O100".

```
sts = plinst (1, 100, 0, "G28X0.Y0.;") ;
```

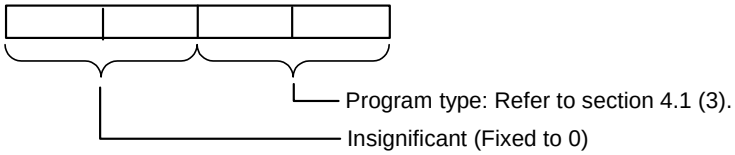
```
O100
G90G92G53X0.Y0.;
G52X0.Y0.;
M02;
%
```

```
O100
G28X0.Y0.;
G90G92G53X0.Y0.;
G52X0.Y0.;
M02;
%
```

4.1.10 pldele () Delete One Block of Machining Program

Format

sts = pldele (mode, name, block)

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 
unsigned	long	name	: Program number (1 to 99999999)
unsigned	long	block	: Block number (block ≥ 1)

Function

Deletes a block from the designated machining program.

Program example

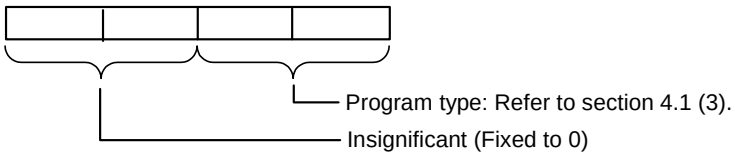
To delete the head of the machining program "O100".

```
sts = pldele (1, 100, 1) ;
```

4.1.11 plcunt () Read Number of Machining Program Blocks

Syntax

sts = plcunt (mode, name, &blockno)

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	: 
unsigned	long	name	: Program number (1 to 99999999)
unsigned	long	blockno	: Number of blocks

Function

Returns the designated number of machining program blocks to "blockno".

Program example

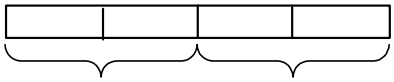
To read the total number of blocks in the machining program "O100".

```
sts = plcunt (1, 100, &blockno) ;
```

4.1.12 plrunblk () Read Machining Program Being Executed

Format

sts = plrunblk (mode, req_num, req_len, read_num, rdbuf)

unsigned	long	sts	: Status number (Refer to section [Status])
unsigned	long	mode	:  Ground designation: 0 FIXED System number: Refer to section 4.1 (3).
unsigned	long	req_num	: 1: Only block being executed 2: Block being executed and next block 3: Block being executed and next two blocks . . . 16: Block being executed and next 15 blocks
unsigned	long	req_len	: Maximum number of characters in one block read (including EOB)
unsigned	long	*read_num	: Read number of blocks is returned.
PLRUNBLK		*rdbuf	: Pointer to read information buffer (When reading number of blocks, the read information buffers are prepared as arrays for the number of blocks to be read, so set the pointer to the first element of the read information buffer array.)

[Read information buffer (PLRUNBLK) structure]

```
typedef struct {
        char    *readbuf;   Pointer to buffer for storing read block data
        long    char_num;   Number of read characters
} PLRUNBLK;
```

Function

This function reads the current machining program block and blocks following the next command in the program being run into the read information buffer array designated with argument "rdbuf".
The number of blocks to be read is designated with "req_num". However, if the remaining number of blocks in the program (from the block being executed) is less than the designated number of blocks, only the remaining number of blocks will be read. The number of blocks actually read is returned to "read_num".
The maximum number of characters in the machining program block to be returned to the read information buffer is designated with "req_len". The area designated with the read information buffer readbuf must be the size designated by "req_len" for all blocks. The number of characters actually read will be returned to "char_num". However, if the blocks to be read exceed the number of characters designated with "req_len", the valid number of characters actually read to each buffer will be the maximum number of characters –1 because a NULL code is added at the end.

Program example

To read the block being executed and the next block during program execution.

```
char actbf [50] ;
char nextbf [50] ;
PLRUNBLK rdbuf [2] ;
long read_num ;

mode = 0 ;
req_num = 2 ;
req_len = 50 ;
rdbuf [0] . readbuf = &actbf [0] ;
rdbuf [1] . readbuf = &nextbf [0] ;
sts = plrunblk (mode, req_num, req_len,
                &read_num, &buff [0]) ;
```

<Machining program>

```
O100;
G28  XYZ ;
G00  X100. ; <-------------- Block being executed
G01  X150.  Y150.  F100. ;
      .
      .
      .
```

<Results>

actbf []

```
G00  X100. ;
```

nextbf []

```
G01  X150.  Y150.  F100. ;
```

rdbuf [0].char_num: 10

rdbuf [1].char_num: 22

read_num: 2

<Note>

(1) Limits to number of read blocks

During the disk operation, there may be cases when the designated number of blocks cannot be read. The actually read blocks will be returned to "read_num". For the block data that could not be read, a NULL code will be added at the head.
(Note that this will occur frequently in programs in which one block exceeds 200 characters.)

(2) Read character strings

Normally, one space is placed between the character strings, but in some cases, two spaces may be placed.

(3) Specifications of read block during search state

The block data can be read when the search is completed (when program is not being run) when using memory operation. For the block data read out at that time, the block being executed will be empty, and the next block will be the head of the program. Note that the number of blocks including the executed block will be returned to "read_num".

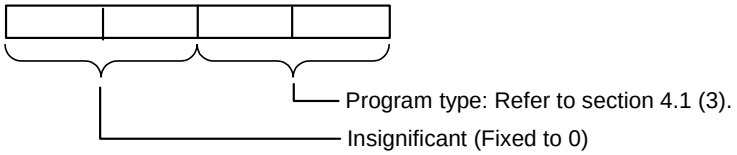
4.1.13 prinfo () (old prsize) Read Machining Program Information

Format

sts = pinfo (mode, &buff [0])

unsigned long sts : Status number (Refer to section [Status])

unsigned long mode :



unsigned long buff [N] : Machining program information (N=4)
buff [0]: Number of registered machining programs
buff [1]: Remaining number of registerable machining programs
buff [2]: Number of registered characters
buff [3]: Remaining number of registerable characters

Function

The information such as the number of registered machining programs, etc., is returned to the buff.

4.1.14 prunchk () (old pdrchk) Check Program Being Run

Format

sts = prunchk (name)

	long	sts	: Check results	0: Not being run –1: Running
unsigned	long	name	: Program number (1 to 99999999)	

Function

Whether the designated machining program is currently being run or not is checked, and the results are returned to "sts".
If the program is being run, do not edit the program.

[Status]

Status No.	Details
0	Normal completion
1	Argument illegal ("mode" value is incorrect)
2	Argument illegal ("name" value is incorrect)
3	Argument illegal ("block" value is incorrect)
5	Argument illegal ("bknum" value is incorrect)
6	Operation mode illegal (Not in memory operation)
8	Argument illegal (System number is incorrect)
10	Same program number found
11	Designated program number not found
12	Designated block number not found
13	Number of registered programs exceeded
14	Number of write characters exceeded
15	No EOB (;) code
16	Number of characters exceeded
17	EOR (%) found before EOB
30	Executing program
80	File is already opened
81	Write protected
82	No DDB function option
99	File system error
100	Not running machining program (generated with plrunblk)

[Note]

- Do not delete or change the file while the program is running.
- Do not use the file input/output function with the initialization function "mopeini()".
- Include "o_def.h" in the file where these functions are called.

5. General Functions

5.1 Data Conversion Functions

Function List

1. Convert binary character string into 32-bit numeric value	abtol ()
2. Convert hexadecimal character string into 32-bit numeric value	ahtol ()
3. Convert decimal character string into 32-bit BCD	atobcd ()
4. Convert decimal character string into 32-bit numeric value	atol ()
5. Convert decimal character string into 16-bit numeric value	atos ()
6. Convert hexadecimal into a character string	dchtoa ()
7. Convert decimal into a character string	ltoa ()
8. Compare two character strings	ostrcmp ()
9. Convert decimal character string with decimal point into 32-bit numeric data	satol ()

5.1.1 abtol () Convert Binary Character String into 32-bit Numeric Value

Format

st = abtol (asc, longd)

short	st	: Error status	Normal completion → 0
			Error occurrence → –1
char	asc []	: Binary character string to be converted (0, 1 or blank)	
long	*longd	: Pointer of variable where converted value is to be stored	

Example

```
char    asc[] = {"10110001"}
                              A null character must be placed at the end of the character string.
main()
{
        short   st;
        long    longd;

        st = abtol (asc,&longd);
}
                              ↓
        The converted value (longd) is 177.
```

5.1.2 ahtol () Convert Hexadecimal Character String into 32-bit Numeric Value

Format

st = ahtol (asc, longd)

short	st	: Error status	Normal completion → 0
			Error occurrence → –1
char	asc []	: Hexadecimal character string to be converted (0 to 9, A to F, or _)	
long	*longd	: Pointer of variable where converted value is to be stored	

Example

```
char    asc[] = {"A0B1CDEF"};
                              A null character must be placed at the end of the character string.
main()
{
        short   st;
        long    longd;

        st = ahtol (asc,&longd);
}
                              ↓
        The converted value (longd) is –1598960145.
```

5.1.3 atobcd () Convert Decimal Character String into 32-bit BCD

Format

st = atobcd (asc, longd)

short	st	: Error status	Normal completion → 0
			Error occurrence → –1
char	asc []	: Decimal character string to be converted (0 to 9 or _)	
long	*longd	: Pointer of variable where converted BCD data is to be stored	

Example

```
char    asc[] = {"12345678"}
                                  A null character must be placed at the end of the character string.
main()
{
        short   st;
        long    longd;

        st = atobcd (asc,&longd);
}
                        ↓
        The converted BCD (longd) is 0x12345678.
                                (305419896)
```

5.1.4 atol () Convert Decimal Character String into 32-bit Numeric Value

Format

st = atol (asc, longd)

short	st	: Error status	Normal completion → 0
			Error occurrence → –1
char	asc []	: Decimal character string to be converted (+, –, 0 to 9 or _)	
long	*longd	: Pointer of variable where converted value is to be stored	

Example

```
char    asc[] = {"1092387456"}
                                  A null character must be placed at the end of the character string.
main()
{
        short   st;
        long    longd;

        st = atol (asc, &longd);
}
                                  The converted value (longd) is–1092387456.
                                                  (0xBEE37D80)
```

5.1.5 atos () Convert Decimal Character String (Integer with Sign) into 16-bit Numeric Value

Format

st = atos (asc, shortd)

short	st	: Error status	Normal completion → 0
			Error occurrence → –1
char	asc []	: Decimal character string to be converted (+, –, 0 to 9 or _)	
short	*shortd	: Pointer of variable where converted value is to be stored	

Example

```
char    asc[] = {"–10923"}
                              ↑
                              └──── A null character must be placed at the end of the character string.
main()
{
        short   st;
        short   shortd;

        st = atos (asc, &shortd);
}                             ↓
                  The converted value (shortd) is –10923.
                                    (0xD555)
```

5.1.6 dchtoa () Convert Hexadecimal into a Character String (0 to 9, A to F)

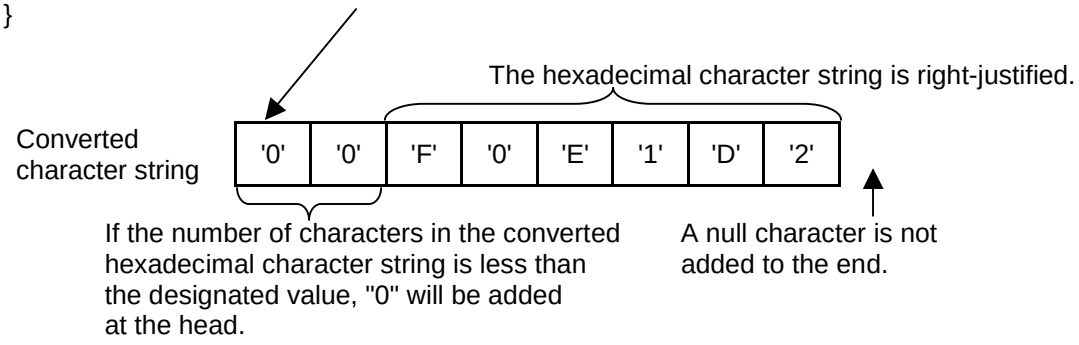
Format

st = dchtoa (longd, keta, asc)

long	longd	: Hexadecimal value to be converted
short	keta	: Designation of number of converted character string digits
char	asc []	: Converted character string

Example

```
main()
{
        char    asc[8];
        long    longd = 0xF0E1D2;

        dchtoa (longd, 8, asc);
}
```



5.1.7 ltoa () Convert Decimal into a Character String (0 to 9)

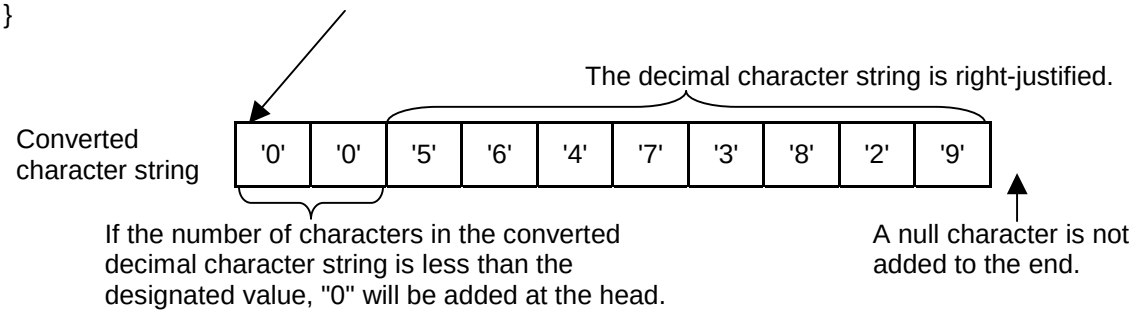
Format

dt = ltoa (longd, keta, asc)

long	dt	: Data (longd value)
long	longd	: Decimal value to be converted
short	keta	: Designation of number of converted character string digits
char	asc []	: Converted character string

Example

```
main()
{
        char    asc[10];
        long    longd = 56473829;
        long    dt;

        dt = ltoa (longd, 10, asc);
}
```



5.1.8 ostrcmp () Compare Two Character Strings

Format

st = ostrcmp (longd, keta, asc)

```
short   st      : Status (Code difference of characters that first differ)
                                Negative value   : str1 < str2
                                0                : str1 = str2
                                Positive value   : str1 > str2
char    *str1   : Character string 1
char    *str2   : Character string 2
```

Example

```
char    str1 = {"MITSUBISHI"} ;
char    str2 = {"METSUBUSHI"} ;
char    str3 = {"MITSUBISHI"} ;

main()
{
        short   code;
        code = ostrcmp (str1, str2);  ──▶ The character string differs.
                                          The code values are 'I'-'E', or in other words, the
                                          code value is 4.

        code = ostrcmp (str1, str3);  ──▶ The character strings are identical.
                                          The code value is 0.
}
```

5.1.9 satol () Convert Decimal Character String (with Sign or Decimal Point) into long Data

Format

st = satol (str, val, num)

short	st	: Error status	Normal completion → 0
			Error occurrence → –1
char	*str	: Pointer of decimal character string to be converted	
long	*val	: Pointer of variable where converted value is to be stored	
short	num	: Number of digits in fraction (num≥0)	

Example

```
static char string[] = "–12345.678";


main()
{                                        Number of digits after the decimal point
        short    st;
        long     val;

        st = satol(string, &val,  3);

}
```

Examples of using this function are shown below.

Character string to be converted	Number of digits after the decimal point	Value after conversion
"–12345.678"	2	Error
"–12345.678"	3	–12345678
"–12345.678"	4	–123456780
"–123 "	3	123000

This function is used when the setting area buffer contents are to be converted into values, etc.

Revision history

Sub-No.	Date of revision	Revision details
E	March 2003	First edition created.

Notice

Every effort has been made to keep up with software and hardware revisions in the contents described in this manual. However, please understand that in some unavoidable cases simultaneous revision is not possible.

Please contact your Mitsubishi Electric dealer with any questions or comments regarding the use of this product.

Duplication Prohibited

MITSUBISHI ELECTRIC CORPORATION
HEAD OFFICE : MITSUBISHI DENKI BLDG., 2-2-3, MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN

MODEL	M60 Series
MODEL CODE	008-254
Manual No.	BNP-B2219E(ENG)

(0303)MEE