Library Reference

Software Development Tolo

Table of Contents

About NetYaroze

What You Need to Know

In order to get started with Net Yaroze, you should have experience of C programming to a competent level and a knowledge of a 2D graphic creation/editing tool. In addition, at least a basic grasp of a 3D modelling package and a sound creation/editing tool would be help you get the best out of you NetYaroze kit.

The NetYaroze Manual Set

There are three books in the set of NetYaroze manuals.

1. Start Up Guide

   An introductory booklet explaining the contents and requirements of the NetYaroze Starter Kit. It also gives step by step instructions on setting up they NetYaroze software on your PC and how to run Net Yaroze software on the system.

2. User Guide

   A reference manual providing details on making software for the NetYaroze system.

3. Library Reference (this document)

   A manual listing and describing the functions and structures in the NetYaroze libraries.

Additional Reading

Please see the Additional Reading list at the end of the Start Up Guide

1

Graphics Functions

RECT

Frame buffer rectangular area

Structure

```
typedef struct {
                    short x, y;
                    short w, h;
} RECT;
```

Members

x, y	Coordinates for the top left-hand corner of the rectangular area
w, h	Width and height of the rectangular area

Comments

RECT specifies the area of the frame buffer to be accessed. Negative values or values that exceed the size of the frame buffer (1024x512) cannot be used.

DRAWENV

Drawing environment

Structure

```
typedef struct {
            RECT  clip;
            short  ofs[2];
            RECT  tw;
            unsigned short  tpage;
            unsigned char  dtd;
            unsigned char  dfe;
            unsigned char  isbg;
            unsigned char  r0, g0, b0;
                  DR_ENV  dr_env;
} DRAWENV;
```

Members

clip	Drawing area. Drawing is limited to the rectangular area specified by clip. Drawing cannot be performed outside the clip area
ofs	Offset. The values (ofs[0], ofs[1]) are added to all coordinate values to give the address values used by all drawing commands when drawing in the frame buffer
tw	Texture window. Repeated use is made of the texture pattern contained in the rectangular area within the texture page defined by tw
tpage	Texture page initial value

dtd	Dither treatment flag 0: OFF 1: ON
dfe	Flag for drawing to the display area 0: Drawing to the display area is blocked 1: Drawing to the display area is allowed
isbg	Clear drawing area flag 0: OFF 1: ON 0: The drawing area is not cleared when the drawing environment is set up 1: The entire clipped area is painted with the brightness values (r0,g0,b0) when the drawing environment is set up.
r0,g0,b0	Background colour. Only available whenisbg = 1.
dr_env	Reserved for this system

Comments

DRAWENV sets the basic parameters relating to drawing offset, drawing clip area, etc.

Notes

Within the drawing space, drawing can actually be carried out in the region (0, 0)-(1023, 511).

Offset values and address values to which the offset has been added are wrapped around using (-1024, -1024)-(1023, 1023).

Values that can be specified for the texture window are limited to the combinations shown in the following table.

Tw.w	0(=256)	16	32	64	128
tw.x	0	multiple of 16	multiple of 32	multiple of 64	multiple of 128
tw.h	0(=256)	16	32	64	128
tw.y	0	multiple of 16	multiple of 32	multiple of 64	multiple of 128

DISPENV

Display environment

Structure

```
typedef struct {
            RECT  disp;
            RECT  screen;
            unsigned char  isinter;
            unsigned char  isrgb24;
            unsigned char  pad0, pad1;
} DISPENV;
```

Members

disp	Display area within the frame buffer The width of the area can be set to 256, 320, 360, 512 or 640 The height of the area can be set to 240 or 480
screen	Display area on the output screen The screen area is calculated on the basis of a standard monitor screen in which the coordinates are (0, 0) for the top left corner and (256, 240) for the bottom right corner, regardless of thedisp value
isinter	Interlaced mode flag 0: Non-interlaced 1: Interlaced
isrgb24	24bit mode flag 0: 16bit mode 1: 24 bit mode

Comments

DISPENV specifies parameters for screen display mode, frame buffer display position, etc.

VECTOR

32bit 3D vectors

Structure

```
typedef struct {
                long  vx, vy;
                long  vz, pad;
} VECTOR;
```

Members

vx, vy, vz	Vector components
pad	Padding

Comments

VECTOR defines the structure of 32 bit 3D vectors.

SVECTOR

16 bit 3D vectors

Structure

```
typedef struct {
                    short  vx, vy;
                    short  vz, pad;
} SVECTOR;
```

Members

vx, vy, vz	Vector components
pad	Padding

Comments

SVECTOR defines the structure of 16bit 3D vectors.

CVECTOR

8bit 3D vectors (colour vectors)

Structure

```
typedef struct {
                              unsigned char  r, g, b, cd;
} CVECTOR;
```

Members

r, g, b	Vector components
cd	Padding

Comments

CVECTOR defines the structure of 8 bit colour vectors.

MATRIX

3 x 3 matrices

Structure

```
typedef struct  {
                        short  m[3][3];
                        long   t[3];
} MATRIX;
```

Members

m	3x3 matrix coefficient values
t	Amount of translation

Comments

Each component is specified using the m[i][j] part of MATRIX.

The amount of translation after conversion is specified using the t[] part of MATRIX.

Ordering table headers

Structure

```
struct GsOT {
                unsigned short  length;
                GsOT_TAG  *org;
                unsigned short  offset;
                unsigned short  point;
                GsOT_TAG  *tag;
};
```

Members

length	OT bit length
org	Top address of the GsOT_TAG table
offset	OT offset on the Z axis in the screencoordinate system
point	OT representative value on the Z axis in the screen coordinate system
tag	Pointer to the current top GsOT_TAG

Comments

GsOT indicates the ordering table header.

This header holds the pointers, org and tag that point to the actual ordering table. org and tag are both initialised using the GsClearOt() function.

tag points to the top of the ordering table.

The GsDrawOt() function draws the ordering table to whichtag points.

The value of tag changes because the top is changed using the GsSortClear() or GsSortOt() functions. org is therefore provided to continue to hold the top of the ordering table.

The size of the ordering table is set by length. length can be set to any value between 1 and 14. When length is set to 1, org points to a 0~1 GsOT_TAG array, and when length is set to 14, org points to a 0~16384 GsOT_TAG array.

The GsClearOt() function initialises an area of memory from org up to the size specified by length. Accordingly, it is important to be aware that if the size of the GsOT_TAG array pointed to by org is less than the size indicated by length, memory may be damaged.

point refers to the representative value of the ordering table when ordering tables are sorted among themselves by means of theGsSortOt() function.

offset sets the ordering table offset on the Z axis. For example, if offset = 256 the top of the ordering table will be at Z = 256. (*1)

Notes

The values of length and org must be set at the initialisation stage. The other members are set using the GsClearOt() function.

*1 Not supported at present.

See also

GsClearOt(),GsDrawOt(),GsSortOt(),GsCutOt()

GsOT_TAG

Ordering table units

Structure

```
struct GsOT_TAG {
                        unsigned  p : 24;
                        unsigned char  num : 8;
};
```

Members

p	OT ring pointer
num	Word number packet

Comments

The ordering table array will be the array of thisGsOT_TAG.

The ordering table is the "list structure" that points to successive addresses. In the case of the 32bit address, the lower order 24bit can be displayed byp.

The GsOT_TAG array of the size set by theGsOT member length is secured when the ordering table is placed in memory.

GsDOBJ2

For GsCOORDINATE2 3D object handler

Structure

```
struct GsDOBJ2 {
                              unsigned long  attribute;
                              GsCOORDINATE2  *coord2;
                              unsigned long  *tmd;
                              unsigned long  id;
};
```

Members

attribute	Object attribute (32bit)
coord2	Pointer to local coordinate system
tmd	Pointer to modelling data
id	Reserved for the system

Comments

3D models can be manipulated via the structure GsDOBJ2, which is used as the handler for each 3D model. GsLinkObject4() is used to link GsDOBJ2 to the modelling data of the TMD file.

Access to linked TMD data is possible via GsDOBJ2. GsSortObject4() is used to register GsDOBJ2 in the ordering table.

coord2 is the pointer to the coordinate system inherent in the object.

The position, gradient and size of the object are reflected in the coordinate system pointed by coord2 by setting the matrix.

tmd holds the top address of the modelling data stored in memory in TMD format. tmd is calculated and set by GsLinkObject4().

attribute is 32bit, and various attributes are set here for the purpose of display. Comments on each bit are as follows.

(a) Light source calculation ON/OFF switch (bit 6)

This bit is used when the light source calculation is removed.

Texture-mapped polygons are displayed in original texture colour when the light source calculation is removed. Unmapped polygons are displayed in modelling data colour as they are.

(b) Automatic division function switch (bit 9-11)

0:	No automatic division
1:	2x2 division
2:	4x4 division
3:	8x8 division
4:	16x16 division
5:	32x32 division

This bit specifies the division number of automatic division. Automatic division is the function for automatically dividing one polygon at the time of execution. It is used for decreasing texture distortion and preventing deficiency in neighbouring polygons. However, division should be kept to a minimum in order to increase the number of polygons in exponential function terms.

(c) Semi-transparency ON/OFF (bit30)

This puts semi-transparency ON/OFF.

The highest order bit (STP bit) of the texture colour field (texture pattern when direct is set, CLUT colour field when indexed is set) must be used together with this bit in order to set semi-transparency. Pixel unit semi-transparency/opacity can also be controlled by using this STP bit.

(d) Display ON/OFF (bit31)

This puts display ON/OFF.

GsCOORDINATE2

Matrix type coordinate system

Structure

```
struct GsCOORDINATE2 {
                    unsigned long  flg;
                    MATRIX coord;
                    MATRIX workm;
                    GsCOORD2PARM  *param;
                    GsCOORDINATE2  *super;
};
```

Members

flg	Flag as to whether or not coord has been rewritten
coord	Matrix
workm	The result from this coordinate system to the WORLD coordinate system
param	Pointer for using scale, rotation and transfer parameters
super	Pointer to the parent coordinates

Comments

GsCOORDINATE2 holds parent coordinates and is defined according to the MATRIX type coord.

When the matrix is multiplied by the GsGetLw() or GsGETLs() function in each node of GsCOORDINATE2 from the WORLD coordinates, its result is held in workm

However, it does not store the result in workm of the coordinate system that is directly connected to the WORLD coordinate system.

At the time of GsGetLw() and GsGetLs() calculation, flg is referred to in order to avoid calculation of nodes that have already been calculated. 1 is to set, 0 is to clear.
The programmer must take responsibility for clearing this flag if the content of coord is changed. Otherwise, the GsGetLw()and GsGetLs() functions will be defective.

GsVIEW2

Viewpoint position (MATRIX type)

Structure

```
struct GsVIEW2 {
                MATRIX view;
                GsCOORDINATE2  *super
};
```

Members

view	Matrix for conversion from parentcoordinates to viewpoint coordinates
super	Pointer to the coordinate system that sets the viewpoint

Comments

GSVIEW2 sets the viewpointcoordinate system. It directly specifies the matrix for converting from the parentcoordinate system to the viewpointcoordinate system in view. The setting function is GsSetView2().

GsRVIEW2

Viewpoint position (REFERENCE type)

Structure

```
struct GsRVIEW2 {
                long  vpx, vpy, vpz;
                long  vpx, vpy, vpz;
                long  rz;
                GsCOORDINATE2  *super
};
```

Members

vpx, vpy, vpz	Viewpoint coordinates
vrx, vry, vrz	Reference point coordinates
rz	Viewpoint twist
super	Pointer to the coordinate system that sets the viewpoint (GsCOORDINATE2 type)

Comments

GsVIEW2 holds the viewpoint information, and is set according to the GsSetRefView2() function.

The coordinates of the viewpoint in the coordinate system displayed by super are set in (vpx, vpy, vpz).

The coordinates of the reference point in the coordinate system displayed by super are set in (vrx, vry, vrz).

rz is specified in fixed decimal point format with the gradient for the screen z axis when the z axis is the vector from the viewpoint to the reference point, set so that 4096 is one degree.

The coordinate systems of the viewpoint and reference point are set in super. For example, a cockpit view can be easily created with this function by setting super in the coordinate system of an aeroplane.

GsF_LIGHT

Parallel light source

Structure

```
struct GsF_LIGHT {
                long  vx, vy, vz;
                unsigned char  r, g, b;
};
```

Members

vx, vy, vz	Light source direction vectors
r, g, b	Light colours

Comments

GsF_LIGHT holds parallel light source information and is set in the system by the GsSetFlatLight() function.

Up to three parallel light sources can be set at the same time.

Sets the direction vectors of the light source in (vx, vy, vz). The programmer does not have to carry out standardisation as this is done by theGsSetFlatLight function.

The light shines strongest on normal vector polygons whose directions are opposite to these vectors.

Sets the colours of the light source in (r,g,b) by 8bit.

GsFOGPARAM

Fog (depth queue) information

Structure

```
struct GsFOGPARAM {
                        short  dqa;
                        long  dqb;
                        unsigned char  rfc, gfc, bfc;
};
```

Members

dqa	Parameter of the degree of merging in relation to depth
dqb	Parameter of the degree of merging in relation to depth
rfc, gfc, bfc	Background colours

Comments

dqa and dqb are the attenuation coefficients to the background colour.

dqa and dqb can be shown according to the following formula.

$$dqa = -df * 4096/64/h$$
$$dqb = 1.25 * 4096 * 4096$$

df is where the attenuation coefficients become one. In other words it is the distance from the viewpoint to the point where the background colour completely merges into the distant view.

h is the distance from the viewpoint to the screen. In other words it indicates the projection distance.

GsIMAGE

Image data configuration information

Structure

```
struct GsIMAGE {
        short  pmode;
        short  px, py;
        unsigned short  pw, ph;
        unsigned long  *pixel;
        short  cx, cy;
        unsigned short  cw, ch;
        unsigned long  *clut;
}
```

Members

pmode	Pixel mode 0: 4bit CLUT 1: 8bit CLUT 2: 16bit DIRECT 3: 24bit DIRECT 4: Other mode mixtures
px, py	Pixel data storage positions
pixel	Pointer to pixel data
cx, cy	CLUT data storage positions

cw, ch	CLUT data width/ height
clut	Pointer to CLUT data

Comments

GsImage is the structure for storing TIM format data information using the GsGetTimInfo() function.

For file format, please refer to the NetYaroze Members' Web site.

Sprite handler

Structure

```
struct GsSPRITE {
                unsigned long  attribute;
                short  x, y;
                unsigned short  w, h;
                unsigned short  tpage;
                unsigned char  u, v;
                short  cx, cy;
                unsigned char  r, g, b;
                short  mx, my;
                short  scalex, scaley;
                long  rotate;
};
```

Members

attribute	32bit length attribute (details are given below)
x, y	Top left-hand point display positions
w, h	Sprite width and height (not displayed when either w or h is 0)
tpage	Sprite pattern texture page number
u, v	Sprite pattern in-page offset
cx, cy	Sprite CLUT address

r, g, b	Brightness is set for each of r, g and b when they are displayed (Original brightness when it is 128)
mx, my	Rotation/ expansion central coordinates
scalex, scaley	x and y direction scaling values
rotate	Rotation angle (Units: 4096 = 1° (degree))
attribute bits	

6: Brightness regulation

0:	ON
1:	OFF

24-25: Sprite pattern bit mode

0:	4bitCLUT
1:	8bitCLUT
2:	15bitDirect

27: Rotation scaling function

0:	ON
1:	OFF

28-29: Semi-transparency rate

0:	0.5 x Back + 0.5 x Forward
1:	1.0 x Back + 1.0 x Forward
2:	1.0 x Back - 1.0 x Forward
3:	1.0 x Back + 0.25 x Forward

30: Semi-transparency ON /OFF

0:	Semi-transparency OFF
1:	Semi-transparency ON

31: Displayed/ Not displayed

0:	Display
1:	No display

Comments

GsSPRITE is the structure that holds information for displaying sprites and prepares one for each sprite displayed. The sprites can be operated via the parameters.

Either GsSortSprite() or GsSortFastSprite() may be used to register GsSPRITE in the ordering table.

The on-screen display position is specified as (x, y). The points specified as (mx, my) in the sprite pattern are the positions specified in the GsSortSprite() function, and the top left-hand points of the sprites are the positions specified in the GsSortFastSprite() function.

The width and length of the sprites are specified in pixel units as (w, h).

Texture page numbers, where there are sprite patterns, are specified as tpage (0~31).

The top left-hand points of the sprite patterns are specified with in-page offset as (u, v). A range (0,0)~(255,255) can be specified.

The top positions of CLUT (Colour palette) are specified by the VRAM address as (cx, cy) (only valid at the time of 4bit/8bit).

Brightness is specified for each of r, g and b as (r, g, b). Values from 0~255 can be specified. The brightness of the original pattern is attained at 128 and double the brightness at 255.

Rotation expansion central coordinates are given as (mx, my) as relative coordinates whose origins are the top left-hand points of the sprites. For example, one half of the width and length is specified if it is rotated at the centre of the sprite.

The scaling values are given for the x and y directions as (scalex, scaley). The unit is 4096 = 1.0 (original size). It can be set up to a maximum of eight times.

rotate sets rotation around the Z axis in fixed decimal point format with 4096 as 1 degree.

attribute is 32bit in which various attributes are set for display.

Comments on each bit are as follows.

(a) Brightness adjustment ON/OFF switch (bit 6)

This sets whether or not the sprite pattern pixel colours are to be drawn with brightness adjusted according to the (r,g,b) values. When it is 1, brightness is not adjusted and the (r,g,b) values are disregarded.

(b) Bit mode (bit 24-25)

In the sprite patterns there are 4bit and 8bit modes that use colour tables and a 15bit mode that displays colour directly. This is specified here.

(c) Rotation scaling function (bit 27)

Switches the sprite expansion function ON/OFF. If it is switched off when sprite rotation and expansion are not carried out, processing will be speeded up.

This bit is also disregarded in the case of the GsSortFastSprite() function, and the expansion function is always turned OFF.

(d) Semi-transparency rate (bit 28-29)

Sets the method of pixel blending when semi-transparency is turned ON with bit 30. Normal semi-transparent processing is performed when set to 0, pixel addition when set to 1, pixel subtraction when set to 2, and 25% addition when set to 3.

(e) Semi-transparency ON/OFF (bit 30)

It turns semi-transparency ON/OFF.

The highest order bit (STP bit) of the texture colour field (texture pattern when direct is set, CLUT colour field when indexed is set) must be used together with this bit in order to set semi-transparency.

Pixel unit semi-transparency/opacity can also be controlled by using this STP bit.

(f) Display ON/OFF (bit 31)

Turns display ON/OFF.

BG (background picture) handler

Structure

```
struct GsBG {
                        unsigned long  attribute;
                        short  x, y;
                        short  w, h;
                        short  scrollx, scrolly;
                        unsigned char  r, g, b;
                        GsMAP  *map;
                        short  mx, my;
                        short  scalex, scaley;
                        long  rotate;
};
```

Members

attribute	Attribute
x, y	Display positions of the top left-hand points
w, h	BG display size (pixel unit)
scrollx, scrolly	x,y scroll value
r, g, b	Brightness is set for each of r, g and b when they are displayed (Original brightness when 128)
map	Pointer to map data

mx, my	Rotation/ expansion central coordinates
scalex, scaley	x and y direction scaling values
rotate	Rotation angle (Units: 4096 = 1° (degree))

Comments

BG (Background) is a function for drawing one large rectangle constructed by the GsMAP data combining small rectangles defined by GsCELL data.
BG can be operated via the structure of this GsBG, which exists in each BG.

The on-screen display position is specified as (x, y).
The display size of BG is specified as (w, h). Units are pixels and do not depend on the cell size or the size of map.
The content of the map is also displayed repeatedly if the display area is larger than the size of the map. (Tiling function)
(scrollx, scrolly) are the display position offsets in the map and are specified in dot units.
Brightness is specified for each of r, g and b as (r, g, b). It becomes the original colour at 128 and double the brightness at 255.
map is the pointer to the GsMAP format map data to which the top address of the map data is specified.
Rotation expansion central coordinates are given as (mx, my) as relative coordinates whose origins are the top left-hand points of BG. For example, one half of the width and length is specified if it is rotated at the centre BG.
The scaling values are given for the x and y directions as (scalex, scaley). The unit is 4096 = 1.0 (original size). It can be set up to a maximum of eight times.
The rotation angle around the z axis is specified as rotate (4096 = 1 degree).
Please refer to GsSprite regarding attribute.

GsMAP

BG composition MAP

Structure

```
struct GsMAP {
        unsigned char  cellw, cellh;
        unsigned short  ncellw, ncellh;
        GsCELL  *base;
        unsigned short  *index;
};
```

Members

cellw, cellh	Cell size (taken as 256 in the case of 0)
ncellw, ncellh	Size of BG (unit is cell)
base	Pointer to the GsCELL structure array
index	Pointer to the cell array information

Comments

GsMAP is map data (cell array information) for composing BG with GsCELL. The map data controls the information by cell index array.

The size of one cell is specified in pixel units as (cellw, cellh). Note also that one BG is formed from a cell of the same size.

The size of map held by BG is specified in cell units as (ncellw, ncellh).

The top address of the GsCell array is set as base.

The top address of the cell array information table is set as index. The cell array information indicates the index value for the above array shown in base as ncellw x ncellh. A NULL cell (transparent cell) is indicated if the index value is 0xffff.

GsCELL

BG configuration cell

Structure

```
struct GsCELL {
                    unsigned char  u, v;
                    unsigned short  cba;
                    unsigned short  flag;
                    unsigned short  tpage;
};
```

Members

u	Offset from within the page (X direction)
v	Offset from within the page (Y direction)
cba	CLUT ID
flag	Inversion information
tpage	Texture page number

Comments

GsCELL is the structure holding information about the cell that composes BG and it is secured in the memory as an array.

The position of the sprite pattern corresponding to its cell is specified as (u, v) by offset in the page specified as tpage.

cba is the data that displays the position within the frame buffer of the CLUT corresponding to its cell, as follows.

Bit	Value
bit0~5	X position of CLUT/16
bit6~15	Y position of CLUT

flag holds information as to whether or not that cell displays the original texture pattern inversely.

Bit	Value
bit0	Vertical inversion (no inversion when set to 0, inversion when set to 1)
bit1	Horizontal inversion (no inversion when set to 0, inversion when set to 1)
bit2~15	Reserved

tpage is the page number displaying the position within the frame buffer of the sprite pattern.

GsLINE

Straight line handler

Structure

```
struct GsLINE {
        unsigned long  attribute;
        short  x0, y0;
        short  x1, y1;
        unsigned char  r, g, b;
};
```

Members

attribute Attribute

28-29: Semi-transparency rate

0:	0.5 x Back + 0.5 x Forward
1:	1.0 x Back + 1.0 x Forward
2:	1.0 x Back - 1.0 x Forward
3:	1.0 x Back + 0.25 x Forward

30: Semi-transparency ON OFF

0:	Semi-transparency OFF
1:	Semi-transparency ON

31: Display ON OFF

0:	Display
1:	No display

x0, y0	Position of drawing start point
x1, y1	Position of drawing end point
r, g, b	Drawing colour

Comments

GsLINE is the structure that holds information necessary for drawing straight lines. The GsSortLine() function is used to registerGsLINE in the ordering table.
attribute is 32bit, and various attributes are set here for the purpose of display.

(a) Semi-transparency rate (bit28-29)

GsLINE sets the pixel blending method when semi-transparency is turned ON by bit30. Normal semi-transparency processing is performed when set to 0, pixel addition when set to 1, pixel subtraction when set to 2, and 25% addition when set to 3.

(b) Semi-transparency ON/OFF (bit30)

Turns semi-transparency ON/OFF

(c) Display ON/OFF (bit31)

Turns display ON/OFF

GsGLINE

Gradation straight line handler

Structure

```
struct GsGLINE {
            unsigned long  attribute;
            short  x0, y0;
            short  x1, y1;
            unsigned char  r0, g0, b0;
            unsigned char  r1, g1, b1;
};
```

Members

attribute   Attribute

28-29: Semi-transparency rate

0:	0.5 x Back + 0.5 x Forward
1:	.0 x Back + 1.0 x Forward
2:	1.0 x Back - 1.0 x Forward
3:	1.0 x Back + 0.25 x Forward

30: Semi-transparency ON OFF

0:	Semi-transparency OFF
1:	Semi-transparency ON

31: Display ON OFF

0:	Display
1:	No display

x0, y0	Position of drawing start point
x1, y1	Position of drawing end point
r0, g0, b0	Start point drawing colour
r1, g1, b1	End point drawing colour

Comments

GsGLINE is the structure that holds information necessary for drawing gradation straight lines. It is the same as for GsLINE except that drawing colour specification can be separately set at the start point and end point.

Structure

```
struct GsBOXF {
                unsigned long  attribute;
                short  x, y;
                unsigned short  w, h;
                unsigned char  r, g, b;
};
```

Members

attribute	Attribute

28-29: Semi-transparency rate

0:	0.5 x Back + 0.5 x Forward
1:	1.0 x Back + 1.0 x Forward
2:	1.0 x Back - 1.0 x Forward
3:	1.0 x Back + 0.25 x Forward

30: Semi-transparency ON OFF

0:	Semi-transparency OFF
1:	Semi-transparency ON

31: Display ON OFF

0:	Display
1:	No display

x, y	Display position (top left-hand point)
x, y	Size of rectangle (width, height)
r, g, b	Drawing colour

Comments

GsBOXF is the structure that holds information necessary for rectangles painted by single colours. The GsSortBoxFill() function is used to registerGsBOXF in the ordering table.

ResetGraph

Initialises graphics system

Format

```
int ResetGraph(
int mode
)
```

Arguments

mode	Set mode
	0: All reset. The drawing environment and display environment are initialised.
	1: The current drawing is cancelled and the command queue is flushed.

Comments

It resets the graphics system with the mode that is specified bymode.

Return Value

None

SetDispMask

Sets display mask

Format

```
void SetDispMask(
int mask
)
```

Arguments

mask	0:	Display is not carried out in ‘Display’.
	1:	Display is carried out in ‘Display’.

Comments

It allows display to ‘Display’

Return Value

None

PutDrawEnv

Sets drawing environment

Format

```
DRAWENV *PutDrawEnv(
DRAWENV *env
)
```

Arguments

env	Drawing environment

Comments

Sets the basic parameters relating to drawing, e.g. drawing offset and drawing clip area.

Return Value

Top address of env

Notes

The drawing environment specified by PutDrawEnv() is valid until PutDrawEnv() is executed or GsSwapDispBuff() is called.

See Also

GsSwapDispBuff(), DRAWENV

PutDispEnv

Sets display environment

Format

```
DISPENV *PutDispEnv(
DISPENV *env
)
```

Arguments

env	Display environment

Comments

PutDispEnv sets the display environment. The display environment is immediately executed at the point in time when the function is called.

Return Value

Top address of env

Notes

The drawing environment specified byPutDispEnv() is valid until PutDispEnv() is executed or GsSwapDispBuff() is called.

See Also

GsSwapDispBuff(), DISPENV

LoadImage

Transmits data to frame buffer

Format

```
int LoadImage(
RECT *recp,
u_long *p
)
```

Arguments

recp	Transmission destination rectangular area
p	Transmission source main memory address

Comments

LoadImage transmits data below the addressp to the rectangular area of the frame buffer specified by recp.

Return Value

Queue number

Notes

Actual completion of the transmission needs to be identified byDrawSync() because it is a non-blocking function.
The transmission area is not affected by the drawing environment (clip and offset).
The transmission area needs to fit into the area in which drawing is possible (0,0) - (1023,511).

StoreImage

Transmits data from frame buffer

Format

```
int StoreImage(
RECT *recp,
u_long *p
)
```

Arguments

recp	Transmission source rectangular area
p	Transmission destination main memory address

Comments

StoreImage transmits the rectangular area of the frame buffer specified by recp to below the address p.

Return Value

Queue number

Notes

Actual completion of the transmission needs to be identified by DrawSync() because it is a non-blocking function.
The transmission area is not affected by the drawing environment (clip and offset).
The transmission area needs to fit into the area in which drawing is possible (0,0) - (1023,511).

MoveImage

Transmits data between frame buffer

Format

```
int MoveImage(
RECT *recp,
int x,
int y
)
```

Arguments

recp	Transmission source rectangular area
x,y	Transmission destination rectangular area top left-hand point

Comments

MoveImage transmits the rectangular area of the frame buffer specified by recp to a rectangular area of the same size starting from x,y.

Return Value

Queue number

Notes

Actual completion of the transmission needs to be identified by DrawSync() because it is a non-blocking function.

The transmission area is not affected by the drawing environment (clip and offset).
The transmission area needs to fit into the area in which drawing is possible (0,0) - (1023,511) for both the transmission source and transmission destination.
The content of the transmission source is stored. Also, the function cannot be guaranteed if the areas of transmission source and transmission destination are overlapping,

ClearImage

Frame buffer high speed painting

Format

```
int ClearImage(
RECT *recp,
u_char r,
u_char g,
u_char b
)
```

Arguments

recp	Painting rectangular area
r, g, b	Painting pixel value

Comments

ClearImage paints the rectangular area of the frame buffer specified by recp with the (r,g,b) brightness value.

Return Value

Queue number

Notes

Actual completion of the transmission needs to be identified by DrawSync() because it is a non-blocking function.

The transmission area is not affected by the drawing environment (clip and offset).

GetTPage

Calculates primitive tpage member value

Format

```
u_short GetTPage (
int tp,
int abr,
int x,
int y
)
```

Arguments

tp	Texture mode	
	0:	4bitCLUT
	1:	8bitCLUT
	2:	16bitDirect
abr	Semi-transparency rate	
	0:	0.5 x Back + 0.5 x Forward
	1:	1.0 x Back + 1.0 x Forward
	2:	1.0 x Back - 1.0 x Forward
	3:	1.0 x Back + 0.25 x Forward
x, y	Texture page address	

Comments

GetTPage calculates the texture page ID and returns it.

Return Value

Texture page ID

Notes

The semi-transparency rate is also valid for polygons that do not carry out texture mapping.
The texture page address is limited to multiples of 64 in the x direction and multiples of 256 in the y direction.

GetClut

Calculates primitive clut member value

Format

```
u_short GetClut (
int x,
int y
)
```

Arguments

x, y	CLUT frame buffer address

Comments

GetClut calculates the texture CLUT ID and returns it.

Return Value

CLUT ID

Notes

The CLUT address is limited to multiples of 16 in the x direction.

DrawSync

Waits for completion of all drawing

Format

int DrawSync(

int mode

)

Arguments

mode	0:	Waits for completion of all non-block functions registered in the queue.
	1:	The current rank number of the queue is checked and returned.

Comments

DrawSync waits for completion of the drawing.

Return Value

Actual queue rank number

VSync

Waits for vertical synchronisation

Format

```
int VSync(
int mode
)
```

Arguments

mode	0:	Blocking until vertical synchronisation occurs.
	1:	The time elapsed from the point in time whenVSync() was previously called is returned in units of one horizontal synchronisation interval.
	n:	(n>1) Counting from the point in time whenVSync() was previously called and blocking up to n times the occurrence of vertical synchronisation.
	n:	(n<0) Absolute time from program activation is returned in vertical synchronisation interval units.

Comments

Vsync waits for vertical synchronisation.

Return Value

mode>=0	Time elapsed from point in time whenVSync() was previously called (horizontal return unit)
mode<0	Time elapsed from program activation (vertical return unit)

VSyncCallback

Sets vertical synchronisation callback function

Format

```
int VSyncCallback(
void (*func)()
)
```

Arguments

func	Callback function

Comments

the function func is called when vertical return section commence.

Callback does not occur when 0 is specified infunc.

Return Value

None

Notes

Subsequent drawing completion interruptions are masked withinfunc. Therefore, func needs to return as soon as possible after completion of the necessary processing.

FntLoad

Transmits font pattern

Format

```
void FntLoad(
int tx,
int ty
)
```

Arguments

tx, ty	Top left coordinate of the area of frame buffer that arranges the font patterns

Comments

FntLoad transmits to the frame buffer the font pattern used for debugging.

Return Value

None

Comments

FntLoad loads the basic font pattern (4bit texture 256x128) to the frame buffer, and initialises all print streams.

Notes

FntLoad() must without fail be executed beforeFntOpen() and FntFlush().
The font area must not conflict with the frame buffer area used by the application.

FntOpen

Opens print stream

Format

```
int FntOpen(
int x,
int y,
int w,
int h,
int isbg,
int n
)
```

Arguments

x, y	Display start positions
w, h	Display area
isbg	Background automatic clearance 0: Background is cleared to (0,0,0) when displayed. 1: Background is not cleared to (0,0,0) when displayed.
n	Number of letters

Comments

FntOpen opens the stream used for printing on screen. Thereafter, the largest n character string of letters can be printed in the rectangular area of the frame buffer (x,y)-(x+w, y+h) using the FntPrint() function.

If 1 is specified in isbg, the background is cleared when a character string is drawn.

Return Value

Print stream ID

Notes

Up to 8 streams can be opened at the same time.

Opened streams cannot be closed until the next FntLoad() is called.

FntPrint

Output to print stream

Format

```
int FntPrint(
int id,
format,
...
)
```

Arguments

id	Print stream ID
format	Print format

Comments

FntPrint sends the character string to the print stream by theprintf() interface.

Return Value

Character string within the stream

Notes

The actual display of the character string occurs whenFntFlush() is executed.

FntFlush

Draws print stream contents

Format

```
u_long *FntFlush(
int id
)
```

Arguments

id	Print Stream ID

Comments

FntFlush draws the print stream in the frame buffer.

Return Value

Temporary OT top pointer used in drawing

Notes

After completion of drawing, the print stream contents are also flushed.

KanjiFntOpen

Opens print stream

Format

```
int KanjiFntOpen(
int x,
int y,
int w,
int h,
int dx,
int dy,
int cx,
int cy,
int isbg,
int n
)
```

Arguments

x, y	Display start positions
w, h	Display area
dx,dy	Kanji font pattern frame buffer address
cx,cy	Kanji clut frame buffer address
isbg	Background automatic clearance 0: Background is cleared to (0,0,0) when displayed. 1: Background is not cleared to (0,0,0) when displayed.
n	Number of letters

Comments

KanjiFntOpen opens the stream used for printing on screen. Thereafter, the largest n character string can be printed in the rectangular area of the frame buffer(x,y)-(x+w, y+h) using the KanjiFntPrint() function.
If 1 is specified in isbg, the background is cleared when a character string is drawn.

Return Value

Print stream ID

Notes

Up to 8 streams can be opened at the same time.
Opened streams cannot be closed until the next KanjiFntLoad() is called.
The Kanji font area must not conflict with the frame buffer area used by the application.

KanjiFntClose

Closes print stream

Format

int KanjiFntClose(void)

Arguments

None

Comments

This function closes all the streams currently openans are used by KanjiFntPrint() and initialize the state.

Return Value

None

Notes

Since KanjiFntClose() only initializes the internal state, this function operations even when there is no stream.

KanjiFntPrint

Outputs to print stream

Format

```
int KanjiFntPrint(
int id,
format,
...
)
```

Arguments

id	Print stream ID
format	Print format

Comments

KanjiFntPrint sends the SHIFT-JIS full-width character string to the print stream by the printf() interface.

Return Value

Character string within the stream

Notes

The Kanji code must be SHIFT-JIS.
Full-width and half-width characters can be mixed in the character string, but they are all changed to full-width at the time of display. Half-width kana are not supported. The actual display of the character string occurs whenKanjiFntFlush() is executed.

KanjiFntFlush

Draws print stream contents

Format

```
u_long *KanjiFntFlush (
int id
)
```

Arguments

id	Print Stream ID

Comments

FntFlush draws the print stream contents in the frame buffer.

Return Value

Temporary OT top pointer used in drawing

Notes

After completion of drawing, the print stream contents are also flushed.

Krom2Tim

Converts SHIFT-JIS character strings to 4 bit CLUT data

Format

```
int Krom2Tim(
u_char *sjis,
u_long *taddr,
int dx,
int dy,
int cx,
int cy,
u_int fg,
u_int bg
)
```

Arguments

sjis	SHIFT-JIS Character String
taddr	Data storage area
dx, dy	px,y coordinates on pixel data VRAM
cx, cy	x,y coordinates on clut data VRAM
fg, bg	Character colour and bg colour

Comments

Krom2Tim converts the SHIFT-JIS character string to 4 bitsclut TIM data and returns to taddr.

Return Value

-1 is returned if an irregular code is transferred.

Notes

The Kanji code must be SHIFT-JIS. Full-width and half-width characters can be mixed in the character string , but they are all changed to full-width at the time of display. Half-width kana are not supported.
For the area specified by taddr, the size shown in the following formula must be secured in advance.

128 x (character string specified by sjis) + 84(byte)

Krom2Tim2

Converts SHIFT-JIS character strings to 4 bit CLUT Tim data

Format

```
int Krom2Tim2(
u_char *sjis,
u_long *taddr,
int dx,
int dy,
int cdx,
int cdy,
u_int fg,
u_int bg
)
```

Arguments

sjis	SHIFT-JIS Character String
taddr	Starting address of the converted TIM data
dx, dy	Pixel data x,y coordinates on VRAM
cx, cy	Clut data x,y coordinates on VRAM
fg, bg	Front and background colour

Comments

Krom2Tim2 converts the SHIFT-JIS character string to 4 bits clut TIM data and returns the starting address in taddr. This is user defined character support version of Krom2Tim.

Return Value

-1 is returned if an invalid code is transferred.

Notes

The Kanji code must be in SHIFT-JIS. Although both ZENKAKU (double byte) and HANKAKU (single byte) can be mixedwitha string, all of them will not be converted to ZENKAKU. Please notethant HANKAKU KANA is not supported.

Prior to calling this function, the area specified by'addr' must be reserved with the size derived from the equation below.

Num: number of characters specified bysjis.

```
If (num<16)
        (32 * num + 16) * 4 (bytes)
else
        (32 * 16* ((num-1/16 + 1) +16) * 4 (bytes)
```

MulMatrix0

Takes product of two matrices

Format

```
MATRIX*MulMatrix0(
MATRIX *m0,
MATRIX *m1,
MATRIX *m2
)
```

Arguments

m0,m1	Input matrix
m2	Output matrix

Comments

MulMatrix0 takes the product of the two matrices m0 and m1. The value is stored inm2.

The argument format is as follows.

m0,m1,m2->m[i][j] : (1,3,12)

Return Value

m2

Notes

The rotation matrix is fragmented

ApplyMatrix

Multiplies vector by matrix

Format

```
VECTOR* ApplyMatrix(
MATRIX *m,
SVECTOR *v0,
VECTOR *v1
)
```

Arguments

m	Input multiplication matrix
v0	Input short vector
v1	Output vector

Comments

ApplyMatrix multiplies from the right the short vectorv0 by the matrix m and stores the result in the vector v1.

The argument format is as follows.

m->m[i][j] : (1,3,12)

v0->vx,vy,vz :(1,15,0)

v1->vx,vy,vz :(1,31,0)

Return Value

v1

Notes

The rotation matrix is fragmented.

ApplyMatrixSV

Multiplies vector by matrix

Format

```
SVECTOR*ApplyMatrixSV(
MATRIX *m,
SVECTOR *v0,
SVECTOR *v1
)
```

Arguments

m	Input multiplication matrix
v0	Input short vector
v1	Output short vector

Comments

ApplyMatrixSV multiplies from the right the short vectorv0 by the matrix m and stores the result in the short vector v1.

The argument format is as follows.

m->m[i][j] : (1,3,12)

v0->vx,vy,vz :(1,15,0)

v1->vx,vy,vz :(1,15,0)

Return Value

v1

Notes

The rotation matrix is fragmented.

ApplyMatrixLV

Multiplies vector by matrix

Format

```
VECTOR* ApplyMatrixLV(
MATRIX *m,
VECTOR *v0,
VECTOR *v1
)
```

Arguments

m	Input multiplication matrix
v0	Input vector
v1	Output vector

Comments

ApplyMatrixSV multiplies from the right the short vectorv0 by the matrix m and stores the result in the short vector v1.

The argument format is as follows.

m->m[i][j] : (1,3,12)

v0->vx,vy,vz :(1,31,0)

v1->vx,vy,vz :(1,31,0)

Return Value

v1

Notes

The rotation matrix is fragmented

RotMatrix

Searches for rotation matrix from rotation angle

Format

```
MATRIX*RotMatrix(
MATRIX *m
SVECTOR *r
)
```

Arguments

m	Output rotation matrix
r	Input rotation angle

Comments

RotMatrix supplies to matrix m the rotation matrix according to the rotation angle (r->vx,r->vy,r->vz). The rotation angle supplies 4096 as 360°, and 4096 is given as 1.0 for the matrix component.

The matrix is an expansion of the following product. Using the GTE coordinate conversion function, the vectors are multiplied from the right, thus the matrix rotates around the Z, Y and X axes in that order.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix} * \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} * \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Angle value

c0=cos(r->vx), s0=sin(r->vx)

c1=cos(r->vy), s1=sin(r->vy)

c2=cos(r->vz), s2=sin(r->vz)

The argument format is as follows.

m->m[i][j] : (1,3,12)

r->vx,vy,vz :(1,3,12) (however 360° is 1.0)

Return Value

m

RotMatrixX

Searches for rotation matrix around the X Axis

Format

```
MATRIX*RotMatrixX(
long r,
MATRIX *m
)
```

Arguments

r	Input rotation angle
m	Input and output rotation matrix

Comments

RotMatrixX supplies to matrix m the matrix multiplied by the rotation matrix around the X axis according to the rotation angle r. The rotation angle supplies 4096 as 360°, and 4096 is given as 1.0 for the matrix component.

The matrix is as follows.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} * m$$

* c=cos(r), s=sin(r)

The argument format is as follows.

m->m[i][j] : (1,3,12)

r:(1,3,12) (however 360° is 1.0)

Return Value

m

RotMatrixY

Searches for rotation matrix around the Y Axis

Format

```
MATRIX*RotMatrixY(
long r,
MATRIX *m
)
```

Arguments

r	Input rotation angle
m	Input and output rotation matrix

Comments

RotMatrixY supplies to matrix m the matrix multiplied by the rotation matrix around the Y axis according to the rotation angle r. The rotation angle supplies 4096 as 360°, and 4096 is given as 1.0 for the matrix component.

The matrix is as follows.

$$\begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} * m$$

* c=cos(r), s=sin(r)

The argument format is as follows.

m->m[i][j] : (1,3,12)

r:(1,3,12)(however 360° is 1.0)

Return Value

m

RotMatrixZ

Searches for rotation matrix around the Z Axis

Format

```
MATRIX*RotMatrixZ(
long r,
MATRIX *m
)
```

Arguments

r	Input rotation angle
m	Input and output rotation matrix

Comments

RotMatrixZ supplies to matrix m the matrix multiplied by the rotation matrix around the Z axis according to the rotation angle r. The rotation angle supplies 4096 as 360°, and 4096 is given as 1.0 for the matrix component.

The matrix is as follows.

$$\begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} * m$$

*	c=cos(r), s=sin(r)

The argument format is as follows.

m->m[i][j] : (1,3,12)

r:(1,3,12)(however 360° is 1.0)

Return Value

m

TransMatrix

Supplies amount of translation

Format

```
MATRIX*TransMatrix(
MATRIX*m,
VECTOR*v
)
```

Arguments

m	Output matrix
v	Input shift vector

Comments

TransMatrix supplies to matrix m the amount of translation shown byv.

The argument format is as follows.

m->m[i][j] : (1,3,12)

m->t[i]: (1,31,0)

v->vx,vy,vz : (1,31,0)

Return Value

m

ScaleMatrix

Supplies scaling factor

Format

```
MATRIX*ScaleMatrix(
MATRIX*m,
VECTOR*v
)
```

Arguments

m	Output matrix
v	Input scale vector

Comments

ScaleMatrix supplies to matrix m the scaling factor shown by v. It is a fixed decimal point number with 4096 as 1.0 for the v component.

$$\text{If } m=\begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix}, v=\begin{bmatrix} sx & sy & sz \end{bmatrix}$$

$$\text{then } m=\begin{bmatrix} a00*sx & a01*sy & a02*sz \\ a10*sx & a11*sy & a12*sz \\ a20*sx & a21*sy & a22*sz \end{bmatrix}$$

The argument format is as follows.

m->m[i][j] : (1,3,12)

v->vx,vy,vz : (1,19,12)

Return Value

m

ScaleMatrixL

Supplies scaling factor

Format

```
MATRIX*ScaleMatrixL (
MATRIX*m,
VECTOR*v
)
```

Arguments

m	Output matrix
v	Input scale vector

Comments

ScaleMatrixL supplies to matrix m the scaling factor shown by v. It is a fixed decimal point number with 4096 as 1.0 for the v component.

$$\text{If } m = \begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix}, v = \begin{bmatrix} sx & sy & sz \end{bmatrix}$$

$$\text{then } m = \begin{bmatrix} a00 * sx & a01 * sy & a02 * sz \\ a10 * sx & a11 * sy & a12 * sz \\ a20 * sx & a21 * sy & a22 * sz \end{bmatrix}$$

The argument format is as follows.

m->m[i][j] : (1,3,12)

v->vx,vy,vz : (1,19,12)

Return Value

m

TransposeMatrix

Supplies rotation value matrix

Format

```
MATRIX*TransposeMatrix(
MATRIX*m0• C
MATRIX*m1
)
```

Arguments

m0	Input matrix
m1	Output matrix

Comments

TransposeMatrix supplies to m1 the rotation value matrix of matrix m0.

The argument format is as follows.

m0->m[i][j] : (1,3,12)

m1->m[i][j] : (1,3,12)

Return Value

m1

CompMatrix

Carries out coordinate conversion synthesis

Format

```
MATRIX*CompMatrix(
MATRIX*m0,
MATRIX*m1,
MATRIX*m2
)
```

Arguments

m0	Input matrix
m1	Input matrix
m2	Output matrix

Comments

CompMatrix carries out synthesis ofcoordinate conversion matrices including translation.

[m2->m] = [m0->m] * [m1->m]

(m2->t) = [m0->m] * (m1->t) + (m0->t)

However the value of them1->t component must be within the range of $\left(-2^{15}, 2^{15}\right)$.

The argument format is as follows.

m0->m[i][j] : (1,3,12)

m0->t[i]: (1,31,0)

m1->m[i][j] : (1,3,12)

m1->t[i]: (1,15,0)

m2->m[i][j] : (1,3,12)

m2->t[i]: (1,31,0)

Return Value

m2

Notes

The rotation matrix is fragmented.

PushMatrix

Evacuates rotation matrix to stack

Format

```
void PushMatrix( void )
```

Arguments

None

Comments

PushMatrix evacuates the rotation matrix to the stack. The stack is up to 20 levels.

Return Value

None

PopMatrix

Resets rotation matrix from stack

Format

```
void PopMatrix( void )
```

Arguments

None

Comments

PopMatrix resets the rotation matrix from the stack.

Return Value

None

gteMIMefunc

Adds differential data array from multiplication of vertex data array by coefficient

Format

```
void gteMIMefunc(
SVECTOR *otp,
SVECTOR *dfp,
long n,
long p
)
```

Arguments

otp	Input/output vertex array
dfp	Input differential array
n	Input vertex (differential) data number
p	Input MIMe weight (control) coefficient

Comments

gteMIMefunc is a subroutine which executes interpolation using the differential data array and the vertex data array used in the multiple interpolation (MIMe) operation.

p is the fixed decimal point data of the decimal 12bit.

This function executes at high speed the same operation as the following program.

```
void gteMIMefunc(SVECTOR *otp, SVECTOR *dfp, long n, long p)
{
                    int i;
                        for( i = 0; i < n; i++){
                                (otp+i)->x += ( (int)((dfp+i)->x) * p )>>12;
                                (otp+i)->y += ( (int)((dfp+i)->y) * p )>>12;
                                (otp+i)->z += ( (int)((dfp+i)->z) * p )>>12;
                        }
}
```

The argument format is as follows.

p : (1,19,12)

otp, dfp optional

Return Value

None

GsInitGraph

Graphics system initialisation

Format

```
void GsInitGraph (
int x_res,
int y_res,
int intl,
int dither,
int vram
)
```

Arguments

x_res	Horizontal resolution (256/320/384/512/640)
y_res	Vertical resolution (240/480)
intl	Interlace display flag (bit 0)
	0: Non-interlace
	1: Interlace
	Double buffer offset mode (bit 2)
	0: GTE offset
	1: GPU offset
dither	Whether or not dither when drawing
	0: OFF
	1: ON
vram	Frame buffer mode
	0: 16bit
	1: 24bit

Comments

GsInitGraph initialises the graphics system.
The GPU setting is notified by the global variablesGsDISPENV andGsDRAWENV, so the program GPU setting can be confirmed and changed by referring toGsDISPENV and GsDRAWENV.
The double buffer offset mode decides whether the double buffer offset is executed by GTE or by GPU. It is easier to handle when executed by GPU because the double buffer offset value is not included in the packet.
In the 24-bit mode, only image display is possible. Polygon drawing etc. is not possible.
Because initialisation of the graphics system includesGsIDMATRIX and GsIDMATRIX2 initialisation, none of the Gs * * * functions operate normally unlessGsInitGraph() has been called.

Return Value

None

GsInit3D

3D graphics system initialisation

Format

void GsInit3D (void)

Arguments

None

Comments

GsInit3D initialises the 3D graphics system within the library.

3D graphics system needs to be initialised by this function first, so that 3D processing functions such as GsSetRefView(), GsInitCoordinate2() and GsSortObject4() can be used.

The following process is executed.

(1)	The screen origin is held in the screen centre.
(2)	The light source defaults to LIGHT_NORMAL.

Return Value

None

Notes

With this function, the graphics system must firstly beintialised by GsInitGraph().

See Also

GsInitGraph(), GsSetRefView(), GsInitCoordinate2(), GsSortObject4()

GsDefDispBuff

Double buffer definition

Format

```
void GsDefDispBuff (
int x0,
int y0,
int x1,
int y1,
)
```

Arguments

x0, y0	Buffer 0 origin (top left-hand)coordinates
x1, y1	Buffer 1 origin (top left-hand)coordinates

Comments

GsDefDispBuff defines the double buffer. (x0, y0) and (x1,y1) are specified by the coordinate value within the frame buffer. In default, the buffer 0 becomes (0, 0) and buffer 1 becomes (0, y_res).

y_res is the vertical resolution specified byGsInitGraph(). The double buffer is cancelled when (x0, y0) and (x1, y1) have the same coordinate values. Switching the double buffer of the even number field and odd number field is automatically carried out if it is left in this mode when the interlace mode is specified.

Double buffer switching is carried out by theGsSwapDispBuff() function.

The double buffer is executed by GPU or GTE offset. GsInitGraph() sets whether execution of offset is by GPU or by GTE. If the double buffer is executed using the GPU offset, the coordinate value is created in the coordinate system whose origin is the top left-hand point of the double buffer in the packet. On the other hand, if the double buffer is executed using the GTE offset, thecoordinate value is created in the coordinate system whose origin is the origin (top left-hand point) of the frame buffer in the packet.

Return Value

None

See Also

GsInitGraph(), GsSwapDispBuff()

GsSwapDispBuff

Double buffer switching

Format

```
void GsSwapDispBuff( void )
```

Arguments

None

Comments

GsSwapDispBuff changes the display buffer and drawing buffer according to double buffer information that has been set byGsDefDispBuff(). Execution is usually carried out immediately after vertical return section surge.
Also, the following processes are executed within the function.

(1) Display commencement address setting
(2) Cancellation of blanking
(3) Double buffer index setting
(4) 2 dimensional clipping switched
(5) GTE or GPU offset setting
(6) Offset setting
(7) PSDCNT increment

The double buffer is executed by the offset. The third argument ofGsInitGraph() decides whether the offset is set by GTE or by GPU (GsOFSGPU orGsOFSGTE is specified).

Return Value

None

Notes

If GPU is drawing, this function does not operate smoothly and it needs to be called immediately after drawing completion has been confirmed byDrawSync(0) or after the drawing has been ended by ResetGraph(1).

See Also

GsDefDispBuff()

GsGetActiveBuff

Gets drawing buffer number

Format

int GsGetActiveBuff(void)

Arguments

None

Comments

GsGetActiveBuff gets the double buffer index (PSDIDX). The index value is either 0 or 1. The frame buffer top 2 dimensional address of the double buffer origin (top left coordinate) is found by entering the index in the external variables PSDOFSX[ ] and PSDOFSY[ ].

Return Value

The double buffer index (0 when buffer 0 and 1 when buffer 1) is returned.

See Also

PSDIDX

GsSetDrawBuffOffset

Drawing offset update

Format

void GsSetDrawBuffOffset(void)

Arguments

None

Comments

GsSetDrawBuffOffset updates the offset for drawing. The set value is represented in the global conversion POSITION.

This offset is relative within the double buffer, and the offset value is maintained even if the double buffer is switched.

The setting of GTE or GPU is executed if this function is called. The third argument of GsInitGraph() decides whether the offset is executed by GTE or by GPU(GsOFSGPU or GsOFSGTE is specified).

Return Value

None

Notes

This function does not operate smoothly if GPU is drawing, and it needs to be called immediately after completion of drawing has been confirmed by DrawSync(0) or after drawing has been ended by ResetGraph(1).

See Also

GsSetOrign(), GsSetOffset(), POSITION

GsSetOffset

Offset setting

Format

```
void GsSetOffset (
int offx,
int offy
)
```

Arguments

offx	Drawing offset X
offy	Drawing offset Y

Comments

GsSetOffset specifies the drawing offset. It is different fromGsSetDrawBuffOffset() in that GsSetDrawBuffOffset() sets the value of the global variable POSITION, whereas GsSetOffset() sets the offset supplied by the argument.
Also, the value set by GsSetOffset() is temporary and the offset values that are set on execution of GsSwapDispBuff() and GsSetDrawBuffOffset() become invalid. On the other hand, the set values of GsSetDrawBuffOffset() are valid until changed byGsSetOrigin().
The offset supplied by the argument is relative within the double buffer. In other words, the offset actually set is the base offset of the double buffer added to the offset supplied by the argument.
The third argument of GsInitGraph() decides whether the offset is executed by GTE or by GPU (GsOFSGPU orGsOFSGTE is specified).

Return Value

None

Notes

This function does not operate smoothly if GPU is drawing, and it needs to be called immediately after completion of drawing has been confirmed byDrawSync(0) or after drawing has been ended by ResetGraph(1).

See Also

GsSetDrawBuffOffset()

GsSetDrawBuffClip

Drawing clipping area update

Format

void GsSetDrawBuffClip(void)

Arguments

None

Comments

GsSetDrawBuffClip updates the drawing clip. It actually represents the clip value set by GsSetClip2D(). The set value is valid until theGsSetDrawBuffClip() function is called once more by a different clip value.
Moreover, this clip value is relative within the double buffer, and the position of the clip does not change even if the double buffer is switched.

Return Value

None

Notes

This function does not operate smoothly if GPU is drawing, and it needs to be called immediately after completion of drawing has been confirmed byDrawSync(0) or after drawing has been ended by ResetGraph(1).

See Also

GsSetClip2D(), GsSetClip()

GsSetClip

Drawing clipping area setting

Format

```
void GsSetClip (
RECT *clip
)
```

Arguments

clip	RECT structure for setting the clipping area

Comments

GSetClip sets the clip for drawing. The set value is valid until theGsSwapDispBuff() function is called next. It is different fromGsSetDrawBuffClip() in that the place where the clip area can be specified by the argument and the validity period of the set value are different.

Moreover, this clip value is relative within the double buffer.

Return Value

None

Notes

This function does not operate smoothly if GPU is drawing, and it needs to be called immediately after completion of drawing has been confirmed byDrawSync(0) or after drawing has been ended by ResetGraph(1).

See Also

GsSetDrawBuffClip()

GsGetTimInfo

Checks TIM format header

Format

```
void GsGetTimInfo(
unsigned long *tim,
GsIMAGE *im
)
```

Arguments

tim	TIM data top address
im	Pointer to image structure

Comments

TIM format information specified by the argument tim is stored in im.
The top of the TIM data is the address that skipped the ID. In other words, it has an offset 4 bytes forward from the top of the TIM file.

For file format, please refer to the NetYaroze Members' Web site.

Return Value

None

See Also

GsIMAGE

GsMapModelingData

Maps TMD data to an actual address

Format

```
void GsMapModelingData(
unsigned long *p
)
```

Arguments

p	Top address of TMD data

Comments

The offset address from the top of the TMD data is stored because at the time of TMD data creation it is uncertain where it is going to be loaded onto the memory.
The GsMapModelingData() function converts this offset address into an actual address, and this conversion must be carried out first of all in order to use the TMD data.
The TMD data top address is the one that skipped the ID. In other words, it has an offset 4 bytes forward from the top of the TMD file.
For file format, please refer to the NetYaroze Members' Web site.

Return Value

None

Notes

A flag stands in the TMD data converted to an actual address, so that no side effects will occur even ifGsMapModelingData() is called for a second time.

GsLinkObject4

Links object and TMD data

Format

```
void GsLinkObject4(
unsigned long *tmd,
GsDOBJ2 *obj_base,
unsigned long n
)
```

Arguments

tmd	Top address of the linking TMD data
obj_base	Array of the object structure to be linked
n	Index of the linking object

Comments

GsLinkObject4 links the TMD data (nth) object with the object structure of GsDOBJ2, so that TMD 3D objects can be handled by GsDOBJ2.

Return Value

None

Notes

Objects linked by GsLinkObject4() can be registered in OT by GsSortObject4().

See Also

GsSortObject4(), GsDOBJ2

GsSetRefView2

Viewpoint position setting

Format

```
int GsSetRefView2(
GsRVIEW2 *pv
)
```

Arguments

pv	Viewpoint position information (viewpoint: steady viewpoint type)

Comments

GsSetRefView2 calculates the WSMATRIX (World Screen Matrix) from the viewpoint information. If the viewpoint does not move, the WSMATRIX does not change and does not need to be called each frame. However, when the viewpoint moves, changes are not represented unless the WSMATRIX is called each frame.
When super of the GsRVIEW2 member is set outside WORLD, even if other parameters are not changed, GsSetRefView2() needs to be called each frame becausethe viewpoint moves if the parent coordinate system parameters change.

Return Value

0 is returned when viewpoint setting is successful, 1 when it fails.

See Also

GsRVIEW2,GsWSMATRIX, GsSetView2()

GsSetView2

Viewpoint setting

Format

```
int GsSetView2(
GsVIEW2 *pv
)
```

Arguments

pv	Viewpoint position information (matrix type)

Comments

GsSetView2 directly sets the WSMATRIX (World Screen Matrix). If the viewpoint is moved, errors can arise due to inaccuracy in the process that searches WSMATRIX from the viewpoint steady viewpoint using GsSetRefView2(), and so it is advantageous to use GsSetView2().

When super of the GsVIEW2 member is set outside WORLD, GsSetRefView2() needs to be called each frame even if other parameters are not changed. This is becausethe viewpoint moves unless the parent coordinate system parameters change.

The screen aspect ratio is regulated automatically if GsIDMATRIX2 is used in the basic matrix.

Return Value

0 is returned if setting is successful, 1 if it fails.

See Also

GsVIEW2, GsWSMATRIX, GsSetRefView2()

GsSetProjection

Projection plane position setting

Format

```
void GsSetProjection(
unsigned short h
)
```

Arguments

h	Distance between viewpoint and projection plane (projection distance), default is 1000.

Comments

GsSetProjection regulates the field of view.
The projection is the distance from the viewpoint to the projection plane.
The size of the projection plane is set by theGsInitGraph() argumentsxres, yres. The field of view narrows if the projection distance is enlarged and expands if it is reduced, because the size of the projection plane is fixed according to the resolution.
Be careful, because sometimes aspect ration is not 1 to 1, depending on the resolution. In this case, the scale of Y coordinates is made 1/2 and the aspect ratio is adjusted.

Resolution	640x480	640x240	320x240
Aspect ratio	1:1	2:1	1:1

Return Value

None

GsSetFlatLight

Parallel light source setting

Format

```
void GsSetFlatLight (
unsigned short id,
GsF_LIGHT *lt
)
```

Arguments

id	Light source number (0,1,2)
lt	Light source information

Comments

GsSetFlatLight sets the parallel light source. The light source can be set up to three (id = 0, 1, 2).

Light source information is given by the GsF_LIGHT structure.

Return Value

None

Notes

Even if the contents of the GsF_LIGHT structure are rewritten, the setting is not represented unless this function is called.

See Also

GsF_LIGHT, GsSetAmbient()

GsSetLightMode

Light source mode setting

Format

```
void GsSetLightMode(
unsigned short mode
)
```

Arguments

mode	Light source mode (0~1) 0: normal lighting 1: normal lighting fog ON

Comments

GsSetLightMode sets the light source mode.
The light source calculation method can also be set by the status bit (attribute) of each object (GsDOBJ2). Setting by the status bit is used in precedence to the status setting.

Return Value

None

GsSetFogParam

Fog parameter setting

Format

```
void GsSetFogParam (
GsFOGPARAM *fogparam
)
```

Arguments

fogparam	Pointer to fog parameter structure

Comments

GsSetFogParam carries out fog parameter setting. Fog is only effective if the light mode is 1.

Return Value

None

See Also

GsFOGPARAM, GsSetLightMode()

GsSetAmbient

Ambient colour setting

Format

```
void GsSetAmbient(
unsigned short r,
unsigned short g,
unsigned short b
)
```

Arguments

r, g, b	RGB value of the ambient colour (0~4095)

Comments

GsSetAmbient sets ambience (ambient light). Setting is carried out in each ofr, g and b according to what fraction of unlit parts there are to lit parts. 1/1 becomes 4096 and 1/8 becomes 4096/8.

Return Value

None

See also

GsSetFlatLight()

GsInitCoordinate2

Local coordinate system initialisation

Format

```
void GsInitCoordinate2(
GsCOORDINATE2 *super,
GsCOORDINATE2 *base
)
```

Arguments

super	Pointer to parent coordinate system
base	Pointer to (initialising) coordinate system

Comments

GsInitCoordinate2 initialises the localcoordinate system. Initialisation ofbase- >coord is by the unit matrix, and base- >super by the coordinate system specified by the argument.

Return Value

None

See Also

GsCOORDINATE2

GsGetLw

Calculates local world matrix

Format

```
void GsGetLw (
GsCOORDINATE2 *coord,
MATRIX *m
)
```

Arguments

coord	Pointer to local coordinate system
m	Pointer to matrix

Comments

GsGetLw calculates the local world perspective conversion matrix from coord of the matrix type coordinate system GsCOORDINATE2 specified by the argument and stores the result in the MATRIX type structure m.

Also, the calculation result of each node of the hierarchical coordinate system is held in order to increase speed, and calculation up to nodes that are not changed is omitted even when the GsGetLw() function is next called.

This is controlled by the GsCOORDINATE2 flag (1 is substituted for the GsCOORDINATE2 flag after calculation). However, even when 1 is substituted for the flag, note that calculation will be carried out if the parent node has been changed.

Return Value

None

See Also

GsGetLws(), GsSetLightMatrix()

GsGetLs

Calculates local screen matrix

Format

```
void GsGetLs (
GsCOORDINATE2 *coord,
MATRIX *m
)
```

Arguments

coord	Pointer to local coordinate system
m	Pointer to matrix

Comments

GsGetLs calculates the perspective conversion matrix of the local screen from coord of the matrix type coordinate system GsCOORDINATE2 specified by the argument, and the result is stored in the MATRIX type structure m.

Also, the calculated result of each node of the hierarchical coordinate system is held in order to increase speed, and calculation up to nodes that are not changed is omitted even when the GsGetLw() function is next called.

This is controlled by the GsCOORDINATE2 flag (1 is substituted for the GsCOORDINATE2 flag after calculation). However, even when 1 is substituted for the flag, note that calculation will be carried out if the parent node has been changed.

Return Value

None

See Also

GsSetLsMatrix()

GsGetLws

Calculates both local world and local screen matrices

Format

```
void GsGetLws (
GsCOORDINATE2 *coord2
MATRIX *lw,
MATRIX *ls
)
```

Arguments

coord2	Pointer to local coordinate system
lw	Pointer to local world coordinate system
ls	Pointer to local screen coordinate system

Comments

GsGetLws calculates both the local worldcoordinates and the local screen coordinates at the same time from the localcoordinate system coord2, and stores them in lw and ls. It is faster than continuously callingGsGetLw() and GsGetLs().

The local world matrix must be specified if light source calculation is carried out at the time of execution, but in this case it is faster to search once withGsGetLws().

Return Value

None

See Also

GsGetLs(), GsGetWs()

GsScaleScreen

Scales screen coordinate system

Format

```
void GsScaleScreen(
SVECTOR *scale
)
```

Arguments

scale	The scale factor (12bit fixed decimal point format) GsScaleScreen sets the scale factor for the original screen coordinate system normally set by GsSetView2() and GsSetRefView2(). By entering ONE forvx, vy and vz , it returns to the original.

Comments

GsScaleScreen carries out scaling of the screencoordinate system with respect to the world coordinate system.
Problems such as the closeness of Far Clip occur because the screencoordinate system is only 16bit whereas the world coordinate system has a 32bit space. GsScaleScreen() is a function that resolves this problem, carries out scaling of the screencoordinates and covers a wider area for the worldcoordinates.
For example, the screen coordinate system expands to a 17bit equivalent size when ONE/2 is specified in (vx,vy,vz). However, as precision is 16bit, the bottom 1 bit is invalid.
At this time, screen coordinate systems with different scales should not be registered in OT with the same scale. For example, registration must be carried out by shifting to one extra bit, in order to register objects, calculated with the screencoordinate system of the

normal scaling, to the OT that registered the objects that were half the scale of the screen coordinate system.

Return Value

None

GsSetLsMatrix

Sets local screen matrix

Format

```
void GsSetLsMatrix(
MATRIX *mp
)
```

Arguments

mp	Local screen matrix to be set

Comments

GsSetLsMatrix sets the local screen matrix in GTE.

If perspective conversion process is carried out using GTE, the local screen matrix needs to be pre-set in GTE.

Because the GsSortObject4() function performs perspective conversion using GTE, GetLsMatrix() needs to be called beforehand.

Return Value

None

See Also

GsSortObject4(), GsGetLs()

GsSetLightMatrix

Sets light matrix

Format

```
void GsSetLightMatrix(
MATRIX *mp
)
```

Arguments

mp	Local screen light matrix to be set

Comments

GsSetLightMatrix multiplies the matrix of three light source vectors and the local screen light matrix mp supplied by the argument, and sets in GTE.
Depending on the type of modelling data to be handled, the GsSortObject4() function may perform light source calculation at the time of execution. In this case too, the light matrix needs to be pre-set using GsSetLightMatrix().
The matrix set as the GsSetLightMatrix() argument is normally the local world matrix.

Return Value

None

See Also

GsSortObject4(), GsGetLw()

GsClearOt

OT initialisation

Format

```
void GsClearOt (
unsigned short offset,
unsigned short point,
GsOT *otp
)
```

Arguments

offset	Ordering table offset value
point	Ordering table representative value Z
otp	Pointer to ordering table

Comments

GsClearOT initialises the ordering table displayed byotp. offset is the Z value at the top of that ordering table, and point is the Z value referred to when inserting that ordering table into another ordering table.
Also, the length of OT must be specified in advance in order to confirm the size to be cleared.

Return Value

None

See Also

GsOT, GsDrawOt()

GsDrawOt

Execution of drawing command allocated to OT

Format

```
void GsDrawOt (
GsOT *otp
)
```

Arguments

otp	Pointer to OT

Comments

GsDrawOt starts execution of the drawing command registered in OT displayed by otp.

GsDrawOt() immediately returns because the drawing process is carried out in the background.

Notes

If GPU is drawing, this function does not operate smoothly and it needs to be called immediately after drawing completion has been confirmed byDrawSync(0) or after drawing has been ended by ResetGraph(1).

Return Value

None

See Also

GsOT, GsClearOt()

GsSortObject4

Allocates object to ordering table

Format

```
void GsSortObject4(
GsDOBJ2 *objp,
GsOT *otp,
long shift,
u_long *scratch
)
```

Arguments

objp	Pointer to object
otp	Pointer to OT
shift	How many bits the value of Z is shifted to the right at the time of allocation to OT
scratch	Specifies scratchpad address

Comments

GsSortObject4 carries out perspective conversion and light source calculation for 3D objects to be handled by GsDOBJ2, and generates the drawing command in the packet area specified by GsSetWorkBase(). Next, it Z sorts the generated drawing command and allocates it to OT displayed byotp.

The precision of Z can be adjusted by the value ofshift. The maximum value of the ordering table size (resolution) is 14bit. However, if for example it is 12bit, then the value of shift is 2 (=14 - 12). At this time take care not to go over the area of the ordering table.

scratch is used as work when automatic division is carried out.

In order to validate the division by attribute which is the member of objp, OR is carried out by GsDIV5, which is the member of macro GsDIV1 objp defined by libps.h. One polygon
is divided into 4 sections of 2x2 at the time of GsDIV1 and into 1024 sections of 32x32 at the time of GsDIV5.
Also, scratchpad is cache memory and 256 words are packaged from 0x1f800000.

Return Value

None

See Also

GsDOBJ2, GsSetWorkBase()

GsSetWorkBase

Sets drawing command storage address

Format

```
void GsSetWorkBase(
PACKET *base_addr
)
```

Arguments

base_addr	Address that stores the drawing command

Comments

GsSetWorkBase sets the memory address that stores the drawing primitives generated by such functions as GsSortObject4() andGsSortSprite().
At the start of the process of each frame, it must be set in the top address of the packet area secured by the user.

Return Value

None

See Also

GsSortObject4(), GsSortSprite(), GsSortFastSprite(), GsOUT_PACKET_P

GsGetWorkBase

Gets current drawing command storage address

Format

```
PACKET *GsGetWorkBase( void )
```

Arguments

None

Comments

GsGetWorkBase gets the current drawing primitive packet address

The top address of the unused area can be got.

Return Value

The address that creates the next drawing primitive packet

See Also

GsSetWorkBase(), GsOUT_PACKET_P

GsSortClear

Registers drawing clear command in OT

Format

```
void GsSortClear (
unsigned char r,
unsigned char g,
unsigned char b,
GsOT *otp
)
```

Arguments

r, g, b	Background colour RGB Value
otp	Pointer to OT

Comments

GsSortClear sets the drawing clear command at the top of OT displayed byotp.

Return Value

None

Notes

GsSortClear only registers the clear command in the ordering table, and is not executed unless the drawing is started by the GsDrawOt() function.

GsSortSprite

Registers sprite in OT

Format

```
void GsSortSprite(
GsSPRITE *sp,
GsOT *otp,
unsigned short pri
)
```

Arguments

sp	Pointer to sprite
otp	Pointer to OT
pri	Position in OT

Comments

GsSortSprite allocates the sprite displayed bysp to the ordering table displayed byotp.
The parameters of sprite display positions, etc. are all supplied by thesp members.
pri is the priority order on the sprite ordering table. The highest value is 0 and the lowest value depends on the size of the ordering table. If a numerical value of the size of the ordering table or more is specified, it is clipped to the maximum value got by the ordering table.

Return Value

None

See Also

GsOT, GsSPRITE, GsSortFastSprite()

GsSortFastSprite

Registers sprite in OT

Format

```
void GsSortFastSprite(
GsSPRITE *sp,
GsOT *otp,
unsigned short pri
)
```

Arguments

sp	Pointer to sprite
otp	Pointer to OT
pri	Position in OT

Comments

GsSortSprite allocates the sprite displayed bysp to the ordering table displayed byotp. The parameters of sprite display positions, etc. are all supplied by thesp members.
pri is the priority order on the sprite ordering table. The highest value is 0 and the lowest value depends on the size of the ordering table. If a numerical value of the size of the ordering table or more is specified, it is clipped to the maximum value got by the ordering table.
In comparison with theGsSortSprite() function, GsSortFastSprite() is processed at high speed, although the scaling rotation function cannot be used. At this time, the value of the sprite structure members, mx, my, scalex, scaley and rotate are disregarded.

Return Value

None

See Also

GsSortSprite(),GsSPRITE

GsInitFixBg16

Initialises high-speed BG working area

Format

```
void GsInitFixBg16 (
GsBG *bg,
unsigned long *work
)
```

Arguments

bg	Pointer to GsBG
work	Pointer to working area (primitive area)

Comments

GsInitFixBg16 initialises the working area used by the GsSortFixBg16 () function. The size of the necessary array varies according to the screen resolution. The size can be found by the following formula (unit is long).

Size = (((ScreenW/CellW+1)*(ScreenH/CellH+1+1)*6+4)*2+2)

ScreenH: Screen height vertical dot number (240/480)
ScreenW: Screen height horizontal dot number (256/320/384/512/640)
CellH: Cell height (pixel number)
CellW: Cell width (pixel number)
GsInitFixBg16() should only be executed once, and does not need to be executed every frame.

Return Value

None

See Also

GsSortFixBg16()

GsSortFixBg16

Registers high-speed BG to OT

Format

```
void GsSortFixBg16(
GsBG *bg,
unsigned long *work,
GsOT *otp,
unsigned short pri
)
```

Arguments

bg	Pointer to GsBG
work	Pointer to working area (primitive area)
otp	Pointer to OT
pri	Position in OT

Comments

GsSortFixBg16 carries out BG data registration processing to the ordering table.

BG rotation/scaling/reduction not possible.

Cell size fixed (16x16).

Texture pattern colour mode 4bit/8bit only.

Map size is optional.

Scrolling possible (1 pixel unit)

Full screen only

This function needs working area for storing the drawing primitives. The working area is prepared as an unsigned long type array, and initialisation by GsInitFixBg16() needs to be carried out in advance.

Packet Area (the area set by GsSetWorkBase()) is not used.

Return Value

None

See Also

GsInitFixBg16()

GsSortLine

Registers straight lines to OT

Format

```
void GsSortLine(
GsLINE *lp,
GsOT *otp,
unsigned short pri
)
```

Arguments

lp	Pointer to GsLINE
otp	Pointer to OT
pri	Position in OT

Comments

GsSortLine allocates straight lines that are displayed bylp to ordering table displayed by otp.

Single colour straight lines are registered in OT byGsSortLine().

Return Value

None

See Also

GsSortGLine()

GsSortGLine

Registers straight lines to OT

Format

```
void GsSortGLine(
GsGLINE *lp,
GsOT *otp,
unsigned short pri
)
```

Arguments

lp	Pointer to GsGLINE
otp	Pointer to OT
pri	Position in OT

Comments

GsSortGLine allocates straight lines that are displayed bylp in the ordering table displayed by otp.
Straight lines with gradation are registered in OT byGsSortGLine().

Return Value

None

See Also

GsSortLine()

GsSortBoxFill

Registers rectangles to OT

Format

```
void GsSortBoxFill(
GsBOXF *bp,
GsOT *otp,
unsigned short pri,
)
```

Arguments

bp	Pointer to GsBOXF
otp	Pointer to OT
pri	Position in OT

Comments

GsSortBoxFill allocates rectangles displayed by bp to ordering table displayed by otp.

Return Value

None

GsSortOt

Allocates OT to another OT

Format

```
GsOT *GsSortOt (
GsOT *ot_src,
GsOT *ot_dest
)
```

Arguments

ot_src	Pointer to assigned source OT
ot_dest	Pointer to assigned destination OT

Comments

GsSortOt assigns the OT displayed byot_src to ot_dest.

The OTZ value used at this time is the representative value in theot_src point field.

The integrated OT is assigned to ot_dest.

Return Value

Pointer to integrated OT

See Also

GsOT

GsSetClip2D

2 dimensional clipping setting

Format

```
void GsSetClip2D(
RECT *rectp
)
```

Arguments

rectp	Clip area

Comments

GsSetClip2D sets the area displayed byrectp as the clipping area.
This setting is not influenced by the double buffer, and so once it is set, the same area is automatically clipped even if the double buffer is switched.
GsSetDrawBuffClip() needs to be called in order to validate this setting immediately afterwards. If GsSetDrawBuffClip() is not called, the setting becomes valid from the next frame.

Return Value

None

GsSetOrign

Screen origin position setting

Format

```
void GsSetOrign (
int x,
int y
)
```

Arguments

x	Screen origin position X
y	Screen origin position Y

Comments

GsSetOrign specifies the drawing offset.
The offset value set byGsSetOffset() is temporary and whereas the offset set when GsSwapDispBuff() orGsSetDrawBuffOffset() is called becomes invalid, the offset value set by GsSetOrign() is valid until next changed byGsSetOrign().
The offset supplied by the argument is relative within the double buffer. In other words, the offset actually set is the offset supplied by the argument added to the offset of the double buffer base. In reality, it is set byoffx and offy of the global variable POSITION.

Notes

The third argument of GsInitGraph() decides whether the offset is executed by GTE or by GPU (GsOFSGPU orGsOFSGTE is specified).

Return Value

None

GsIncFrame

Updates frame ID

Format

GsIncFrame()

Arguments

None

Comments

GsIncFrame is the macro called insideGsSwapDispBuff(). It applies one increment to PSDCNT. Although PSDCNT is 32bit, it does not become 0 even if it is recycled, and it starts from 1.

PSDCNT is referred to when the validity of the matrix cache is determined byGsGetLw(), GsGetLs() and GsGetLws().

If the double buffer is switched without usingGsSwapDispBuff() and GsGetLw(), GsGetLs() and GsGetLws() are used, this macro needs to be called every time the double buffer is switched.

See Also

PSDCNT, GsGetLw(), GsGetLs(), GsGetLws(), GsSwapDispBuff()

Table: Graphics External Variables

Global	Type	Description
CLIP2	RECT	2 dimensional clipping area
PSDOFSX [2]	unsigned short	Double buffer base point (X coordinate) Set by GsDefDispbuff()
PSDOFSY [2]	unsigned short	Double buffer base point (Y coordinate) Set by GsDefDispbuff()
PSDIDX	unsigned short	Double buffer index
PSDCNT	unsigned long	Number incremented by frame switch
POSITION	_GsPOSITION	2 dimensional offset
GsDRAWENV	DRAWENV	Drawing Environment
GsDISPENV	DISPENV	Display Environment
GsLSMATRIX	MATRIX	Local screen matrix Set by GsSetLs()
GsWSMATRIX	MATRIX	World screen matrix Set by GsSetRefView(), etc.
GsLIGHT_MODE	int	Default light mode
GsLIGHTWSMATRIX	MATRIX	Light matrix Set by GsSetFlatLight()
GsIDMATRIX	MATRIX	Unit matrix
GsIDMATRIX2	MATRIX	Unit matrix (including aspect conversion)
GsOUT_PACKET_P	unsigned long	Pointer holding top of packet area Set by GsSetWorkBase()
GsLMODE	unsigned long	Attribute decoding result (light mode)
GsLIGNR	unsigned long	Attribute decoding result (light disregarded)
GsLIOFF	unsigned long	Attribute decoding result (without shading)
GsNDIV	unsigned long	Attribute decoding result (division number)
GsTON	unsigned long	Attribute decoding result (semi-transparency)
GsDISPON	unsigned long	Attribute decoding result (display/ no display

2

Sound Functions

SndVolume

Structure

```
struct SndVolume {
                unsigned short left;
                unsigned short right;
};
```

Members

left	L channel volume value
right	R channel volume value

SsVabTransfer

Recognises and transmits sound source data

Format

```
short SsVabTransfer (
unsigned char vh_addr,
unsigned char vb_addr,
short vabid,
short i_flag
)
```

Arguments

vh_addr	VH data top address
vb_addr	VB data top address
vabid	VAB identification number
i_flag	Fixed at 1

Comments

SsVabTransfer recognises the sound source header list (VH data) specified byvh_addr, and transmits the sound source data (VB data) specified byvb_addr to the SPU sound buffer. It specifies the VAB identification number invabid. It searches and allocates an available VAB identification number (0 - 15) whenvabid is -1.

Return Value

VAB identification number

In the case of failure, the following values are returned according to the cause.

-1	VAB ID cannot be assigned or VH abnormality

-2	VB abnormality
-3 or below	Other abnormalities

See Also

SsVabClose()

SsVabClose

Closes VAB data

Format

```
void SsVabClose(
short vab_id
)
```

Arguments

vab_id	VAB data id

Comments

SsVabClose closes VAB data that holds vab_id.

Return Value

None

See Also

SsVabTransfer()

SsSeqOpen

Opens SEQ data

Format

```
short SsSeqOpen(
unsigned long* addr,
short vab_id
)
```

Arguments

addr	SEQ data main memory top address
vab_id	VAB id

Comments

SsSeqOpen analyses the SEQ data in the main memory, and returns the SEQ access number.

A maximum of 32SEQ data can be opened at the same time and if more than that are opened, -1 becomes the return value.

Return Value

SEQ access number (the number to be used within the SEQ data access function and the number of the SEQ data control table held internally).

See Also

SsSeqClose()

SsSeqClose

Closes SEQ data

Format

```
void SsSeqClose(
short seq_access_num
)
```

Arguments

seq_access_num	SEQ access number

Comments

SsSeqClose closes the SEQ data holding theseq_acces_numthat is no longer necessary.

Return Value

None

See Also

SsSeqOpen()

SsSeqPlay

SEQ data reading (musical performance)

Format

```
void SsSeqPlay(
short seq_access_num,
char play_mode,
short l_count
)
```

Arguments

seq_access_num	SEQ access number	
play_mode	Performance mode	
	SSPLAY_PAUSE	Switches to pause state
	SSPLAY_PLAY	Performs immediately
l_count	Number of tune repetitions	

Comments

According to the play_mode value, SsSeqPlay can select whether to begin reading (performing) the SEQ data immediately or switch to the pause state at the SEQ data top (tune top). At this time, it specifies the number of tune repetitions in l_count.

SSPLAY_INFINITY is specified if there is an infinite number of performances

Return Value

None

See Also

SsSeqPause(), SsPlayBack(), SsSeqStop()

SsSeqPause

Temporarily stops SEQ data reading (pause)

Format

```
void SsSeqPause(
short seq_access_num
)
```

Arguments

seq_access_num	SEQ access number

Comments

SsSeqPause temporarily stops the reading (performance) of SEQ data holding seq_access_num

Return Value

None

See Also

SsSeqPlay(), SsSeqReplay()

SsSeqReplay

Restarts SEQ data reading (replay)

Format

```
void SsSeqReplay(
short seq_access_num
)
```

Arguments

seq_access_num	SEQ access number

Comments

SsSeqReplay restarts the reading of the SEQ data holding seq_access_num that has been temporarily suspended by SsSeqPause.

Return Value

None

See Also

SsSeqPlay(), SsSeqPause()

SsSeqStop

Stops SEQ data reading (stop)

Format

```
void SsSeqStop(
short seq_access_num
)
```

Arguments

seq_access_num	SEQ access number

Comments

SsSeqStop ends the reading (performance) of the SEQ data holding seq_access_num

Return Value

None

See Also

SsSeqPlay()

SsSeqSetVol

SEQ volume setting

Format

```
void SsSeqSetVol(
short seq_access_num,
short voll,
short volr
)
```

Arguments

seq_access_num	SEQ access number
voll	L channel main volume value
volr	R channel main volume value

Comments

SsSeqSetVol sets the main volume of the tune holdingseq_access_numin sizes specified in the L and R channels respectively. 0 to 127 can be set.

Return Value

None

See Also

SsSeqGetVol()

SsSeqGetVol

Gets SEQ volume

Format

```
void SsSeqGetVol(
short access_num,
short seq_num,
short *voll,
short *volr
)
```

Arguments

access_num	SEQ access number
seq_num	Fixed at 0
voll	SEQ L volume value
volr	SEQ R volume value

Comments

SsSeqGetVol returns the current L and R volume values of SEQ to voll and volr respectively.

Return Value

None

See Also

SsSeqSetVol()

SsSeqSetNext

Next SEQ data specification

Format

```
void SsSeqSetNext (
short seq_access_num1,
short seq_access_num2
)
```

Arguments

seq_access_num1	SEQ access number
seq_access_num2	SEQ access number

Comments

SsSeqSetNext specifies the access numberseq_access_num2of the SEQ data next to be performed from SEQ data holdingseq_access_num1

Return Value

None

SsSeqSetRitardando

Slows tempo

Format

```
void SsSeqSetRitardando(
short seq_access_num,
long tempo,
long v_time
)
```

Arguments

seq_access_num	SEQ access number
tempo	Tune tempo
v_time	Time (tick unit)

Comments

SsSeqSetRitardando slows the data holding seq_access_num until resolution of tempo in v_time.

However, if the specified resolution is greater (faster) than the current resolution, the same operation as SsSeqSetAccelerando is carried out.

Return Value

None

See Also

SsSeqSetAccelerando()

SsSeqSetAccelerando

Accelerates tempo

Format

```
void SsSeqSetAccelerando(
short seq_access_num,
long tempo,
long v_time
)
```

Arguments

seq_access_num	SEQ access number
tempo	Tune tempo
v_time	Time (tick unit)

Comments

SsSeqSetAccelerando accelerates the data holding seq_access_num until resolution of tempo in v_time.

However, if the specified resolution is smaller (slower) than the current resolution, the same operation as SsSeqSetRitardando is carried out.

Return Value

None

See Also

SsSeqSetRitardando()

SsSetMVol

Main volume value setting

Format

```
void SsSetMVol(
short voll,
short volr
)
```

Arguments

voll	L channel volume value
volr	R channel volume value

Comments

SsSetMVol sets the main volume value involl and volr respectively. Each can be set from 0 to 127.

It is essential to set it before SEQ data is played.

Return Value

None

See Also

SsGetMVol()

SsGetMVol

Gets main volume value

Format

```
void SsGetMVol(
SndVolume *m_vol
)
```

Arguments

m_vol	Main volume value

Comments

SsGetMVol assigns the main volume value tom_vol.

Return Value

None

See Also

SsSetMVol()

SsSetMute

Mute setting

Format

```
void SsSetMute(
char mode
)
```

Arguments

mode	Setting mode	
	SS_MUTE_ON	Mute on
	SS_MUTE_OFF	Mute off

Comments

SsSetMute carries out mute setting.

Return Value

None

See Also

SsGetMute()

SsGetMute

Gets mute attributes

Format

char SsGetMute (void)

Comments

SsGetMute gets mute attributes.

Return Value

Mute attributes.

SS_MUTE_ON	Mute on
SS_MUTE_OFF	Mute off

See Also

SsSetMute()

SsPlayBack

SEQ data reading

Format

```
void SsPlayBack (
short access_num,
short seq_num,
short l_count
)
```

Arguments

access_num	SEQ access number
seq_num	Fixed at 0
l_count	Number of tune repetitions

Comments

SsPlayBack stops the tune during the current performance, and starts performance by returning to the top of that tune.

It specifies the number of tune repetitions inl_count. SSPLAY_INFINITY is specified in the case of an infinite number of performances.

Return Value

None

See Also

SsSeqPlay()

SsSetTempo

Sets tempo

Format

```
void SsSetTempo(
short access_num,
short seq_num,
short tempo
)
```

Arguments

access_num	SEQ access number
seq_num	Fixed at 0
tempo	Tune tempo

Comments

SsSetTempo sets the tempo.

This is valid if the tempo set by SsSeqPlay() is to be changed. After this function has been called, the performance is changed to the newly set tempo and played.

Return Value

None

SsIsEos

Judges whether or not in mid-performance

Format

```
short SsIsEos (
short access_num,
short seq_num
)
```

Arguments

access_num	SEQ access number
seq_num	Fixed at 0

Comments

SsIsEos judges whether or not the specified tune is in mid-performance.

Return Value

1 is returned if in mid-performance, 0 if not.

SsSetSerialAttr

CD audio attribute setting

Format

```
void SsSetSerialAttr (
char s_num,
char attr,
char mode
)
```

Arguments

s_num	Fixed as SS_CD
attr	Attribute value
mode	Setting mode

Comments

SsSetSerialAttr carries out attribute setting relating to CD audio.

attr = SS_MIX	Mixing
attr = SS_REV	Reverberation

mode= SS_SON	attr on
mode= SS_SOFF	attr off

Return Value

None

See Also

SsGetSerialAttr()

SsGetSerialAttr

Gets CD audio attribute value

Format

```
char SsGetSerialAttr (
char s_num,
char attr
)
```

Arguments

s_num	Fixed at SS_CD
attr	Attribute

Comments

SsGetSerialAttr returns the CD audio attribute value.

attr = SS_MIX	Mixing
attr = SS_REV	Reverberation

Return Value

Attribute value: 1 is returned if on and 0 if off.

See Also

SsSetSerialAttr()

SsSetSerialVol

CD audio volume value setting

Format

```
void SsSetSerialVol(
short s_num,
short voll,
short volr
)
```

Arguments

s_num	Fixed as SS_CD
voll	L channel volume value
volr	R channel volume value

Comments

SsSetSerialVol sets the CD volume value involl and volr.

The volume value can be set from 0 to 127.

Return Value

None

See Also

SsGetSerialVol()

SsGetSerialVol

Gets CD audio volume value

Format

```
void SsGetSerialVol (
char s_num,
SndVolume *s_vol
)
```

Arguments

s_num	Fixed at SS_CD
s_vol	CD audio volume value

Comments

SsGetSerialVol returns the CD audio volume value tos_vol.

Return Value

None

See Also

SsSetSerialVol()

SsUtKeyOn

Keys on voice

Format

```
short SsUtKeyOn (
short vabId,
short prog,
short tone,
short note,
short fine,
short voll,
short volr
)
```

Arguments

vabId	VAB number
prog	Program number
tone	Tone number
note	Half tone unit pitch specification (note number)
fine	Detailed pitch specification (100/127 cent specification)
voll	Volume (left)
volr	Volume (right)

Comments

SsUtKeyOn specifies and keys on the volume number (0 to 127), tone number (0 to 15) and VAB number for SE, and returns the allocated voice number.

Return Value

The voice number (0 to 23) used by key-on is returned.

-1 is returned in the event of failure.

See Also

SsUtKeyOff(),SsUtAllKeyOff()

SsUtKeyOff

Keys off voice

Format

```
short SsUtKeyOff(
short voice,
short vabId,
short prog,
short tone,
short note
)
```

Arguments

voice	Voice number
vabId	VAB number
prog	Program number
tone	Tone number
note	Half tone unit pitch specification (note number)

Comments

SsUtKeyOff keys off the voice that was keyed on by SsUtKeyOn.

Return Value

0 is returned if successful, -1 if it fails.

See Also

SsUtKeyOn(), SsUtAllKeyOff()

SsUtPitchBend

Bends pitch

Format

```
short SsUtPitchBend(
short voice,
short vabId,
short prog,
short note,
short pbend
)
```

Arguments

voice	Voice number
vabId	VAB number
prog	Program number
note	Half tone unit pitch specification (note number)
pbend	Pitch bend value

Comments

SsUtPitchBend bends pitch of voice keyed on bySsUtKeyOn().

Return Value

0 is returned if successful, -1 if it fails.

See Also

SsUtChangePitch()

SsUtChangePitch

Changes pitch

Format

```
short SsUtChangePitch(
short voice,
short vabId,
short prog,
short old_note,
short old_fine,
short new_note,
short new_fine
)
```

Arguments

voice	Voice number
vabId	VAB number
prog	Program number
old_note	Note number at the time of SsUtKeyOn
olde_fine	Detailed pitch at the time of SsUtKeyOn (note number)
new_note	Note number to be changed
new_fine	Detailed pitch to be changed (note number)

Comments

SsUtChangePitch changes the pitch of the voice keyed on by SsUtKeyOn().

Return Value

0 is returned if successful, -1 if it fails.

See Also

SsUtPitchBend()

SsUtSetVVol

Sets voice volume

Format

```
short SsUtSetVVol(
short vc,
short voll,
short volr
)
```

Arguments

vc	Voice number
voll	Volume (left)
volr	Volume (right)

Comments

SsUtSetVVol sets in detail the voice volume keyed on bySsUtKeyOn().

Return Value

0 is returned if successful, -1 if it fails.

See Also

SsUtGetVVol()

SsUtGetVVol

Gets voice volume

Format

```
short SsUtGetVVol(
short vc,
short *voll,
short *volr
)
```

Arguments

vc	Voice number
voll	Volume (left)
volr	Volume (right)

Comments

SsUtGetVVol returns the detailed value of the voice volume keyed on bySsUtKeyOn().

Return Value

0 is returned if successful, -1 if it fails.

See Also

SsUtSetVVol()

SsUtReverbOn

Reverberation on

Format

void SsUtReverbOn(void)

Arguments

None

Comments

SsUtReverbOn turns on the reverberation with the set type and depth.

Return Value

None

See Also

SsUtReverbOff()

SsUtReverbOff

Reverberation off

Format

```
void SsUtReverbOff( void )
```

Arguments

None

Comments

SsUtReverbOff turns the reverberation off.

Return Value

None

See Also

SsUtReverbOn()

SsUtSetReverbType

Sets reverberation type

Format

```
short SsUtSetReverbType(
short type
)
```

Arguments

type	Reverberation type

Type	Mode	Delay time *	Feedback*
SS_REV_TYPE_OFF	Off	X	X
SS_REV_TYPE_ROOM	Room	X	X
SS_REV_TYPE_STUDIO_A	Studio (small)	X	X
SS_REV_TYPE_STUDIO_B	Studio (medium)	X	X
SS_REV_TYPE_STUDIO_C	Studio (large)	X	X
SS_REV_TYPE_HALL	Hall	X	X
SS_REV_TYPE_SPACE	Space echo	X	X
SS_REV_TYPE_ECHO	Echo	O	O
SS_REV_TYPE_DELAY	Delay	O	O
SS_REV_TYPE_PIPE	Pipe echo	X	X

* Delay time and Feedback specification by reverberation type is possible

Comments

SsUtSetReverbType sets the reverberation type.

The reverberation depth is automatically set to 0 when the reverberation type is set.

When data is left in the reverberation work area, noise appears as soon as the depth is set, so the following procedure should be used.

```
SsUtSetReverbType(SS_REV...);
SsUtReverbOn();
                                    :
Takes several seconds
                                    :
SsUtSetReverbDepth(64,64);
```

Number and type response as above

Return Value

If setting is carried out correctly, the set type number is returned.

If setting is carried out incorrectly, -1 is returned.

See Also

SsUtGetReverbType(),SsUtSetReverbDepth(), SsUtSetReverbFeedback(), SsUtSetReverbDelay()

SsUtGetReverbType

Gets reverberation type

Format

short SsUtGetReverbType(void)

Arguments

None

Comments

SsUtGetReverbType gets the current reverberation type value.

Return Value

Current reverberation type value

See Also

SsUtSetReverbType()

SsUtSetReverbDepth

Sets reverberation depth

Format

```
void SsUtSetReverbDepth(
short ldepth,
short rdepth
)
```

Arguments

ldepth	0~127
rdepth	0~127

Comments

SsUtSetReverbDepth sets the reverberation depth.

Return Value

None

See Also

SsUtSetReverbType()

SsUtSetReverbFeedback

Sets feedback amount

Format

```
void SsUtSetReverbFeedback(
short feedback
)
```

Arguments

feedback	0~127

Comments

SsUtSetReverbFeedback sets the feedback amount if the echo type reverberation is used.

Return Value

None

See Also

SsUtSetReverbType()

SsUtSetReverbDelay

Sets delay amount

Format

```
void SsUtSetReverbDelay(
short delay
)
```

Arguments

delay	0~127

Comments

SsUtSetReverbDelay sets the delay amount if the echo and delay type reverberation is used.

Return Value

None

See Also

SsUtSetReverbType()

SsUtAllKeyOff

Keys off all voices

Format

```
void SsUtAllKeyOff(
short mode
)
```

Arguments

mode	Always 0

Comments

SsUtAllKeyOff compulsorily keys off all voices used by the sound service.

Return Value

None

See Also

SsUtKeyOn(), SsUtKeyOff(), SsSeqPlay()

3

Standard C Functions

abs

Calculates absolute value

Format

```
#include <stdlib.h>
long abs (
long i
)
```

Arguments

i	Integer value

Comments

abs calculates the absolute value of the integeri. This function is primarily for searching the absolute value of int type integers. However, as int type and long type have the same meaning in R3000, on this system it is a function equivalent to labs described next.

Return Value

The absolute value of the argument is returned.

See Also

labs()

labs

Calculates absolute value

Format

```
#include <stdlib.h>
long labs (
long i
)
```

Arguments

i	Integer value

Comments

labs calculates the absolute value of the integer i. On this system, it is a function equivalent to abs described previously.

Return Value

The absolute value of the argument is returned.

See Also

abs()

Converts character strings to integers

Format

```
#include <stdlib.h>
long atoi (
const char *s
)
```

Arguments

s	Character string

Comments

atoi is the same as (long)strtol(s,(char**)NULL). On this system it is a function equivalent to atol, which follows on next page.

Return Value

The result of converting the input values to an integer is returned.

See Also

atol(), strtol()

atol

Converts character strings to integers

Format

```
#include <stdlib.h>
long atol(
const char *s
)
```

Arguments

s	Character string

Comments

atol is the same as (long)strtol(s,(char**)NULL).

Return Value

The result of converting the input values to an integer is returned.

See Also

atoi(),strtol()

bzero

Pads memory blocks with zeros

Format

```
#include <memory.h>
void *bzero(
unsigned char *p,
int n
}
```

Arguments

p	Pointer to write start position
n	Write byte number

Comments

Writes n byte zeros from the address specified byp.

Return Value

Returns the pointer to the address where write starts.

See Also

bcopy(), bcmp()

Copies memory blocks

Format

```
#include <memory.h>
void bcopy(
char *src,
char *dest,
long n
)
```

Arguments

src	Copy source
dest	Copy destination
n	Copy byte number

Comments

bcopy copies the firstn byte of src to dest.

Return Value

None

See Also

memcpy()

Compares memory blocks

Format

```
#include <memory.h>
long bcmp(
char *b1,
char *b2,
long n
)
```

Arguments

b1	Comparison source 1
b2	Comparison source 2
n	Comparison byte number

Comments

bcmp compares the first n bytes of b1 and b2.

Return Value

The next value depending on the comparison result of b1 and b2 is returned.

Result	Return Value
b1<b2	<0
b1=b2	=0
b1>b2	>0

See Also

memcmp()

bsearch

Carries out binary searches

Format

```
#include <stdlib.h>
void *bsearch (
const void *key,
const void *base,
size_t n,
size_t w,
long(*fcmp)(const void *, const void *)
)
```

Arguments

key	Storage destination of retrieved value
base	Storage destination of retrieved array
n	Number of elements
w	Size of 1 element
fcmp	Comparison function

Comments

With fcmp as a comparison function, bsearch carries out a binary search of tables ofn items (size of item = w) starting from base, looking for items matchingkey.

Return Value

The address of the first item matching the retrieval key is returned. 0 is returned if there is no matching item.

calloc

Allocates main memory

Format

```
#include <stdlib.h>
void *calloc (
size_t n,
size_t s
)
```

Arguments

n	Number of articles
s	Block size

Comments

calloc secures the n x s byte block from the heap memory.

Return Value

The pointer to the secured memory block is returned.

NULL is returned in the event of failure.

See Also

malloc(), realloc(), free()

Allocates main memory

Format

```
#include <stdlib.h>
void *malloc (
size_t s
)
```

Arguments

s	Characters to be tested

Comments

malloc secures the s byte block from the heap memory.

Return Value

The pointer to the secured memory block is returned.

NULL is returned in the event of failure to secure.

* At the time of user program activation the heap memory is defined as follows.

Lowest address	Module's highest address + 4
Highest address	Package memory• 64KB

See Also

calloc(), realloc(), free()

realloc

Reallocates heap memory

Format

```
#include <stdlib.h>
void *realloc (
void *block,
size_t s
)
```

Arguments

block	Area to be reallocated
s	Area size

Comments

realloc reduces or enlarges the blockblock that was previously secured tos byte. Ifblock is NULL, it has the same operation asmalloc().

Return Value

The reallocated block address is returned. This address may be different from the original address. NULL is returned in the event of failure to allocate. At this time the original block cannot be opened.

See Also

calloc(), malloc(), free()

Opens allocated memory blocks

Format

```
#include <stdlib.h>
void fr ee (
void*block
)
```

Arguments

block	Area to be opened

Comments

free opens the memory block secured bycalloc(), malloc() and realloc().

Return Value

None

See Also

calloc(), malloc(), realloc()

memchr

Searches for characters in memory blocks

Format

```
#include <memory.h>
void *memchr (
const void *s,
long c,
size_t n
)
```

Arguments

s	Retrieved characters storage destination
c	Retrieved characters
n	Number of retrieved bytes

Comments

memchr locates the first appearance of the characterc in the memory block of then byte starting from s.

Return Value

The pointer to the located character is returned. NULL is returned whenc cannot be discovered.

memcmp

Carries out memory block comparison

Format

```
#include <memory.h>
long memcmp (
const void *s1,
const void *s2,
size_t n
)
```

Arguments

s1	Comparison source 1
s2	Comparison source 2
n	Comparison byte number

Comments

memcmp compares the first n bytes of s1 and s2.

Return Value

The following values are returned depending on the comparison result of s1 and s2.

Result	Return Value
s1<s2	<0
s1=s2	=0
s1>s2	>0

See Also

bcmp()

memcpy

Copies memory blocks

Format

```
#include <memory.h>
void *memcpy(
void *dest,
const void *src,
size_t n
)
```

Arguments

dest	Copy destination
src	Copy source
n	Copy byte number

Comments

memcpy copies the firstn byte of src to dest.

Return Value

dest is returned.

See Also

bcopy()

memmove

Copies memory blocks

Format

```
#include <memory.h>
void  *memmove(
void *dest,
const void *src,
size_t n
)
```

Arguments

dest	Copy destination
src	Copy source
n	Copy byte number

Comments

memmove copies the firstn byte of src to dest.

Accurate copying is performed even among duplicated objects.

Return Value

dest is returned.

Writes specified characters to memory blocks

Format

```
#include <memory.h>
void *memset (
const void *s,
long c,
size_t n
)
```

Arguments

s	Memory block
c	Character
n	Character number

Comments

memset writes c to the n byte memory block starting froms.

Return Value

s is returned.

qsort

Carries out quick sort

Format

```
#include <stdlib.h>
void qsort (
void *base,
size_t n,
size_t w,
long (*fcmp)(const void *, const void *)
)
```

Arguments

base	Storage destination of array to be sorted
n	Number of elements
w	Size of 1 element
fcmp	Comparison function

Comments

With fcmp as a comparison function, qsort sorts a table of n number of items (size of item = w) starting from base.

Take care with the empty heap area because malloc() is called internally.

Return Value

None

srand

Initialises random number generator

Format

```
#include <stdlib.h>
void srand (
unsigned int seed
)
```

Arguments

seed	Random number

Comments

srand sets the new starting point of the random number generation. Default is 1.

Return Value

None

See Also

rand()

rand

Generates random numbers

Format

```
#include <stdlib.h>
long rand ( void )
```

Arguments

None

Comments

rand generates pseudo random numbers between RAND_MAX(0x7FFF=32767) from 0.

Return Value

A generated pseudo random number is returned.

See Also

srand()

Adds one character string to another

Format

```
#include <strings.h>
char *strcat (
char *dest,
const char *src
)
```

Arguments

dest	Link destination character string
src	Link source character string

Comments

strcat adds src to the end of the character string dest.

Return Value

dest is returned.

See Also

strncat()

Searches for position of first appearance of a specified character in a character string

Format

```
#include <strings.h>
char *strchr (
const char *s,
long c
)
```

Arguments

s	Retrieved character string
c	Retrieved character

Comments

strchr searches for the position where the characterc first appears in the character string s.

Return Value

The address of the appearance position ofc is returned. NULL is returned ifc does not appear.

Compares character strings

Format

```
#include <strings.h>
long strcmp (
const char *s1,
const char *s2
)
```

Arguments

s1	Comparison source 1
s2	Comparison source 2

Comments

strcmp compares each character of s1 and s2 as unsigned char.

Return Value

The following values are returned depending on the comparison result of s1 and s2.

Result	Return Value
s1<s2	<0
s1=s2	=0
s1>s2	>0

strcpy

Copies one character string to another

Format

```
#include <strings.h>
char *strcpy (
char *dest,
const char *src
)
```

Arguments

dest	Copy destination character string
src	Copy source character string

Comments

strcpy copies src to the character string dest.

Return Value

dest is returned.

See Also

strncpy()

strcspn

Searches for first part of a character string comprising only characters not included in specified character set

Format

```
#include <strings.h>
size_t strcspn (
const char *s1,
const char *s2
)
```

Arguments

s1	Character string
s2	Character group

Comments

strcspn returns the length of the first part of a character string comprising only characters not included in the character string s2 within the character string s1.

Return Value

The length of the found section of the character string is returned.

Finds the number of characters in character string

Format

```
#include <strings.h>
long strlen (
const char *s
)
```

Arguments

s	Character string

Comments

strlen counts number of characters in the character strings.

Return Value

The character number is returned.

strncat

Adds one character string to another

Format

```
#include <strings.h>
char  *strncat (
char *dest,
const char *src,
size_t n
)
```

Arguments

dest	Link destination array
src	Link source character string
n	Link character number

Comments

strncat adds the largest n character from src to end of character string dest.

Return Value

dest is returned.

Compares character strings

Format

```
#include <strings.h>
long stncmp (
const char *s1,
const char *s2,
size_t n
)
```

Arguments

s1	Comparison source 1
s2	Comparison source 2
n	Comparison character number

Comments

strncmp compares as unsigned char all characters as far ass1 and s2 top n characters.

Return Value

The following values are returned depending on the result of the comparison.

Result	Return
s1<s2	<0
s1=s2	=0
s1>s2	>0

strncpy

Copies one character to another

Format

```
#include <strings.h>
char *strncpy(
char *dest,
const char *src,
size_t n
)
```

Arguments

dest	Copy destination character string
src	Copy source character string
n	Copy byte number

Comments

strncpy copies n bytes of src to the character string dest. It stops copying when the number of characters added reaches n.

Return Value

dest is returned.

strpbrk

Searches for position of first appearance of a specified character in a character set

Format

```
#include <strings.h>
char *strpbrk(
const char *s1,
const char *s2
)
```

Arguments

s1	Retrieved character string
s2	Character group

Comments

strpbrk checks the character string s1 and searches the position where any one character included in the character group s2 first appears.

Return Value

The address of the found character is returned. NULL is returned if it is not found.

Searches for position of last appearance of a specified character in a character string

Format

```
#include <strings.h>
char *strrchr (
const char *s,
long c
)
```

Arguments

s	Retrieved character string
c	Retrieved character

Comments

strrchr searches the position where the character c last appears in the character string s.

Return Value

The address of the appearance position of c is returned. NULL is returned if c does not appear.

strspn

Searches for first part of a character string comprising only characters in a specified character set

Format

```
#include <strings.h>
size_t strspn (
const char *s1,
const char *s2
)
```

Arguments

s1	Retrieved character string
s2	Character group

Comments

strspn returns the length of the first section that comprises only characters that are included in the character group s2 within the character string s1.

Return Value

The length of the found section of the character string is returned.

Searches for position of appearance of specified partial character string

Format

```
#include <strings.h>
char *strstr (
const char *s1,
const char *s2
)
```

Arguments

s1	Retrieved character string
s2	Retrieved character string

Comments

strstr checks the character string s1 and searches the position where the character string s2 first appears.

Return Value

The address of the position found is returned. NULL is returned if it is not found.

strtok

Searches for a character string bounded by characters in a specified character set

Format

```
#include <strings.h>
char *strtok (
char *s1,
const char *s2
)
```

Arguments

s1	Retrieved character string
s2	Bounded character group

Comments

strtok takes the character string s1 as a set of tokens bounded by one or more characters within the separate character string s2.

The first token top address of s1 is returned when strtok is first called, and directly after the token, the character NULL is written. After the s1 address is stored in the function, when NULL is entered in the first argument and strtok is called, a search is carried out until the token in the character string s1 disappears.

Return Value

The top address of the tokens found in s1 is returned. NULL is returned if nothing is found.

Converts character strings to integers

Format

```
#include <stdlib.h>
long str tol (
const char *s,
char **endp
)
```

Arguments

s	Character string
endp	Storage destination of pointer to non-convertible character string

Comments

strtol converts the character strings to long type (same as int type in R3000).

s must be in the following format.

[ws][sn][ddd]

[ws]	White space (can be omitted)
[sn]	Sign (can be omitted)
[ddd]	Number string (can be omitted)

strtol stops conversion when a character is encountered that cannot be converted and, unless endp is NULL, it sets the pointer to the character that stopped conversion to endp.

Return Value

The result of converting the input values to an integer is returned. 0 is returned when an error occurs.

See Also

strtoul()

strtoul

Converts character string into unsigned integer

Format

```
#include <stdlib.h>
unsigned long strtoul (
const char *s,
char **endp
)
```

Arguments

s	Character string
endp	Storage destination of pointer to non-convertible character string

Comments

strtoul converts the character strings to unsigned long type (same as unsignedint type in R3000).
s must be in the following format.

[ws][sn][ddd]

[ws]	White space (can be omitted)
[sn]	Sign (can be omitted)
[ddd]	Number string (can be omitted)

strtoul stops conversion when a character is encountered that cannot be converted and, unless endp is NULL, it sets the pointer to the character that stopped conversion toendp.

Return Value

The result of converting the input values to an integer is returned.

See Also

strtol()

isXXX

Carries out character testing

Format

```
#include <ctype.h>
long isXXX (
long c
)
```

Arguments

c	Character

Comments

isXXX carries out testing of characterc. They are all macros. The test conditions are as follows.

Function Name	Condition
isalnum	isalpha(c) ||isdigit(c)
isalpha	isupper(c) ||islower(c)
isascii	ASCII characters
iscntrl	Control characters
isdigit	10 base
isgraph	Printable characters except spaces
islower	Lower case characters
isprint	Printable characters including spaces
ispunct	Printable characters except spaces, English letters and numbers
isspace	Spaces, page breaks, line feeds, character returns, tabs
isupper	Upper case letters
isxdigit	Hexadecimal

Return Value

A value other than 0 is returned if the input value c satisfies the conditions, and 0 is returned if the conditions are not satisfied.

toascii

Masks 7th bit of an input value

Format

```
#include <ctype.h>
long toascii (
long c
)
```

Arguments

c	Input value

Comments

toascii is a macro for masking the 7th bit.

Return Value

The value masking the 7th bit of the input valuec is returned.

tolower

Converts characters to lower case characters

Format

```
#include <ctype.h>
long tolower (
long c
)
```

Arguments

c	Input value

Comments

tolower is a macro for converting the input valuec to a lower case character.

Return Value

The lower case character corresponding to the input valuec.

toupper

Converts characters to upper case characters

Format

```
#include <ctype.h>
long toupper (
long c
)
```

Arguments

c	Input value 1

Comments

toupper is a macro for converting the input valuec to an upper case character.

Return Value

The upper case character corresponding to the input valuec.

getc

Gets a single character from the stream

Format

```
#include <stdio.h>
char getc (
FILE *stream
)
```

Arguments

stream	Input stream

Comments

Gets a single character from input streamstream.

Return Value

NULL is returned in the case of file end or error.

See Also

getchar(), gets()

getchar

Gets a single character from the standard input stream

Format

```
#include <stdio.h>
char getchar( void )
```

Arguments

None

Comments

getchar gets a single character from the standard input stream. It is the same asgetc (stdin).

Return Value

Same as getc.

See Also

getc(), gets()

gets

Reads in a character string from the standard input stream

Format

```
#include <stdio.h>
char *gets (
char *s
)
```

Arguments

s	Input array storage destination

Comments

gets reads in the array that ends with a line feed character from the standard input stream (stdin) and stores it in s.

Return Value

The character string arguments is returned when successful. NULL is returned in the case of file end or error.

See Also

getc(), getchar()

putc

Outputs a single character to the stream

Format

```
#include <stdio.h>
void putc (
long c,
FILE *stream
)
```

Arguments

c	Output character
stream	Output stream

Comments

putc outputs the character c to the output stream stream.

Return Value

None

See Also

putchar(), puts()

putchar

Outputs a single character to standard output stream

Format

```
#include <stdio.h>
long putchar(
char c,
)
```

Arguments

c	Output character

Comments

putchar outputs a single character to the standard output stream. It is the same as putc (stdout).

Return Value

None

See Also

putc(), puts()

puts

Outputs a character string to the standard output stream

Format

```
#include <stdio.h>
void puts (
const char *s
)
```

Arguments

s	Output character string

Comments

puts outputs the character string closed by NULL to the standard output stream (stdout), and finally outputs the line feed character.

Return Value

None

See Also

putc(), putchar()

printf

Carries out formatted output to standard output stdout

Format

```
#include <stdio.h>
long printf(
const char *fmt[,argument ...]
)
```

Arguments

fmt	Input format character string

Comments

Please refer to C language reference books for a detailed explanation of input format.

Not compatible with conversion specifiers "f", "e", "E", "g" and "G".

printf2() of the mathematical function service is used in floating-point display.

Return Value

The length of the output character string is returned. NULL is returned when an error occurs.

See Also

sprintf(), printf2()

sprintf

Format output to array

Format

```
#include <stdio.h>
long sprintf(
char *s,
const char *fmt[,argument...]
)
```

Arguments

s	Storage destination of conversion character string
fmt	Input format character string

Comments

Please refer to C language reference books for a detailed explanation of input format.

Not compatible with conversion specifiers “f”, “e”, “E”, “g” and “G”.

sprintf2() of the mathematical function service is used in floating-point display.

Return Value

The length of the output character string is returned. NULL is returned when an error occurs.

See Also

printf(), sprintf2()

setjmp

Defines arrival point of non-local jump

Format

```
#include <setjmp.h>
int setjmp (
jmp_buf p
)
```

Arguments

p	Environment evacuation variable

Comments

Stores non-local jump arrival point information inp. When longjmp(p,val) is executed, it returns from setjmp().

Return Value

With direct calling 0 is returned.
When jump is carried out the value supplied to the second argument of longjmp() is returned.

See Also

longjmp()

Non-local jump

Format

```
#include <setjmp.h>
void longjmp (
jmp_buf p,
int val
)
```

Arguments

p	Environment evacuation variable
val	Return value of setjmp()

Comments

Jumps non-locally to arrival point specified byp.

Return Value

None. Not returned when executed normally.

See Also

setjmp()

4

Mathematical Functions

fabs

Absolute value (macro)

Format

```
fabs (
double x
)
```

Arguments

x	Floating-point value

Comments

fabs looks for the absolute value.

Return Value

The absolute value of x

Notes

This is a macro

atof

Converts character strings to floating-point numbers

Format

```
double atof(
const char *s
)
```

Arguments

s	Character string

Comments

atof converts character string to floating-point numbers (double type).

Return Value

The result of converting the input values to double type is returned. If the correct value exceeds the range that can be expressed, either +HUGE_VAL(1.797693134862316e+308) or -HUGE_VAL is returned according to the sign. 0 is returned if an underflow occurs.

Notes

Error processing is as follows.

Condition	Return Value	Error
Outside the range that can be expressed	+/- HUGE_VAL	Domain error
Underflow occurrence	0	Domain error

See Also

strtod()

Converts character strings to floating-point numbers

Format

```
double strtod(
const char *s,
char **endp
)
```

Arguments

s	Character string
endp	Storage destination of pointer to non-convertible character string

Comments

strtod converts the character strings to double type.
s must be in the following format.

[ws][sn][ddd]

[ws]	White space (can be omitted)
[sn]	Sign (can be omitted)
[ddd]	Number string (can be omitted)

strtod stops conversion when a character is encountered that cannot be converted and, unless endp is NULL, it sets the pointer to the character that stopped conversion toendp.

Return Value

The result of converting the input values to double type is returned. If the correct value exceeds the range that can be expressed, either +HUGE_VAL(1.797693134862316e+308) or -HUGE_VAL is returned, according to the sign. 0 is returned if an underflow occurs.

Notes

Error processing is as follows.

Condition	Return Value	Error
Outside the range that can be expressed	+/- HUGE_VAL	Domain error
Underflow occurrence	0	Domain error

pow

x to the power of y

Format

```
double pow (
double x,
double y
)
```

Arguments

x	Number value
y	Power

Comments

pow calculates x to the power of y.

Return Value

x to the power of y (x^y)

Notes

Error processing is as follows.

Condition	Return Value	Error
x==0 && y>0	0	
x==0 && y<=0	1	Domain error
x<0 && "y is not Integer value"	0	Domain error

See Also

exp()

Exponent

Format

```
double exp (
double x
)
```

Arguments

x	Floating-point value

Comments

exp looks for the exponent function ofx.

Return Value

e to the power ofx (e^x)

See Also

pow(), log()

log

Natural logarithm

Format

double log (
double x
)

Arguments

x	Logarithm calculated value

Comments

log looks for the logarithm function ofx.

Return Value

x logarithm (ln(x))

Notes

x is greater than 0. Range error in the case of others.

Condition	Return Value	Error
x<0	0	Domain error
x==0	1	Range error

See Also

exp(), log10()

Base 10 logarithm

Format

double log10 (

double x

)

Arguments

x	Logarithm calculated value

Comments

log looks for the base 10 logarithm function ofx.

Return Value

x base 10 logarithm (log10(x))

Notes

x is greater than 0. Range error in the case of others.

Condition	Return Value	Error
x<0	0	Domain error
x==0	1	Range error

See Also

log()

floor

Largest integer not greater than x (base function)

Structure

```
double floor (
double x
)
```

Arguments

x	Floating-point value

Comments

floor looks for the largest integer (double type) that is not greater thanx.

Return value

Largest integer (double type) that is not greater thanx

See Also

ceil()

ceil

Smallest integer not smaller than x (ceiling function)

Structure

```
double ceil (
double x
)
```

Arguments

x	Floating-point value

Comments

ceil looks for the smallest integer (double type) that is not smaller thanx.

Return value

Smallest integer (double type) that is not smaller than x

See Also

floor()

fmod

x/y floating-point number remainder

Structure

```
double fmod (
double x,
double y
)
```

Arguments

x	Floating-point value
y	Floating-point value

Comments

fmod looks for the remainder of the floating-point number resulting from x/y.

Return value

Floating-point number remainder of x/y

Notes

Return value sign is the same as x. 0 is returned if y is 0.

modf

Separation into integer parts and fractional parts

Structure

```
double modf (
double x,
double *y
)
```

Arguments

x	Floating-point value
y	Pointer to the buffer for storing integer part

Comments

modf separates x into integer parts and fractional parts.

The integer part is stored in y, and the fractional part becomes the return value.

Return value

Fractional part of x

Notes

The sign for both integer parts and fractional parts is the same asx.

sin

Sine

Structure

```
double sin (
double x
)
```

Arguments

x	Angle in radian units

Comments

sin looks for the sine function ofx.

Return value

sine function of x (sin(x))

See Also

cos(), tan(), asin()

Cosine

Structure

```
double cos (
double x
)
```

Arguments

x	Angle in radian units

Comments

cos looks for the cosine function ofx.

Return value

cosine function ofx (cos(x))

See Also

sin(), tan(), acos()

Tangent

Structure

```
double tan (
double x
)
```

Arguments

x	Angle in radian units

Comments

tan looks for the tangent function ofx.

Return value

tangent function ofx (tan(x))

See Also

sin(), cos(), atan()

Arcsine

Structure

```
double asin (
double x
)
```

Arguments

x	Arcsine calculation value. Range is [-1 to 1].

Comments

asin looks for the arcsine function ofx.

Return value

Arcsine function ofx. The range is [-pi/2, pi/2].

Error processing is as follows.

Condition	Return value	Error
fabs(x)>1	0	Domain error

Notes

[ ] shows the closed area.

See Also

sin(), acos(), atan()

Arccosine

Structure

```
double acos (
double x
)
```

Arguments

x	Arccosine calculation value. Range is [-1 to 1].

Comments

acos looks for the arccosine function ofx

Return value

Arccosine function ofx. The range is [0 to pi].

Error processing is as follows.

Condition	Return value	Error
fabs(x)>1	0	Domain error

Notes

[ ] shows the closed area.

See Also

cos(), asin(), atan()

Arctangent

Structure

```
double atan (
double x
)
```

Arguments

x	Arctangent calculation value

Comments

atan looks for the arctangent function ofx.

Return value

Arctangent function ofx. The range is [-pi/2 to pi/2]

Notes

[ ] shows the closed area.

See Also

tan(), asin(), acos(), atan2()

Arctangent

Structure

```
double atan2 (
double x,
double y
)
```

Arguments

x	Floating-point value
y	Floating-point value

Comments

atan2 looks for the arctangent function ofx/y.

Return value

Arctangent function ofx/y. The range is [-pi to pi].

Error processing is as follows.

Condition	Return value	Error
x==0 && y==0	0	Domain error

Notes

[ ] shows the closed area.

See Also

atan()

sinh

Hyperbolic sine

Structure

```
double sinh (
double x
)
```

Arguments

x	Angle in radian units

Comments

sinh looks for the hyperbolic sine function ofx.

Return value

Hyperbolic sine function ofx (sinh(x))

See Also

cosh(), tanh()

Hyperbolic cosine

Structure

double cosh (

double x

)

Arguments

x	Angle in radian units

Comments

cosh looks for the hyperbolic cosine function ofx.

Return value

hyperbolic cosine function ofx (cosh(x))

See Also

sinh(), tanh()

tanh

Hyperbolic tangent

Structure

double tanh (

double x

)

Arguments

x	Angle in radian units

Comments

tanh looks for the hyperbolic tangent function ofx.

Return value

Hyperbolic tangent function ofx (tanh(x))

See Also

sinh(), cosh()

sqrt

Square root

Structure

```
double sqrt (
double x
)
```

Arguments

x	Floating-point value that is not negative

Comments

sqrt looks for the square root ofx

Return value

Square root ofx

Error processing is as follows.

Condition	Return value	Error
x<0	0	Domain error

hypot

Complex number absolute value

Structure

```
double hypot (
double x,
double y
)
```

Arguments

x	Floating-point value
y	Floating-point value

Comments

hypot looks for the absolute value of the complex number (x+iy).

Return value

Square root of the sum of x^2 and y^2

ldexp

Calculates real number from mantissa and exponent ($x \times 2^n$)

Structure

```
double ldexp (
double x,
long n
)
```

Arguments

x	Floating-point value
n	Integer exponent

Comments

ldexp calculates the real number from the mantissa and exponent.

Return value

The value of $x \times 2^n$

frexp

Resolution into normalised fractional part and 2^n part

Structure

```
double frexp (
double x,
int *n
)
```

Arguments

x	Floating-point value
n	Pointer to the buffer that stores the 2^n part

Comments

frexp resolves x into fractional parts normalised to [1/2,1) and 2^n parts. The fractional part becomes the return value and the 2^n part is stored in n.

Return value

Normalised fractional part [1/2, 1)

Notes

[ ] shows the closing section and () the opening section.

printf2

Formatted output of standard output stdout (supports float and double type)

Structure

```
long printf2(
const char *fmt, [argument...]
)
```

Arguments

fmt	Output format character string

Comments

The conversion specifiers “f”, “e”, “E”, “g” and “G” can be used.

The stack consumption amount is greater than printf.

Return value

The length of the output character string is returned.

See Also

sprintf2()

sprintf2

Formatted output to array (supports float and double type)

Structure

```
long sprintf2(
char *s,
const char *fmt, [argument...]
)
```

Arguments

s	Storage destination of converted character string
fmt	Output format character string

Comments

The conversion specifiers “f”, “e”, “E”, “g” and “G” can be used.

The stack consumption amount is greater than printf.

Return value

The length of the output character string is returned.

See Also

printf2()

5

Other Functions

EXEC

Executable file data structure

Structure

```
struct EXEC {
                        unsigned long pc0;
                        unsigned long gp0;
                        unsigned long t_addr;
                        unsigned long t_size;
                        unsigned long d_addr;
                        unsigned long d_size;
                        unsigned long s_addr;
                        unsigned long s_size;
                        unsigned long sp, fp, gp, base;
};
```

Members

pc0	Execution start address
gp0	gp register initial value
t_addr	Data session top address with text session + initial value
t_size	Data session size with text session + initial value
d_addr	Reserved for the system
d_size	Reserved for the system
b_addr	Data session top address without initial value
b_size	Data session size without initial value
s_addr	Stack area top address (for user specification)
s_size	Stack area size (for user specification)
sp,fp,gp,base	Register evacuation area

Comments

EXEC is arranged in the top 2k bytes of the executable file (PS-X EXE structure). It holds information for loading and executing the program that is stored in the file.
It activates the program by adding stack information and delivering it to the Exec() function.

See Also

Exec()

DIRENTRY

Directory entry data structure

Structure

```
struct DIRENTRY {
                char name[20];
                long attr;
                long size;
                struct DIRENTRY *next
                long head;
                char system[8];
}
```

Members

name	Filename
attr	Attribute (depends on file system)
size	File size (byte units)
next	Next file entry (for user)
head	Head sector
system	Reserved for the system

Comments

DIRENTRY stores information relating to files that are registered in the file system.

See Also

firstfile(), nextfile()

CdlLOC

CD-ROM location

Structure

```
typedef struct {
                u_char minute;
                u_char second;
                u_char sector;
                u_char track;
} CdlLOC;
```

Members

minute	Minute
second	Second
sector	Sector
track	Track number

Comments

CD location specification structure.

Notes

track members are not currently used.

CdlFILE

ISO-9660 file descriptor

Structure

```
typedef struct {
                CdlLOC pos;
                u_long size;
                char name[16];
} CdlFILE;
```

Members

pos	File position
size	File size
name	Filename

Comments

CdlFILE gets the ISO-9660 CD-ROM file location and size.

GetRCnt

Getting root counter

Structure

```
long GetRCnt (
unsigned long spec
)
```

Arguments

spec	Root counter specification

Comments

GetRCnt returns the current value of the root counterspec.

Return value

The counter value that is expanded without the sign in 32bit is returned when successful, and -1 is returned in the event of failure.

See Also

StartRCnt(), ResetRCnt()

ResetRCnt

Resetting root counter

Structure

```
long ResetRCnt(
unsigned long spec
)
```

Arguments

spec	Root counter specification

Comments

ResetRCnt resets the root counterspec.

Return value

1 is returned when successful, and 0 in the event of failure.

See Also

GetRCnt(), StartRCnt()

StartRCnt

Root counter activation

Structure

```
long StartRCnt (
unsigned long spec
)
```

Arguments

spec	Root counter specification

Comments

StartRCnt activates the root counterspec.

Return value

1 is returned when successful, and 0 in the event of failure.

See Also

GetRCnt(), ResetRCnt()

Enter/ExitCriticalSection

Interruption inhibited/permitted

Structure

```
void EnterCriticalSection(void)
void ExitCriticalSection(void)
```

Arguments

None

Comments

EnterCriticalSection() inhibits interruption

ExitCriticalSection() permits interruption.

Return value

None

Opening file

Structure

```
int open (
char *devname,
int flag
)
```

Arguments

devname	Filename
flag	Open mode

Comments

open opens the file devnameand returns its descriptor.

Macros that can be specified in flag are as follows.

Macro	Open mode
O_RDONLY	Read only
O_WRONLY	Write only
O_RDWR	Read and write
O_CREAT	Create file
O_NOBUF	No buffer mode
O_NOWAIT	No synchronisation mode

Return value

The file descriptor is returned when successful, and -1 in the event of failure.

See Also

close()

Closing file

Structure

```
int close (
int fd
)
```

Arguments

fd	File descriptor

Comments

close releases the file descriptor.

Return value

fd is returned when successful, and -1 in all other cases.

See Also

open()

lseek

Moving file pointer

Structure

```
int lseek (
int fd,
unsigned int offset,
int flag
)
```

Arguments

fd	File descriptor
offset	Offset
flag	Refer to the comments

Comments

lseek moves the file pointer of the device showing the descriptor specified by fd.

offset is the movement byte number. The movement start point changes according to the value of flag.

It cannot be applied to character type drivers.

Macros that can be specified in flag are as follows.

Flag	Macro function
SEEK_SET	Top of file
SEEK_CUR	Current location

Return value

The current file pointer is returned when successful, and -1 in all other cases.

See Also

open(), read(), write()

Reads data from file

Structure

```
int read (
int fd,
char *buf,
int n
)
```

Arguments

fd	File descriptor
buf	Read buffer address
n	Read byte number

Comments

read reads n bytes from the descriptor specified byfd to the buf specified area.

Return value

The byte number read in the area at the time of normal termination is returned, and -1 in all other cases.

See Also

open()

write

Writes data to file

Structure

```
int write (
int fd,
char *buf,
int n
)
```

Arguments

fd	File descriptor
buf	Write data address
n	Write byte number

Comments

write writes n bytes from the descriptor specified byfd to the buf specified area.

Return value

The byte number written in the area at the time of normal termination is returned, and -1 in all other cases.

See Also

open()

firstfile

First file retrieval

Structure

```
struct DIRENTRY *firstfile (
char *name,
struct DIRENTRY *dir
)
```

Arguments

name	Filename
dir	Buffer that stores information relating to retrievable files

Comments

firstfile retrieves files corresponding to the filename pattern name, and stores information relating to them in dir.

Return value

dir is returned when successful, and 0 in all other cases.

Notes

(one optional character) * (entire character string of optional length) can be used as a wildcard character in the filename pattern. The character specification after * is disregarded.

See Also

DIRENTRY structure, nextfile()

nextfile

Next file retrieval

Structure

```
struct DIRENTRY *nexttfile (
struct DIRENTRY *dir
)
```

Arguments

dir	Buffer that stores information relating to retrievable files

Comments

nextfile continuously carries out retrieval in the same way as the firstfile() function executed directly before. When relevant files are found, information relating to them is stored in dir.

Return value

dir is returned when successful, and 0 in all other cases.

Notes

Execution will be unsuccessful if the CD-ROM drive shell cover is opened after firstfile(), and there will be a report that the file cannot be found.

See Also

DIRENTRY structure, firstfile()

Deletes files

Structure

```
int delete (
char *name
)
```

Arguments

name	Filename

Comments

delete deletes the file name.

Return value

1 is returned when successful, and 0 in all other cases.

Initialises file system

Structure

```
int format (
char *fs
)
```

Arguments

fs	File system name

Comments

format initialises the file system fs.

Return value

1 is returned when successful, and 0 in all other cases.

Notes

Valid only for file systems that can be written.

Renaming files

Structure

```
int rename (
char *src,
char *dest
)
```

Arguments

src	Source filename
dest	New filename

Comments

rename changes the filename from src to dest. It specifies the full path from the device name to both src and dest.

Return value

1 is returned when successful, and 0 in all other cases.

Notes

Valid only for file systems that can be written.

LoadTest

Load test execution

Structure

```
long LoadTest (
char *name,
struct EXEC *exec
)
```

Arguments

name	Filename
exec	Executable file information

Comments

LoadTest writes the information contained in the PS-EXE format file name to exec.

Return value

The execution start address is returned when successful, and 0 if unsuccessful.

See Also

EXEC structure, Load()

Load

Loading executable file

Structure

```
long Load (
char *name,
struct EXEC *exec
)
```

Arguments

name	Filename
exec	Executable file information

Comments

Load reads the PS-EXE format filename in the address specified by its internal header, and writes the internal information to exec.

Return value

1 is returned when successful, and 0 if unsuccessful.

See Also

EXEC structure, Exec()

Executing executable files

Structure

```
long Exec (
struct EXEC *exec,
long argc,
char *argv
)
```

Arguments

exec	Executable file information
argc	Argument number
argv	Argument

Comments

Exec executes the module loaded on the memory in accordance with the executable file information specified byexec.

Neither the stack nor the frame buffer are set ifexec->s_addr is 0.

The contents of the operation are as follows.

(1) Data session is zero cleared without an initial value.

(2) sp, fp and gp are initialised after evacuation (the value of fp is equal to that of sp)

(3) The argument of main() is set (by the a0 and a1 registers)

(4) The execution start address is called.

(5) sp, fp and gp are returned after return.

Return value

1 is returned when successful, and 0 in the event of failure.

Notes

Must be executed by critical section.

See Also

EXEC structure, Load()

InitHeap

Initialisation of heap area

Structure

```
void InitHeap (
void *head,
long size
)
```

Arguments

head	Heap head address
size	Heap size (multiples of 4 byte units)

Comments

InitHeap initialises the group of memory control functions. Thereafter, malloc(), etc. can be used. Not all the size bytes can be used because of the presence of overhead.

Return value

None

Notes

Do not carry out multiple execution.

See Also

malloc()

FlushCache

Flushing I cache

Structure

void FlushCache (void)

Arguments

None

Comments

FlushCache flushes the I cache.

It is executed when the program code is written in the memory.

Return value

None

Notes

Memory content cannot be changed.

_get_errno

Gets adjacent input/output error code

Structure

long _get_errno (void)

Arguments

None

Comments

_get_errno gets adjacent error code through all file descriptors.

The error code is defined in sys/errno.h.

Return value

Error code

GetPadBuf

Gets controller buffers

Structure

```
void GetPadBuf (
volatile unsigned char **buf1,
volatile unsigned char **buf2
)
```

Arguments

buf1	Pointer to the buffer that stores data from the port 1 controller.
buf2	Pointer to the buffer that stores data from the port 2 controller.

Comments

Communication with the controller is carried out every vertical synchronisation interruption, and the result stored in controller buffers within the system. The GetPadBuf function can get the pointers to those buffers.
Two sets of controller buffers are available for the ports, and the following data is stored.

Bytes	Content
0	0xff: Without controller 0x00: With controller
1	Upper 4bit: Terminal type Lower 4bit: Received data size (1/2 byte number)
2~	Reception data (largest 32 bytes)

The received data is different according to the controller type shown by ‘terminal type’. The terminal types supported by this library are as follows.

Terminal Classification	Device Name
0x1	Mouse
0x2	NeGCon
0x4	Standard controller
0x5	Joystick

Please refer to the "Programmer's Guide" for the contents of received data corresponding to terminal type.

Return value

None

CdPlay

Plays back CD-DA tracks

Structure

```
int CdPlay (
int mode,
int *tracks,
int offset
)
```

Arguments

mode	Mode
tracks	Array that specifies track to be played. Ends with 0.
offset	index of tracks starting the performance

Comments

CdPlay plays consecutively in the background multiple tracks specified by the array tracks. When the last track of the array is played, it repeats or ends the performance, according to the mode.

Values that can be specified in mode are as follows.

Value	Description
0	Stops performance
1	The tracks specified by tracks are played consecutively, and the performance is stopped when all the specified tracks have been played.
2	The tracks specified by tracks are played consecutively, and the performance is returned to the start and repeated when all the specified tracks have been played.
3	The index of the tracks array for the track currently being played is returned.

Return value

The track currently being played. The index of thetracks array is returned instead of the track number. The performance is shown as ended if -1 is returned.

Notes

The performance is carried out in track units, Performance and stopping etc. in mid track is not possible.

CdReadFile

Reads files on CD-ROM

Structure

```
int CdReadFile(
char *file,
u_long *addr,
int nbyte
)
```

Arguments

file	Filename
addr	Read memory address
nbyte	Read size

Comments

CdReadFile reads nbyte of a file on CD-ROM.

The entire file is read if 0 is specified in nbyte.

If NULL is specified infile, reading starts from the last location read by CdReadFile immediately before.

Return value

The data number (bytes) read is returned if successful, and 0 is returned in the case of a reading error.

Notes

The filename must be an absolute path.

Lower case characters are automatically changed to upper case characters.

Reading is carried out in the background, and CdReadSync() is used to determine the end of reading.

CdReadExec

Loading executable files from CD-ROM

Structure

```
struct EXEC *CdReadExec(
char *file
)
```

Arguments

file	Executable filename

Comments

Executable files specified by file are loaded by CdReadExec from CD-ROM to the appropriate address in the main memory.

Reading is carried out in the background, and CdReadSync() is used to determine the end of reading.

The loaded file is executed as a child process by using Exec().

Return value

EXEC structure that holds executable files that have been read.

Notes

The load address of the executable file should not overlap the area used by the parent process

CdReadSync

Waits for termination of CdRead

Structure

```
int CdReadSync (
int mode,
u_char *result
)
```

Arguments

mode	0: Waits for termination of read
	1: Current condition is checked and immediately returned
result	Status of most recently terminated command

Comments

CdReadSync waits for reading by CdReadFile() and CdReadExec() to terminate.

Return value

The following values are returned.

Return value	Content
Standard integer	Remaining sector number
0	Termination
-1	Read error

CdSearchFile

Gets location and size from filename on CD-ROM

Structure

```
CdlFILE *CdSearchFile (
CdlFILE *fp,
char *name
)
```

Arguments

fp	CD-ROM file structure pointer
name	Filename

Comments

CdSearchFile recognises the absolute location (minute, second, sector) and size from the filename on CD-ROM.

The result is stored in fp.

Return value

The pointer of the CD-ROM file structure obtained is returned.

0 is returned if the file is not found, and -1 is returned if the search fails.

Notes

The filename must be an absolute path.

File location information in the same directory as files specified by fp are cached in memory. For this reason, if CdSearchFile() is carried out continuously in files within the same directory, access becomes faster from the second time.

Cases where the return value is -1 show that the directory read has failed for some reason.

GetVideoMode()

Obtains the present video signalling system

Structure

long GetVideoMode (void)

Arguments

None

Comments

Returns the present video signaling system declared in SetVideoMode().

Return value

Return value contents is the video signaling system mode

MODE_NTSC:	NTSC system video signaling system
MODE_PAL:	PAL system video signaling system

Notes

When SetVideoMode () is not called, no matter what the machine, it will return MODE_NTSC.

See Also

SetVideoMode()

SetVideoMode()

Declares current video signalling system

Structure

```
long SetVideoMode (
long mode
)
```

Arguments

mode	Video signaling system mode

Comments

Declares the video signaling system indicated by *mode* to the libraries.

Related libraries will be able to conform to the actions of the declared video signaling system environment.

Return value

Previously-set video signaling system mode

Mode Contents

MODE_NTSC:	NTSC system video signaling system
MODE_PAL:	PAL system video signaling system

Notes

Gets called in advance of all library functions.

See Also

GetVideoMode()

TestCard

Memory card test

Structure

```
long TestCard (
long chan
)
```

Arguments

chan	Slot numbers 0: Slot 1 1: Slot 2

Comments

TestCard tests the memory card set in the slot specified bychan and returns the result.

Card initialisation is carried out on the memory card control screen of the PlayStation.

One to four vertical synchronisation interruptions at the end of the operation are necessary (17m to 68m seconds).

Return value

0:	No card
1:	Card present
2:	New card detected
3:	Communication or card abnormality detected
4:	Non-initialised card detected

6

Index

All structures, functions and external variables explained in this Library Reference are listed in alphabetical order together with the relavent page number.

Library Reference
Software Development Tool

Published February 1997
©1997 Sony Computer Entertainment Inc. All Rights Reserved.

Written and produced by :
Sony Computer Entertainment Inc.
Akasaka Oji Building
8-1-22 Akasaka, Minato-ku, Tokyo, Japan 107
Enquiries to: Network Business Project
E-mail:ny-info@scei.co.jp
TEL:+81 (0) 3-3475-1711

Sony Computer Entertainment Europe
Waverley House
7-12 Noel Street
London W1V 4HH, England
Inquiries to: TheYaroze Team
E-mail: yaroze-info@scee.sony.co.uk
TEL:+44 (0) 171 447 1616 / +44 (0) 7000 YAROZE

Sony Computer Entertainment America
919 E. Hillsdale Blvd., 2nd Floor
Foster City, CA 94404, USA
Inquiries to: TheYaroze Team
E-mail: yaroze@interactive.sony.com
TEL:+1-415-655-3600