PlayStation™ Operating System

Developer

Reference

Series

September 1995

Foreword

This document explains the mechanics of the target box operating system, and the concepts it implements.

OS Handbook

- This Handbook is distributed to Sony PlayStation licensed developers. Information in this Handbook is subject to change without notice. Unauthorised reproduction, distribution or disclosure to third parties of the whole or any part of this Handbook, in any form or by any means, for any purpose, is expressly prohibited by the Licensee Developer Agreement.

- The terminology used in this Handbook is of specific application to the R3000. Please note that, when applied to CPUs the same terminology may have a different meaning.

- Where applicable the company and product names used in this Handbook are registered trademark of their respective owners.

Published August 1994

Sony Computer Entertainment
13 Great Marlborough Street
London
W1V 2LP
Tel No: +44 (0) 71 911 8900 (switchboard)

Contents

<PS-X OS> Overview

A flexible and powerful operating system is part of the PS-X. By using this operating system, you may maximize the capabilities of the PS-X.

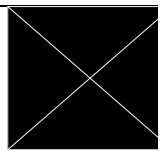
What is <PS-X OS>

<PS-X OS> was developed for the R3000, the CPU of PS-X. The efficiency of program development is very dependent on the environment and services that the operating system in a machine provides. If it has a fast enough CPU and peripheral devices, there is no need to take time to figure out a method of maximizing the capabilities of the hardware and one may concentrate on programming by skillfully using the services provided by the OS.

The design concept of the <PS-X OS> lies in providing an environment for the developer of game programs and for enabling you to control programs with interrupts more easily. Based on this concept the kernel of the <PS-X OS> is provided as the aggregate of the services (subroutines) controlling the R3000 and the PS-X hardware.

In addition each service is provided as a C language function. By using C, not only may we aim for better source code readability and maintainability, it also makes block structure description and function calls easier and makes it possible to program with greater ease.

Characteristics of <PS-X OS>



There are many characteristics of <PS-X OS> which implement its concept of computing.

Programming with C

For control of the R3000 CPU, control of the PS-X hardware units etc., all services are provided as C language functions. Thus, programming may be performed throughout in C.

Easy access to the features of the R3000

Though the interrupt processing procedures in the R3000 are said to be complex, the operating system under the <PS-X OS> makes use of a substitute “dispatcher” system and provides a simple interface. Of course the overhead that accompanies the dispatcher is kept very low. In addition, in ordinary operating systems the low level implementation details are not disclosed. In this way, operating system chip capabilities are maximized and a higher level of tuning may be achieved. Of course because everything can be done in C, there is no need to learn the intricacies of R3000 assembler.

Light volume, small scale, emphasis on performance

If the operating system is slow in speed it could ruin a speedy game program. In order to place emphasis on performance of applications, the size in RAM of <PS-X OS> has been kept to a maximum of 64 Kilobytes, and it was also designed so that the CPU occupied time is minimized. In addition the OS system tables have been disclosed and consideration has been given to expansion of the operating system and acquisition of internal data.

Furthermore, in the <PS-X OS> operating system, checks of prohibited items seen in other operating systems have been eliminated in order to aim for greater speed. These checks should be performed by the application. Thus, though there is the danger that essentially prohibited operations will end up being performed, cautious programming makes possible higher level tuning .

Until now in hardware control of consoles, one had to analyze hardware driver code and painstakingly work through everything with an assembler. In order to lighten this burden all hardware features are provided as C language functions by the <PS-X OS>. The overhead of each function is kept to a minimum.

Single task or multi-tasking

The <PS-X OS> is basically a multi-tasking operating system that can perform multiple processing asynchronously. Multi-task control is appropriate for controlling a CD-ROM, a comparatively slow device, or things like playing background music.

In addition, we have made it possible to select multi-tasking mode as an option for those who are accustomed to traditional programming, in other words, those who are used to single task. Immediately after starting the OS it is single task mode, and it is possible to select operating mode by making a selection at this point.

Characteristics of <PS-X OS> (cont.)

Device driver based file system

The file system (i.e. files of data on CD-ROM) in <PS-X OS> is accessed via a device driver. This allows multiple file systems to co-exist and increases development time by avoiding low level file manipulation problems.

<PS-X OS> in Reality

What sort of structure does <PS-X OS> actually have? In this chapter we shall explain how to start the OS, the memory map, and other things that one must know when using this operating system.

Starting and operating the OS

<PS-X OS> is an operating system that provides an “environment for game program developers.” Thus, it is basically not outfitted with an interface for the user to operate hardware directly (except for the debug monitor in the debugging environment). It is necessary for the application to provide the user interface.

Starting the Operating System

There are two start modes with the <PS-X OS> operating system. The mode is determined by the dip switches of the target box and is changed when the target box is turned on or reset. Please refer to the “Hardware Guide” for setting the dip switches.

Hardware mode

This is the mode which corresponds to booting the actual PS-X hardware.

Application boot

First jumps to a specific address in ROM and performs a check of the hardware connected (CD-ROM drive etc.).

Next, makes a decision on type of disk in the CD-ROM drive. If the disk is the appropriate one, it retrieves a system designation file “system.cnf” and executes this.

If there is no disk it goes into a looping demonstration.

Booting in a PC/AT compatible development environment

When the dipswitch is set so that the system may be booted from a developer host (PC/AT compatible machine), the kernel is set in the default configuration (please refer to “library reference”) and then the remote debugger is started. The reading and execution of the execute file which corresponds to the final stage of the boot sequence is performed manually. The kernel configuration cannot be altered.

System designation file SYSTEM.CNF

Here the PS-X system settings, and the applications that will be started are described.

The parameters are described from the beginning of a line in “<key word> = <content>” format. Description is all done in 1 byte upper case alphanumeric characters and a space is necessary on both sides of the equal sign. Lower case characters may not be used. When the same parameter reoccurs in a file the first version found takes precedence. The following are the parameters that may be described.

Table 1 System Designation File Keyword List

Keyword	Content
BOOT	Specify the file name to be started. The default is GAME.PSX. Example: BOOT=PSXAPP.EXE
TCB	Specify the number of tasks. The default is 4. Example: TCB=5
EVENT	Specify the number of events. The default is 16. Example: EVENT=5
STACK	The stack pointer value when the file specified by BOOT is started. Default is 0x8001FF00. Example: STACK=800FFF0

Debug mode

This is a mode which is valid in the debug environment, in other words only with the target box. There are the following three applications in debug mode.

Debug monitor

The command interpreter which takes an external RS-232C channel as a standard input/output device and starts. Debugging is possible, using printf( ), whose output is piped through the RS-232C link.

SCSI monitor

SCSI channel 0 is a CPU device, and may be used for execution of remote commands from a host, debugging at C source level, as a DOS device driver and for file transfers.

File system

In the PS-X, CD-ROM is used as the application supply medium. That is to say, the file system of the <PS-X OS> takes as a premise that files are on CD-ROM.

The file system of PS-X is based on ISO 9660 level 1 which is an international standard.

File name conventions

File names take the format of "<BASENAME><EXT>." The <BASENAME> is eight 1 byte alphanumeric characters and the <EXT> is 3 alphanumeric characters. 1 byte katakana and 2 byte Chinese characters (kanji) cannot be used. <BASENAME> and <EXT> are separated by a period. This is the same name format as used in MS-DOS.

Subdirectories

Subdirectories cannot be created. Files are all arranged in the root directory.

Number of Files

There is no limit on the number of files that may be placed in the root directory.

Memory management

Using memory space on the R3000 is important when using <PS-X OS>. Here we shall explain the memory management of the <PS-X OS>, focusing on the memory map.

Memory Map

We will explain the memory map of <PS-X OS>.

To begin with, we call the space that a program uses “logic space.” Logic space is expressed by a 32 bit address, and is mapped on physical space in multi layer. In C language it is accessed with a long pointer.

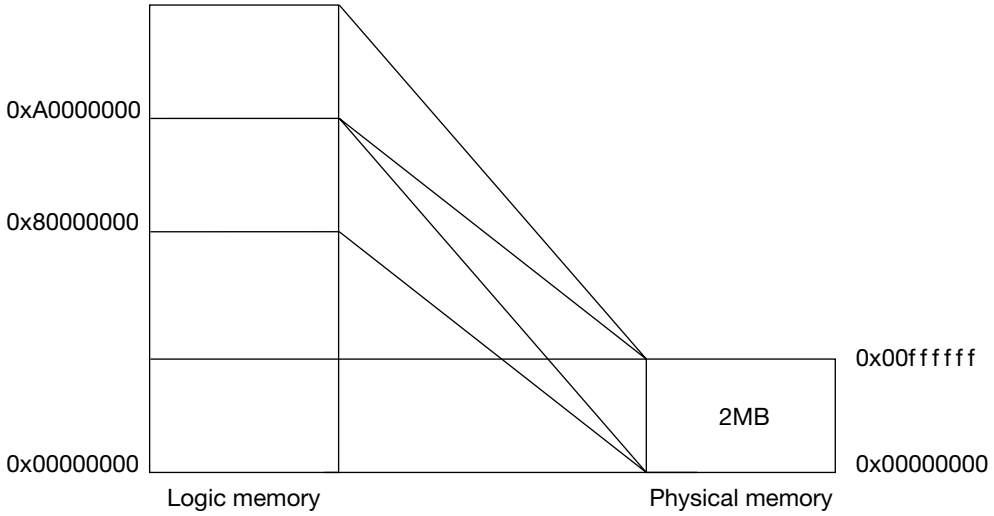


Fig. 2-1 Logic memory/physical memory allocation

“Physical space” is an address in the installed memory device (i.e. an address on the actual memory chip). If a logic space address is accessed that is not mapped in physical space, a “bus error” will be generated.

Interface registers with peripheral devices are also mapped in logic space (i.e. memory mapped I/O). Memory devices may be accessed with 1 to 4 byte units but accessing the interface register is done by a method that uses specific size units. When accessed by units other than this a bus error will be generated.

Memory management (cont.)

Table 2 Memory Map

Logic space (32 bit)	Segment name	Cache
0x00000000~max0x00ffffff)	A	Valid
0x1f800000~0x1f8003ff	S	Invalid
0x1f801000~0x1fbfffff	X	Invalid
0x1fc00000~0x1fc7ffff	P	Valid
0x80000000~0x80000000+max	B	Valid
0x9fc00000~0x9fc7ffff	Q	Valid
0xa0000000~0xa0000000+max	C	Invalid
0xbfc00000~0xbfc7ffff	R	Invalid

Physical space	Corresonding logic segment	Device
0x000000~max0x00ffffff)	A・B・C	RAM
0x1f800000~0x1f8003ff	S	RAM in CPU
0x1f801000~0x1fbfffff	X	Device
0x1fc00000~0x1fc7ffff	P・Q・R	ROM

I Cache

The validity/ invalidity of the Instruction Cache (hereafter I cache) is determined by the 4 upper bits of the logic space. With the segments which correspond to the memory devices mentioned earlier I cache is valid for A, B, P and Q, but is invalid with C and R. Segments, in the R3000 refer to block units separating memory space from features and differ in this respect from other CPU's (8086 type) so please be careful.

When I cache is valid, machine code is read into I cache in a block. If the desired instruction is in I cache, main memory does not need to be accessed via the bus. Speed of execution of applications is improved.

The I cache is packaged with 1K words (4 kilobytes), and one way mapped. In other words, logic space is divided into 4 kilobyte units, and these units are mapped onto the I cache.

D cache

The D cache is configured by the high speed memory loaded in the CPU. Internally it is structured as a scratch pad and is mapped in the memory space as S segment so that "game program developers" can access it.

Both data and programs may be placed in this segment. However it cannot be the target of a DMA transfer.

System tables (ToT)

In order to uniformly access each type of system table used internally in the operating system, system table information is disclosed as the structure ToT (Table of Tables). ToT is placed at address 0x00000100.

The contents of ToT are as follows.

Table 3 System tables

Entries	Nature
0	Queue header requires interrupt processing
1	Task state queue header
2	Task management block
3	System reserved
4	Event management block
5~31	System reserved

Definition of ToT data structure

The ToT data structure is described by the following C structure.

The structure is defined in the header file include/kernel.h.

```
struct ToT {                    /*system table table*/
        unsigned long *head;    /*system table lead address*/
        long size;              /*system table size (byte)*/
};
```

R3000 Registers

Here we shall explain the R3000 registers. For higher level programming knowledge of the R3000 registers is necessary but in normal C programming there is no need to be especially conscious of it.

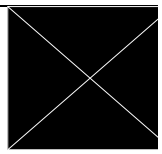
The R3000 registers may be divided into “General purpose registers” and a “Program counter.”

General Purpose registers

There are 32, 32 bit general purpose registers. Each register is assigned to the following specific uses by the compiler. When using a threaded data base and developing with an assembler, it is necessary to use the registers in the following way.

Table 4 R3000 General Purpose Registers

Macro (1)	Macro (2)	Assignment
R_ZERO	R_R0	0 fixed
R_AT	R_R1	Assembler reserves
R_V0	R_R2	Return value
R_V1	R_R3	Return value (for double type)
R_A0	R_R4	Argument #1
R_A1	R_R5	Argument #2
R_A2	R_R6	Argument #3
R_A3	R_R7	Argument #4
R_T0	R_R8	Temporaries
R_T1	R_R9	" "
R_T2	R_R10	" "
R_T3	R_R11	" "
R_T4	R_R12	" "
R_T5	R_R13	" "
R_T6	R_R14	" "
R_T7	R_R15	Saved temporaries
R_S0	R_R16	" "
R_S1	R_R17	" "
R_S2	R_R18	" "
R_S3	R_R19	" "
R_S4	R_R20	" "
R_S5	R_R21	" "
R_S6	R_R22	" "
R_S7	R_R23	" "
R_T8	R_R24	Temporaries
R_T9	R_R25	" "
R_K0	R_R26	Kernel reserve #0
R_K1	R_R27	Kernel reserve#2
R_GP	R_R28	Global Pointer
R_SP	R_R29	Stack Pointer
R_FP	R_R30	Frame pointer
R_RA	R_R31	Return previous address



Precautions to be taken when using the general purpose registers are as follows:

AT register

The assembler uses the AT register as a work area. The C compiler and assembler program are prohibited from using this register.

Return address

The R3000 does not have a subroutine call concept. Therefore, this is substituted by a jump command saving the return address in a register. The assembler may specify the register where it is saved but the compiler is restricted to the RA register.

C Language function arguments

When there are less than 4 arguments they are stored in A0 to A3 registers in order from the left. When there are more than 4 arguments they are accumulated on the stack. Space for 4 arguments is allocated on the stack, but is dummy. The remaining arguments are passed on the stack; the first 4 are in registers as usual.

C Language Function Return Value

The return value of a function is stored in the V0 register when less than 32 bits. When 64 bits (double) the 32 high order bits are stored in the V1 register.

Stack

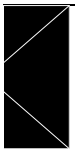
There is no stack concept in the R3000. Therefore, the compiler stores the pointer in the SP register and implements a stack. In order to efficiently use a function frame (a memory area for automatic variables and a work area) it stores “the lead address of the frame for that function” in the FP register. This value is determined initially by SP. When the module is started, FP is set to SP.

Global pointer

The R3000 memory is accessed by “encoded 16 bit offset register indirect mode.” In order to effectively accomplish this, the compiler consolidates the variables up to 64 Kilobytes in a “bss section” block. Here the block’s start address is stored in the GP register and using the above mode access is performed by one word. This address value is called a “global pointer” and it does not vary in the module.

Program counter

The program counter cannot be accessed directly. The actions of the program counter are as follows.



R3000 Registers (cont.)

Immediately after power-on reset

The program counter will be 0xbfc00000.

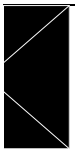
When there is an external interrupt and when an error (except a debug error) occurs

1) The content of the program counter will be saved to the EPC register of coprocessor 0 (the section which handles errors and interrupts in the R3000).

2) The processor jumps to 0x00000080.

<PS-X OS> Service

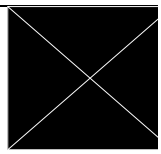
The most significant characteristic of <PS-X OS> is that all features controlling the R3000 and PS-X hardware are provided as C functions. Because of this, the burden on the program developer is lightened, he or she may proceed with development more efficiently.



<PS-X OS> Service

The services provided by <PS-X OS> are called application programmer interfaces (API). The programs which call the OS must have their function libraries linked. All of the PS-X features are covered by the API. Program readability will increase if only API calls are used.

Application interfaces (API)



A general service for controlling PS-X hardware including the CPU. All of the PS-X features are covered by the API.

The API has the following services. Please refer to “library reference” concerning functions provided by the API, and details about data structures.

Module management service

Module management is a basic service which loads and executes user application programs.

Event management service

Event management is a service controlling program execution triggered by asynchronously occurring events.

Controller service

This is a service which deals with the controller which is the PS-X ‘s chief input device. It supports up to 64 controllers which are configured with a maximum of 16 buttons.

I/O management service

This is a service which supports low level input/output to files and logic devices. All file control in PS-X is performed using this service.

Thread management service

“Threads” are aggregates of resources necessary for executing programs. Specifically they are composed of register states, stack areas and CPU states. Multiple threads, multi-tasking and multi-interrupts can be implemented using thread management.

Other services

There are various other services, for example, setting up the I cache and the R3000 etc., and authorizing or prohibiting processing of interrupts.

Standard C library

Provides standard C functions based on K & R such as character string operations, standard input and outputs.

Root counter management service

Provides a counting feature which is used by game programmers for time restrictions and timing adjustments.

A “Root counter” causes automatic count timing. There are 4 elements to this.

(a) Display pixels
(b) Horizontal synchronization
(c) System clock
(d) Vertical synchronization

In all of these counters except (d) a 16 bit target value can be set.

Counters count from zero and when they reach the target value generate an interrupt, then automatically clear to zero and begin counting again. The target value of (d) is fixed at 1.

PS-X library

This is a group of C functions which are for controlling expression features like graphics and sound. It provides the following library files. Please refer to “library reference” for details.

Table 5 Types of library Files

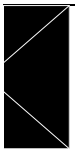
Name of library	Feature
libgpu	CPU basic operation
libgte	GTE basic operation
libgs	Expanded graphics system
libpress	Data compression
libstr	Streaming
libsnd	Sound
libapi	Kernel service
libetc	Other

File Format

This defines the internal structure of the files that the PS-X library processes. PS-X special tools also generate and process files in this format. Details are explained in Chapter 5.

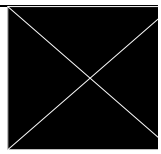
Programming Tips

In this chapter we will cover precautions to be taken and tips when programming with <PS-X OS>.



Use in single task mode

<PS-X OS> is basically a multi-tasking operating system which starts out in single task mode. If the “game program developer” does not start multiple threads it will operate in single task mode. One does not necessarily have to use the multi-tasking feature.



Prohibition of interrupts

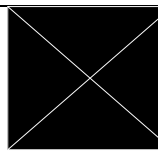
By calling the EnterCritical Section( ) function, interrupts can be stopped. Authorization is performed by EnterCritical Section( ).

Overlay processing

Overlay processing in the PS-X program takes the form of executing a child process from a resident module.

The operating system drives the hardware and starts the file to be executed, but in overlay processing it is necessary for the application to do this.

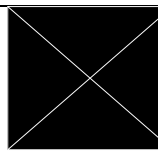
Character string display



There is no character generator and fonts are not provided in PS-X. To display character strings the application must provide data and must perform this as part of a graphics operation.

Storage of game data

A special PS-X back up cartridge is provided as a game data storage device. Since special functions are also provided for the cartridge, this may be used during programming.



Acceleration of execution speed

Here we shall provided a number of tips for increasing the speed of execution of applications.

File partition and placement

By partitioning files into appropriate sizes and placing them carefully on the CD-ROM, the speed of starting and reading of data can be increased.

Using the I cache

Of the segments that correspond to the memory devices described in Chapter 2, I cache is valid with A, B, P and Q and is invalid with C and R. Since the size of I cache is 4 kilobytes, it is very important where and how code is placed. Careful placement of code can result in increased speed.

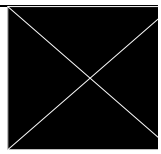
Using the S segment

Using the S segment that is part of the CPU's high speed memory is another factor in increasing speed.

Heap memory

The C language heap memory management function group is provided as one part of the C library but <PS-X OS> does not do the initialization necessary for its use. That is because the “game program developer” must manage all memory available.

When using the heap memory management functions, first execute the heap memory initialization functions belonging to “other services” and then specify the start address and size of the heap memory.



Program entry point

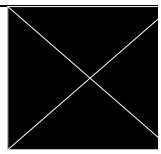
The first program started in running PS-X is prepared as an execute file. The file name may be specified following the boot sequence (Please refer to the “Hardware Guide”). The first address to be executed of the execute file, in other words, the entry point, may be specified by a function name at link time. When not specified, main( ) will be selected (C language default). The function that will be the entry point cannot be given an argument.

Vertical blanking period detection

If the root counter and the event management service are used, the beginning of the vertical blanking period can be detected. The beginning of the vertical blank acts as a trigger so that polling, calling of queue functions and starting of handler functions can be performed. Call queuing functions in the following way:

```
Sample()
{
        unsigned long EV;
        EnterCriticalSection();
        EV = OpenEvent (RCntCNT3, EvSpINT, EvMdNOINTR, NULL);
        EnableEvent (EV);
        SetRCnt (RCntCNT3, 1, RCntMdINTR);
        StartRCnt (RCntCNT3);
        ........
        /* Initialization of controller */
        ........
        while (1){
                WaitEvent (EV);
                ........
        }
}
```

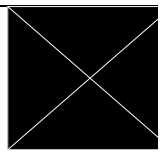
Standard input/output



Immediately after starting, standard output is allocated to file descriptor 0 and standard input to file descriptor 1. The input output device is RS232C channel 0 (device name “tty00”). Input output functions such as printf( ) and getchar( ) may be used. The circuit designation of this port is 8 bit characters, 1 stop bit, no parity, hardware flow control. Control with PC/AT can be performed with the special cable packaged together with the target box. A 25 pin type host may be connected by cable.

CD-ROM simulator

The “CD-ROM simulator” feature which uses the developer host file system is provided as one aspect of the development environment. Please refer to “Psy-Q Development Environment.”



Settings necessary when activating a program

In addition to designating a starting address, in other words the program counter initial value, a stack pointer (SP) value should be set. When executing a program and a value corresponding to an address that cannot be accessed is in the stack pointer a “bus error” will be generated.

File Formats

Here we explain graphics data file formats that are processed by the PS-X library. Data is created with a special PS-X tool.

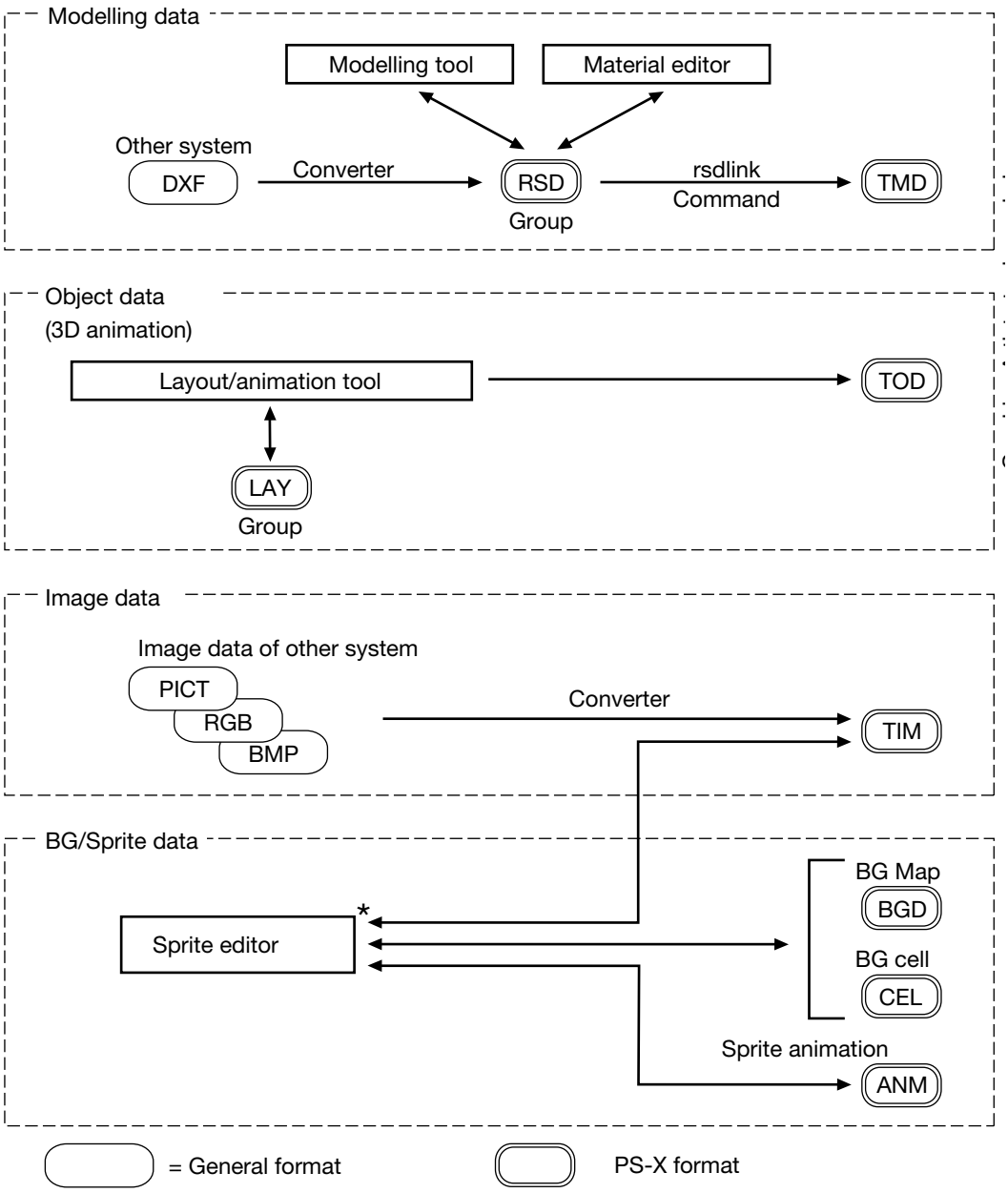
Types of Graphics data

There are 2 types of PS-X graphics, 3 dimensional graphics which characterize this system and the 2 dimensional graphics.

In 3 dimensional data, there are “modeling data (TMD)” which express the shape and surface attributes of an object and “animation data (TOD)” which includes information on object placement.

In 2 dimensional data, in addition to “image data (TIM),” which are sprite patterns and texture materials, there are “BG map data (BGD)” for BG screen map data and sprite animation, “cell data (CEL),” and “information data (ANM).”

We shall explain each type of data in order below.



PS-X file format relations

Animation data (TOD format)

TOD format is for describing a 3 dimensional object through time and it corresponds to functions in the expanded graphics library “libgs.”

Specifically, it describes for each frame the data for a three dimensional object which appears, changes or disappears in a 3 dimensional animation and arranges data for each frame following the flow of time.

File Format

TOD files consist of file header and frame data as follows:

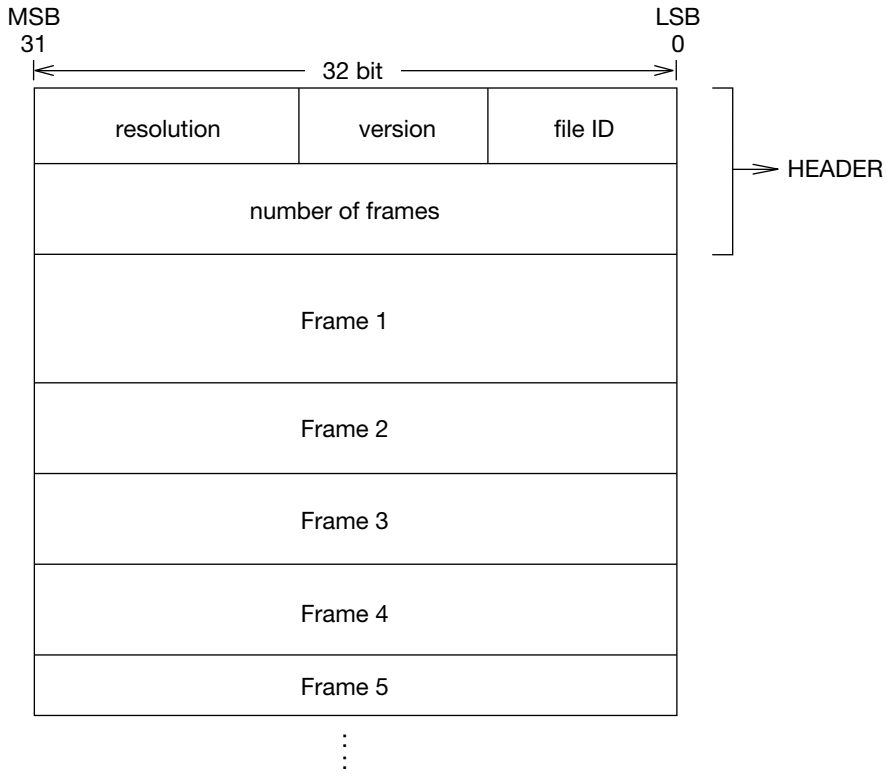


Fig. 5-2-1 TOD file format

Header

At the beginning of TOD files there is a 2 word (64 bit) header. The following four types of information are described in headers.

(a) *file ID (8 bits)*
This indicates that it is an animation file. Value is 0x50.

(b) *version (8 bits)*
This indicates the version of the animation. Value begins from 0x00.

(c) *resolution (16 bits)*
This is the time that one frame is displayed (unit tick = 1/60 second).

Animation data (TOD format) (cont.)

(d) *number of frames (32 bits)*
The number of frames described in the file.

Frame

Following the header the frame is described. Frames are arranged chronologically (See Figure 5-2-2).

Frame Section

Each frame is composed of a frame header and a succeeding packet as follows.

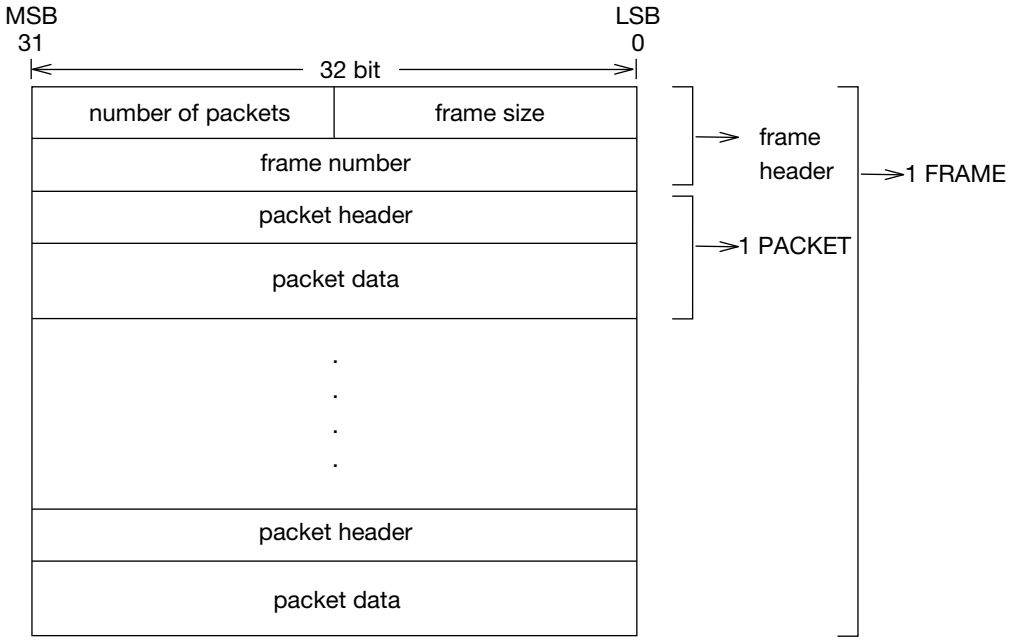


Fig. 5-2-2 Vertex structure

Frame header

There is a 2 word frame header at the beginning of each frame. The following information is described in a frame header.

(a) *frame size (16 bits)*
Frame length (including header) in words.

(b) *number of packets (16 bits)*
Number of packets.

(c) *frame numbers (32 bits)*
Frame number.

PACKET

After a frame header there is a packet. Each packet consists of a 1 word packet header at the beginning followed by packet data. (See Figure 5-3) There are various things in a packet.

The size of packet data in each packet will not be the same whether the packets are of the same type or not.

Packet section

Packets are composed of packet headers and packet data as follows:

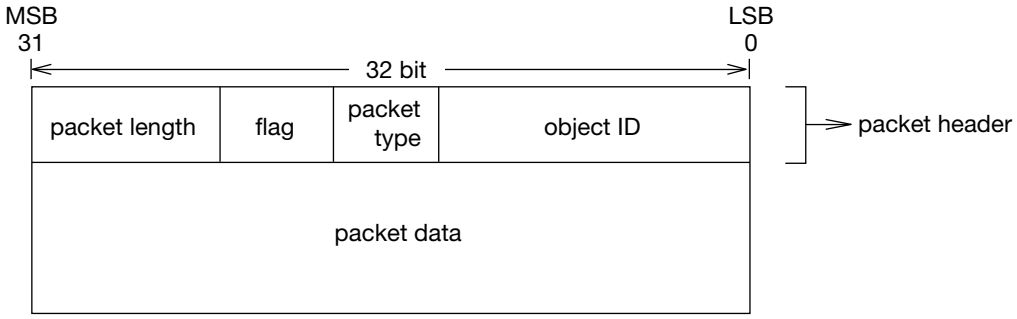


Fig. 5-3 Packet format

Packet header

The following information is described in a packet header.

(a) *Object ID (16 bits)*
The identification of the object concerned.

(b) *Packet type (4 bits)*
Type of packet data.

(c) *Flag (4 bits)*
Meaning differs with packet.

(d) *Packet length (8 bits)*
Packet length (including header) word (4 bytes) unit size.

An “object” is a 3 dimensional object (GsDOBJ) handled by libgs (expanded graphics library) and indicates the object that will reflect the packet data.

In packet type the type of data stored in packet data is stored. The meaning of flag will differ according to this packet type.

Packet length expresses packet length in word (4 bytes) units.

Packet data

Various types of data such as the GS COORDINATE structure, RST value and TMD data identification (modeling ID) etc. are stored in packet data.

The type of a packet is indicated by the packet type in the header. The relationship between the packet type value and the type of data is as follows:

Animation data (TOD format) (cont.)

Table 6 Packet type value and nature of packet data

0	Attribute
1	Coordinate (RST)
10	TMD data ID
11	Parent object ID
100	Matrix value
101	TMD data body
110	Light source
111	Camera
1000	Object control
1001~1101	(User defined)
1110	(System reserved)
1111	Special commands

In the following we explain data types.

Packet data - attribute

When packet type is 0000, the data that designates attribute of the GsDOBJ structure in the packet data is stored. In this case a flag is not used.

Packet data is composed of 2 words as follows:

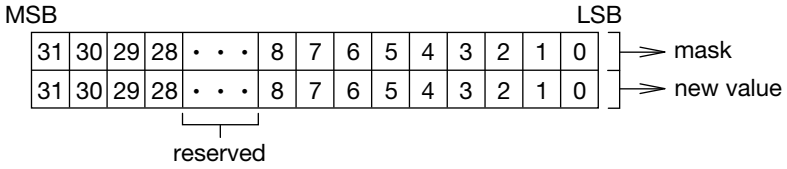


Fig. 5-4 packet data configuration when attribute

The first word is a mask which indicates the section which changes value and the section which does not change value. 0 is set in the bit which corresponds to the item which will change and 1 is set in the bit for the value which will not change.

In the second word, new data is entered in the bit which corresponds to the item which will change. Set other bits at 0.

Here be careful to set the default value for the bit which will not change so that the first word will be 1 and the second word will be 0 so that the first word and second word will be different.

The meaning of each bit in the packet data of the second word of Fig. 5-4 is as follows:

Table 7 The meaning of each bit in packet data

Bit (0) – Bit (2)	material decrease 00: material decrease 0 01: material decrease 1 02: material decrease 2 03: material decrease 3
Bit (3)	Number 1 lighting mode 0: fog off (no fog) 1: fog on (fog)
Bit (4)	Number 2 lighting mode 0: material on (material) 1: material off (no material)
Bit (5)	Number 3 lighting mode 0: use lighting mode 1: use default lighting mode
Bit (6)	Light source 0: No lighting calculation 1: Lighting calculation restriction ON
Bit (7)	Action when overflow 0: z overflow clip 1: z overflow not clip
Bit (8)	Existence or non-existence of back clip 0: Valid 1: Invalid
Bit (9) – Bit (27)	System reserved (initialised at 0)
Bit (28) – Bit (29)	Translucence rate 00: 50% 01: Add 100% 02: Add 50% 03: Add 25%
Bit (30)	Translucence 0: OFF 1: ON
Bit (31)	Display 0: Display 1: Nondisplay

For example, when “light source computation ON,” packet data will be as shown in Fig. 5-5.

In the first word, bit (6) specify 0 to indicate that light source will change, and specify 1 to indicate no change. Thus, the first word will be 0xffbf.

In the second word, set 1 in bit (6) which indicates “light source computation ON” and in other bits whose values do not change set the default, 0. Thus the second word will be 0x0040.

Animation data (TOD format) (cont.)

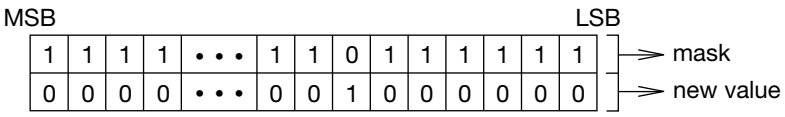


Fig. 5-5 Packet data when Illumination calculation restriction is ON

Packet data - Coordinate (RST)

translation	scaling	rotation	matrix type

Fig. 5-6 Flag when Coordinate (RST)

Table 8 Meanings and values of items in Fig 5-6

matrix type	RST matrix type 0: Absolute value 1: Differential matrix from preceding frame
rotation	Rotation (R) flag 0: None 1: Has
scaling	Screening (S) flag 0: None 1: Has
translation	Parallel movement (T) flag 0: None 1: Has

When packet type is 0001, data that sets the coord of the GsDOBJ structure is stored in packet data. In this case the flag will have the following meaning.

The configuration of packet data will differ according to the values of the flag rotation bit, the scaling bit, and the translation bit as per Fig. 5-7.

In Figure 5-7, Rx, Ry, and Rz express the X axis component, the Y axis component and the Z axis component of the turning angle, respectively.

In the same way Sx, Sy, and Sz express the X axis component, the Y axis component and the Z axis component of scaling, and Tx, Ty, and Tz express the X axis component, the Y axis component and the Z axis component of parallel movement, respectively.

(a) flag: 1110 or 1111

Rx	
Ry	
Rz	
Sy	Sx
******	Sz
Tx	
Ty	
Tz	

(b) flag: 0110 or 0111

Rx	
Ry	
Rz	
Sy	Sx
******	Sz

(c) flag: 1010 or 1011

Rx
Ry
Rz
Tx
Ty
Tz

(d) flag: 1100 or 1101

Sy	Sx
******	Sz
Tx	
Ty	
Tz	

(e) flag: 0010 or 0011

Rx
Ry
Rz

(f) flag: 0100 or 0101

Sy	Sx
******	Sz

(g) flag: 1000 or 1001

Tx
Ty
Tz

Fig. 5-7 Packet Data configuration when Coordinate (RST)

Packet data - TMD data ID

When packet type is 0010, the modeling data ID (TMD data) of the real object is stored in the packet data (See Figure 5-8). The TMD data ID is composed of 2 bytes. In this case no flag is used.

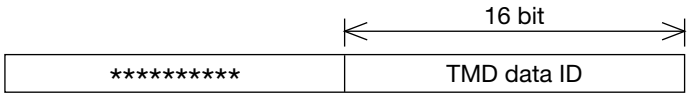


Figure 5-8 Packet data configuration when TMD data ID

Packet data- parent object ID

When packet type is 0011, the parent object ID of the object specified is stored in packet data (See Figure 5-9). The parent object ID is composed of 2 bytes. In this case no flag is used.

Animation data (TOD format) (cont.)

Packet data - MATRIX value

When the packet type is 0100, the data which designates coord members of the GsCOORDINATE2 structure to which GsDOBJ2 structure points is stored in packet data. In this case a flag is not used.

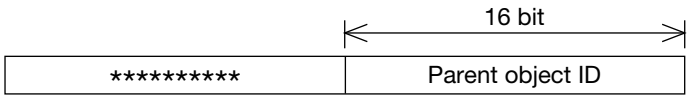


Fig. 5-9 Packet data configuration when Parent object

The configuration of packet data is as follows:

Packet data - TMD data body

When packet type is 0101, TMD data is stored. Not presently supported.

R01	R00
R10	R02
R11	R11
R21	R20
**********	R22
Tx	
Ty	
Tz	

Fig. 5-10 Packet data configuration when matrix value

Packet data - light source

When packet type is 0110, the data that designates light source is stored in packet data.
When this is the case, the object ID is separate from the normal object ID and becomes the light source ID. Flags have the following meanings.

**********	colour	direction	data type

Fig. 5-11 Flag when light source packet

Table 9 The meanings and values of the items in Fig 5-11

data type	Data type 0: Absolute value 1: Difference from preceding frame
direction	Direction flag 0: None 1: Has
color	Color flag 0: None 1: Has

The configuration of packet data will differ with the value of the flag direction bit, and the color bit as follows:

(a) flag: 0110 or 0111

X			
Y			
Z			
**	B	G	R

(b) flag: 0100 or 0101

X
Y
Z

(c) flag: 0010 or 0011

**	B	G	R

Fig. 5-12 Packet data when light source packet

Packet data - camera

When packet type is 0111, data which designates viewpoint location information is stored in the packet. When this is the case, the object ID is separate from the normal object ID and becomes the camera ID. Flags have the meaning indicated in Figure 5-13. Please be careful to note that the meaning of other bits will change depending on the type bit.

(a) camera type: 0

z angle	position & reference	data type	camera type = 0

(a) camera type: 1

translation	rotation	data type	camera type = 1

Fig. 5-13 Flag for camera

Animation data (TOD format) (cont.)

When camera type bit is 0 other bits are:

Table 10 Other bits when the camera type bit is 0

data type	Data type 0: Absolute value 1: Difference from preceding frame
position & reference	Position and reference position flag 0: None 1: Has
z angle	Reference angle flag from level 0: None 1: Has

When camera type bit is 1 other bits are:

Table 11 Other bits when the camera type bit is 1

data type	Data type 0: Absolute value 1: Difference from preceding frame
rotation	Rotation (R) flag 0: None 1: Has
translation	Horizontal movement (T) flag 0: None 1: Has

The configuration of packet data changes depending on the flag as in Figures 5-14 and 5-15.

(a) flag: 1100 or 1110

Px
Py
Pz
Rx
Ry
Rz
Z

(b) flag: 0100 or 0110

Px
Py
Pz
Rx
Ry
Rz

(c) flag: 1000 or 1010

Z

Fig. 5-14 Number 1 Packet data configuration for camera

(a) flag: 0111

Rx
Ry
Rz
Tx
Ty
Tz
tz

(b) flag: 0011

Rx
Ry
Rz

(c) flag: 0101

Tx
Ty
Tz

Fig. 5-15 Number 2 Packet data configuration for camera

Object control

When packet type is 1000, this designates control for the object. When this is the case, there is no packet data. Flags will have the following meanings.

Table 12 Flag meaning and value when object

0	create
1	kill
0010~1111	system reserved

Special Commands

When packet type is 1111, control of animation is performed. Details are not yet set.

Modeling data (TMD format)

TMD format is a modeling data format which is compatible with the PS-X expanded graphics library. TMD data is downloaded to memory and may be passed as an argument to functions provided by libgs.

TMD files are created by rsdlink command at the time a program is created from the RSD file (text data of a high degree of abstraction) created by artist's tools such as the 3D Graphics tool made by Sony.

The data described by the TMD are primitive aggregates such as the polygons and straight lines that make up an object. One TMD file may hold the data for multiple objects.

Coordinate value

The coordinate values in TMD data follow the space handled by the PS-X system's expanded graphics library and the positive direction for the X axis is right, for the Y axis, down, and the Z axis the back of the screen. Coordinates are 16 bit coded integer values and the value that the coordinate value of each axis may take is -32767 to +32768.

Because in the design stage, in other words in RSD format, the VERTEX value is a floating point, there is a need to expand or reduce and align the scale when going from RSD to TMD. The scale adjustment value at this time is included in the object structure that will be described later as a reference value (please refer to the section on OBJ TABLE section).

However, this value is an index value of what scale should be used in mapping world coordinates, and at present this scale value is ignored by libgs.

File format

TMD files are configured by 4 blocks. They have 3 dimensional object tables, and 3 types of data entities, PRIMITIVE, VERTEX, and NORMAL which configure these.

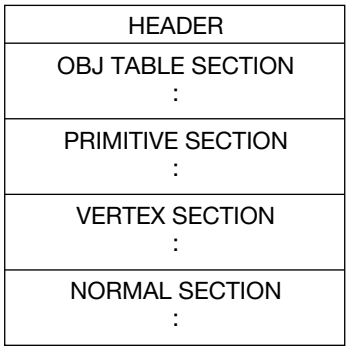


Fig. 5-16 TMD File Format

HEADER section

The HEADER section is 3 words (12 bytes) of data which have information concerning the data structure and is as follows.

ID
FLAGS
NOBJ

Fig. 5-17 Header structure

ID	32 bit length data, expresses TMD file version. Present version is 0x00000041.
FLAGS	32 bit length data, expresses TMD data configuration information. Least significant bit (LSB) is FIXP bit and the remaining bits are reserved. Value for all is 0. The FIXP bit expresses whether the pointer value held by the object structure that will be described later is the actual address or not.
NOBJ	Integer that expresses the number of objects.

OBJ TABLE section

The OBJ TABLE section is a table in which structures with pointer information are stored. These indicate where the data for each object is stored. The table is configured as follows.

OBJECT #1
OBJECT #2
⋮

Fig. 5-18 OBJ table structure

Object structures are as follows:

```
struct object
     {
     unsigned long *vert_top;         /*VERTEX leading address*/
     unsigned long n_vert:            /*VERTEX number*/
     unsigned long *normal_top:       /*NORMAL leading address*/
     unsigned long n_normal;          /*NORMAL number*/
     unsigned long *primitive_top;    /*PRIMITIVE leading address*/
     unsigned long n_primitive;       /*POLYGON number*/
     long          scale;             /*Scaling factor*/
     }
```

Of the members of the structure, the meaning of the pointer value (vert_top, normal_top, primitive_top) will vary with the value of the FIXP bit of the HEADER section. When FIXP is 1, it will indicate the actual address. When FIXP is 0 it expresses the relative address with the top of the object section as 0.

The scaling factor is encoded in long form, and the second exponential value becomes the scale value. In other words, when the scaling factor is 0, it is 0, when 2 the scale value is 4, when -1, the scale value is 1/2.

PRIMITIVE section

The primitive section is an arrangement of drawing packets of object primitives and one primitive is expressed in one packet. (See Figure 5-19).

Primitives defined by TMD differ from the drawing primitives handled by libgpu. With TMD primitives, transparent conversion processing is done by libgs function and they are converted to drawing primitives.

Each packet has a variable length and the size and structure vary depending on the type of primitive.

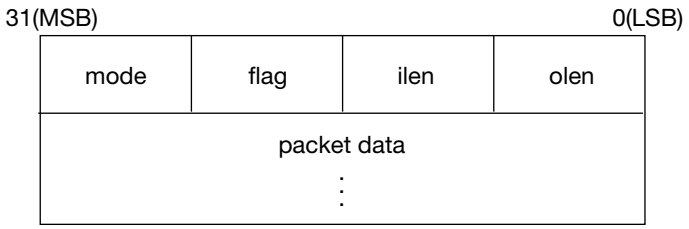
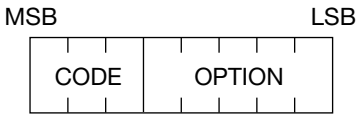


Fig. 5-19 Drawing packet general structure

Each item in Figure 5-19 is as follows:

mode (8 bit)

mode indicates the type of primitive and added attributes. They have the following bit structure.
CODE: 3 bit code expressing entities



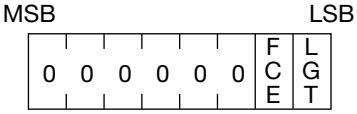
001 = polygon (triangle, rectangle)
010 = Straight line
011 = Sprite

OPTION: varies with the option bit and CODE values
(Listed with the list of packet data configurations described later)

Fig. 5-20 MODE

flag (8 bit)

flag indicates option information when rendering and has the following bit configuration.



FCE: When 1, two-sided polygon, when 0, one-sided polygon.
(Only valid when the CODE value is polygon)
LGT: When 1, no light source computation, when 0, light source computation.

Fig. 5-21 flag

Ilen (8 bit)

Indicates packet data section word length.

Olen (8 bit)

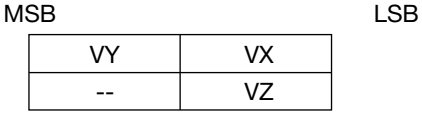
Indicates the word length of the 2D drawing primitives that are generated by intermediate processing.

Packet data

Content varies depending on type of primitive. Please refer to “Packet data configuration” which will be discussed later.

VERTEX section

The VERTEX section is a string of structures expressing vertex. The format of the structure is as shown in Figure 5-22.



VX,VY, VZ Vertex coordinates X,Y, Z values (16 bit integers)

Fig. 5-22 VERTEX structure

Modeling data (TMD format) (cont.)

NORMAL section

The NORMAL section is a string of structures expressing a vertex normal. The format of the structure is as shown in Figure 5-23.

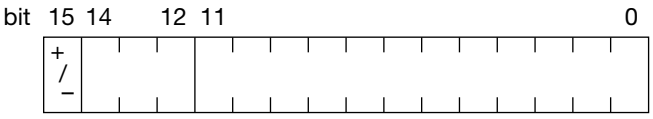
MSB LSB

NY	NX
--	NZ

NX, NY, NZ: X, Y, Z elements of normal (16 bit fixed point)

Fig. 5-23 NORMAL structure

The values of NX, NY and NZ are signed 16 bit fixed points with 4096 treated as 1.0. (See Figure 5-24)



Sign: 1 bit
Integer section: 3 bits
Decimal section: 12 bits

Fig. 5-24 Fixed point format

Packet data configuration

There are the following in the parameters included in the packet data of the primitive section.

Vertex (n)

16 bit length index value pointing to VERTEX. Expresses the number element from the top of the VERTEX section of the object that the polygon belongs to.

Normal (n)

A 16 bit length index value pointing to NORMAL, like Vn.

Un, Vn

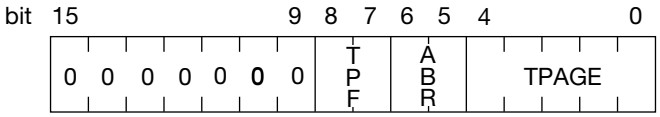
X and Y coordinate values in the texture source space of each vertex.

Rn, Gn, Bn

RGB value expressing color of polygon, an 8 bit unsigned integer value. When no light source computation, it is necessary to specify the calculated intensity in advance.

TSB

Has information concerning texture and sprite patterns. Is as follows:

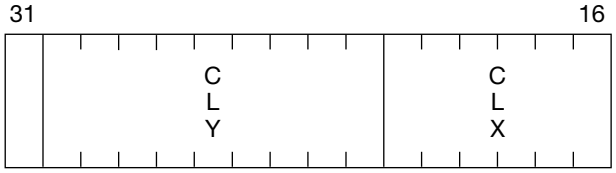


TPAGE: Texture page number (0-31)
ABR: Translucency rate (mixed ratio). Only valid when ABE is 1.
00: 50%back + 50% polygon
01: 100%back + 100% polygon
10: 100%back + 50% polygon
11: 100%back - 100% polygon
TPF: Texture color mode
00: 4bit CLUT
01: 8bit CLUT
10: 15bit

Figure 5-25 TSB format

CBA

Indicates the storage location of CLUT in VRAM



CLY CLUT lead Y coordinate (9 bits)
CLX CLUT lead X coordinate (6 bits)

Fig. 5-26 CBA

Please use the above as a reference since we describe the configuration of packet data for each type of primitive.

Packet data configuration example - 3 vertex polygon with light source calculation

A 3 vertex polygon with light source calculation will be as follows: The mode and flag values in this example express a one sided polygon with translucency in the OFF state.

Modeling data (TMD format) (cont.)

Bit configuration of mode value

The mode value bit configuration of the primitive section is as follows:

mode value bit configuration

MSB LSB

0	0	1	IIP	1	TME	ABE	TGE	

IIP: Shading mode
0: Flat shading
1: Gouraud shading

TME: Texture specification
0:Off
1:On

ABE: Translucency processing
0:Off
1:On

TGE: Brightness calculation at time of texture mapping
0:On
1:Off (Draws texture as is)

Fig. 5-27 Mode structure

Packet data configuration

Packet data configuration is as follows.

Flat, No-Texture

0x20	0x00	0x03	0x04
Note	B	G	R
Vertex0		Normal0	
Vertex2		Vertex1	

Gouraud, No-Texture

0x30	0x00	0x04	0x06
Note	B	G	R
Vertex0		Normal0	
Vertex1		Normal1	
Vertex2		Normal2	

Flat, Texture

0x24	0x00	0x05	0x07
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
Vertex0		Normal0	
Vertex2		Vertex1	

Note: Has the same value as mode.

Gouraud, Texture

0x34	0x00	0x06	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
Vertex0		Normal0	
Vertex1		Normal1	
Vertex2		Normal2	

Fig. 5-28 Packet data for polygons

Packet data configuration example - For a polygon with 3 vertices and no light source calculation

Bit configuration of mode value

The primitive section mode value bit configuration is as follows. For the value of each bit please refer to “3 vertex polygon with light source calculation.”

Mode value bit configuration

MSB							LSB
0	0	1	IIP	0	TME	ABE	TGE

Fig. 5-29 Mode byte

Packet data configuration

Packet date configuration will be as follows:

Flat, No Texture

0x21	0x01	0x03	0x04
Note	B	G	R
Vertex1		Vertex0	
--		Vertex2	

Gouraud, No Texture

0x31	0x01	0x05	0x06
Note	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
Vertex1		Vertex0	
--		Vertex2	

Flat, Texture

0x25	0x01	0x06	0x07
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	B	G	R
Vertex1		Vertex0	
--		Vertex2	

Note: Has same value as mode.

Gouraud, Texture

0x35	0x01	0x08	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	B0	G0	R0
--	B1	G1	G1
--	B2	G2	G2
Vertex1		Vertex0	
--		Vertex2	

Fig. 5-30 Packet data structures for 4 vertex polygons

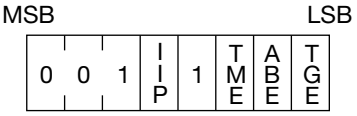
Modeling data (TMD format) (cont.)

Packet data configuration example - For a polygon with 4 vertices and light source calculation

Bit configuration of mode value

The primitive section mode value bit configuration is as follows: For the value of each bit please refer to “3 vertex polygon with light source calculation.”

Mode value bit configuration



Note: Bit 3 is set to 1 to designate a 4 vertex primitive.

Fig. 5-31 Mode byte

Packet data configuration

Packet data configuration is as follows:

Flat, No Texture

0x28	0x00	0x04	0x05
Note	B	G	R
Vertex0		Normal0	
Vertex2		Vertex1	
--		Vertex3	

Gouraud, No Texture

0x38	0x00	0x05	0x08
Note	B	G	R
Vertex0		Normal0	
Vertex1		Normal1	
Vertex1		Normal2	
Vertex2		Normal3	

Flat, Texture

0x2c	0x00	0x07	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
Vertex0		Normal0	
Vertex2		Vertex1	
--		Vertex3	

Note: Has same value as mode.

Gouraud, Texture

0x3c	0x00	0x08	0x0c
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
Vertex0		Normal0	
Vertex1		Normal1	
Vertex2		Normal2	
Vertex3		Normal3	

Fig. 5-32 Mode

Packet data configuration example - For a polygon with 4 vertices and no light source calculation

Bit configuration of mode value

The primitive section mode value bit configuration is as follows: For the value of each bit please refer to “3 angle polygon with light source calculation.”

Mode value bit configuration

MSB								LSB
0	0	1	IIP	1	TME	ABE	TGE	

Note: Bit 3 is set to 1 to designate a 4 vertex primitive.

Fig. 5-33 Mode byte

Packet data configuration

Packet data configuration is as follows:

Flat, No Texture

0x29	0x01	0x03	0x05
Note	B	G	R
Vertex1		Vertex0	
Vertex3		Vertex2	

Gouraud, No Texture

0x39	0x01	0x06	0x08
Note	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex1		Vertex0	
Vertex3		Vertex2	

Flat, Texture

0x2d	0x01	0x07	0x09
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
--	B	G	R
Vertex1		Vertex0	
Vertex3		Vertex2	

Note: Has the same value as mode.

Gouraud, Texture

0x3d	0x01	0x0a	0x0c
CBA		V0	U0
TSB		V1	U1
--	--	V2	U2
--	--	V3	U3
--	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
--	B3	G3	R3
Vertex1		Vertex0	
Vertex3		Vertex2	

Fig. 5-34 Packet Data

Modeling data (TMD format) (cont.)

Packet data configuration example - For a straight line

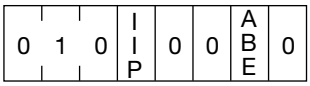
Bit configuration of mode value

The primitive section mode value bit configuration is as follows:

IIP: With or without gradation
0: Gradation off (Monochrome)

Mode value bit configuration

MSB LSB



1: Gradation on
ABE: Translucency processing on/off
0: off
1: on

Fig. 5-35 mode

Packet data configuration

Packet data configuration is as follows:

Gradation off

0x40	0x01	0x02	0x03
Note	B	G	R
Vertex1		Vertex0	

Note: Has the same value as mode.

Gradation on

0x50	0x01	0x04	0x04
Note	B0	G0	R0
--	B1	G1	R1
--	B2	G2	R2
Vertex1		Vertex0	

Fig. 5-36 packet data

Packet data configuration example - For a 3 dimensional sprite

A 3 dimensional sprite is a sprite with 3-D coordinates and the drawing content is the same as a normal sprite.

Bit configuration of mode value

The primitive section mode value bit configuration is as follows:

Mode value bit configuration

MSB							LSB
0	1	1	SIZ		1	ABE	0

SIZ: Sprite size
00: Free size (Specified by W, H)
01: 1 x 1
10: 8 x 8
11: 16 x 16

ABE: Translucency processing
0: off
1: on

Fig. 5-37 Mode

Packet data configuration

Packet data configuration is as follows:

Free size

0x64	0x01	0x03	0x05
TSB		Vertex0	
CBA		V0	U0
H		W	

1 x 1

0x6c	0x01	0x02	0x04
TSB		Vertex0	
CBA		V0	U0

8 x 8

0x74	0x01	0x02	0x04
TSB		Vertex0	
CBA		V0	U0

16 x 16

0x7c	0x01	0x02	0x04
TSB		Vertex0	
CBA		V0	U0

Fig. 5-38 Packet data for sprites

Screen image data (TIM format)

The TIM format is the screen image standard handled by the PS-X system. TIM data may be directly transferred to the PS-X system frame buffer. It may also represent sprite pattern and 3 dimensional texture mapping materials.

Image data modes (color numbers)

There are four image data modes that can be handled by the PS-X system.

(a) 4 bit CLUT (color look-up table) (16 colors)
(b) 8 bit CLUT (256 colors)
(c) 16 bit Direct color (32768 colors)
(d) 24 bit Direct color (Full color)

Since the VRAM of the PS-X system has a 16 bit configuration, only 16 bit or 24 bit data can be directly transferred to the frame buffer. However, sprite patterns and polygon texture mapping patterns can be 4 bit, 8 bit or 16 bit.

File format

TIM files have a file header (ID) at the top and consist of several different blocks.

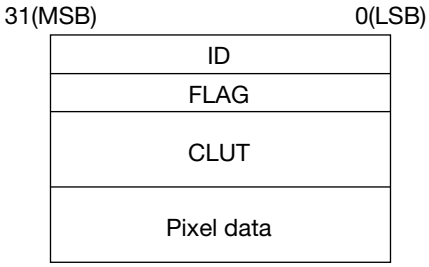


Fig. 5-39 TIM file format

All data is in 32 bit binary data strings. And because it is Little Endian (like the 80 x 86), the byte order of multiple byte data is such that the low order byte comes first as in Figure 5-40.

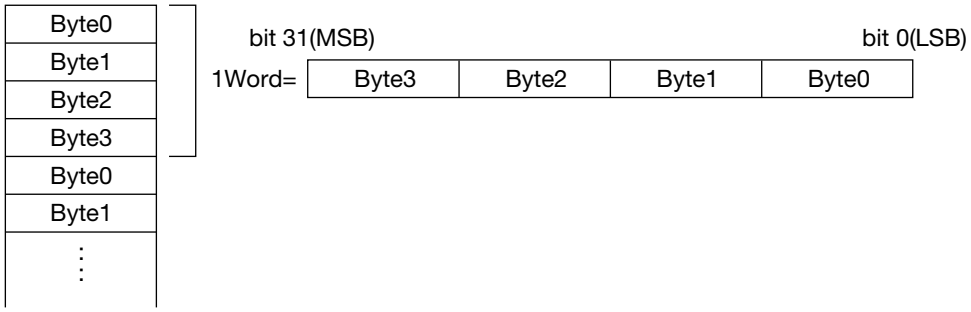


Fig. 5-40 Little Endian byte order within files

ID section

Expresses file ID. File ID is one word of data and the bit configuration is as follows.

bit 0 - 7: ID Value is 0x10
bit 8 - 15: Version number. Value is 0x00

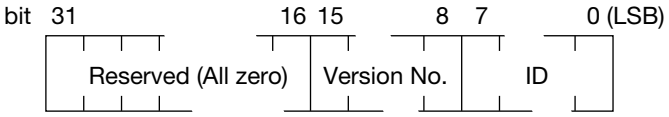


Fig. 5-41 File ID word

Flag section

Flags are 32 bit data containing information concerning the file structure. The bit configuration is as in Figure 5-42.

When a single TIM data file contains numerous sprites and texture data, the value of PMODE is 4 (mixed), since data of multiple types is intermingled.

bit 0 - 2 (PMODE) Pixel mode (Bit length)
0: 4 bit CLUT
1: 8 bit CLUT
2: 15 bit Direct
3: 24 bit Direct
4: Mixed
bit 3 (CF) Whether there is a CLUT or not
0: No CLUT section
1: Has CLUT section
Other: reserved

Fig. 5-42 Flag word

CLUT section

The CF flag of the flag word specifies whether the TIM file has a CLUT section or not. A CLUT is a color palette used by 4 bit, and 8 bit mode image data.

As shown in Figure 5-42, the CLUT section is configured so that the number of CLUT section bytes is at the start of the structure followed by the VRAM location information, image size and data body.

Screen image data (TIM format) (cont.)

bit 31(MSB) bit 0(LSB)

bnum	
DX	DY
H	CLUT 0
CLUT 1	
⋮	
CLUT n	CLUT n-1

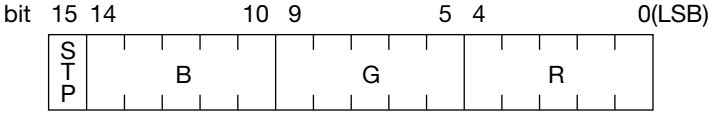
Bnum	CLUT Section data length. Unit is bytes. (Includes bnum's four bytes).
DX	The x coordinate in VRAM.
DY	The y coordinate in VRAM.
H	Vertical direction data size.
W	Horizontal direction data size.
CLUT 1 - n	CLUT entry (16 bit/ entry).

Fig. 5-43 CLUT structure

In 4 bit mode, 16 CLUT entries make up one CLUT. In 8 bit mode, 256 CLUT entries make up 1 CLUT.

In the PS-X system, because CLUT is placed in VRAM, the CLUT section of the TIM file is handled as a rectangular VRAM image. In other words, one CLUT entry equals one pixel in VRAM. Because of this, one CLUT is handled as a rectangular image and in 4 bit mode the height is 1, width is 16 and in 8 bit mode the height is 1, and the width is 256.

It is possible for more than one CLUT to be held in a TIM file. In this case, the multiple CLUT data is stored as one image. The configuration of a CLUT entry is as follows.



STP	Semi-transparent control bit
R	Red component (5 bits)
G	Green component (5 bits)
B	Blue component (5 bits)

Fig. 5-44 A CLUT entry

The Transparent control bit (STP) is valid when used as sprite data and texture data. This is what controls whether the sprites and pixels that pertain to polygons are transparent or not. When the STP value is 1, pixels of that color will be translucent. Otherwise they will be a non-transparent color. The R,G and B bits control the color component.

If the R, G and B bits are 0 and the STP bit is 0 the pixel will always be transparent. If the R, G and B bits are 0 and the STP bit is 1 the pixel will always be non-transparent. These relationships are shown in the chart below.

STP/R,G,B	Translucent Processing ON	Translucent Processing OFF
0,0,0,0	Transparent	Transparent
0,X,X,X	Not transparent	Not transparent
1,X,X,X	Translucent	Not transparent
1,0,0,0	Non-transparent black	Non-transparent black

Figure 5-45 STP bit function in combination with R, G, B data

Pixel data section

The pixel data section is the image data body. Since the VRAM of the PS-X system has a 16 bit configuration, the image data is separated in 16 bit units. The pixel data section is configured as follows:

bit 31(MSB) bit 0(LSB)

bnum	
DX	DY
H	W
DATA 1	DATA 0
⋮	
DATA n	DATA n-1

bnum	Pixel data section data length. Unit is bytes. Includes the 4 bytes of bnum.
DX	The x coordinate in VRAM
DY	The y coordinate in VRAM
H	Vertical direction data size
W	Horizontal direction data size (16 bit units)
DATA 1-n	VRAM data (16 bit)

Fig. 5-46 Pixel data

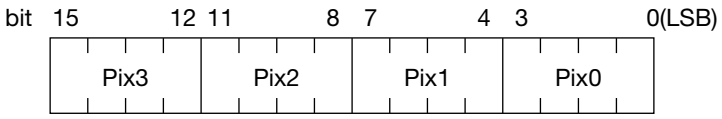
The configuration of 1 piece of VRAM data (16 bits) varies with the image data mode. The configuration of each mode is as shown in diagram 9.

Be careful when handling the pixel data size in TIM data. Because the width value (horizontal width) in Fig. 5-46 is in 16 bit pixel units, the actual image size in 4 and 8 bit mode will be 1/4 and 1/2,

Screen image data (TIM format) (cont.)

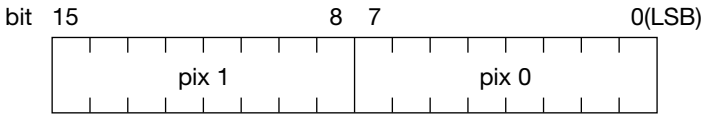
respectively. Thus the image size horizontal width in 4 bit mode must be a multiple of 4 and in 8 bit mode must be an even number.

(a) In 4 bit mode



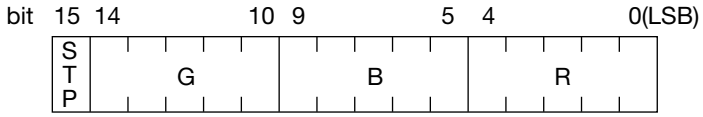
pix 0-3 pixel value (CLUT Index)
The order on the screen is from the left, i.e. 0, 1, 2, 3.

(b) In 8 bit mode



pix 0-1 pixel (CLUT Index)
The order on the screen is from the left, i.e. 0, 1.

(c) When 16 bit mode



STP	Transparent control bit (See CLUT section)
R	Red component (5 bits)
G	Green component (5 bits)
B	Blue component (5 bits)

Fig. 5-47 VRAM data (pixel data)

BG map data (BGD format)

BGD format is the standard for data configuring BG (background) surfaces using 2 dimensional systems.

BGD files normally use the same name as the TIM file (the file base name is the same) and the actual pixel images are held by the TIM file.

File format

A BGD file has a header file at the top and is divided into three blocks. However it is possible to omit the ATTR section.

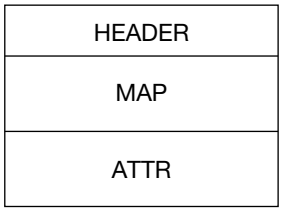


Fig. 5-48 BG File format

Header section

This is the file header. It is configured as follows.

31(MSB) 0(LSB)

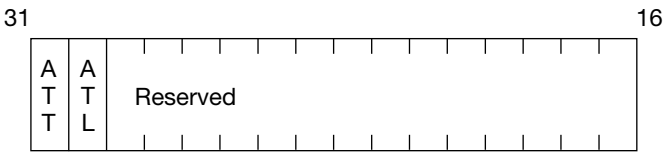
FLAG		VERSION	ID
CELLH	CELLW	MAPH	MAPW

ID	0x23
VERSION	0x00
FLAG	See Fig. 5-49
MAPH	BG Map data vertical direction size (Unit is cell)
MAPW	BG Map data horizontal direction size (Unit is cell)
CELLH	Cell data vertical direction size (Unit is cell)
CELLW	Cell data horizontal direction size (Unit is cell)

Fig. 5-49 BG Header format

BG map data (BGD format) (cont.)

The FLAG in Figure 5-48 has the following bit configuration



ATT	Whether or not the ATTR block is included in this file 0: ATTR is not included 1: ATTR is included
ATL	ATTR data length 0: 8 bits 1: 16 bits

Figure 5-50 Flag half word structure

MAP Section

A map is considered as a set of the cells of MAPH x MAPW (a matrix of the vertical and horizontal size) which describes the order of arrangement of these cells. For example the arrangement of the cells of an 8 x 8 map would be as follows.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Fig. 5-51 Cell arrangement in MAP (when 8 x8)

The Map section is an aggregate of cell numbers arranged in numerical order in a form like that in Figure 5-50 (See Figure 5-51). Cell number is a number which indicates the number of the cell in the CEL file.

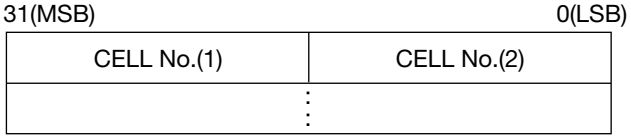


Fig. 5-52 MAP

ATTR section

Indicates attribute data. Attribute data is additional information concerning the MAP and is arranged in the same order as MAP.

There are two types of attribute data, 8 bit data and 16 bit data and each is as follows. Data length is indicated in the Header section, in the ATL bit in the FLAG.

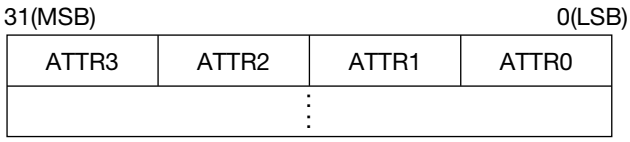


Fig. 5-53 ATTR (8 bit)

Fig. 5-54 ATTR (16 bit)

Cell data (CEL format)

CEL format describes the pointer table in VRAM of CELL, a component of BG surfaces.

File format

There is a header file at the top of a CEL file and a CEL file is divided into 3 blocks.

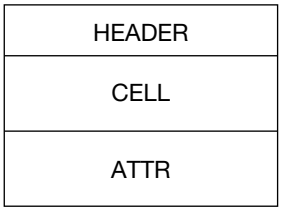
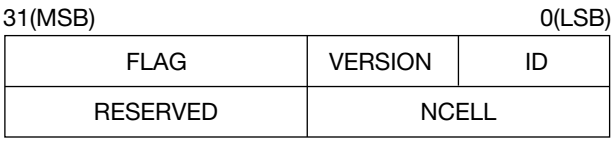


Fig. 5-55 Cell File Format

Header Section

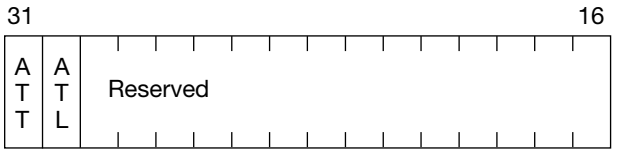
This is the file header. It has the following configuration.



FLAG	See Figure 5-57
ID	0x22
VERSION	0x00
NCELL	Cell data number (unit is cell)

Fig. 5-56 CEL file header

The FLAG in Figure 5-56 has the following bit configuration.

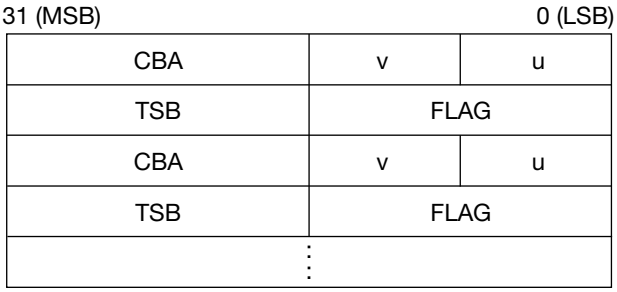


ATT:	Whether or not the ATTR block is included in this file
	0: ATTR is not included
	1: ATTR is included
ATL	ATTR data length
	0: 8 bit
	1: 16 bit

Figure 5-57 CEL file flag half word format

CELL section

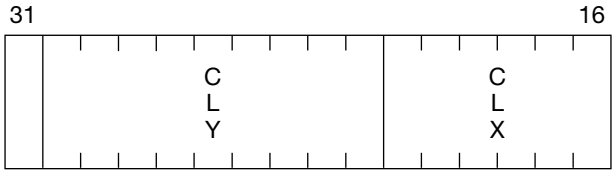
This is a table of pointers to cells in VRAM, which is a component of BG. 4 bytes corresponds to one cell.



CBA	See Figure 5-59
v	The vertical offset from the texture page base address (8 bits)
u	The horizontal offset from the texture page base address (8 bits)
TSB	See Figure 5-60
FLAG	See Figure 5-61

Figure 5-58 CEL section format

The CBA in Figure 5-58 has the following bit configuration.

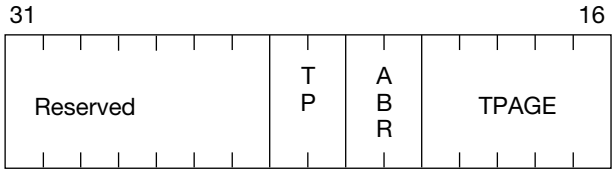


CLY	CLUT lead Y coordinate (9 bits)
CLX	CLUT lead X coordinate (6 bits)

Fig. 5-59 CBA half word format

Cell data (CEL format) (cont.)

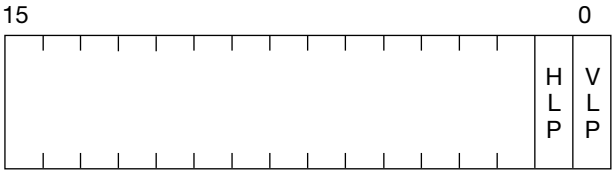
The TSB in Figure 5-58 has the following bit configuration.



TP	Pixel depth of texture pattern
	00: 4 bit CLUT
	01: 8 bit CLUT
	10: 16 bit
ABR	Translucence rate (F = foreground, B = background)
	00: 0.50xF +0.50xB
	01: 1.00F + 1.00xB
	10: 0.50F + 1.00B
	11: 0.25xF + 1.00B
TPAGE	Texture Page number
	01000: (512,0)
	11000: (512, 256)
	01100: (768, 0)
	11100: (768, 256)

Figure 5-60 TSB format

The FLAG in Figure 5-58 has the following bit configuration.



TP	Texture pattern pixel depth
HLP	Horizontal inversion information
VLP	Vertical inversion information

Figure 5-61 FLAG format for CEL entries

ATTR section

Expresses attribute data. Attribute data is additional information concerning the cell and is arranged in the same order as CEL.

There are two types of attribute data, 8 bit length data and 16 bit length data and each would be as follows. Data length is indicated in the Header section, in the ATL bit in the FLAG half word.

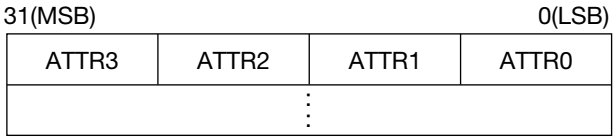


Figure 5-62 ATTR format (8 bit)

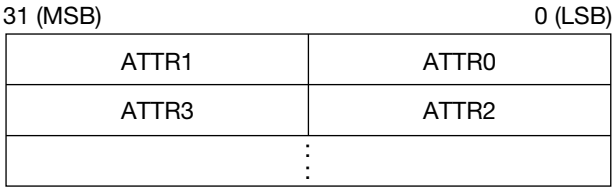


Fig. 5-63 ATTR format (16 bit)

Information data (ANM format)

ANM format prescribes information for animating pictorial image data. The ANM file is normally used in tandem with the TIM file and the actual pixel images are held by the TIM file.

File format

ANM files have a header at the beginning and are divided into 4 parts.

HEADER
SEQUENCE
SPRITEGp
CLUTGp

Fig. 5-64 ANM File Format

HEADER section

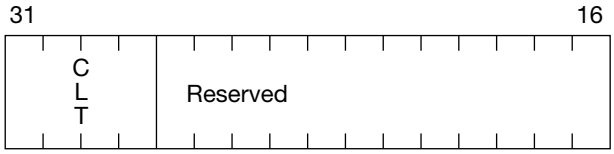
This is the file header. It has the following configuration.

31(MSB)		0(LSB)
FLAG	VERSION	ID
NSEQUENCE	NSPRITEGp	

ID	0x21
VERSION	0x00
FLAG	See Figure 5-66
NSEQUENCE	Sequence data number
NSPRITEGp	Number of Sprite groups

Fig. 5-65 ANM header format

The FLAG in Figure 6-65 has the following bit configuration.

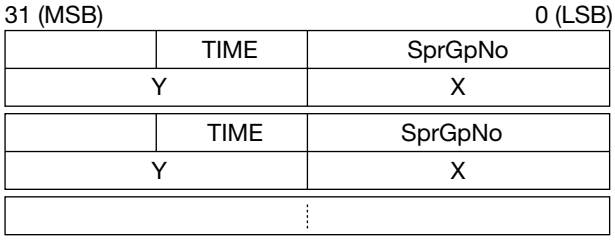


CLT	CLUT number for color animation

Fig. 5-66 FLAG half word format

SEQUENCE section

This is sequence data. Sequence data is a list of coordinates, display times and sprite group numbers for hot spots for each frame.



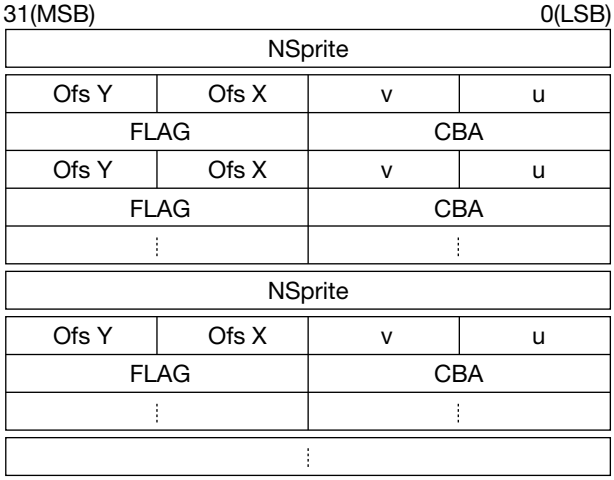
TIME	Display time (number of repeats)
SprGpNo	The sprite group number
X	X coordinate of hot spot
Y	Y coordinate of hot spot

Fig. 5-67 Sequence

SPRITEGp section

This is a Sprite group set. A sprite group describes a location that a given sprite will be displayed at in one frame.

The configuration of a sprite group section will vary with the size of the sprite as in Figures 5-68 and 5-69.



Fig, 5-68 SPRITEGp (fixed sprite size)

Information data (ANM format) (cont.)

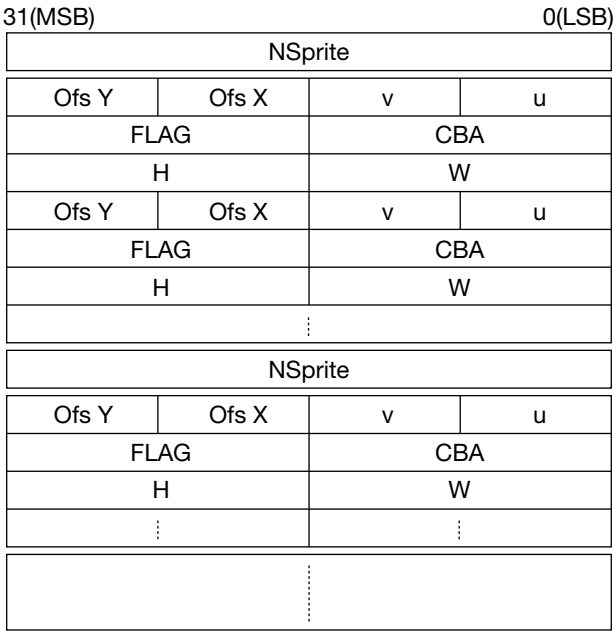
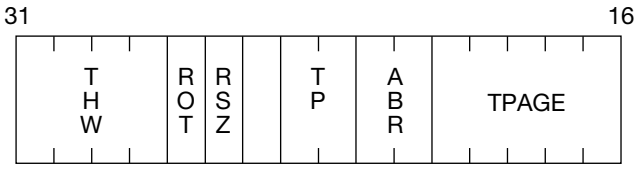


Fig. 5-69 SPRITEGp (variable sprite size)

Nsprite	The number of sprites within in one sprite group
FLAG	See Figure 5-70
v	The vertical offset from the texture page base address
u	The horizontal offset from the texture page base address
Ofs Y	The vertical offset from the hot spot in the frame
Ofs X	The horizontal offset from the hot spot in the frame
CBA	Described earlier
H	The texture width of the desired size
W	The texture height of the desired size

The FLAG in Figures 5-68 and 5-69 have the following bit configuration.



THW		The rectangular area size of the sprite divided by 8 (When cannot be divided by 8, leave as 0x0 and specify the actual size in H and W above).
ROT		Whether rotates or not.
	0:	Does not rotate.
	1:	Does rotate.
RSZ		Whether expands and contracts or not.
	0:	Does not expand and contract.
	1:	Does expand and contract.
TP		Pixel depth of texture pattern.
	00:	4 bit CLUT.
	01:	8 bit CLUT.
	10:	16 bit.
ABR		Translucence rate.
	00:	0.50xF + 0.50xB.
	01:	1.00xF + 1.00xB.
	10:	0.50xF + 1.00xB.
	11:	0.25xF + 1.00xB.
TPAGE		Texture Page number (0-31)

Figure 5-70 FLAG half word format

When rotation, or expansion/contraction has been specified, the following data will be appended after the sprite definition.

31 (MSB) 0(LSB)

Reserved	ROT
Y	X

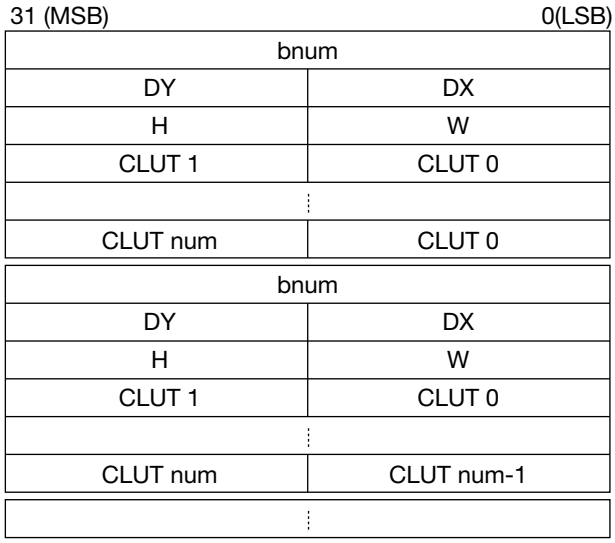
ROT	Rotation angle
Y, X	Expansion/contraction rate (fixed point designation)

Fig. 5-71 Rotation, expansion/contraction designation

Information data (ANM format) (cont.)

CLUTGp section

A set of CLUT groups used when doing color animation. The CLUT number is specified by the CLT of the flag structure of the header section.



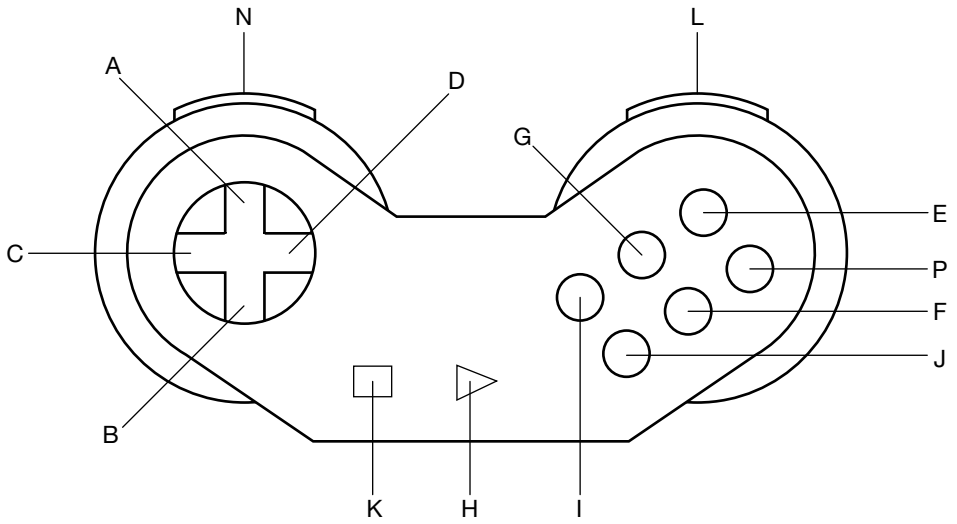
bnum	CLUT data length (bytes)
DX	The x coordinate in VRAM
DY	The y coordinate in VRAM
W	The horizontal data size
H	The vertical data size
CLUT 1-n	CLUT entry (16 bit/entry)

Figure 5-72 CLUTGp structure

Controller Pad (DTL-H500C) Button Layout

Controller Pad (DTL-H500C) Button Layout

A1. Controller (DTL-H500C) button arrangement/bit correspondence diagram



Bit no.	Corresponding button
15	C
14	B
13	D
12	A
11	H
10	J
9	I
8	K
7	G
6	F
5	P
4	E
3	L
2	N
1	

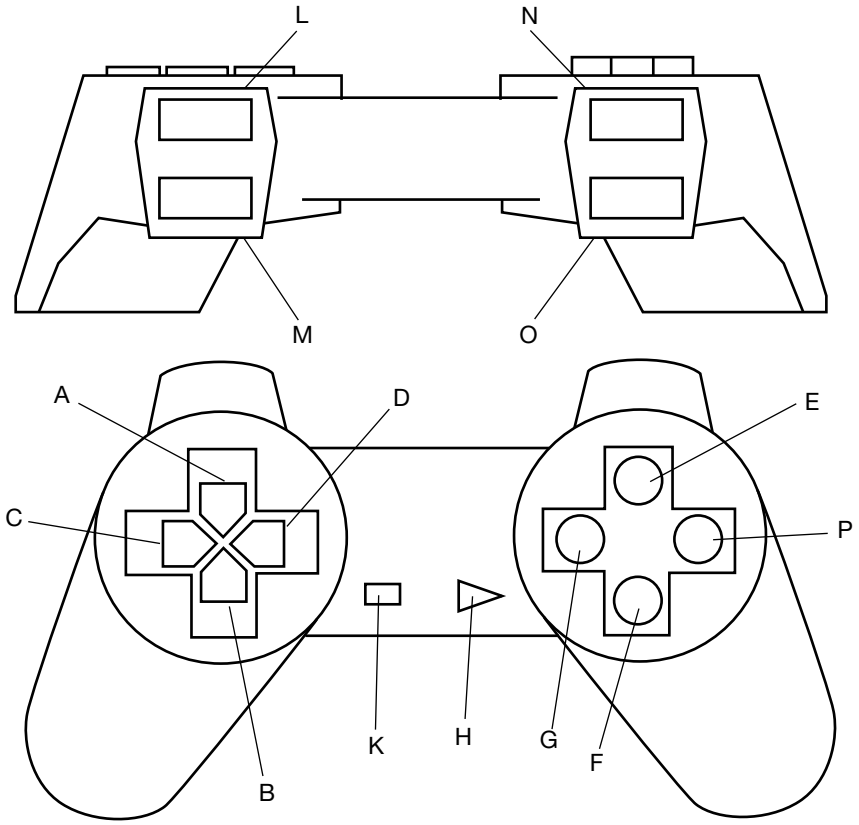
Value of each bit
0: Not Pressed
1: Pressed

Note: This diagram indicates the button configuration arrangement of the controller used in the target box for development. Please see the following page for the controller for the actual PS-X machine (the product with the final specifications.

Controller Pad (Final Specification) Button Layout

Controller Pad (Final Specification) Button Layout

B1. Controller (product with final specifications) button arrangement/bit correspondence diagram



Bit no.	Corresponding button
15	C
14	B
13	D
12	A
11	H
10	
9	
8	K
7	G
6	F
5	P
4	E
3	L
2	N
1	M
0	O

Value of each bit
0: Not Pressed
1: Pressed