Contents

Nocash PSXSPX Playstation Specifications
Memory Map
I/O Map
Graphics Processing Unit (GPU)
Geometry Transformation Engine (GTE)
Macroblock Decoder (MDEC)
Sound Processing Unit (SPU)
Interrupts
DMA Channels
Timers
CDROM Drive
Controllers and Memory Cards
Pocketstation
Serial Port (SIO)
Expansion Port (PIO)
Memory Control
Unpredictable Things
CPU Specifications
Kernel (BIOS)
Arcade Cabinets
Hardware Numbers
Pinouts
About & Credits

Latest Research
CDROM Internal Info on PSX CDROM Controller
Plus updated MDEC and DMA chapters.
And, CDROM BIOS Dumping

Memory Map

Memory Map

```
KUSEG     KSEG0     KSEG1
00000000h 80000000h A0000000h  2048K  Main RAM (first 64K reserved for BIOS)
1F000000h 9F000000h BF000000h  8192K  Expansion Region 1 (ROM/RAM)
1F800000h 9F800000h    --      1K     Scratchpad (D-Cache used as Fast RAM)
1F801000h 9F801000h BF801000h  8K     I/O Ports
1F802000h 9F802000h BF802000h  8K     Expansion Region 2 (I/O Ports)
1FA00000h 9FA00000h BFA00000h  2048K  Expansion Region 3 (whatever purpose)
1FC00000h 9FC00000h BFC00000h  512K   BIOS ROM (Kernel) (4096K max)
      FFFE0000h (KSEG2)        0.5K   I/O Ports (Cache Control)
```

Additionally, there are a number of memory mirrors.

Additional Memory (not mapped to the CPU bus)

```
1024K   VRAM (Framebuffers, Textures, Palettes) (with 2KB Texture Cache)
512K    Sound RAM (Capture Buffers, ADPCM Data, Reverb Workspace)
0.5K    CDROM controller RAM (see CDROM Test commands)
16.5K   CDROM controller ROM (Firmware and Bootstrap for MC68HC05 cpu)
32K     CDROM Buffer (IC303) (32Kx8) (BUG: only two sectors accessible?)
128K    External Memory Card(s) (EEPROMs)
```

KUSEG,KSEG0,KSEG1,KSEG2 Memory Regions

```
Address   Name   Size   Privilege    Code-Cache  Data-Cache
00000000h KUSEG  2048M  Kernel/User  Yes         (Scratchpad)
80000000h KSEG0  512M   Kernel       Yes         (Scratchpad)
A0000000h KSEG1  512M   Kernel       No          No
C0000000h KSEG2  1024M  Kernel       (No code)   No
```

Kernel Memory: KSEG1 is the normal physical memory (uncached), KSEG0 is a mirror thereof (but with cache enabled). KSEG2 is usually intended to contain virtual kernel memory, in the PSX it's containing Cache Control I/O Ports.
User Memory: KUSEG is intended to contain 2GB virtual memory (on extended MIPS processors), the PSX doesn't support virtual memory, and KUSEG simply contains a mirror of KSEG0/KSEG1 (in the first 512MB) (trying to access memory in the remaining 1.5GB causes an exception).

Code Cache
Works in the cached regions (KUSEG and KSEG0).
There are reportedly some restrictions... not sure there... eventually it is using the LSBs of the address as cache-line number... so, for example, it couldn't simultaneously memorize opcodes at BOTH address 80001234h, AND at address 800F1234h (?)

Data Cache aka Scratchpad
The MIPS CPU usually have a Data Cache, but, in the PSX, Sony has misused it as "Scratchpad", that is, the "Data Cache" is mapped to a fixed memory location at 1F800000h..1F8003FFh (ie. it's used as Fast RAM, rather than as cache).
There <might> be a way to disable that behaviour (via Port FFFE0130h or so), but, the Kernel is accessing I/O ports via KUSEG, so activating Data Cache would cause the Kernel to access cached I/O ports.
Not tested yet, but most probably the Scratchpad can be used only for Data (ie. NOT for program Code?).

Memory Mirrors
As described above, the 512Mbyte KUSEG, KSEG0, and KSEG1 regions are mirrors of each other. Additional mirrors within these 512MB regions are:

```
2MB RAM can be mirrored to the first 8MB (strangely, enabled by default)
512K BIOS ROM can be mirrored to the last 4MB (disabled by default)
Expansion hardware (if any) may be mirrored within expansion region
The seven DMA Control Registers at 1F8010x8h are mirrored to 1F8010xCh
```

The size of the RAM, BIOS, Expansion regions can be configured by software, for Expansion Region it's also possible to change base address, see:
Memory Control
The Scratchpad is mirrored only in KUSEG and KSEG0, but not in KSEG1.

Memory Exceptions

```
Memory Error ------> Misalignments
```

```
              (and probably also KSEG access in User mode)
  Bus Error   ------> Unused Memory Regions (including Gaps in I/O Region)
              (unless RAM/BIOS/Expansion mirrors are mapped to "unused" area)
```

More Memory Info
For Info on Exception vectors, Unused/Garbage memory locations, I/O Ports, Expansion ROM Headers, and Memory Waitstate Control, etc. see:
I/O Map
Memory Control
EXP1 Expansion ROM Header
BIOS Memory Map
BIOS Memory Allocation
COP0 - Exception Handling
Unpredictable Things

I/O Map

Expansion Region 1
```
  1F000000h 80000h Expansion Region (default 512 Kbytes, max 8 MBytes)
  1F000000h 100h   Expansion ROM Header (IDs and Entrypoints)
```
Scratchpad
```
  1F800000h 400h Scratchpad (1K Fast RAM) (Data Cache mapped to fixed address)
```
Memory Control 1
```
  1F801000h 4    Expansion 1 Base Address (usually 1F000000h)
  1F801004h 4    Expansion 2 Base Address (usually 1F802000h)
  1F801008h 4    Expansion 1 Delay/Size (usually 0013243Fh; 512Kbytes 8bit-bus)
  1F80100Ch 4    Expansion 3 Delay/Size (usually 00003022h; 1 byte)
  1F801010h 4    BIOS ROM    Delay/Size (usually 0013243Fh; 512Kbytes 8bit-bus)
  1F801014h 4    SPU_DELAY   Delay/Size (usually 200931E1h)
  1F801018h 4    CDROM_DELAY Delay/Size (usually 00020843h or 00020943h)
  1F80101Ch 4    Expansion 2 Delay/Size (usually 00070777h; 128-bytes 8bit-bus)
  1F801020h 4    COM_DELAY / COMMON_DELAY (00031125h or 0000132Ch or 00001325h)
```
Peripheral I/O Ports
```
  1F801040h 1/4  JOY_DATA Joypad/Memory Card Data (R/W)
  1F801044h 4    JOY_STAT Joypad/Memory Card Status (R)
  1F801048h 2    JOY_MODE Joypad/Memory Card Mode (R/W)
  1F80104Ah 2    JOY_CTRL Joypad/Memory Card Control (R/W)
  1F80104Eh 2    JOY_BAUD Joypad/Memory Card Baudrate (R/W)
  1F801050h 1/4  SIO_DATA Serial Port Data (R/W)
  1F801054h 4    SIO_STAT Serial Port Status (R)
  1F801058h 2    SIO_MODE Serial Port Mode (R/W)
  1F80105Ah 2    SIO_CTRL Serial Port Control (R/W)
  1F80105Ch 2    SIO_MISC Serial Port Internal Register (R/W)
  1F80105Eh 2    SIO_BAUD Serial Port Baudrate (R/W)
```
Memory Control 2
```
  1F801060h 4/2  RAM_SIZE (usually 00000B88h; 2MB RAM mirrored in first 8MB)
```
Interrupt Control
```
  1F801070h 2    I_STAT - Interrupt status register
  1F801074h 2    I_MASK - Interrupt mask register
```
DMA Registers
```
  1F80108xh      DMA0 channel 0 - MDECin
  1F80109xh      DMA1 channel 1 - MDECout
  1F8010Axh      DMA2 channel 2 - GPU (lists + image data)
  1F8010Bxh      DMA3 channel 3 - CDROM
  1F8010Cxh      DMA4 channel 4 - SPU
  1F8010Dxh      DMA5 channel 5 - PIO (Expansion Port)
  1F8010Exh      DMA6 channel 6 - OTC (reverse clear OT) (GPU related)
  1F8010F0h      DPCR - DMA Control register
  1F8010F4h      DICR - DMA Interrupt register
  1F8010F8h      unknown
  1F8010FCh      unknown
```
Timers (aka Root counters)
```
  1F80110xh      Timer 0 Dotclock
  1F80111xh      Timer 1 Horizontal Retrace
  1F80112xh      Timer 2 1/8 system clock
```
CDROM Registers (Address.Read/Write.Index)
```
  1F801800h.x.x   1   CD Index/Status Register (Bit0-1 R/W, Bit2-7 Read Only)
  1F801801h.R.x   1   CD Response Fifo (R) (usually with Index1)
  1F801802h.R.x   1/2 CD Data Fifo - 8bit/16bit (R) (usually with Index0..1)
  1F801803h.R.0   1   CD Interrupt Enable Register (R)
  1F801803h.R.1   1   CD Interrupt Flag Register (R/W)
  1F801803h.R.2   1   CD Interrupt Enable Register (R) (Mirror)
  1F801803h.R.3   1   CD Interrupt Flag Register (R/W) (Mirror)
  1F801801h.W.0   1   CD Command Register (W)
  1F801802h.W.0   1   CD Parameter Fifo (W)
  1F801803h.W.0   1   CD Request Register (W)
  1F801801h.W.1   1   Unknown/unused
  1F801802h.W.1   1   CD Interrupt Enable Register (W)
  1F801803h.W.1   1   CD Interrupt Flag Register (R/W)
  1F801801h.W.2   1   Unknown/unused
  1F801802h.W.2   1   CD Audio Volume for Left-CD-Out to Left-SPU-Input (W)
  1F801803h.W.2   1   CD Audio Volume for Left-CD-Out to Right-SPU-Input (W)
  1F801801h.W.3   1   CD Audio Volume for Right-CD-Out to Right-SPU-Input (W)
  1F801802h.W.3   1   CD Audio Volume for Right-CD-Out to Left-SPU-Input (W)
  1F801803h.W.3   1   CD Audio Volume Apply Changes (by writing bit5=1)
```
GPU Registers
```
  1F801810h.Write 4   GP0 Send GP0 Commands/Packets (Rendering and VRAM Access)
  1F801814h.Write 4   GP1 Send GP1 Commands (Display Control)
  1F801810h.Read  4   GPUREAD Read responses to GP0(C0h) and GP1(10h) commands
  1F801814h.Read  4   GPUSTAT Read GPU Status Register
```
MDEC Registers
```
  1F801820h.Write 4   MDEC Command/Parameter Register (W)
  1F801820h.Read  4   MDEC Data/Response Register (R)
  1F801824h.Write 4   MDEC Control/Reset Register (W)
  1F801824h.Read  4   MDEC Status Register (R)
```
SPU Voice 0..23 Registers
```
  1F801C00h+N*10h 4   Voice 0..23 Volume Left/Right
  1F801C04h+N*10h 2   Voice 0..23 ADPCM Sample Rate
```

```
  1F801C06h+N*10h 2   Voice 0..23 ADPCM Start Address
  1F801C08h+N*10h 4   Voice 0..23 ADSR Attack/Decay/Sustain/Release
  1F801C0Ch+N*10h 2   Voice 0..23 ADSR Current Volume
  1F801C0Eh+N*10h 2   Voice 0..23 ADPCM Repeat Address
```

SPU Control Registers

```
  1F801D80h 4  Main Volume Left/Right
  1F801D84h 4  Reverb Output Volume Left/Right
  1F801D88h 4  Voice 0..23 Key ON (Start Attack/Decay/Sustain) (W)
  1F801D8Ch 4  Voice 0..23 Key OFF (Start Release) (W)
  1F801D90h 4  Voice 0..23 Channel FM (pitch lfo) mode (R/W)
  1F801D94h 4  Voice 0..23 Channel Noise mode (R/W)
  1F801D98h 4  Voice 0..23 Channel Reverb mode (R/W)
  1F801D9Ch 4  Voice 0..23 Channel ON/OFF (status) (R)
  1F801DA0h 2  Unknown? (R) or (W)
  1F801DA2h 2  Sound RAM Reverb Work Area Start Address
  1F801DA4h 2  Sound RAM IRQ Address
  1F801DA6h 2  Sound RAM Data Transfer Address
  1F801DA8h 2  Sound RAM Data Transfer Fifo
  1F801DAAh 2  SPU Control Register (SPUCNT)
  1F801DACh 2  Sound RAM Data Transfer Control
  1F801DAEh 2  SPU Status Register (SPUSTAT) (R)
  1F801DB0h 4  CD Volume Left/Right
  1F801DB4h 4  Extern Volume Left/Right
  1F801DB8h 4  Current Main Volume Left/Right
  1F801DBCh 4  Unknown? (R/W)
```

SPU Reverb Configuration Area

```
  1F801DC0h 2  dAPF1  Reverb APF Offset 1
  1F801DC2h 2  dAPF2  Reverb APF Offset 2
  1F801DC4h 2  vIIR   Reverb Reflection Volume 1
  1F801DC6h 2  vCOMB1 Reverb Comb Volume 1
  1F801DC8h 2  vCOMB2 Reverb Comb Volume 2
  1F801DCAh 2  vCOMB3 Reverb Comb Volume 3
  1F801DCCh 2  vCOMB4 Reverb Comb Volume 4
  1F801DCEh 2  vWALL  Reverb Reflection Volume 2
  1F801DD0h 2  vAPF1  Reverb APF Volume 1
  1F801DD2h 2  vAPF2  Reverb APF Volume 2
  1F801DD4h 4  mSAME  Reverb Same Side Reflection Address 1 Left/Right
  1F801DD8h 4  mCOMB1 Reverb Comb Address 1 Left/Right
  1F801DDCh 4  mCOMB2 Reverb Comb Address 2 Left/Right
  1F801DE0h 4  dSAME  Reverb Same Side Reflection Address 2 Left/Right
  1F801DE4h 4  mDIFF  Reverb Different Side Reflection Address 1 Left/Right
  1F801DE8h 4  mCOMB3 Reverb Comb Address 3 Left/Right
  1F801DECh 4  mCOMB4 Reverb Comb Address 4 Left/Right
  1F801DF0h 4  dDIFF  Reverb Different Side Reflection Address 2 Left/Right
  1F801DF4h 4  mAPF1  Reverb APF Address 1 Left/Right
  1F801DF8h 4  mAPF2  Reverb APF Address 2 Left/Right
  1F801DFCh 4  vIN    Reverb Input Volume Left/Right
```

SPU Internal Registers

```
  1F801E00h+N*04h  4 Voice 0..23 Current Volume Left/Right
  1F801E60h      20h Unknown? (R/W)
  1F801E80h     180h Unknown? (Read: FFh-filled) (Unused or Write only?)
```

Expansion Region 2 (default 128 bytes, max 8 KBytes)

```
  1F802000h      80h Expansion Region (8bit data bus, crashes on 16bit access?)
```

Expansion Region 2 - Dual Serial Port (for TTY Debug Terminal)

```
  1F802020h/1st    DUART Mode Register 1.A (R/W)
  1F802020h/2nd    DUART Mode Register 2.A (R/W)
  1F802021h/Read   DUART Status Register A (R)
  1F802021h/Write  DUART Clock Select Register A (W)
  1F802022h/Read   DUART Toggle Baud Rate Generator Test Mode (Read=Strobe)
  1F802022h/Write  DUART Command Register A (W)
  1F802023h/Read   DUART Rx Holding Register A (FIFO) (R)
  1F802023h/Write  DUART Tx Holding Register A (W)
  1F802024h/Read   DUART Input Port Change Register (R)
  1F802024h/Write  DUART Aux. Control Register (W)
  1F802025h/Read   DUART Interrupt Status Register (R)
  1F802025h/Write  DUART Interrupt Mask Register (W)
  1F802026h/Read   DUART Counter/Timer Current Value, Upper/Bit15-8 (R)
  1F802026h/Write  DUART Counter/Timer Reload Value,  Upper/Bit15-8 (W)
  1F802027h/Read   DUART Counter/Timer Current Value, Lower/Bit7-0 (R)
  1F802027h/Write  DUART Counter/Timer Reload Value,  Lower/Bit7-0 (W)
  1F802028h/1st    DUART Mode Register 1.B (R/W)
  1F802028h/2nd    DUART Mode Register 2.B (R/W)
  1F802029h/Read   DUART Status Register B (R)
  1F802029h/Write  DUART Clock Select Register B (W)
  1F80202Ah/Read   DUART Toggle 1X/16X Test Mode (Read=Strobe)
  1F80202Ah/Write  DUART Command Register B (W)
  1F80202Bh/Read   DUART Rx Holding Register B (FIFO) (R)
  1F80202Bh/Write  DUART Tx Holding Register B (W)
  1F80202Ch/None   DUART Reserved Register (neither R nor W)
  1F80202Dh/Read   DUART Input Port (R)
  1F80202Dh/Write  DUART Output Port Configuration Register (W)
  1F80202Eh/Read   DUART Start Counter Command (Read=Strobe)
  1F80202Eh/Write  DUART Set Output Port Bits Command (Set means Out=LOW)
  1F80202Fh/Read   DUART Stop Counter Command (Read=Strobe)
  1F80202Fh/Write  DUART Reset Output Port Bits Command (Reset means Out=HIGH)
```

Expansion Region 2 - Int/Dip/Post

```
  1F802000h 1 DTL-H2000: ATCONS STAT (R)
  1F802002h 1 DTL-H2000: ATCONS DATA (R and W)
  1F802004h 2 DTL-H2000: Whatever 16bit data ?
  1F802030h 1/4 DTL-H2000: Secondary IRQ10 Flags
  1F802032h 1 DTL-H2000: Whatever IRQ Control ?
  1F802040h 1 DTL-H2000: Bootmode "Dip switches" (R)
  1F802041h 1 PSX: POST (external 7 segment display, indicate BIOS boot status)
  1F802042h 1 DTL-H2000: POST/LED (similar to POST) (other addr, 2-digit wide)   1F802070h 1 PS2: POST2 (similar to POST, but PS2 BIOS uses this addre
```

Expansion Region 2 - Nocash Emulation Expansion

```
  1F802060h Emu-Expansion ID1 "E" (R)
  1F802061h Emu-Expansion ID2 "X" (R)
  1F802062h Emu-Expansion ID3 "P" (R)
  1F802063h Emu-Expansion Version (01h) (R)
  1F802064h Emu-Expansion Enable1 "O" (R/W)
  1F802065h Emu-Expansion Enable2 "N" (R/W)
  1F802066h Emu-Expansion Halt (R)
  1F802067h Emu-Expansion Turbo Mode Flags (R/W)
```

Expansion Region 3 (default 1 byte, max 2 MBytes)
```
1FA00000h - Not used by BIOS or any PSX games
1FA00000h - POST3 (similar to POST, but PS2 BIOS uses this address)
```
BIOS Region (default 512 Kbytes, max 4 MBytes)
```
1FC00000h 80000h   BIOS ROM (512Kbytes) (Reset Entrypoint at BFC00000h)
```
Memory Control 3 (Cache Control)
```
FFFE0130h 4        Cache Control
```

Coprocessor Registers
```
COP0 System Control Coprocessor          - 32 registers (not all used)
COP1 N/A
COP2 Geometry Transformation Engine (GTE) - 64 registers (most are used)
COP3 N/A
```

Graphics Processing Unit (GPU)

The GPU can render Polygons, Lines, or Rectangles to the Drawing Buffer, and sends the Display Buffer to the Television Set. Polygons are useful for 3D graphics (or rotated/scaled 2D graphics), Rectangles are useful for 2D graphics and Text output.

GPU I/O Ports, DMA Channels, Commands, VRAM
GPU Render Polygon Commands
GPU Render Line Commands
GPU Render Rectangle Commands
GPU Rendering Attributes
GPU Memory Transfer Commands
GPU Other Commands
GPU Display Control Commands (GP1)
GPU Status Register
GPU Depth Ordering
GPU Video Memory (VRAM)
GPU Texture Caching
GPU Timings
GPU (MISC)

GPU I/O Ports, DMA Channels, Commands, VRAM

GPU I/O Ports (1F801810h and 1F801814h in Read/Write Directions)
```
Port            Name    Expl.
1F801810h-Write GP0     Send GP0 Commands/Packets (Rendering and VRAM Access)
1F801814h-Write GP1     Send GP1 Commands (Display Control) (and DMA Control)
1F801810h-Read  GPUREAD Receive responses to GP0(C0h) and GP1(10h) commands
1F801814h-Read  GPUSTAT Receive GPU Status Register
```
It (=GP0 only?) has a 64-byte (16-word) command FIFO buffer.
Optionally, Port 1F801810h (Read/Write) can be also accessed via DMA2.

GPU Timers / Synchronization
Most of the Timers are bound to GPU timings, see
Timers
Interrupts

GPU-related DMA Channels (DMA2 and DMA6)
```
Channel                  Recommended for
DMA2 in Linked Mode    - Sending rendering commands  ;GP0(20h..7Fh,E1h..E6h)
DMA2 in Continous Mode - VRAM transfers to/from GPU  ;GP0(A0h,C0h)
DMA6                   - Initializing the Link List  ;Main RAM
```
Note: Before using DMA2, set up the DMA Direction in GP1(04h).
DMA2 is equivalent to accessing Port 1F801810h (GP0/GPUREAD) by software.
DMA6 just initializes data in Main RAM (not physically connected to the GPU).

GPU Command Summary
Commands/Packets consist of a 8bit command number (MSBs) and a 24bit parameter (LSBs), which are written as 32bit value to GP0 or GP1.
```
GP0(00h)                 - Nop?
GP0(01h,02h,80h,A0h,C0h) - Direct VRAM Access
GP0(03h)                 - Unknown (does take up FIFO space!!!)
GP0(1Fh)                 - Interrupt Request (IRQ1)
GP0(20h..3Fh)            - Render Polygons
GP0(40h..5Fh)            - Render Lines
GP0(60h..7Fh)            - Render Rectangles
GP0(E1h..E6h)            - Rendering Attributes
GP1(00h..09h,10h,20h)    - Display Control (these via GP1 register)
```
Some GP0 commands require additional parameters, which are written (following to the command) as further 32bit values to GP0. The execution of the command starts when all parameters have been received (or, in case of Polygon/Line commands, when the first 3/2 vertices have been received).

VRAM Overview / VRAM Addressing
VRAM is 1MByte (not mapped to the CPU bus) (it can be read/written only via I/O or DMA). The memory is used for:
```
Framebuffer(s)      ;Usually 2 buffers (Drawing Area, and Display Area)
Texture Page(s)     ;Required when using Textures
Texture Palette(s)  ;Required when using 4bit/8bit Textures
```
The 1MByte VRAM is organized as 512 lines of 2048 bytes. It is accessed via coordinates, ranging from (0,0)=Upper-Left to (N,511)=Lower-Right.
```
Unit  = 4bit  8bit  16bit  24bit   Halfwords   | Unit   = Lines
Width = 4096  2048  1024   682.66  1024        | Height = 512
```
The horizontal coordinates are addressing memory in 4bit/8bit/16bit/24bit/halfword units (depending on what data formats you are using) (or a mixup thereof, eg. a halfword-base address, plus a 4bit texture coordinate).

GPU Render Polygon Commands

GP0(20h) - Monochrome three-point polygon, opaque

GP0(22h) - Monochrome three-point polygon, semi-transparent
GP0(28h) - Monochrome four-point polygon, opaque
GP0(2Ah) - Monochrome four-point polygon, semi-transparent

```
  1st  Color+Command     (CcBbGgRrh)
  2nd  Vertex1           (YyyyXxxxh)
  3rd  Vertex2           (YyyyXxxxh)
  4th  Vertex3           (YyyyXxxxh)
 (5th) Vertex4           (YyyyXxxxh) (if any)
```

GP0(24h) - Textured three-point polygon, opaque, texture-blending
GP0(25h) - Textured three-point polygon, opaque, raw-texture
GP0(26h) - Textured three-point polygon, semi-transparent, texture-blending
GP0(27h) - Textured three-point polygon, semi-transparent, raw-texture
GP0(2Ch) - Textured four-point polygon, opaque, texture-blending
GP0(2Dh) - Textured four-point polygon, opaque, raw-texture
GP0(2Eh) - Textured four-point polygon, semi-transparent, texture-blending
GP0(2Fh) - Textured four-point polygon, semi-transparent, raw-texture

```
  1st  Color+Command     (CcBbGgRrh) (color is ignored for raw-textures)
  2nd  Vertex1           (YyyyXxxxh)
  3rd  Texcoord1+Palette (ClutYyXxh)
  4th  Vertex2           (YyyyXxxxh)
  5th  Texcoord2+Texpage (PageYyXxh)
  6th  Vertex3           (YyyyXxxxh)
  7th  Texcoord3         (0000YyXxh)
 (8th) Vertex4           (YyyyXxxxh) (if any)
 (9th) Texcoord4         (0000YyXxh) (if any)
```

GP0(30h) - Shaded three-point polygon, opaque
GP0(32h) - Shaded three-point polygon, semi-transparent
GP0(38h) - Shaded four-point polygon, opaque
GP0(3Ah) - Shaded four-point polygon, semi-transparent

```
  1st  Color1+Command    (CcBbGgRrh)
  2nd  Vertex1           (YyyyXxxxh)
  3rd  Color2            (00BbGgRrh)
  4th  Vertex2           (YyyyXxxxh)
  5th  Color3            (00BbGgRrh)
  6th  Vertex3           (YyyyXxxxh)
 (7th) Color4            (00BbGgRrh) (if any)
 (8th) Vertex4           (YyyyXxxxh) (if any)
```

GP0(34h) - Shaded Textured three-point polygon, opaque, texture-blending
GP0(36h) - Shaded Textured three-point polygon, semi-transparent, tex-blend
GP0(3Ch) - Shaded Textured four-point polygon, opaque, texture-blending
GP0(3Eh) - Shaded Textured four-point polygon, semi-transparent, tex-blend

```
  1st  Color1+Command    (CcBbGgRrh)
  2nd  Vertex1           (YyyyXxxxh)
  3rd  Texcoord1+Palette (ClutYyXxh)
  4th  Color2            (00BbGgRrh)
  5th  Vertex2           (YyyyXxxxh)
  6th  Texcoord2+Texpage (PageYyXxh)
  7th  Color3            (00BbGgRrh)
  8th  Vertex3           (YyyyXxxxh)
  9th  Texcoord3         (0000YyXxh)
 (10th) Color4           (00BbGgRrh) (if any)
 (11th) Vertex4          (YyyyXxxxh) (if any)
 (12th) Texcoord4        (0000YyXxh) (if any)
```

GP0(35h,37h,3Dh,3Fh) - Undocumented/Nonsense (Raw Texture + UNUSED shading)
These are undocumented inefficient nonsense commands: Parameters are same as for GP0(34h,36h,3Ch,3Eh), ie. with colors for all vertices, but without actually using that colors. Instead, the commands are rendering raw textures without blending.
In other words, the commands have same function as GP0(25h,27h,2Dh,2Fh), but with additional/unused parameters (and possible additional/unused internal gouraud shading calculations).
For whatever reason, Castlevania is actually using these nonsense commands, namely GP0(3Dh) and GP0(3Fh).

GP0(21h,23h,29h,2Bh,31h,33h,39h,3Bh) - Undocumented/Nonsense
These commands have texture-blending disabled, which is nonsense because they are using untextured polygons anyways, ie. they are probably same as GP0(20h,22h,28h,2Ah,30h,32h,38h,3Ah).

Notes
Polygons are displayed up to <excluding> their lower-right coordinates.
Four-point polygons are internally processed as two Three-point polygons, the first consisting of Vertices 1,2,3, and the second of Vertices 2,3,4.
Within the Three-point polygons, the ordering of the vertices is don't care at the GPU side (a front-back check, based on clockwise or anti-clockwise ordering, can be implemented at the GTE side).
Dither enable (in Texpage command) affects ONLY polygons that do use Gouraud Shading or Texture Blending.

GPU Render Line Commands

GP0(40h) - Monochrome line, opaque
GP0(42h) - Monochrome line, semi-transparent
GP0(48h) - Monochrome Poly-line, opaque
GP0(4Ah) - Monochrome Poly-line, semi-transparent

```
  1st   Color+Command     (CcBbGgRrh)
  2nd   Vertex1           (YyyyXxxxh)
  3rd   Vertex2           (YyyyXxxxh)
 (...)  VertexN           (YyyyXxxxh) (poly-line only)
 (Last) Termination Code  (55555555h) (poly-line only)
```

GP0(50h) - Shaded line, opaque
GP0(52h) - Shaded line, semi-transparent
GP0(58h) - Shaded Poly-line, opaque
GP0(5Ah) - Shaded Poly-line, semi-transparent

```
  1st   Color1+Command    (CcBbGgRrh)
```

```
  2nd   Vertex1           (YyyyXxxxh)
  3rd   Color2            (00BbGgRrh)
  4th   Vertex2           (YyyyXxxxh)
 (...)  ColorN            (00BbGgRrh) (poly-line only)
 (...)  VertexN           (YyyyXxxxh) (poly-line only)
 (Last) Termination Code  (55555555h) (poly-line only)
```

Note
Lines are displayed up to <including> their lower-right coordinates (ie. unlike as for polygons, the lower-right coordinate is not excluded).
If dithering is enabled (via Texpage command), then both monochrome and shaded lines are drawn with dithering (this differs from monochrome polygons and monochrome rectangles).

Termination Codes for Poly-Lines (aka Linestrips)
The termination code should be usually 55555555h, however, Wild Arms 2 uses 50005000h (unknown which exact bits/values are relevant there).

Wire-Frame
Poly-Lines can be used (among others) to create Wire-Frame polygons (by setting the last Vertex equal to Vertex 1).

GPU Render Rectangle Commands

Rectangles are drawn much faster than polygons. Unlike for polygons, gouroud shading is not possible, dithering isn't applied, the rectangle must forcefully have horizontal and vertical edges, textures cannot be rotated or scaled, and, of course, the GPU does render Rectangles at once (without splitting them into triangles).

GP0(60h) - Monochrome Rectangle (variable size) (opaque)
GP0(62h) - Monochrome Rectangle (variable size) (semi-transparent)
GP0(68h) - Monochrome Rectangle (1x1) (Dot) (opaque)
GP0(6Ah) - Monochrome Rectangle (1x1) (Dot) (semi-transparent)
GP0(70h) - Monochrome Rectangle (8x8) (opaque)
GP0(72h) - Monochrome Rectangle (8x8) (semi-transparent)
GP0(78h) - Monochrome Rectangle (16x16) (opaque)
GP0(7Ah) - Monochrome Rectangle (16x16) (semi-transparent)

```
  1st  Color+Command     (CcBbGgRrh)
  2nd  Vertex            (YyyyXxxxh)
 (3rd) Width+Height      (YsizXsizh) (variable size only) (max 1023x511)
```

GP0(64h) - Textured Rectangle, variable size, opaque, texture-blending
GP0(65h) - Textured Rectangle, variable size, opaque, raw-texture
GP0(66h) - Textured Rectangle, variable size, semi-transp, texture-blending
GP0(67h) - Textured Rectangle, variable size, semi-transp, raw-texture
GP0(6Ch) - Textured Rectangle, 1x1 (nonsense), opaque, texture-blending
GP0(6Dh) - Textured Rectangle, 1x1 (nonsense), opaque, raw-texture
GP0(6Eh) - Textured Rectangle, 1x1 (nonsense), semi-transp, texture-blending
GP0(6Fh) - Textured Rectangle, 1x1 (nonsense), semi-transp, raw-texture
GP0(74h) - Textured Rectangle, 8x8, opaque, texture-blending
GP0(75h) - Textured Rectangle, 8x8, opaque, raw-texture
GP0(76h) - Textured Rectangle, 8x8, semi-transparent, texture-blending
GP0(77h) - Textured Rectangle, 8x8, semi-transparent, raw-texture
GP0(7Ch) - Textured Rectangle, 16x16, opaque, texture-blending
GP0(7Dh) - Textured Rectangle, 16x16, opaque, raw-texture
GP0(7Eh) - Textured Rectangle, 16x16, semi-transparent, texture-blending
GP0(7Fh) - Textured Rectangle, 16x16, semi-transparent, raw-texture

```
  1st  Color+Command     (CcBbGgRrh) (color is ignored for raw-textures)
  2nd  Vertex            (YyyyXxxxh) (upper-left edge of the rectangle)
  3rd  Texcoord+Palette  (ClutYyXxh) (for 4bpp Textures Xxh must be even!)
 (4th) Width+Height      (YsizXsizh) (variable size only) (max 1023x511)
```

Unlike for Textured-Polygons, the "Texpage" must be set up separately for Rectangles, via GP0(E1h). Width and Height can be up to 1023x511, however, the maximum size of the texture window is 256x256 (so the source data will be repeated when trying to use sizes larger than 256x256).

Texture Origin and X/Y-Flip
Vertex & Texcoord specify the upper-left edge of the rectangle. And, normally, screen coords and texture coords are both incremented during rendering the rectangle pixels.
Optionally, X/Y-Flip bits can be set in Texpage.Bit12/13, these bits cause the texture coordinates to be decremented (instead of incremented). The X/Y-Flip bits do affect only Rectangles (not Polygons, nor VRAM Transfers).
Caution: Reportedly, the X/Y-Flip feature isn't supported on old PSX consoles (unknown which ones exactly, maybe such with PU-7 mainboards, and unknown how to detect flipping support; except of course by reading VRAM).

Note
There are also two VRAM Transfer commands which work similar to GP0(60h) and GP0(65h). Eventually, that commands might be even faster... although not sure if they do use the Texture Cache?
The difference is that VRAM Transfers do not clip to the Drawig Area boundary, do not support fully-transparent nor semi-transparent texture pixels, and do not convert color depths (eg. without 4bit texture to 16bit framebuffer conversion).

GPU Rendering Attributes

Vertex (Parameter for Polygon, Line, Rectangle commands)

```
  0-10   X-coordinate (signed, -1024..+1023)
  11-15  Not used (usually sign-extension, but ignored by hardware)
  16-26  Y-coordinate (signed, -1024..+1023)
  26-31  Not used (usually sign-extension, but ignored by hardware)
```

Size Restriction: The maximum distance between two vertices is 1023 horizontally, and 511 vertically. Polygons and lines that are exceeding that dimensions are NOT rendered. For example, a line from Y1=-300 to Y2=+300 is NOT rendered, a line from Y1=-100 to Y2=+400 is rendered (as far as it is within the drawing area).
If portions of the polygon/line/rectangle are located outside of the drawing area, then the hardware renders only the portion that is inside of the drawing area. Not sure if the hardware is skipping all clipped pixels at once (within a single clock cycle), or if it's (slowly) processing them pixel by pixel?

Color Attribute (Parameter for all Rendering commands, except Raw Texture)

```
  0-7    Red   (0..FFh)
```

```
  8-15   Green (0..FFh)
  16-23  Blue  (0..FFh)
  24-31  Command (in first paramter) (don't care in further parameters)
```
Caution: For untextured graphics, 8bit RGB values of FFh are brightest. However, for texture blending, 8bit values of 80h are brightest (values 81h..FFh are "brighter than bright" allowing to make textures about twice as bright as than they were originially stored in memory; of course the results can't exceed the maximum brightness, ie. the 5bit values written to the framebuffer are saturated to max 1Fh).

Texpage Attribute (Parameter for Textured-Polygons commands)
```
  0-8    Same as GP0(E1h).Bit0-8 (see there)
  9-10   Unused (does NOT change GP0(E1h).Bit9-10)
  11     Same as GP0(E1h).Bit11  (see there)
  12-13  Unused (does NOT change GP0(E1h).Bit12-13)
  14-15  Unused (should be 0)
```
This attribute is used in all Textured-Polygons commands.

Clut Attribute (Color Lookup Table, aka Palette)
This attribute is used in all Textured Polygon/Rectangle commands. Of course, it's relevant only for 4bit/8bit textures (don't care for 15bit textures).
```
  0-5      X coordinate X/16  (ie. in 16-halfword steps)
  6-14     Y coordinate 0-511 (ie. in 1-line steps)
  15       Unknown/unused (should be 0)
```
Specifies the location of the CLUT data within VRAM.

GP0(E1h) - Draw Mode setting (aka "Texpage")
```
  0-3   Texture page X Base   (N*64) (ie. in 64-halfword steps)    ;GPUSTAT.0-3
  4     Texture page Y Base   (N*256) (ie. 0 or 256)               ;GPUSTAT.4
  5-6   Semi Transparency     (0=B/2+F/2, 1=B+F, 2=B-F, 3=B+F/4)   ;GPUSTAT.5-6
  7-8   Texture page colors   (0=4bit, 1=8bit, 2=15bit, 3=Reserved);GPUSTAT.7-8
  9     Dither 24bit to 15bit (0=Off/strip LSBs, 1=Dither Enabled) ;GPUSTAT.9
  10    Drawing to display area (0=Prohibited, 1=Allowed)          ;GPUSTAT.10
  11    Texture Disable (0=Normal, 1=Disable if GP1(09h).Bit0=1)   ;GPUSTAT.15
  12    Textured Rectangle X-Flip   (BIOS does set this bit on power-up...?)
  13    Textured Rectangle Y-Flip   (BIOS does set it equal to GPUSTAT.13...?)
  14-23 Not used (should be 0)
  24-31 Command  (E1h)
```
The GP0(E1h) command is required only for Lines, Rectangle, and Untextured-Polygons (for Textured-Polygons, the data is specified in form of the Texpage attribute; except that, Bit9-10 can be changed only via GP0(E1h), not via the Texpage attribute).
Texture page colors setting 3 (reserved) is same as setting 2 (15bit).
Note: GP0(00h) seems to be often inserted between Texpage and Rectangle commands, maybe it acts as a NOP, which may be required between that commands, for timing reasons...?

GP0(E2h) - Texture Window setting
```
  0-4    Texture window Mask X   (in 8 pixel steps)
  5-9    Texture window Mask Y   (in 8 pixel steps)
  10-14  Texture window Offset X (in 8 pixel steps)
  15-19  Texture window Offset Y (in 8 pixel steps)
  20-23  Not used (zero)
  24-31  Command  (E2h)
```
Mask specifies the bits that are to be manipulated, and Offset contains the new values for these bits, ie. texture X/Y coordinates are adjusted as so:
```
  Texcoord = (Texcoord AND (NOT (Mask*8))) OR ((Offset AND Mask)*8)
```
The area within a texture window is repeated throughout the texture page. The data is not actually stored all over the texture page but the GPU reads the repeated patterns as if they were there.

GP0(E3h) - Set Drawing Area top left (X1,Y1)
GP0(E4h) - Set Drawing Area bottom right (X2,Y2)
```
  0-9    X-coordinate (0..1023)
  10-19  Y-coordinate (0..1023) (really 10bit, not 9bit) (should be 512 max)
  20-23  Not used (zero)
  24-31  Command  (Exh)
```
Sets the drawing area corners. The Render commands GP0(20h..7Fh) are automatically clipping any pixels that are outside of this region.

GP0(E5h) - Set Drawing Offset (X,Y)
```
  0-10   X-offset (-1024..+1023) (usually within X1,X2 of Drawing Area)
  11-21  Y-offset (-1024..+1023) (usually within Y1,Y2 of Drawing Area)
  22-23  Not used (zero)
  24-31  Command  (E5h)
```
If you have configured the GTE to produce vertices with coordinate "0,0" being located in the center of the drawing area, then the Drawing Offset must be "X1+(X2-X1)/2, Y1+(Y2-Y1)/2". Or, if coordinate "0,0" shall be the upper-left of the Drawing Area, then Drawing Offset should be "X1,Y1". Where X1,Y1,X2,Y2 are the values defined with GP0(E3h-E4h).

GP0(E6h) - Mask Bit Setting
```
  0     Set mask while drawing (0=TextureBit15, 1=ForceBit15=1)   ;GPUSTAT.11
  1     Check mask before draw (0=Draw Always, 1=Draw if Bit15=0) ;GPUSTAT.12
  2-23  Not used (zero)
  24-31 Command  (E6h)
```
When bit0 is off, the upper bit of the data written to the framebuffer is equal to bit15 of the texture color (ie. it is set for colors that are marked as "semi-transparent") (for untextured polygons, bit15 is set to zero).
When bit1 is on, any (old) pixels in the framebuffer with bit15=1 are write-protected, and cannot be overwritten by (new) rendering commands.
The mask setting affects all rendering commands, as well as CPU-to-VRAM and VRAM-to-VRAM transfer commands (where it acts on the separate halfwords, ie. as for 15bit textures). However, Mask does NOT affect the Fill-VRAM command.

Note
GP0(E3h..E5h) do not take up space in the FIFO, so they are probably executed immediately (even if there're still other commands in the FIFO). Best use them only if you are sure that the FIFO is empty (otherwise the new Drawing Area settings might accidently affect older Rendering Commands in the FIFO).

GPU Memory Transfer Commands

GP0(01h) - Clear Cache
```
  1st  Command           (Cc000000h)
```
"Seems to be the same as the GP1 command." Uh, which GP1 command?
Before using GP(A0h) or GP(C0h) one should reportedly send:
Clear Cache (01000000h)
"Reset command buffer (write to GP1 or GP0)" Uh? Bullshit.

However, there <may> be some situations in which it is neccessary to flush the texture cache.

GP0(02h) - Fill Rectangle in VRAM
```
  1st  Color+Command     (CcBbGgRrh)  ;24bit RGB value (see note)
  2nd  Top Left Corner   (YyyyXxxxh)  ;Xpos counted in halfwords, steps of 10h
  3rd  Width+Height      (YsizXsizh)  ;Xsiz counted in halfwords, steps of 10h
```
Fills the area in the frame buffer with the value in RGB. Horizontally the filling is done in 16-pixel (32-bytes) units (see below masking/rounding).
The "Color" parameter is a 24bit RGB value, however, the actual fill data is 16bit: The hardware automatically converts the 24bit RGB value to 15bit RGB (with bit15=0).
Fill is NOT affected by the Mask settings (acts as if Mask.Bit0,1 are both zero).

GP0(80h) - Copy Rectangle (VRAM to VRAM)
```
  1st  Command           (Cc000000h)
  2nd  Source Coord      (YyyyXxxxh)  ;Xpos counted in halfwords
  3rd  Destination Coord (YyyyXxxxh)  ;Xpos counted in halfwords
  4th  Width+Height      (YsizXsizh)  ;Xsiz counted in halfwords
```
Copys data within framebuffer. The transfer is affected by Mask setting.

GP0(A0h) - Copy Rectangle (CPU to VRAM)
```
  1st  Command           (Cc000000h)
  2nd  Destination Coord (YyyyXxxxh)  ;Xpos counted in halfwords
  3rd  Width+Height      (YsizXsizh)  ;Xsiz counted in halfwords
  ...  Data              (...)      <--- usually transferred via DMA
```
Transfers data from CPU to frame buffer. If the number of halfwords to be sent is odd, an extra halfword should be sent (packets consist of 32bit units). The transfer is affected by Mask setting.

GP0(C0h) - Copy Rectangle (VRAM to CPU)
```
  1st  Command           (Cc000000h) ;\
  2nd  Source Coord      (YyyyXxxxh) ; write to GP0 port (as usually)
  3rd  Width+Height      (YsizXsizh) ;/
  ...  Data              (...)       ;<--- read from GPUREAD port (or via DMA)
```
Transfers data from frame buffer to CPU. Wait for bit27 of the status register to be set before reading the image data. When the number of halfwords is odd, an extra halfword is read at the end (packets consist of 32bit units).

Masking and Rounding for FILL Command parameters
```
  Xpos=(Xpos AND 3F0h)                      ;range 0..3F0h, in steps of 10h
  Ypos=(Ypos AND 1FFh)                      ;range 0..1FFh
  Xsiz=((Xsiz AND 3FFh)+0Fh) AND (NOT 0Fh)  ;range 0..400h, in steps of 10h
  Ysiz=((Ysiz AND 1FFh))                    ;range 0..1FFh
```
Fill does NOT occur when Xsiz=0 or Ysiz=0 (unlike as for Copy commands). Xsiz=400h works only indirectly: Param=400h is handled as Xsiz=0, however, Param=3F1h..3FFh is rounded-up and handled as Xsiz=400h.

Masking for COPY Commands parameters
```
  Xpos=(Xpos AND 3FFh)                      ;range 0..3FFh
  Ypos=(Ypos AND 1FFh)                      ;range 0..1FFh
  Xsiz=((Xsiz-1) AND 3FFh)+1                ;range 1..400h
  Ysiz=((Ysiz-1) AND 1FFh)+1                ;range 1..200h
```
Parameters are just clipped to 10bit/9bit range, the only special case is that Size=0 is handled as Size=max.

Notes
The coordinates for the above VRAM transfer commands are absolute framebuffer addresses (not relative to Draw Offset, and not clipped to Draw Area).
Non-DMA transfers seem to be working at any time, but GPU-DMA Transfers seem to be working ONLY during V-Blank (outside of V-Blank, portions of the data appear to be skipped, and the following words arrive at wrong addresses), unknown if it's possible to change that by whatever configuration settings...? That problem appears ONLY for continous DMA aka VRAM transfers (linked-list DMA aka Ordering Table works even outside V-Blank).

Wrapping
If the Source/Dest starting points plus the width/height value exceed the 1024x512 pixel VRAM size, then the Copy/Fill operations wrap to the opposite memory edge (without any carry-out from X to Y, nor from Y to X).

GPU Other Commands

GP0(1Fh) - Interrupt Request (IRQ1)
```
  1st  Command           (Cc000000h)                 ;GPUSTAT.24
```
Requests IRQ1. Can be acknowledged via GP1(02h). This feature is rarely used.
Note: The command is used by Blaze'n'Blade, but the game doesn't have IRQ1 enabled, and the written value (1F801810h) looks more like an I/O address, rather than like a command, so not sure if it's done intentionally, or if it is just a bug.

GP0(03h) - Unknown?
Unknown. Doesn't seem to be used by any games. Unlike the "NOP" commands, GP0(03h) does take up space in FIFO, so it is apparently not a NOP.

GP0(00h) - NOP (?)
This command doesn't take up space in the FIFO (eg. even if a VRAM-to-VRAM transfer is still busy, one can send dozens of GP0(00h) commands, without the command FIFO becoming full. So, either the command is ignored (or, if it has a function, it is executed immediately, even while the transfer is busy).
...
GP0(00h) unknown, used with parameter = 08A16Ch... or rather 08FDBCh ... the written value seems to be a bios/ram memory address, anded with 00FFFFFFh... maybe a bios bug?
GP0(00h) seems to be often inserted between Texpage and Rectangle commands, maybe it acts as a NOP, which may be required between that commands, for timing reasons...?

GP0(04h..1Eh,E0h,E7h..EFh) - Mirrors of GP0(00h) - NOP (?)
Like GP0(00h), these commands don't take up space in the FIFO. So, maybe, they are same as GP0(00h), however, the Drawing Area/Offset commands GP0(E3h..E5h) don't take up FIFO space either, so not taking up FIFO space doesn't neccessarily mean that the command has no function.

GP0(81h..9Fh) - Mirror of GP0(80h) - Copy Rectangle (VRAM to VRAM)
GP0(A1h..BFh) - Mirror of GP0(A0h) - Copy Rectangle (CPU to VRAM)
GP0(C1h..DFh) - Mirror of GP0(C0h) - Copy Rectangle (VRAM to CPU)
Mirrors.

GPU Display Control Commands (GP1)

GP1 Display Control Commands are sent by writing the 8bit Command number (MSBs), and 24bit parameter (LSBs) to Port 1F801814h. Unlike GP0 commands, GP1 commands are passed directly to the GPU (ie. they can be sent even when the FIFO is full).

GP1(00h) - Reset GPU
```
  0-23  Not used (zero)
```
Resets the GPU to the following values:
```
  GP1(01h)      ;clear fifo
  GP1(02h)      ;ack irq (0)
  GP1(03h)      ;display off (1)
  GP1(04h)      ;dma off (0)
  GP1(05h)      ;display address (0)
  GP1(06h)      ;display x1,x2 (x1=200h, x2=200h+256*10)
  GP1(07h)      ;display y1,y2 (y1=010h, y2=010h+240)
  GP1(08h)      ;display mode 320x200 NTSC (0)
  GP0(E1h..E6h) ;rendering attributes (0)
```
Accordingly, GPUSTAT becomes 14802000h. The x1,y1 values are too small, ie. the upper-left edge isn't visible. Note that GP1(09h) is NOT affected by the reset command.

GP1(01h) - Reset Command Buffer
```
  0-23  Not used (zero)
```
Resets the command buffer.

GP1(02h) - Acknowledge GPU Interrupt (IRQ1)
```
  0-23  Not used (zero)                                     ;GPUSTAT.24
```
Resets the IRQ flag in GPUSTAT.24. The flag can be set via GP0(1Fh).

GP1(03h) - Display Enable
```
  0     Display On/Off   (0=On, 1=Off)                      ;GPUSTAT.23
  1-23  Not used (zero)
```
Turns display on/off. "Note that a turned off screen still gives the flicker of NTSC on a PAL screen if NTSC mode is selected."
The "Off" settings displays a black picture (and still sends /SYNC signals to the television set). (Unknown if it still generates vblank IRQs though?)

GP1(04h) - DMA Direction / Data Request
```
  0-1  DMA Direction (0=Off, 1=FIFO, 2=CPUtoGP0, 3=GPUREADtoCPU) ;GPUSTAT.29-30
  2-23 Not used (zero)
```
Notes: Manually sending/reading data by software (non-DMA) is ALWAYS possible, regardless of the GP1(04h) setting. The GP1(04h) setting does affect the meaning of GPUSTAT.25.

Display start/end
Specifies where the display area is positioned on the screen, and how much data gets sent to the screen. The screen sizes of the display area are valid only if the horizontal/vertical start/end values are default. By changing these you can get bigger/smaller display screens. On most TV's there is some black around the edge, which can be utilised by setting the start of the screen earlier and the end later. The size of the pixels is NOT changed with these settings, the GPU simply sends more data to the screen. Some monitors/TVs have a smaller display area and the extended size might not be visible on those sets. "(Mine is capable of about 330 pixels horizontal, and 272 vertical in 320*240 mode)"

GP1(05h) - Start of Display area (in VRAM)
```
  0-9   X (0-1023)    (halfword address in VRAM)  (relative to begin of VRAM)
  10-18 Y (0-511)     (scanline number in VRAM)   (relative to begin of VRAM)
  19-23 Not used (zero)
```
Upper/left Display source address in VRAM. The size and target position on screen is set via Display Range registers; target=X1,Y2; size=(X2-X1/cycles_per_pix), (Y2-Y1).

GP1(06h) - Horizontal Display range (on Screen)
```
  0-11   X1 (260h+0)       ;12bit       ;\counted in 53.222400MHz units,
  12-23  X2 (260h+320*8)   ;12bit       ;/relative to HSYNC
```
Specifies the horizontal range within which the display area is displayed. For resolutions other than 320 pixels it may be necessary to fine adjust the value to obtain an exact match (eg. X2=X1+pixels*cycles_per_pix).
The number of displayed pixels per line is "(((X2-X1)/cycles_per_pix)+2) AND NOT 3" (ie. the hardware is rounding the width up/down to a multiple of 4 pixels). Most games are using a width equal to the horizontal resolution (ie. 256, 320, 368, 512, 640 pixels). A few games are using slightly smaller widths (probably due to programming bugs). Pandemonium 2 is using a bigger "overscan" width (ensuring an intact picture without borders even on mis-calibrated TV sets).
The 260h value is the first visible pixel on normal TV Sets, this value is used by MOST NTSC games, and SOME PAL games (see below notes on Mis-Centered PAL games).

GP1(07h) - Vertical Display range (on Screen)
```
  0-9   Y1 (NTSC=88h-(224/2), (PAL=A3h-(264/2))  ;\scanline numbers on screen,
  10-19 Y2 (NTSC=88h+(224/2), (PAL=A3h+(264/2))  ;/relative to VSYNC
  20-23 Not used (zero)
```
Specifies the vertical range within which the display area is displayed. The number of lines is Y2-Y1 (unlike as for the width, there's no rounding applied to the height). If Y2 is set to a much too large value, then the hardware stops to generate vblank interrupts (IRQ0).
The 88h/A3h values are the middle-scanlines on normal TV Sets, these values are used by MOST NTSC games, and SOME PAL games (see below notes on Mis-Centered PAL games).
The 224/264 values are for fullscreen pictures. Many NTSC games display 240 lines (overscan with hidden lines). Many PAL games display only 256 lines (underscan with black borders).

GP1(08h) - Display mode
```
  0-1   Horizontal Resolution 1     (0=256, 1=320, 2=512, 3=640) ;GPUSTAT.17-18
  2     Vertical Resolution         (0=240, 1=480, when Bit5=1)  ;GPUSTAT.19
  3     Video Mode                  (0=NTSC/60Hz, 1=PAL/50Hz)    ;GPUSTAT.20
  4     Display Area Color Depth    (0=15bit, 1=24bit)           ;GPUSTAT.21
  5     Vertical Interlace          (0=Off, 1=On)                ;GPUSTAT.22
  6     Horizontal Resolution 2     (0=256/320/512/640, 1=368)   ;GPUSTAT.16
  7     "Reverseflag"               (0=Normal, 1=Distorted)      ;GPUSTAT.14
  8-23  Not used (zero)
```
Note: Interlace must be enabled to see all lines in 480-lines mode (interlace is causing ugly flickering, so a non-interlaced low resolution image is typically having better quality than a high resolution interlaced image, a pretty bad example are the intro screens shown by the BIOS). The Display Area Color Depth does NOT affect the Drawing Area (the Drawing Area is <always> 15bit).
When the "Reverseflag" is set, the display scrolls down 2 lines or so, and colored regions are getting somehow hatched/distorted, but black and white regions are still looking okay. Don't know what that's good for? Probably relates to PAL/NTSC-Color Clock vs PSX-Dot Clock mismatches: Bit7=0 causes Flimmering errors (errors at different locations in each frame), and Bit7=1 causes Static errors (errors at same locations in all frames)?

GP1(09h) - Texture Disable
```
  0     Texture Disable (0=Normal, 1=Allow Disable via GP0(E1h).11) ;GPUSTAT.15
  1-23  Unknown (seems to have no effect)
```
This feature seems to be intended for debugging purposes (most released games do contain program code for disabling textures, but do never execute it). Also, GP1(09h) doesn't seem to be supported on all GPUs. See "GPU Versions" notes below.

GP1(0Bh) - Unknown/Internal?
```
  0-10  Unknown (GPU crashes after a while when set to 274h..7FFh)
  11-23 Unknown (seems to have no effect)
```
The register doesn't seem to be used by any games.

GP1(0Ah) - N/A
GP1(0Ch) - N/A
GP1(0Dh) - N/A
GP1(0Eh) - N/A
GP1(0Fh) - N/A
Not used?

GP1(10h) - Get GPU Info
GP1(11h..1Fh) - Mirrors of GP1(10h), Get GPU Info
After sending the command, the result can be read (immediately) from GPUREAD register (there's no NOP or other delay required) (namely GPUSTAT.Bit27 is used only for VRAM-Reads, but NOT for GPU-Info-Reads, so do not try to wait for that flag).
```
  0-3   Select Information which is to be retrieved (via following GPUREAD)
          00h-01h = Returns Nothing (old value in GPUREAD remains unchanged)
          02h     = Read Texture Window setting  ;GP0(E2h) ;20bit/MSBs=Nothing
          03h     = Read Draw area top left      ;GP0(E3h) ;20bit/MSBs=Nothing
          04h     = Read Draw area bottom right  ;GP0(E4h) ;20bit/MSBs=Nothing
          05h     = Read Draw offset             ;GP0(E5h) ;22bit
          06h     = Returns Nothing (old value in GPUREAD remains unchanged)
          07h     = Read GPU Type (usually 2) See "GPU Versions" notes below
          08h     = Unknown (Returns 00000000h)
          09h-0Fh = Returns Nothing (old value in GPUREAD remains unchanged)
  4-23  Not used (no effect / should be zero)
```
The selected data is latched in GPUREAD, the same/latched value can be read multiple times, but, the latch isn't automatically updated when changing GP0 registers.

GP1(20h) - Ancient Texture Disable
```
  0-23  Unknown (501h=Texture Enable, 504h=Texture Disable, or so?)
```
Seems to be a used only on whatever older GPUs, instead of GP1(09h). See "GPU Versions" notes below.

GP1(21h..3Fh) - N/A
Not used?

GP1(40h..FFh) - N/A (Mirrors)
Mirrors of GP1(00h..3Fh).

GPU Versions
Most or all PSX games seem to detect and support at least two slightly different GPU versions. The games are using GP1(10h) with index 7 to obtain the GPU version number, the standard version seems to be 00000002h, used in PSone's and reportedly also in PSX consoles. Not sure if/which consoles do actually have different version numbers... maybe <very> old PSX'es... or maybe the program code is only a relict for compatibility with even older early prototypes?
The differences between the versions seem to apply mainly to the Texture Disable feature, and to how many bits of the Texpage attribute are reflected to which bits of GPUSTAT.
The standard GPU version (version=2) seems to use GP1(09h)=1 to disable textures. For non-standard GPUs (version<>2), the games seem to additional check whether Texpage.Bit12 can be written to GPUSTAT.Bit12, by setting "GP0(E1h)=GPUSTAT AND 3FFFh OR 2000h", and do then eventually use GP1(20h)=504h to disable textures, or GP1(20h)=501h to enable them.
Although the games do contain program code for texture disable, that feature seems to be mainly intended for debugging purposes, and most or all released games do never actually execute that code.
Note: The Textured Rectangle X/Y-Flip feature is said to be not working on very old GPU versions.

Mis-Centered PAL Games (wrong GP1(06h)/GP1(07h) settings)
NTSC games are typically well centered (using X1=260h, and Y1/Y2=88h+/-N).
PAL games should be centered as X1=260h, and Y1/Y2=A3h+/-N) - these values would be looking well on a Philips Philetta TV Set, and do also match up with other common picture positions (eg. as used by Nintendo's SNES console).
However, most PAL games are using completely different "random" centering values (maybe caused by different developers trying to match the centering to the different TV Sets) (although it looks more as if the PAL developers just went amok: Many PAL games are even using different centerings for their Intro, Movie, and actual Game sequences).
In result, most PAL games are looking like crap when playing them on a real PSX. For PSX emulators it may be recommended to ignore the GP1(06h)/GP1(07h) centering, and instead, apply auto-centering to PAL games.
For PAL game developers, it may be recommended to add a screen centering option (as found in Tomb Raider 3, for example). Unknown if this is really required... or if X1=260h, and Y1/Y2=A3h+/-N would work fine on most or all PAL TV Sets?

GPU Status Register

1F801814h - GPUSTAT - GPU Status Register (R)
```
  0-3   Texture page X Base   (N*64)                          ;GP0(E1h).0-3
  4     Texture page Y Base   (N*256) (ie. 0 or 256)           ;GP0(E1h).4
  5-6   Semi Transparency     (0=B/2+F/2, 1=B+F, 2=B-F, 3=B+F/4)  ;GP0(E1h).5-6
  7-8   Texture page colors   (0=4bit, 1=8bit, 2=15bit, 3=Reserved)GP0(E1h).7-8
  9     Dither 24bit to 15bit (0=Off/strip LSBs, 1=Dither Enabled);GP0(E1h).9
  10    Drawing to display area (0=Prohibited, 1=Allowed)        ;GP0(E1h).10
  11    Set Mask-bit when drawing pixels (0=No, 1=Yes/Mask)      ;GP0(E6h).0
  12    Draw Pixels           (0=Always, 1=Not to Masked areas)  ;GP0(E6h).1
  13    "reserved"            (seems to be always set?)
  14    "Reverseflag"         (0=Normal, 1=Distorted)            ;GP1(08h).7
  15    Texture Disable       (0=Normal, 1=Disable Textures)     ;GP0(E1h).11
  16    Horizontal Resolution 2     (0=256/320/512/640, 1=368)   ;GP1(08h).6
  17-18 Horizontal Resolution 1     (0=256, 1=320, 2=512, 3=640) ;GP1(08h).0-1
  19    Vertical Resolution         (0=240, 1=480, when Bit22=1) ;GP1(08h).2
  20    Video Mode                  (0=NTSC/60Hz, 1=PAL/50Hz)    ;GP1(08h).3
```

```
21    Display Area Color Depth    (0=15bit, 1=24bit)           ;GP1(08h).4
22    Vertical Interlace          (0=Off, 1=On)                ;GP1(08h).5
23    Display Enable              (0=Enabled, 1=Disabled)      ;GP1(03h).0
24    Interrupt Request (IRQ1)    (0=Off, 1=IRQ)       ;GP0(1Fh)/GP1(02h)
25    DMA / Data Request, meaning depends on GP1(04h) DMA Direction:
        When GP1(04h)=0 ---> Always zero (0)
        When GP1(04h)=1 ---> FIFO State  (0=Full, 1=Not Full)
        When GP1(04h)=2 ---> Same as GPUSTAT.28
        When GP1(04h)=3 ---> Same as GPUSTAT.27
26    Ready to receive Cmd Word   (0=No, 1=Ready)  ;GP0(...) ;via GP0
27    Ready to send VRAM to CPU   (0=No, 1=Ready)  ;GP0(C0h) ;via GPUREAD
28    Ready to receive DMA Block  (0=No, 1=Ready)  ;GP0(...) ;via GP0
29-30 DMA Direction (0=Off, 1=?, 2=CPUtoGP0, 3=GPUREADtoCPU)    ;GP1(04h).0-1
31    Drawing even/odd lines in interlace mode (0=Even or Vblank, 1=Odd)
```

In 480-lines mode, bit31 changes per frame. And in 240-lines mode, the bit changes per scanline. The bit is always zero during Vblank (vertical retrace and upper/lower screen border).

Note
Further GPU status information can be retrieved via GP1(10h) and GP0(C0h).

Ready Bits
Bit28: Normally, this bit gets cleared when the command execution is busy (ie. once when the command and all of its parameters are received), however, for Polygon and Line Rendering commands, the bit gets cleared immediately after receiving the command word (ie. before receiving the vertex parameters). The bit is used as DMA request in DMA Mode 2, accordingly, the DMA would probably hang if the Polygon/Line parameters are transferred in a separate DMA block (ie. the DMA probably starts ONLY on command words).
Bit27: Gets set after sending GP0(C0h) and its parameters, and stays set until all data words are received; used as DMA request in DMA Mode 3.
Bit26: Gets set when the GPU wants to receive a command. If the bit is cleared, then the GPU does either want to receive data, or it is busy with a command execution (and doesn't want to receive anything).
Bit25: This is the DMA Request bit, however, the bit is also useful for non-DMA transfers, especially in the FIFO State mode.

GPU Depth Ordering

Absent Depth Buffer
The PlayStation's GPU stores only RGB colors in the framebuffer (ie. unlike modern 3D processors, it's NOT buffering Depth values; leaving apart the Mask bit, which could be considered as a tiny 1bit "Depth" or "Priority" value). In fact, the GPU supports only X,Y coordinates, and it's totally unaware of Z coordinates. So, when rendering a polygon, the hardware CANNOT determine which of the new pixels are in front/behind of the old pixels in the buffer.

Simple Ordering
The rendering simply takes place in the ordering as the data is sent to the GPU (ie. the most distant objects should be sent first). For 2D graphics, it's fairly easy follow that order (eg. even multi-layer 2D graphics can be using DMA2-continous mode).

Depth Ordering Table (OT)
For 3D graphics, the ordering of the polygons may change more or less randomly (eg. when rotating/moving the camera). To solve that problem, the whole rendering data is usually first stored in a Depth Ordering Table (OT) in Main RAM, and, once when all polygons have been stored in the OT, the OT is sent to the GPU via "DMA2-linked-list" mode.

Initializing an empty OT (via DMA6)
DMA channel 6 can be used to set up an empty linked list, in which each entry points to the previous:

```
DPCR    - enable bits                                ;Example=x8xxxxxh
D6_MADR - pointer to the LAST table entry            ;Example=8012300Ch
D6_BCR  - number of list entries                     ;Example=00000004h
D6_CHCR - control bits (should be 11000002h)         ;Example=11000002h
```

Each entry has a size of 00h words (upper 8bit), and a pointer to the previous entry (lower 24bit). With the above Example values, the generated table would look like so:

```
[80123000h]=00FFFFFFh  ;1st entry, points to end code (xxFFFFFFh)
[80123004h]=00123000h  ;2nd entry, points to 1st entry
[80123008h]=00123004h  ;3rd entry, points to 2nd entry
[8012300Ch]=00123008h  ;last entry, points to 3rd entry (table entrypoint)
```

Inserting Entries (Passing GTE data to the OT) (by software)
The GTE commands AVSZ3 and AVSZ4 can be used to calculate the Average Z coordinates of a polygon (based on its three or four Z coordinates). The result is returned as a 16bit Z value in GTE register OTZ, the commands do also allow to divide the result, to make it less than 16bit (the full 16bit would require an OT of 256KBytes - for the EMPTY table, which would be a waste of memory, and which would slowdown the DMA2/DMA6 operations) (on the other hand, a smaller table means less depth resolution).

```
[PacketAddr+0]      = [80123000h+OTZ*4] + (N SHL 24)   <--internal link chain
[PacketAddr+4..N*4] = GP0 Command(s) and Parameters    <--data (send to GP0)
[80123000h+OTZ*4]   = PacketAddr AND FFFFFFh           <--internal link chain
```

If there's been already an entry (at the same OTZ index), then the new polygon will be processed first (ie. it will appear "behind" of the old entry).
Not sure if the packet size must be limited to max N=16 words (ie. as for the DMA2-continous block size) (due to GP0 FIFO size limits)?

Sending the OT to the CPU (via DMA2-linked-list mode)

```
1 - Wait until GPU is ready to receive commands ;GPUSTAT.28
2 - Enable DMA channel 2                 ;DPCR
3 - Set GPU to DMA cpu->gpu mode         ;[GP1]=04000002h aka GP1(04h)
3 - Set D2_MADR to the start of the list  ;(LAST Entry) ;Example=80123010h
4 - Set D2_BCR to zero                    ;(length unused, end at END-CODE)
5 - Set D2_CHCR to link mode, mem->GPU and dma enable   ;=01000401h
```

GPU Video Memory (VRAM)

Framebuffer
The framebuffer contains the image that is to be output to the Television Set. The GPU supports 10 resolutions, with 16bit or 24bit per pixel.

Resolution	16bit	24bit	Resolution	16bit	24bit
256x240	120Kbytes	180Kbytes	256x480	240Kbytes	360Kbytes
320x240	150Kbytes	225Kbytes	320x480	300Kbytes	450Kbytes
368x240	xx0Kbytes	xx0Kbytes	368x480	xx0Kbytes	xx0Kbytes
512x240	240Kbytes	360Kbytes	512x480	480Kbytes	720Kbytes
640x240	300Kbytes	450Kbytes	640x480	600Kbytes	900Kbytes

Note: In most cases, you'll need TWO framebuffers (one being displayed, and used as rendering target) (unless you are able to draw the whole new image during vblank, or unless when using single-layer 2D graphics). So, resolutions that occupy more than 512K would exceed the available 1MB VRAM when using 2 buffers.

Also, high resolutions mean higher rendering load, and less texture memory.

```
15bit Direct Display (default) (works with polygons, lines, rectangles)
0-4   Red       (0..31)
5-9   Green     (0..31)
10-14 Blue      (0..31)
15    Mask flag (0=Normal, 1=Do not allow to overwrite this pixel)
24bit Direct Display (works ONLY with direct vram transfers)
0-7    Red      (0..255)
8-15   Green    (0..255)
16-23  Blue     (0..255)
```

Note: The 24bit pixels occupy 3 bytes (not 4 bytes with unused MSBs), so each 6 bytes contain two 24bit pixels. The 24bit display mode works only with VRAM transfer commands like GP0(A0h); the rendering commands GP0(20h..7Fh) cannot output 24bit data. Ie. 24bit mode is used mostly for MDEC videos (and some 2D games like Heart of Darkness).

Texture Bitmaps

A texture is an image put on a polygon or sprite. The data of a texture can be stored in 3 different modes:

```
16bit Texture (Direct Color)            ;(One 256x256 page = 128Kbytes)
0-4   Red       (0..31)         ;\Color 0000h        = Fully-Transparent
5-9   Green     (0..31)         ; Color 0001h..7FFFh = Non-Transparent
10-14 Blue      (0..31)         ; Color 8000h..FFFFh = Semi-Transparent (*)
15    Semi Transparency Flag    ;/(*) or Non-Transparent for opaque commands
8bit Texture (256 Color Palette)        ;(One 256x256 page = 64Kbytes)
0-7   Palette index for 1st pixel (left)
8-15  Palette index for 2nd pixel (right)
4bit Texture (16 Color Palette)         ;(One 256x256 page = 32Kbytes)
0-3   Palette index for 1st pixel (left)
4-7   Palette index for 2nd pixel (middle/left)
8-11  Palette index for 3rd pixel (middle/right)
12-15 Palette index for 4th pixel (right)
```

A Texture Page is a 256x256 texel region in VRAM (the Polygon rendering commands are using Texcoords with 8bit X,Y coordinates, so polygons cannot use textures bigger than 256x256) (the Rectangle rendering commands with width/height parameters could theoretically use larger textures, but the hardware clips their texture coordinates to 8bit, too).
The GP0(E2h) Texture Window (aka Texture Repeat) command can be used to reduce the texture size to less than 256x256 texels.
The Texture Pages can be located in the frame buffer on X multiples of 64 halfwords and Y multiples of 256 lines.

Texture Palettes - CLUT (Color Lookup Table)

The clut is a the table where the colors are stored for the image data in the CLUT modes. The pixels of those images are used as indexes to this table. The clut is arranged in the frame buffer as a 256x1 image for the 8bit clut mode, and a 16x1 image for the 4bit clut mode.

```
0-4   Red       (0..31)         ;\Color 0000h        = Fully-Transparent
5-9   Green     (0..31)         ; Color 0001h..7FFFh = Non-Transparent
10-14 Blue      (0..31)         ; Color 8000h..FFFFh = Semi-Transparent (*)
15    Semi Transparency Flag    ;/(*) or Non-Transparent for opaque commands
```

The clut data can be arranged in the frame buffer at X multiples of 16 (X=0,16,32,48,etc) and anywhere in the Y range of 0-511.

Texture Color Black Limitations

On the PSX, texture color 0000h is fully-transparent, that means textures cannot contain Black pixels. However, in some cases, Color 8000h (Black with semi-transparent flag) can be used, depending on the rendering command:

```
opaque command, eg. GP0(24h)      --> 8000h = Non-Transparent Black
semi-transp command, eg. GP0(26h) --> 8000h = Semi-Transparent Black
```

So, with semi-transparent rendering commands, it isn't possible to use Non-Transparent Black pixels in textures, the only workaround is to use colors like 0001h (dark red) or 0400h (dark blue). However, due to the PSX's rather steeply increasing intensity ramp, these colors are clearly visible to be brighter than black.

RGB Intensity Notes

The Playstations RGB values aren't linear to normal RGB values (as used on PCs). The min/max values are of course the same, but the medium values differ:

Intensity	PC	PSX
Minimum	0	0
Medium (circa)	16	8
Maximum	31	31

Ie. on the PSX, the intensity increases steeply from 0 to 15, and less steeply from 16 to 31.

GPU Texture Caching

The GPU has 2 Kbyte Texture Cache
The Texture Cache is (maybe) also used for CLUT data - or is there a separate CLUT Cache - or is the CLUT uncached - but that'd be trash?

If polygons with texture are displayed, the GPU needs to read these from the frame buffer. This slows down the drawing process, and as a result the number of polygons that can be drawn in a given timespan. To speed up this process the GPU is equipped with a texture cache, so a given piece of texture needs not to be read multiple times in succession.
The texture cache size depends on the color mode used for the textures.
In 4 bit CLUT mode it has a size of 64x64, in 8 bit CLUT it's 32x64 and in 15bitDirect is 32x32. A general speed up can be achieved by setting up textures according to these sizes. For further speed gain a more precise knowledge of how the cache works is necessary.

Cache blocks

The texture page is divided into non-overlapping cache blocks, each of a unit size according to color mode. These cache blocks are tiled within the texture page.

```
+-----+-----+-----+--
|cache|     |     |
|block|     |
|    0|   1 |   2   ..
+-----+-----+--
|..   |     |
```

Cache entries

Each cache block is divided into 256 cache entries, which are numbered sequentially, and are 8 bytes wide. So a cache entry holds 16 4bit clut pixels 8 8bit clut pixels, or 4 15bitdirect pixels.

```
4bit and 8bit clut:        15bitdirect:
+----+----+----+----+     +----+----+----+----+----+----+----+----+
|   0|   1|   2|   3|     |   0|   1|   2|   3|   4|   5|   6|   7|
+----+----+----+----+     +----+----+----+----+----+----+----+----+
|   4|   5|   6|   7|     |   8|   9|   a|   b|   c|   d|   e|   f|
+----+----+----+----+     +----+----+----+----+----+----+----+----+
|   8|   9|  ..           |  10|  11|  ..
+----+----+--             +----+----+--
|   c|  ..|               |  18|  ..|
```

```
  +----+--                +----+--
  |  ..                   |  ..
```

The cache can hold only one cache entry by the same number, so if f.e. a piece of texture spans multiple cache blocks and it has data on entry 9 of block 1, but also on entry 9 of block 2, these cannot be in the cache at once.

GPU Timings

Video Clock

The PSone/PAL video clock is the cpu clock multiplied by 11/7.

```
  CPU Clock   =  33.868800MHz (44100Hz*300h)
  Video Clock =  53.222400MHz (44100Hz*300h*11/7)
```

For other PSX/PSone PAL/NTSC variants, see:
Pinouts - CLK Pinouts

Vertical Timings

```
  PAL:  314 scanlines per frame (13Ah)
  NTSC: 263 scanlines per frame (107h)
```

Timer1 can use the hblank signal as input, allowing to count scanlines (unless the display is configured to 0 pixels width, which would cause an endless hblank). The hblank signal is generated even during vertical blanking/retrace.

Horizontal Timings

```
  PAL:  3406 video cycles per scanline (or 3406.1 or so?)
  NTSC: 3413 video cycles per scanline (or 3413.6 or so?)
```

Dotclocks:

```
  PSX.256-pix Dotclock =  5.322240MHz (44100Hz*300h*11/7/10)
  PSX.320-pix Dotclock =  6.652800MHz (44100Hz*300h*11/7/8)
  PSX.368-pix Dotclock =  7.603200MHz (44100Hz*300h*11/7/7)
  PSX.512-pix Dotclock = 10.644480MHz (44100Hz*300h*11/7/5)
  PSX.640-pix Dotclock = 13.305600MHz (44100Hz*300h*11/7/4)
  Namco GunCon 385-pix =  8.000000MHz (from 8.00MHz on lightgun PCB)
```

Dots per scanline are, depending on horizontal resolution, and on PAL/NTSC:

```
  320pix/PAL: 3406/8  = 425.75 dots     320pix/NTSC: 3413/8  = 426.625 dots
  640pix/PAL: 3406/4  = 851.5 dots      640pix/NTSC: 3413/4  = 853.25 dots
  256pix/PAL: 3406/10 = 340.6 dots      256pix/NTSC: 3413/10 = 341.3 dots
  512pix/PAL: 3406/5  = 681.2 dots      512pix/NTSC: 3413/5  = 682.6 dots
  368pix/PAL: 3406/7  = 486.5714 dots   368pix/NTSC: 3413/7  = 487.5714 dots
```

Timer0 can use the dotclock as input, however, the Timer0 input "ignores" the fractional portions (in most cases, the values are rounded down, ie. with 340.6 dots/line, the timer increments only 340 times/line; the only value that is rounded up is 425.75 dots/line) (for example, due to the rounding, the timer isn't running exactly twice as fast in 512pix/PAL mode than in 256pix/PAL mode). The dotclock signal is generated even during horizontal/vertical blanking/retrace.

Frame Rates

```
  PAL:  53.222400MHz/314/3406 = ca. 49.76 Hz (ie. almost 50Hz)
  NTSC: 53.222400MHz/263/3413 = ca. 59.29 Hz (ie. almost 60Hz)
```

Note

Above values include "hidden" dots and scanlines (during horizontal and vertical blanking/retrace).

GPU (MISC)

GP0(20h..7Fh) - Render Command Bits

```
  0-23  Color for (first) Vertex                  (Not for Raw-Texture)
  24    Texture Mode      (0=Blended, 1=Raw)      (Textured-Polygon/Rect only)
  25    Semi Transparency (0=Off, 1=On)           (All Render Types)
  26    Texture Mapping   (0=Off, 1=On)           (Polygon/Rectangle only)
  27-28 Rect Size   (0=Var, 1=1x1, 2=8x8, 3=16x16) (Rectangle only)
  27    Num Vertices      (0=Triple, 1=Quad)      (Polygon only)
  27    Num Lines         (0=Single, 1=Poly)      (Line only)
  28    Shading           (0=Flat, 1=Gouroud)     (Polygon/Line only)
  29-31 Primitive Type    (1=Polygon, 2=Line, 3=Rectangle)
```

Perspective (in-)correct Rendering

The PSX doesn't support perspective correct rendering: Assume a polygon to be rotated so that it's right half becomes more distant to the camera, and it's left half becomes closer. Due to the GTE's perspective division, the right half should appear smaller than the left half.
The GPU supports only linear interpolations for rendering - that is correct concerning the X and Y screen coordinates (which are still linear to each other, even after perspective division, since both are divided by the same value).
However, texture coordinates (and Gouraud shaded colors) are NOT linear to the screen coordinates, and so, the linear interpolated PSX graphics are often looking rather distorted, that especially for textures that contain straight lines. For color shading the problem is less obvious (since shading is kinda blurry anyways).

Perspective correct Rendering

For perspective correct rendering, the polygon's Z-coordinates would be needed to be passed from the GTE to the GPU, and, the GPU would then need to use that Z-coordinates to "undo" the perspective division for each pixel (that'd require some additional memory, and especially a powerful division unit, which isn't implemented in the hardware).
As a workaround, you can try to reduce the size of your polygons (the interpolation errors increase in the center region of larger polygons). Reducing the size would be only required for polygons that occupy a larger screen region (which may vary depending on the distance to the camera).
Ie. you may check the size AFTER perspective division, if it's too large, then break it into smaller parts (using the original coordinates, NOT the screen coordinates), and then pass the fragments to the GTE another time.
Again, perspective correction would be relevant only for certain textures (not for randomly dithered textures like sand, water, fire, grass, and not for untextured polygons, and of course not for 2D graphics, so you may exclude those from size reduction).

24bit RGB to 15bit RGB Dithering (enabled in Texpage attribute)

For dithering, VRAM is broken to 4x4 pixel blocks, depending on the location in that 4x4 pixel region, the corresponding dither offset is added to the 8bit R/G/B values, the result is saturated to +00h..+FFh, and then divided by 8, resulting in the final 5bit R/G/B values.

```
  -4  +0  -3  +1   ;\dither offsets for first two scanlines
  +2  -2  +3  -1   ;/
  -3  +1  -4  +0   ;\dither offsets for next two scanlines
  +3  -1  +2  -2   ;/(same as above, but shifted two pixels horizontally)
```

POLYGONs (triangles/quads) are dithered ONLY if they do use gouraud shading or texture blending.
LINEs are dithered (no matter if they are mono or do use gouraud shading).
RECTs are NOT dithered (no matter if they do use texture blending).

Shading information
"Texture RGB values control the brightness of the individual colors ($00-$7f). A value of $80 in a color will take the former value as data." (What...? probably means the "double brightness" effect... or does it want to tell that ALL colors of 80h..FFh have only single brightness.. rather than reaching double brightness at FFh...?)

Shading
The GPU has a shading function, which will scale the color of a primitive to a specified brightness. There are 2 shading modes: Flat shading, and gouraud shading. Flat shading is the mode in which one brightness value is specified for the entire primitive. In Gouraud shading mode, a different brightness value can be given for each vertex of a primitive, and the brightness between these points is automatically interpolated.

Semi Transparency
When semi transparency is set for a pixel, the GPU first reads the pixel it wants to write to, and then calculates the color it will write from the 2 pixels according to the semitransparency mode selected. Processing speed is lower in this mode because additional reading and calculating are necessary. There are 4 semitransparency modes in the GPU.

```
B=Back  (the old pixel read from the image in the frame buffer)
F=Front (the new halftransparent pixel)
* 0.5 x B + 0.5 x F    ;aka B/2+F/2
* 1.0 x B + 1.0 x F    ;aka B+F
* 1.0 x B - 1.0 x F    ;aka B-F
* 1.0 x B +0.25 x F    ;aka B+F/4
```

Draw to display enable
This will enable/disable any drawing to the area that is currently displayed. Not sure yet WHY one should want to disable that?
Also not sure HOW and IF it works... the SIZE of the display area is implied by the screen size - which is horizontally counted in CLOCK CYCLES, so, to obtain the size in PIXELS, the hardware would require to divide that value by the number of cycles per pixel, depending on the current resolution...?

Geometry Transformation Engine (GTE)

GTE Overview
GTE Registers
GTE Saturation
GTE Opcode Summary
GTE Coordinate Calculation Commands
GTE General Purpose Calculation Commands
GTE Color Calculation Commands
GTE Division Inaccuracy

GTE Overview

GTE Operation
The GTE doesn't have any memory or I/O ports mapped to the CPU memory bus, instead, it's solely accessed via coprocessor opcodes:

```
mov  cop0r12,rt          ;-enable/disable COP2 (GTE) via COP0 status register
mov  cop2r0-63,rt        ;\write parameters to GTE registers
mov  cop2r0-31,[rs+imm]  ;/
mov  cop2cmd,imm25       ;-issue GTE command
mov  rt,cop2r0-63        ;\read results from GTE registers
mov  [rs+imm],cop2r0-31  ;/
jt   cop2flg,dest        ;-jump never  ;\implemented (no exception), but,
jf   cop2flg,dest        ;-jump always ;/flag seems to be always "false"
```

GTE (memory-?) load and store instructions have a delay of 2 instructions, for any GTE commands or operations accessing that register. Any? That's wrong!
GTE instructions and functions should not be used in

```
- Delay slots of jumps and branches
- Event handlers or interrupts (sounds like nonsense?) (need push/pop though)
```

If an instruction that reads a GTE register or a GTE command is executed before the current GTE command is finished, the CPU will hold until the instruction has finished. The number of cycles each GTE instruction takes is shown in the command list.

GTE Command Encoding (COP2 imm25 opcodes)

```
31-25  Must be 0100101b for "COP2 imm25" instructions
20-24  Fake GTE Command Number (00h..1Fh) (ignored by hardware)
19     sf - Shift Fraction in IR registers (0=No fraction, 1=12bit fraction)
17-18  MVMVA Multiply Matrix    (0=Rotation. 1=Light, 2=Color, 3=Reserved)
15-16  MVMVA Multiply Vector    (0=V0, 1=V1, 2=V2, 3=IR/long)
13-14  MVMVA Translation Vector (0=TR, 1=BK, 2=FC/Bugged, 3=None)
11-12  Always zero                      (ignored by hardware)
10     lm - Saturate IR1,IR2,IR3 result (0=To -8000h..+7FFFh, 1=To 0..+7FFFh)
6-9    Always zero                      (ignored by hardware)
0-5    Real GTE Command Number (00h..3Fh) (used by hardware)
```

The MVMVA bits are used only by the MVMVA opcode (the bits are zero for all other opcodes).
The "sf" and "lm" bits are usually fixed (either set, or cleared, depending on the command) (for MVMVA, the bits are variable) (also, "sf" can be changed for some commands like SQR) (although they are usually fixed for most other opcodes, changing them might have some effect on some/all opcodes)?

GTE Data Register Summary (cop2r0-31)

```
cop2r0-1   3xS16 VXY0,VZ0             Vector 0 (X,Y,Z)
cop2r2-3   3xS16 VXY1,VZ1             Vector 1 (X,Y,Z)
cop2r4-5   3xS16 VXY2,VZ2             Vector 2 (X,Y,Z)
cop2r6     4xU8  RGBC                 Color/code value
cop2r7     1xU16 OTZ                  Average Z value (for Ordering Table)
cop2r8     1xS16 IR0                  16bit Accumulator (Interpolate)
cop2r9-11  3xS16 IR1,IR2,IR3          16bit Accumulator (Vector)
cop2r12-15 6xS16 SXY0,SXY1,SXY2,SXYP  Screen XY-coordinate FIFO  (3 stages)
cop2r16-19 4xU16 SZ0,SZ1,SZ2,SZ3      Screen Z-coordinate FIFO   (4 stages)
cop2r20-22 12xU8 RGB0,RGB1,RGB2       Color CRGB-code/color FIFO (3 stages)
cop2r23    4xU8  (RES1)               Prohibited
cop2r24    1xS32 MAC0                 32bit Maths Accumulators (Value)
cop2r25-27 3xS32 MAC1,MAC2,MAC3       32bit Maths Accumulators (Vector)
cop2r28-29 1xU15 IRGB,ORGB            Convert RGB Color (48bit vs 15bit)
cop2r30-31 2xS32 LZCS,LZCR            Count Leading-Zeroes/Ones (sign bits)
```

GTE Control Register Summary (cop2r32-63)

```
cop2r32-36 9xS16 RT11RT12,..,RT33 Rotation matrix     (3x3)        ;cnt0-4
```

```
cop2r37-39 3x 32 TRX,TRY,TRZ      Translation vector  (X,Y,Z)      ;cnt5-7
cop2r40-44 9xS16 L11L12,..,L33    Light source matrix (3x3)        ;cnt8-12
cop2r45-47 3x 32 RBK,GBK,BBK      Background color    (R,G,B)      ;cnt13-15
cop2r48-52 9xS16 LR1LR2,..,LB3    Light color matrix source (3x3)  ;cnt16-20
cop2r53-55 3x 32 RFC,GFC,BFC      Far color           (R,G,B)      ;cnt21-23
cop2r56-57 2x 32 OFX,OFY          Screen offset       (X,Y)        ;cnt24-25
cop2r58 BuggyU16 H                Projection plane distance.       ;cnt26
cop2r59      S16 DQA              Depth queing parameter A (coeff) ;cnt27
cop2r60       32 DQB              Depth queing parameter B (offset);cnt28
cop2r61-62 2xS16 ZSF3,ZSF4        Average Z scale factors          ;cnt29-30
cop2r63      U20 FLAG             Returns any calculation errors   ;cnt31
```

GTE Registers

Note in some functions format is different from the one that's given here.

Matrix Registers

```
Rotation matrix (RT)   Light matrix (LLM)     Light Color matrix (LCM)
cop2r32.lsbs=RT11      cop2r40.lsbs=L11       cop2r48.lsbs=LR1
cop2r32.msbs=RT12      cop2r40.msbs=L12       cop2r48.msbs=LR2
cop2r33.lsbs=RT13      cop2r41.lsbs=L13       cop2r49.lsbs=LR3
cop2r33.msbs=RT21      cop2r41.msbs=L21       cop2r49.msbs=LG1
cop2r34.lsbs=RT22      cop2r42.lsbs=L22       cop2r50.lsbs=LG2
cop2r34.msbs=RT23      cop2r42.msbs=L23       cop2r50.msbs=LG3
cop2r35.lsbs=RT31      cop2r43.lsbs=L31       cop2r51.lsbs=LB1
cop2r35.msbs=RT32      cop2r43.msbs=L32       cop2r51.msbs=LB2
cop2r36     =RT33      cop2r44     =L33       cop2r52     =LB3
```

Each element is 16bit (1bit sign, 3bit integer, 12bit fraction). Reading the last elements (RT33,L33,LB3) returns the 16bit value sign-expanded to 32bit.

Translation Vector (TR) (Input, R/W?)

```
cop2r37 (cnt5) - TRX - Translation vector X (R/W?)
cop2r38 (cnt6) - TRY - Translation vector Y (R/W?)
cop2r39 (cnt7) - TRZ - Translation vector Z (R/W?)
```

Each element is 32bit (1bit sign, 31bit integer).
Used only for MVMVA, RTPS, RTPT commands.

Background Color (BK) (Input?, R/W?)

```
cop2r45 (cnt13) - RBK - Background color red component
cop2r46 (cnt14) - GBK - Background color green component
cop2r47 (cnt15) - BBK - Background color blue component
```

Each element is 32bit (1bit sign, 19bit integer, 12bit fraction).

Far Color (FC) (Input?) (R/W?)

```
cop2r53 (cnt21) - RFC - Far color red component
cop2r54 (cnt22) - GFC - Far color green component
cop2r55 (cnt23) - BFC - Far color blue component
```

Each element is 32bit (1bit sign, 27bit integer, 4bit fraction).

Screen Offset and Distance (Input, R/W?)

```
cop2r56 (cnt24) - OFX - Screen offset X
cop2r57 (cnt25) - OFY - Screen offset Y
cop2r58 (cnt26) - H   - Projection plane distance
cop2r59 (cnt27) - DQA - Depth queing parameter A.(coeff.)
cop2r60 (cnt28) - DQB - Depth queing parameter B.(offset.)
```

The X and Y values are each 32bit (1bit sign, 15bit integer, 16bit fraction).
The H value is 16bit unsigned (0bit sign, 16bit integer, 0bit fraction). BUG: When reading the H register, the hardware does accidently <sign-expand> the <unsigned> 16bit value (ie. values +8000h..+FFFFh are returned as FFFF8000h..FFFFFFFFh) (this bug applies only to "mov rd,cop2r58" opcodes; the actual calculations via RTPS/RTPT opcodes are working okay).
The DQA value is only 16bit (1bit sign, 7bit integer, 8bit fraction).
The DQB value is 32bit (1bit sign, 7bit integer, 24bit? fraction).
Used only for RTPS/RTPT commands.

Average Z Registers (ZSF3/ZSF4=Input, R/W?) (OTZ=Result, R)

```
cop2r61 (cnt29) ZSF3 |  0|ZSF3 1,3,12| Z3 average scale factor (normally 1/3)
cop2r62 (cnt30) ZSF4 |  0|ZSF4 1,3,12| Z4 average scale factor (normally 1/4)
cop2r7       OTZ (R) |   |OTZ 0,15, 0| Average Z value (for Ordering Table)
```

Used only for AVSZ3/AVSZ4 commands.

Screen XYZ Coordinate FIFOs

```
cop2r12 - SXY0  rw|SY0 1,15, 0|SX0 1,15, 0| Screen XY fifo (older)
cop2r13 - SXY1  rw|SY1 1,15, 0|SX1 1,15, 0| Screen XY fifo (old)
cop2r14 - SXY2  rw|SY2 1,15, 0|SX2 1,15, 0| Screen XY fifo (new)
cop2r15 - SXYP  rw|SYP 1,15, 0|SXP 1,15, 0| SXY2-mirror with move-on-write
cop2r16 - SZ0   rw|          0|SZ0 0,16, 0| Screen Z fifo (oldest)
cop2r17 - SZ1   rw|          0|SZ1 0,16, 0| Screen Z fifo (older)
cop2r18 - SZ2   rw|          0|SZ2 0,16, 0| Screen Z fifo (old)
cop2r19 - SZ3   rw|          0|SZ3 0,16, 0| Screen Z fifo (new)
```

SX,SY,SZ are used as Output for RTPS/RTPT. Additionally, SX,SY are used as Input for NCLIP, and SZ is used as Input for AVSZ3/AVSZ4.
The SZn Fifo has 4 stages (required for AVSZ4 command), the SXYn Fifo has only 3 stages, and a special mirrored register: SXYP is a mirror of SXY2, the difference is that writing to SXYP moves SXY2/SXY1 to SXY1/SXY0, whilst writing to SXY2 (or any other SXYn or SZn registers) changes only the written register, but doesn't move any other Fifo entries.

16bit Vectors (R/W)

```
Vector 0 (V0)         Vector 1 (V1)       Vector 2 (V2)       Vector 3 (IR)
cop2r0.lsbs - VX0     cop2r2.lsbs - VX1   cop2r4.lsbs - VX2   cop2r9  - IR1
cop2r0.msbs - VY0     cop2r2.msbs - VY1   cop2r4.msbs - VY2   cop2r10 - IR2
cop2r1      - VZ0     cop2r3      - VZ1   cop2r5      - VZ2   cop2r11 - IR3
```

All elements are signed 16bit. The IRn and VZn elements occupy a whole 32bit register, reading these registers returns the 16bit value sign-expanded to 32bit. Note: IRn can be also indirectly accessed via IRGB/ORGB registers.

Color Register and Color FIFO

```
cop2r6  - RGBC  rw|CODE |B    |G    |R    | Color/code
cop2r20 - RGB0  rw|CD0  |B0   |G0   |R0   | Characteristic color fifo.
cop2r21 - RGB1  rw|CD1  |B1   |G1   |R1   |
```

```
  cop2r22 - RGB2  rw|CD2  |B2   |G2   |R2   |
  cop2r23 - (RES1)  |                       | Prohibited
```
RES1 seems to be unused... looks like an unused Fifo stage... RES1 is read/write-able... unlike SXYP (for SXYn Fifo) it does not mirror to RGB2, nor does it have a move-on-write function...

Interpolation Factor
```
  cop2r8   IR0   rw|Sign       |IR0 1, 3,12| Intermediate value 0.
```
Used as Output for RTPS/RTPT, and as Input for various commands.

XX...
```
  cop2r24  MAC0  rw|MAC0 1,31,0           | Sum of products value 0
```

XX...
```
  cop2r25  MAC1  rw|MAC1 1,31,0           | Sum of products value 1
  cop2r26  MAC2  rw|MAC2 1,31,0           | Sum of products value 2
  cop2r27  MAC3  rw|MAC3 1,31,0           | Sum of products value 3
```

cop2r28 - IRGB - Color conversion Input (R/W)
Expands 5:5:5 bit RGB (range 0..1Fh) to 16:16:16 bit RGB (range 0000h..0F80h).
```
  0-4    Red   (0..1Fh) (R/W)  ;multiplied by 80h, and written to IR1
  5-9    Green (0..1Fh) (R/W)  ;multiplied by 80h, and written to IR2
  10-14  Blue  (0..1Fh) (R/W)  ;multiplied by 80h, and written to IR3
  15-31  Not used (always zero) (Read only)
```
After writing to IRGB, the result can be read from IR3 after TWO nop's, and from IR1,IR2 after THREE nop's (for uncached code, ONE nop would work). When using IR1,IR2,IR3 as parameters for GTE commands, similar timing restrictions might apply... depending on when the specific commands use the parameters?

cop2r29 - ORGB - Color conversion Output (R)
Collapses 16:16:16 bit RGB (range 0000h..0F80h) to 5:5:5 bit RGB (range 0..1Fh). Negative values (8000h..FFFFh/80h) are saturated to 00h, large positive values (1000h..7FFFh/80h) are saturated to 1Fh, there are no overflow or saturation flags set in cop2r63 though.
```
  0-4    Red   (0..1Fh) (R)  ;IR1 divided by 80h, saturated to +00h..+1Fh
  5-9    Green (0..1Fh) (R)  ;IR2 divided by 80h, saturated to +00h..+1Fh
  10-14  Blue  (0..1Fh) (R)  ;IR3 divided by 80h, saturated to +00h..+1Fh
  15-31  Not used (always zero) (Read only)
```
Any changes to IR1,IR2,IR3 are reflected to this register (and, actually also to IRGB) (ie. ORGB is simply a read-only mirror of IRGB).

cop2r30 - LZCS - Count Leading Bits Source data (R/W)
cop2r31 - LZCR - Count Leading Bits Result (R)
Reading LZCR returns the leading 0 count of LZCS if LZCS is positive and the leading 1 count of LZCS if LZCS is negative. The results are in range 1..32.

cop2r63 (cnt31) - FLAG - Returns any calculation errors.
See GTE Saturation chapter.

GTE Saturation

Maths overflows are indicated in FLAG register. In most cases, the result is saturated to MIN/MAX values (except MAC0,MAC1,MAC2,MAC3 which aren't saturated). For IR1,IR2,IR3 many commands allow to select the MIN value via "lm" bit of the GTE opcode (though not all commands, RTPS/RTPT always act as if lm=0).

cop2r63 (cnt31) - FLAG - Returns any calculation errors.
```
  31   Error Flag (Bit30..23, and 18..13 ORed together) (Read only)
  30   MAC1 Result larger than 43 bits and positive
  29   MAC2 Result larger than 43 bits and positive
  28   MAC3 Result larger than 43 bits and positive
  27   MAC1 Result larger than 43 bits and negative
  26   MAC2 Result larger than 43 bits and negative
  25   MAC3 Result larger than 43 bits and negative
  24   IR1 saturated to +0000h..+7FFFh (lm=1) or to -8000h..+7FFFh (lm=0)
  23   IR2 saturated to +0000h..+7FFFh (lm=1) or to -8000h..+7FFFh (lm=0)
  22   IR3 saturated to +0000h..+7FFFh (lm=1) or to -8000h..+7FFFh (lm=0)
  21   Color-FIFO-R saturated to +00h..+FFh
  20   Color-FIFO-G saturated to +00h..+FFh
  19   Color-FIFO-B saturated to +00h..+FFh
  18   SZ3 or OTZ saturated to +0000h..+FFFFh
  17   Divide overflow. RTPS/RTPT division result saturated to max=1FFFFh
  16   MAC0 Result larger than 31 bits and positive
  15   MAC0 Result larger than 31 bits and negative
  14   SX2 saturated to -0400h..+03FFh
  13   SY2 saturated to -0400h..+03FFh
  12   IR0 saturated to +0000h..+1000h
  0-11 Not used (always zero) (Read only)
```
Bit30-12 are read/write-able, ie. they can be set/reset by software, however, that's normally not required - all bits are automatically reset at the begin of a new GTE command.
Bit31 is apparently intended for RTPS/RTPT commands, since it triggers only on flags that are affected by these two commands, but even for that commands it's totally useless since one could as well check if FLAG is nonzero.
Note: Writing 32bit values to 16bit GTE registers by software does not trigger any overflow/saturation flags (and does not do any saturation), eg. writing 12008900h (positive 32bit) to a signed 16bit register sets that register to FFFF8900h (negative 16bit).

GTE Opcode Summary

GTE Command Summary (sorted by Real Opcode bits) (bit0-5)
```
  Opc  Name   Clk Expl.
  00h  -          N/A (modifies similar registers than RTPS...)
  01h  RTPS   15  Perspective Transformation single
  0xh  -          N/A
  06h  NCLIP  8   Normal clipping
  0xh  -          N/A
  0Ch  OP(sf) 6   Outer product of 2 vectors
  0xh  -          N/A
  10h  DPCS   8   Depth Cueing single
  11h  INTPL  8   Interpolation of a vector and far color vector
  12h  MVMVA  8   Multiply vector by matrix and add vector (see below)
  13h  NCDS   19  Normal color depth cue single vector
```

```
14h  CDP    13  Color Depth Que
15h  -          N/A
16h  NCDT   44  Normal color depth cue triple vectors
1xh  -          N/A
1Bh  NCCS   17  Normal Color Color single vector
1Ch  CC     11  Color Color
1Dh  -          N/A
1Eh  NCS    14  Normal color single
1Fh  -          N/A
20h  NCT    30  Normal color triple
2xh  -          N/A
28h  SQR(sf)5   Square of vector IR
29h  DCPL   8   Depth Cue Color light
2Ah  DPCT   17  Depth Cueing triple (should be fake=08h, but isn't)
2xh  -          N/A
2Dh  AVSZ3  5   Average of three Z values
2Eh  AVSZ4  6   Average of four Z values
2Fh  -          N/A
30h  RTPT   23  Perspective Transformation triple
3xh  -          N/A
3Dh  GPF(sf)5   General purpose interpolation
3Eh  GPL(sf)5   General purpose interpolation with base
3Fh  NCCT   39  Normal Color Color triple vector
```

Unknown if/what happens when using the "N/A" opcodes?

GTE Command Summary (sorted by Fake Opcode bits) (bit20-24)

The fake opcode number in bit20-24 has absolutely no effect on the hardware, it seems to be solely used to (or not to) confuse developers. Having the opcodes sorted by their fake numbers gives a more or less well arranged list:

```
Fake Name   Clk Expl.
00h  -          N/A
01h  RTPS   15  Perspective Transformation single
02h  RTPT   23  Perspective Transformation triple
03h  -          N/A
04h  MVMVA  8   Multiply vector by matrix and add vector (see below)
05h  -          N/A
06h  DCPL   8   Depth Cue Color light
07h  DPCS   8   Depth Cueing single
08h  DPCT   17  Depth Cueing triple (should be fake=08h, but isn't)
09h  INTPL  8   Interpolation of a vector and far color vector
0Ah  SQR(sf)5   Square of vector IR
0Bh  -          N/A
0Ch  NCS    14  Normal color single
0Dh  NCT    30  Normal color triple
0Eh  NCDS   19  Normal color depth cue single vector
0Fh  NCDT   44  Normal color depth cue triple vectors
10h  NCCS   17  Normal Color Color single vector
11h  NCCT   39  Normal Color Color triple vector
12h  CDP    13  Color Depth Que
13h  CC     11  Color Color
14h  NCLIP  8   Normal clipping
15h  AVSZ3  5   Average of three Z values
16h  AVSZ4  6   Average of four Z values
17h  OP(sf) 6   Outer product of 2 vectors
18h  -          N/A
19h  GPF(sf)5   General purpose interpolation
1Ah  GPL(sf)5   General purpose interpolation with base
1Bh  -          N/A
1Ch  -          N/A
1Dh  -          N/A
1Eh  -          N/A
1Fh  -          N/A
```

For the sort-effect, DCPT should use fake=08h, but Sony seems to have accidently numbered it fake=0Fh in their devkit (giving it the same fake number as for NCDT). Also, "Wipeout 2097" accidently uses 0140006h (fake=01h and distorted bit18) instead of 1400006h (fake=14h) for NCLIP.

Additional Functions

The LZCS/LZCR registers offer a Count-Leading-Zeroes/Leading-Ones function.
The IRGB/ORGB registers allow to convert between 48bit and 15bit RGB colors.
These registers work without needing to send any COP2 commands. However, unlike for commands (which do automatically halt the CPU when needed), one must insert dummy opcodes between writing and reading the registers.

GTE Coordinate Calculation Commands

COP2 0180001h - 15 Cycles - RTPS - Perspective Transformation (single)
COP2 0280030h - 23 Cycles - RTPT - Perspective Transformation (triple)

RTPS performs final Rotate, translate and perspective transformation on vertex V0. Before writing to the FIFOs, the older entries are moved one stage down. RTPT is same as RTPS, but repeats for V1 and V2. The "sf" bit should be usually set.

```
IR1 = MAC1 = (TRX*1000h + RT11*VX0 + RT12*VY0 + RT13*VZ0) SAR (sf*12)
IR2 = MAC2 = (TRY*1000h + RT21*VX0 + RT22*VY0 + RT23*VZ0) SAR (sf*12)
IR3 = MAC3 = (TRZ*1000h + RT31*VX0 + RT32*VY0 + RT33*VZ0) SAR (sf*12)
SZ3 = MAC3 SAR ((1-sf)*12)                          ;ScreenZ FIFO 0..+FFFFh
MAC0=(((H*20000h/SZ3)+1)/2)*IR1+OFX, SX2=MAC0/10000h ;ScrX FIFO -400h..+3FFh
MAC0=(((H*20000h/SZ3)+1)/2)*IR2+OFY, SY2=MAC0/10000h ;ScrY FIFO -400h..+3FFh
MAC0=(((H*20000h/SZ3)+1)/2)*DQA+DQB, IR0=MAC0/1000h  ;Depth cueing 0..+1000h
```

If the result of the "(((H*20000h/SZ3)+1)/2)" division is greater than 1FFFFh, then the division result is saturated to +1FFFFh, and the divide overflow bit in the FLAG register gets set; that happens if the vertex is exceeding the "near clip plane", ie. if it is very close to the camera (SZ3<=H/2), exactly at the camara position (SZ3=0), or behind the camera (negative Z coordinates are saturated to SZ3=0). For details on the division, see:
GTE Division Inaccuracy
For "far plane clipping", one can use the SZ3 saturation flag (MaxZ=FFFFh), or the IR3 saturation flag (MaxZ=7FFFh) (eg. used by Wipeout 2097), or one can compare the SZ3 value with any desired MaxZ value by software.
Note: The command does saturate IR1,IR2,IR3 to -8000h..+7FFFh (regardless of lm bit). When using RTP with sf=0, then the IR3 saturation flag (FLAG.22) gets set <only> if "MAC3 SAR 12" exceeds -8000h..+7FFFh (although IR3 is saturated when "MAC3" exceeds -8000h..+7FFFh).

COP2 1400006h - 8 Cycles - NCLIP - Normal clipping

```
MAC0 =   SX0*SY1 + SX1*SY2 + SX2*SY0 - SX0*SY2 - SX1*SY0 - SX2*SY1
```

The sign of the result indicates whether the polygon coordinates are arranged clockwise or anticlockwise (ie. whether the front side or backside is visible). If the result is zero, then it's neither one (ie. the vertices are all arranged in a straight line). Note: The GPU probably renders straight lines as invisble 0 pixel width lines?

COP2 158002Dh - 5 Cycles - AVSZ3 - Average of three Z values (for Triangles)
COP2 168002Eh - 6 Cycles - AVSZ4 - Average of four Z values (for Quads)
```
MAC0 =  ZSF3*(SZ1+SZ2+SZ3)       ;for AVSZ3
MAC0 =  ZSF4*(SZ0+SZ1+SZ2+SZ3)   ;for AVSZ4
OTZ  =  MAC0/1000h               ;for both (saturated to 0..FFFFh)
```
Adds three or four Z values together and multplies them by a fixed point value. The result can be used as index in the GPU's Ordering Table (OT).
GPU Depth Ordering
The scaling factors would be usually ZSF3=N/30h and ZSF4=N/40h, where "N" is the number of entries in the OT (max 10000h). SZn and OTZ are unsigned 16bit values, for whatever reason ZSFn registers are signed 16bit values (negative values would allow a negative result in MAC0, but would saturate OTZ to zero).

GTE General Purpose Calculation Commands

COP2 0400012h - 8 Cycles - MVMVA(sf,mx,v,cv,lm)
Multiply vector by matrix and vector addition.
```
Mx = matrix specified by mx  ;RT/LLM/LCM - Rotation, light or color matrix
Vx = vector specified by v   ;V0, V1, V2, or [IR1,IR2,IR3]
Tx = translation vector specified by cv  ;TR or BK or Bugged/FC, or None
```
Calculation:
```
MAC1 = (Tx1*1000h + Mx11*Vx1 + Mx12*Vx2 + Mx13*Vx3) SAR (sf*12)
MAC2 = (Tx2*1000h + Mx21*Vx1 + Mx22*Vx2 + Mx23*Vx3) SAR (sf*12)
MAC3 = (Tx3*1000h + Mx31*Vx1 + Mx32*Vx2 + Mx33*Vx3) SAR (sf*12)
[IR1,IR2,IR3] = [MAC1,MAC2,MAC3]
```
Multiplies a vector with either the rotation matrix, the light matrix or the color matrix and then adds the translation vector or background color vector.
The GTE also allows selection of the far color vector (FC), but this vector is not added correctly by the hardware: The return values are reduced to the last portion of the formula, ie. MAC1=(Mx13*Vx3) SAR (sf*12), and similar for MAC2 and MAC3, netherthelss, some bits in the FLAG register seem to be adjusted as if the full operation would have been executed. Setting Mx=3 selects a garbage matrix (with elements -60h, +60h, IR0, RT13, RT13, RT13, RT22, RT22, RT22).

COP2 0A00428h+sf*80000h - 5 Cycles - SQR(sf) - Square vector
```
[MAC1,MAC2,MAC3] = [IR1*IR1,IR2*IR2,IR3*IR3] SHR (sf*12)
[IR1,IR2,IR3]    = [MAC1,MAC2,MAC3]    ;IR1,IR2,IR3 saturated to max 7FFFh
```
Calculates the square of a vector. The result is, of course, always positive, so the "lm" flag for negative saturation has no effect.

COP2 170000Ch+sf*80000h - 6 Cycles - OP(sf,lm) - Outer product of 2 vectors
```
[MAC1,MAC2,MAC3] = [IR3*D2-IR2*D3, IR1*D3-IR3*D1, IR2*D1-IR1*D2] SAR (sf*12)
[IR1,IR2,IR3]    = [MAC1,MAC2,MAC3]                    ;copy result
```
Calculates the outer product of two signed 16bit vectors. Note: D1,D2,D3 are meant to be the RT11,RT22,RT33 elements of the RT matrix "misused" as vector. lm should be usually zero.

LZCS/LZCR registers - ? Cycles - Count-Leading-Zeroes/Leading-Ones
The LZCS/LZCR registers offer a Count-Leading-Zeroes/Leading-Ones function.

GTE Color Calculation Commands

COP2 0C8041Eh - 14 Cycles - NCS - Normal color (single)
COP2 0D80420h - 30 Cycles - NCT - Normal color (triple)
COP2 108041Bh - 17 Cycles - NCCS - Normal Color Color (single vector)
COP2 118043Fh - 39 Cycles - NCCT - Normal Color Color (triple vector)
COP2 0E80413h - 19 Cycles - NCDS - Normal color depth cue (single vector)
COP2 0F80416h - 44 Cycles - NCDT - Normal color depth cue (triple vectors)
In: V0=Normal vector (for triple variants repeated with V1 and V2), BK=Background color, RGBC=Primary color/code, LLM=Light matrix, LCM=Color matrix, IR0=Interpolation value.
```
[IR1,IR2,IR3] = [MAC1,MAC2,MAC3] = (LLM*V0) SAR (sf*12)
[IR1,IR2,IR3] = [MAC1,MAC2,MAC3] = (BK*1000h + LCM*IR) SAR (sf*12)
[MAC1,MAC2,MAC3] = [R*IR1,G*IR2,B*IR3] SHL 4          ;<--- for NCDx/NCCx
[MAC1,MAC2,MAC3] = MAC+(FC-MAC)*IR0                   ;<--- for NCDx only
[MAC1,MAC2,MAC3] = [MAC1,MAC2,MAC3] SAR (sf*12)       ;<--- for NCDx/NCCx
Color FIFO = [MAC1/16,MAC2/16,MAC3/16,CODE], [IR1,IR2,IR3] = [MAC1,MAC2,MAC3]
```

COP2 138041Ch - 11 Cycles - CC(lm=1) - Color Color
COP2 1280414h - 13 Cycles - CDP(...) - Color Depth Que
In: [IR1,IR2,IR3]=Vector, RGBC=Primary color/code, LCM=Color matrix, BK=Background color, and, for CDP, IR0=Interpolation value, FC=Far color.
```
[IR1,IR2,IR3] = [MAC1,MAC2,MAC3] = (BK*1000h + LCM*IR) SAR (sf*12)
[MAC1,MAC2,MAC3] = [R*IR1,G*IR2,B*IR3] SHL 4
[MAC1,MAC2,MAC3] = MAC+(FC-MAC)*IR0                   ;<--- for CDP only
[MAC1,MAC2,MAC3] = [MAC1,MAC2,MAC3] SAR (sf*12)
Color FIFO = [MAC1/16,MAC2/16,MAC3/16,CODE], [IR1,IR2,IR3] = [MAC1,MAC2,MAC3]
```

COP2 0680029h - 8 Cycles - DCPL - Depth Cue Color light
COP2 0780010h - 8 Cycles - DPCS - Depth Cueing (single)
COP2 0x8002Ah - 17 Cycles - DPCT - Depth Cueing (triple)
COP2 0980011h - 8 Cycles - INTPL - Interpolation of a vector and far color
In: [IR1,IR2,IR3]=Vector, FC=Far Color, IR0=Interpolation value, CODE=MSB of RGBC, and, for DCPL, R,G,B=LSBs of RGBC.
```
[MAC1,MAC2,MAC3] = [R*IR1,G*IR2,B*IR3] SHL 4          ;<--- for DCPL only
[MAC1,MAC2,MAC3] = [IR1,IR2,IR3] SHL 12               ;<--- for INTPL only
[MAC1,MAC2,MAC3] = [R,G,B] SHL 16                     ;<--- for DPCS/DPCT
[MAC1,MAC2,MAC3] = MAC+(FC-MAC)*IR0
[MAC1,MAC2,MAC3] = [MAC1,MAC2,MAC3] SAR (sf*12)
Color FIFO = [MAC1/16,MAC2/16,MAC3/16,CODE], [IR1,IR2,IR3] = [MAC1,MAC2,MAC3]
```
DPCT executes thrice, and reads the R,G,B values from RGB0 (ie. reads from the Bottom of the Color FIFO, instead of from the RGBC register) (the CODE value is kept read from RGBC as usually), so, after DPCT execution, the RGB0,RGB1,RGB2 Fifo entries are modified.

COP2 190003Dh - 5 Cycles - GPF(sf,lm) - General purpose Interpolation
COP2 1A0003Eh - 5 Cycles - GPL(sf,?) - General Interpolation with base
```
[MAC1,MAC2,MAC3] = [0,0,0]                            ;<--- for GPF only
[MAC1,MAC2,MAC3] = [MAC1,MAC2,MAC3] SHL (sf*12)       ;<--- for GPL only
[MAC1,MAC2,MAC3] = (([IR1,IR2,IR3] * IR0) + [MAC1,MAC2,MAC3]) SAR (sf*12)
Color FIFO = [MAC1/16,MAC2/16,MAC3/16,CODE], [IR1,IR2,IR3] = [MAC1,MAC2,MAC3]
```
Note: Although the SHL in GPL is theoretically undone by the SAR, 44bit overflows can occur internally when sf=1.

Details on "MAC+(FC-MAC)*IR0"
```
[IR1,IR2,IR3] = (([RFC,GFC,BFC] SHL 12) - [MAC1,MAC2,MAC3]) SAR (sf*12)
[MAC1,MAC2,MAC3] = (([IR1,IR2,IR3] * IR0) + [MAC1,MAC2,MAC3])
```
Note: Above "[IR1,IR2,IR3]=(FC-MAC)" is saturated to -8000h..+7FFFh (ie. as if lm=0), anyways, further writes to [IR1,IR2,IR3] (within the same command) are saturated as usually (ie. depening on lm setting).

Details on "(LLM*V0) SAR (sf*12)" and "(BK*1000h + LCM*IR) SAR (sf*12)"
Works like MVMVA command (see there), but with fixed Tx/Vx/Mx parameters, the sf/lm bits can be changed and do affect the results (although normally both bits should be set for use with color matrices).

Notes
The 8bit RGB values written to the top of Color Fifo are the 32bit MACn values divided by 16, and saturated to +00h..+FFh, and of course, the older Fifo entries are moved downwards. Note that, at the GPU side, the meaning of the RGB values depends on whether or not texture blending is used (for untextured polygons FFh is max brightness) (for texture blending FFh is double brightness and 80h is normal brightness).
The 8bit CODE value is intended to contain a GP0(20h..7Fh) Rendering command, allowing to automatically merge the 8bit command number, with the 24bit color value.
The IRGB/ORGB registers allow to convert between 48bit and 15bit RGB colors.
Although the result of the commands in this chapter is written to the Color FIFO, some commands like GPF/GPL may be also used for other purposes (eg. to scale or scale/translate single vertices).

GTE Division Inaccuracy

GTE Division Inaccuracy (for RTPS/RTPT commands)
Basically, the GTE division does (attempt to) work as so (using 33bit maths):
```
n = (((H*20000h/SZ3)+1)/2)
```
alternatly, below would give (almost) the same result (using 32bit maths):
```
n = ((H*10000h+SZ3/2)/SZ3)
```
in both cases, the result is saturated about as so:
```
if n>1FFFFh or division_by_zero then n=1FFFFh, FLAG.Bit17=1, FLAG.Bit31=1
```
However, the real GTE hardware is using a fast, but less accurate division mechanism (based on Unsigned Newton-Raphson (UNR) algorithm):
```
if (H < SZ3*2) then                          ;check if overflow
  z = count_leading_zeroes(SZ3)              ;z=0..0Fh (for 16bit SZ3)
  n = (H SHL z)                              ;n=0..7FFF8000h
  d = (SZ3 SHL z)                            ;d=8000h..FFFFh
  u = unr_table[(d-7FC0h) SHR 7] + 101h      ;u=200h..101h
  d = ((2000080h - (d * u)) SHR 8)           ;d=10000h..0FF01h
  d = ((0000080h + (d * u)) SHR 8)           ;d=20000h..10000h
  n = min(1FFFFh, (((n*d) + 8000h) SHR 16))  ;n=0..1FFFFh
else n = 1FFFFh, FLAG.Bit17=1, FLAG.Bit31=1  ;n=1FFFFh plus overflow flag
```
the GTE's unr_table[000h..100h] consists of following values:
```
FFh,FDh,FBh,F9h,F7h,F5h,F3h,F1h,EFh,EEh,ECh,EAh,E8h,E6h,E4h,E3h ;\
E1h,DFh,DDh,DCh,DAh,D8h,D6h,D5h,D3h,D1h,D0h,CEh,CDh,CBh,C9h,C8h ; 00h..3Fh
C6h,C5h,C3h,C1h,C0h,BEh,BDh,BBh,BAh,B8h,B7h,B5h,B4h,B2h,B1h,B0h ;
AEh,ADh,ABh,AAh,A9h,A7h,A6h,A4h,A3h,A2h,A0h,9Fh,9Eh,9Ch,9Bh,9Ah ;/
99h,97h,96h,95h,94h,92h,91h,90h,8Fh,8Dh,8Ch,8Bh,8Ah,89h,87h,86h ;\
85h,84h,83h,82h,81h,7Fh,7Eh,7Dh,7Ch,7Bh,7Ah,79h,78h,77h,75h,74h ; 40h..7Fh
73h,72h,71h,70h,6Fh,6Eh,6Dh,6Ch,6Bh,6Ah,69h,68h,67h,66h,65h,64h ;
63h,62h,61h,60h,5Fh,5Eh,5Dh,5Dh,5Ch,5Bh,5Ah,59h,58h,57h,56h,55h ;/
54h,53h,53h,52h,51h,50h,4Fh,4Eh,4Dh,4Dh,4Ch,4Bh,4Ah,49h,48h,48h ;\
47h,46h,45h,44h,43h,43h,42h,41h,40h,3Fh,3Fh,3Eh,3Dh,3Ch,3Ch,3Bh ; 80h..BFh
3Ah,39h,39h,38h,37h,36h,36h,35h,34h,33h,33h,32h,31h,31h,30h,2Fh ;
2Eh,2Eh,2Dh,2Ch,2Ch,2Bh,2Ah,2Ah,29h,28h,28h,27h,26h,26h,25h,24h ;/
24h,23h,22h,22h,21h,20h,20h,1Fh,1Eh,1Eh,1Dh,1Dh,1Ch,1Bh,1Bh,1Ah ;\
19h,19h,18h,18h,17h,16h,16h,15h,15h,14h,14h,13h,12h,12h,11h,11h ; C0h..FFh
10h,0Fh,0Fh,0Eh,0Eh,0Dh,0Dh,0Ch,0Ch,0Bh,0Ah,0Ah,09h,09h,08h,08h ;
07h,07h,06h,06h,05h,05h,04h,04h,03h,03h,02h,02h,01h,01h,00h,00h ;/
00h    ;<-- one extra table entry (for "(d-7FC0h)/80h"=100h)    ;-100h
```
Above can be generated as "unr_table[i]=min(0,(40000h/(i+100h)+1)/2-101h)".
Some special cases: NNNNh/0001h uses a big multiplier (d=20000h), in practice, this can occur only for 0000h/0001h and 0001h/0001h (due to the H<SZ3*2 overflow check).
The min(1FFFFh) limit is needed for cases like FE3Fh/7F20h, F015h/780Bh, etc. (these do produce UNR result 20000h, and are saturated to 1FFFFh, but without setting overflow FLAG bits).

Macroblock Decoder (MDEC)

The MDEC is a JPEG-style Macroblock Decoder, that can decompress pictures (or a series of pictures, for being displayed as a movie).

MDEC I/O Ports
MDEC Commands
MDEC Decompression
MDEC Data Format

MDEC I/O Ports

1F801820h - MDEC0 - MDEC Command/Parameter Register (W)
```
31-0  Command or Parameters
```
Used to send command word, followed by parameter words to the MDEC (usually, only the command word is written to this register, and the parameter words are transferred via DMA0).

1F801820h.Read - MDEC Data/Response Register (R)
```
31-0  Macroblock Data (or Garbage if there's no data available)
```
The data is always output as a 8x8 pixel bitmap, so, when manually reading from this register and using colored 16x16 pixel macroblocks, the data from four 8x8 blocks must be re-ordered accordingly (usually, the data is received via DMA1, which is doing the re-ordering automatically). For monochrome 8x8 macroblocks, no re-ordering is needed (that works with DMA1 too).

1F801824h - MDEC1 - MDEC Status Register (R)
```
31    Data-Out Fifo Empty (0=No, 1=Empty)
```

```
  30    Data-In Fifo Full   (0=No, 1=Full, or Last word received)
  29    Command Busy  (0=Ready, 1=Busy receiving or processing parameters)
  28    Data-In Request  (set when DMA0 enabled and ready to receive data)
  27    Data-Out Request (set when DMA1 enabled and ready to send data)
  26-25 Data Output Depth  (0=4bit, 1=8bit, 2=24bit, 3=15bit)      ;CMD.28-27
  24    Data Output Signed (0=Unsigned, 1=Signed)                  ;CMD.26
  23    Data Output Bit15  (0=Clear, 1=Set) (for 15bit depth only) ;CMD.25
  22-19 Not used (seems to be always zero)
  18-16 Current Block (0..3=Y1..Y4, 4=Cr, 5=Cb) (or for mono: always 4=Y)
  15-0  Number of Parameter Words remaining minus 1  (FFFFh=None)  ;CMD.Bit0-15
```

If there's data in the output fifo, then the Current Block bits are always set to the current output block number (ie. Y1..Y4; or Y for mono) (this information is apparently passed to the DMA1 controller, so that it knows if and how it must re-order the data in RAM). If the output fifo is empty, then the bits indicate the currently processsed incoming block (ie. Cr,Cb,Y1..Y4; or Y for mono).

1F801824h - MDEC1 - MDEC Control/Reset Register (W)

```
  31    Reset MDEC (0=No change, 1=Abort any command, and set status=80040000h)
  30    Enable Data-In Request  (0=Disable, 1=Enable DMA0 and Status.bit28)
  29    Enable Data-Out Request (0=Disable, 1=Enable DMA1 and Status.bit27)
  28-0  Unknown/Not used - usually zero
```

The data requests are required to be enabled for using DMA (and for reading the request status flags by software). The Data-Out request acts a bit strange: It gets set when a block is available, but, it gets cleared after reading the first some words of that block (nethertheless, one can keep reading the whole block, until the fifo-empty flag gets set).

DMA

MDEC decompression uses a lot of DMA channels,

```
  1) DMA3 (CDROM)    to send compressed data from CDROM to RAM
  2) DMA0 (MDEC.In)  to send compressed data from RAM to MDEC
  3) DMA1 (MDEC.Out) to send uncompressed macroblocks from MDEC to RAM
  4) DMA2 (GPU)      to send uncompressed macroblocks from RAM to GPU
```

DMA0 and DMA1 should be usually used with a blocksize of 20h words. If necessary, the parameters for the MDEC(1) command should be padded with FE00h halfwords to match the 20h words (40h halfwords) DMA blocksize.

MDEC Commands

MDEC(1) - Decode Macroblock(s)

```
  31-29 Command (1=decode_macroblock)
  28-27 Data Output Depth  (0=4bit, 1=8bit, 2=24bit, 3=15bit)      ;STAT.26-25
  26    Data Output Signed (0=Unsigned, 1=Signed)                  ;STAT.24
  25    Data Output Bit15  (0=Clear, 1=Set) (for 15bit depth only) ;STAT.23
  24-16 Not used (should be zero)
  15-0  Number of Parameter Words (size of compressed data)
```

This command is followed by one or more Macroblock parameters (usually, all macroblocks for the whole image are sent at once).

MDEC(2) - Set Quant Table(s)

```
  31-29 Command (2=set_iqtab)
  28-1  Not used (should be zero)  ;Bit25-28 are copied to STAT.23-26 though
  0     Color   (0=Luminance only, 1=Luminance and Color)
```

The command word is followed by 64 unsigned parameter bytes for the Luminance Quant Table (used for Y1..Y4), and if Command.Bit0 was set, by another 64 unsigned parameter bytes for the Color Quant Table (used for Cb and Cr).

MDEC(3) - Set Scale Table

```
  31-29 Command (3=set_scale)
  28-0  Not used (should be zero)  ;Bit25-28 are copied to STAT.23-26 though
```

The command is followed by 64 signed halfwords with 14bit fractional part, the values should be usually/always the same values (based on the standard JPEG constants, although, MDEC(3) allows to use other values than that constants).

MDEC(0) - No function

This command has no function. Command bits 25-28 are reflected to Status bits 23-26 as usually. Command bits 0-15 are reflected to Status bits 0-15 (similar as the "number of parameter words" for MDEC(1), but without the "minus 1" effect, and without actually expecting any parameters).

MDEC(4..7) - Invalid

These commands act identical as MDEC(0).

MDEC Decompression

decode_colored_macroblock ;MDEC(1) command (at 15bpp or 24bpp depth)

```
  rl_decode_block(Crblk,src,iq_uv)                ;Cr (low resolution)
  rl_decode_block(Cbblk,src,iq_uv)                ;Cb (low resolution)
  rl_decode_block(Yblk,src,iq_y), yuv_to_rgb(0,0)  ;Y1 (and upper-left Cr,Cb)
  rl_decode_block(Yblk,src,iq_y), yuv_to_rgb(0,8)  ;Y2 (and upper-right Cr,Cb)
  rl_decode_block(Yblk,src,iq_y), yuv_to_rgb(8,0)  ;Y3 (and lower-left Cr,Cb)
  rl_decode_block(Yblk,src,iq_y), yuv_to_rgb(8,8)  ;Y4 (and lower-right Cr,Cb)
```

decode_monochrome_macroblock ;MDEC(1) command (at 4bpp or 8bpp depth)

```
  rl_decode_block(Yblk,src,iq_y), y_to_mono        ;Y
```

rl_decode_block(blk,src,qt)

```
  for i=0 to 63, blk[i]=0, next i   ;initially zerofill all entries (for skip)
 @@skip:
  n=[src], src=src+2, k=0           ;get first entry, init dest addr k=0
  if n=FE00h then @@skip            ;ignore padding (FE00h as first halfword)
  q_scale=(n SHR 10) AND 3Fh        ;contains scale value (not "skip" value)
  val=signed10bit(n AND 3FFh)*qt[k] ;calc first value (without q_scale/8) (?)
 @@lop:
  if q_scale=0 then val=signed10bit(n AND 3FFh)*2   ;special mode without qt[k]
  val=minmax(val,-400h,+3FFh)            ;saturate to signed 11bit range
  val=val*scalezag[i]               ;<-- for "fast_idct_core" only
  blk[zagzig[k]]=val                ;store entry
  n=[src], src=src+2                ;get next entry (or FE00h end code)
  k=k+((n SHR 10) AND 3Fh)+1        ;skip zerofilled entries
  val=(signed10bit(n AND 3FFh)*qt[k]*q_scale+4)/8  ;calc value for next entry
  if k<=63 then jump @@lop          ;should end with n=FE00h (that sets k>63)
```

```
  idct_core(blk)
  return (with "src" address advanced)
```

fast_idct_core(blk) ;fast "idct_core" version

Fast code with only 80 multiplications, works only if the scaletable from MDEC(3) command contains standard values (which is the case for all known PSX games).

```
  src=blk, dst=temp_buffer
  for pass=0 to 1
    for i=0 to 7
      if src[(1..7)*8+i]=0 then      ;when src[(1..7)*8+i] are all zero:
        dst[i*8+(0..7)]=src[0*8+i]   ;quick fill by src[0*8+i]
      else
        z10=src[0*8+i]+src[4*8+i], z11=src[0*8+i]-src[4*8+i]
        z13=src[2*8+i]+src[6*8+i], z12=src[2*8+i]-src[6*8+i]
        z12=(1.414213562*z12)-z13          ;=sqrt(2)
        tmp0=z10+z13, tmp3=z10-z13, tmp1=z11+z12, tmp2=z11-z12
        z13=src[3*8+i]+src[5*8+i], z10=src[3*8+i]-src[5*8+i]
        z11=src[1*8+i]+src[7*8+i], z12=src[1*8+i]-src[7*8+i]
        z5  =(1.847759065*(z12-z10))       ;=sqrt(2)*scalefactor[2]
        tmp7=z11+z13
        tmp6=(2.613125930*(z10))+z5-tmp7   ;=scalefactor[2]*2
        tmp5=(1.414213562*(z11-z13))-tmp6  ;=sqrt(2)
        tmp4=(1.082392200*(z12))-z5+tmp5   ;=sqrt(2)/scalefactor[2]
        dst[i*8+0]=tmp0+tmp7, dst[i*8+7]=tmp0-tmp7
        dst[i*8+1]=tmp1+tmp6, dst[i*8+6]=tmp1-tmp6
        dst[i*8+2]=tmp2+tmp5, dst[i*8+5]=tmp2-tmp5
        dst[i*8+4]=tmp3+tmp4, dst[i*8+3]=tmp3-tmp4
      endif
    next i
    swap(src,dst)
  next pass
```

real_idct_core(blk) ;low level "idct_core" version

Low level code with 1024 multiplications, using the scaletable from the MDEC(3) command. Computes dst=src*scaletable (using normal matrix maths, but with "src" being diagonally mirrored, ie. the matrices are processed column by column, instead of row by column), repeated with src/dst exchanged.

```
  src=blk, dst=temp_buffer
  for pass=0 to 1
    for x=0 to 7
      for y=0 to 7
        sum=0
        for z=0 to 7
          sum=sum+src[y+z*8]*(scaletable[x+z*8]/8)
        next z
        dst[x+y*8]=(sum+0fffh)/2000h              ;<-- or so?
      next y
    next x
    swap(src,dst)
  next pass
```

The "(sum+0fffh)/2000h" part is meant to strip fractional bits, and to round-up the result if the fraction was BIGGER than 0.5. The hardware appears to be working roughly like that, still the results aren't perfect.

Maybe the real hardware is doing further roundings in other places, possibly stripping some fractional bits before summing up "sum", possibly stripping different amounts of bits in the two "pass" cycles, and possibly keeping a final fraction passed on to the y_to_mono stage.

yuv_to_rgb(xx,yy)

```
  for y=0 to 7
    for x=0 to 7
      R=[Crblk+((x+xx)/2)+((y+yy)/2)*8], B=[Cbblk+((x+xx)/2)+((y+yy)/2)*8]
      G=(-0.3437*B)+(-0.7143*R), R=(1.402*R), B=(1.772*B)
      Y=[Yblk+(x)+(y)*8]
      R=MinMax(-128,127,(Y+R))
      G=MinMax(-128,127,(Y+G))
      B=MinMax(-128,127,(Y+B))
      if unsigned then BGR=BGR xor 808080h  ;aka add 128 to the R,G,B values
      dst[(x+xx)+(y+yy)*16]=BGR
    next x
  next y
```

Note: The exact fixed point resolution for "yuv_to_rgb" is unknown. And, there's probably also some 9bit limit (similar as in "y_to_mono").

y_to_mono

```
  for i=0 to 63
    Y=[Yblk+i]
    Y=Y AND 1FFh                  ;clip to signed 9bit range
    Y=MinMax(-128,127,Y)          ;saturate from 9bit to signed 8bit range
    if unsigned then Y=Y xor 80h  ;aka add 128 to the Y value
    dst[i]=Y
  next i
```

set_iqtab ;MDEC(2) command

```
  iqtab_core(iq_y,src), src=src+64       ;luminance quant table
  if command_word.bit0=1
    iqtab_core(iq_uv,src), src=src+64    ;color quant table (optional)
  endif
```

iqtab_core(iq,src) ;src = 64 unsigned paramter bytes

```
  for i=0 to 63, iq[i]=src[i], next i
```

Note: For "fast_idct_core" one could precalc "iq[i]=src[i]*scalezag[i]", but that would conflict with the RLE saturation/rounding steps (though those steps aren't actually required, so a very-fast decoder could omit them).

scalefactor[0..7] =

```
  1.000000000, 1.387039845, 1.306562965, 1.175875602,
  1.000000000, 0.785694958, 0.541196100, 0.275899379
```

zigzag[0..63] =

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

```
35,36,48,49,57,58,62,63
```

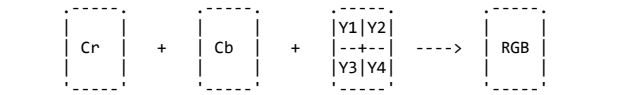
scalezag[0..63] (precalulated factors, for "fast_idct_core")
```
for y=0 to 7
 for x=0 to 7
  scalezag[zigzag[x+y*8]] = scalefactor[x] * scalefactor[y] / 8
 next x
next y
```

zagzig[0..63] (reversed zigzag table)
```
for i=0 to 63, zagzig[zigzag[i]]=i, next i
```

set_scale_table: ;MDEC(3) command
This command defines the IDCT scale matrix, which should be usually/always:
```
5A82 5A82 5A82 5A82 5A82 5A82 5A82 5A82
7D8A 6A6D 471C 18F8 E707 B8E3 9592 8275
7641 30FB CF04 89BE 89BE CF04 30FB 7641
6A6D E707 8275 B8E3 471C 7D8A 18F8 9592
5A82 A57D A57D 5A82 5A82 A57D A57D 5A82
471C 8275 18F8 6A6D 9592 E707 7D8A B8E3
30FB 89BE 7641 CF04 CF04 7641 89BE 30FB
18F8 B8E3 6A6D 8275 7D8A 9592 471C E707
```
Note that the hardware does actually use only the upper 13bit of those 16bit values. The values are choosen like so,
```
+s0  +s0  +s0  +s0  +s0  +s0  +s0  +s0
+s1  +s3  +s5  +s7  -s7  -s5  -s3  -s1
+s2  +s6  -s6  -s2  -s2  -s6  +s6  +s2
+s3  -s7  -s1  -s5  +s5  +s1  +s7  -s3
+s4  -s4  -s4  +s4  +s4  -s4  -s4  +s4
+s5  -s1  +s7  +s3  -s3  -s7  +s1  -s5
+s6  -s2  +s2  -s6  -s6  +s2  -s2  +s6
+s7  -s5  +s3  -s1  +s1  -s3  +s5  -s7
```
whereas, s0..s7 = scalefactor[0..7], multiplied by sqrt(2) (ie. by 1.414), and multiplied by 4000h (ie. with 14bit fractional part).

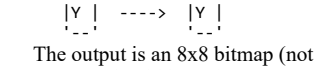
MDEC Data Format

Colored Macroblocks (16x16 pixels) (in 15bpp or 24bpp depth mode)
Each macroblock consists of six blocks: Two low-resolution blocks with color information (Cr,Cb) and four full-resolution blocks with luminance (grayscale) information (Y1,Y2,Y3,Y4). The color blocks are zoomed from 8x8 to 16x16 pixel size, merged with the luminance blocks, and then converted from YUV to RGB format.



Native PSX files are usually containing vertically arranged Macroblocks (eg. allowing to send them to the GPU as 16x240 portion) (JPEG-style horizontally arranged Macroblocks would require to send the data in 16x16 pixel portions to the GPU) (something like 320x16 won't work, since that'd require to wrap from the bottom of the first macroblock to the top of the next macroblock).

Monochrome Macroblocks (8x8 pixel) (in 4bpp or 8bpp depth mode)
Each macroblock consist of only one block: with luminance (grayscale) information (Y), the data comes out as such (it isn't converted to RGB).



The output is an 8x8 bitmap (not 16x16), so it'd be send to the GPU as 8x8 pixel rectangle, or multiple blocks at once as 8x240 pixel rectangle. Since the data isn't RGB, it should be written to Texture memory (and then it can be forwarded to the frame buffer in form of a texture with monochrome 15bit palette with 32 grayscales). Alternately, one could convert the 8bpp image to 24bpp by software (this would allow to use 256 grayscales).

Blocks (8x8 pixels)
An (uncompressed) block consists of 64 values, representing 8x8 pixels. The first (upper-left) value is an absolute value (called "DC" value), the remaining 63 values are relative to the DC value (called "AC" values). After decompression and zig-zag reordering, the data in unfiltered horizontally and vertically (IDCT conversion, ie. the relative "AC" values are converted to absolute "DC" values).

.STR Files
PSX Video files are usually having file extension .STR (for "Streaming").

MDEC vs JPEG
The MDEC data format is very similar to the JPEG file format, the main difference is that JPEG uses Huffman compressed blocks, whilst MDEC uses Run-Length (RL) compressed blocks.
The (uncompressed) blocks are same as in JPEGs, using the same zigzag ordering, AC to DC conversion, and YUV to RGB conversion (ie. the MDEC hardware can be also used to decompress JPEGs, when handling the file header and huffman decompression by software).
Some other difference are that MDEC has only 2 fixed-purpose quant tables, whilst JPEGs <can> use up to 4 general-purpose quant tables. Also, JPEGs <can> use other color resolutions than the 8x8 color info for 16x16 pixels. Whereas, JPEGs <can> do that stuff, but most standard JPEG files aren't actually using 4 quant tables, nor higher color resolution.

Run-Length compressed Blocks
Within each block the DCT information and RLE compressed data is stored:
```
DCT              ;1 halfword
RLE,RLE,RLE,etc. ;0..63 halfwords
EOB              ;1 halfword
```

DCT (1st value)
DCT data has the quantization factor and the Direct Current (DC) reference.
```
15-10 Q    Quantization factor (6 bits, unsigned)
9-0   DC   Direct Current reference (10 bits, signed)
```
Contains the absolute DC value (the upper-left value of the 8x8 block).

RLE (Run length data, for 2nd through 64th value)
```
15-10 LEN  Number of zero AC values to be inserted (6 bits, unsigned)
9-0   AC   Relative AC value (10 bits, signed)
```
Example: AC values "000h,000h,123h" would be compressed as "(2 shl 10)+123h".

EOB (End Of Block)
```
15-0  End-code (Fixed, FE00h)
```
Indicates the end of a 8x8 pixel block. The rest of the block is padded with zero AC values.

Dummy halfwords
Data is sent in units of words (or, when using DMA, even in units of 32-words), which is making it neccessary to send some dummy halfwords (unless the compressed data size should match up the transfer unit). The value FE00h can be used as dummy value: When FE00h appears at the begin of a new block, or after the end of block, then it is simply ignored by the hardware (if it occurs elsewhere, then it acts as EOB end code, as described above).

Sound Processing Unit (SPU)

SPU Overview
SPU ADPCM Samples
SPU ADPCM Pitch
SPU Volume and ADSR Generator
SPU Voice Flags
SPU Noise Generator
SPU Control and Status Register
SPU Memory Access
SPU Interrupt
SPU Reverb Registers
SPU Reverb Formula
SPU Reverb Examples
SPU Unknown Registers

SPU Overview

SPU I/O Port Summary
```
1F801C00h..1F801D7Fh - Voice 0..23 Registers (eight 16bit regs per voice)
1F801D80h..1F801D87h - SPU Control (volume)
1F801D88h..1F801D9Fh - Voice 0..23 Flags (six 1bit flags per voice)
1F801DA2h..1F801DBFh - SPU Control (memory, control, etc.)
1F801DC0h..1F801DFFh - Reverb configuration area
1F801E00h..1F801E5Fh - Voice 0..23 Internal Registers
1F801E60h..1F801E7Fh - Unknown?
1F801E80h..1F801FFFh - Unused?
```

SPU Memory layout (512Kbyte RAM)
```
00000h-003FFh  CD Audio left  (1Kbyte) ;\CD Audio before Volume processing
00400h-007FFh  CD Audio right (1Kbyte) ;/signed 16bit samples at 44.1kHz
00800h-00BFFh  Voice 1 mono   (1Kbyte) ;\Voice 1 and 3 after ADSR processing
00C00h-00FFFh  Voice 3 mono   (1Kbyte) ;/signed 16bit samples at 44.1kHz
01000h-xxxxxh  ADPCM Samples  (first 16bytes usually contain a Sine wave)
xxxxxh-7FFFFh  Reverb work area
```
As shown above, the first 4Kbytes are used as special capture buffers, and, if desired, one can also use the Reverb hardware to capture output from other voice(s). The SPU memory is not mapped to the CPU bus, it can be accessed only via I/O, or via DMA transfers (DMA4).

Voices
The SPU has 24 hardware voices. These voices can be used to reproduce sample data, noise or can be used as frequency modulator on the next voice. Each voice has it's own programmable ADSR envelope filter. The main volume can be programmed independently for left and right output.

Voice Capabilities
All 24 voices are having exactly the same capabilities(?), with the exception that Voice 1 and 3 are having a special Capture feature (see SPU Memory map).
There seems to be no way to produce square waves (without storing a square wavefrom in memory... although, since SPU RAM isn't connected to the CPU bus, the "useless" DMA for square wave data wouldn't slowdown the CPU bus)?

Additional Sound Inputs
External Audio can be input (from the Expansion Port?), and the CDROM drive can be commanded to playback normal Audio CDs (via Play command), or XA-ADPCM sectors (via Read command), and to pass that data to the SPU.

Unstable and Delayed I/O
The SPU occassionally seems to "miss" I/O writes (not sure if that can be fixed by any Memory Control settings?), a stable workaround is too write all values twice (except of course, Fifo writes). The SPU seems to process written values at 44100Hz rate (so it may take 1/44100 seconds (300h clock cycles) until it has actually realized the new value).

Mono/Stereo Audio Output
The standard PSX Audio cables have separate Left/Right signals, that is good for stereo TVs, but, when using a normal mono TV, only one of the two audio signals (Left or Right) can be connected. PSX programs should thus offer an option to disable stereo effects, and to output an equal volume to both cables.

SPU Bus-Width
The SPU is connected to a 16bit databus. 8bit/16bit/32bit reads and 16bit/32bit writes are implemented. However, 8bit writes are NOT implemented: 8bit writes to ODD addresses are simply ignored (without causing any exceptions), 8bit writes to EVEN addresses are executed as 16bit writes (eg. "movp r1,12345678h, movb [spu_port],r1" will write 5678h instead of 78h).

SPU ADPCM Samples

The SPU supports only ADPCM compressed samples (uncompressed samples seem to be totally unsupported; leaving apart that one can write uncompressed 16bit PCM samples to the Reverb Buffer, which can be then output at 22050Hz, as long as they aren't overwritten by the hardware).

1F801C06h+N*10h - Voice 0..23 ADPCM Start Address (R/W)
This register holds the sample start address (not the current address, ie. the register doesn't increment during playback).
```
15-0   Startaddress of sound in Sound buffer (in 8-byte units)
```
Writing to this register has no effect on the currently playing voice.
The start address is copied to the current address upon Key On.

1F801C0Eh+N*10h - Voice 0..23 ADPCM Repeat Address (R/W)
If the hardware finds an ADPCM header with Loop-Start-Bit, then it copies the current address to the repeat addresss register.
If the hardware finds an ADPCM header with Loop-Stop-Bit, then it copies the repeat addresss register setting to the current address; that, <after> playing the current ADPCM block.
```
15-0  Address sample loops to at end (in 8-byte units)
```
Normally, repeat works automatically via the above start/stop bits, and software doesn't need to deal with the Repeat Address Register. However, reading from it may be useful to sense if the hardware has reached a start bit, and writing may be also useful in some cases, eg. to redirect a one-shot sample (with stop-bit, but without any start-bits) to a silent-loop located elsewhere in memory.

Sample Data (SPU-ADPCM)
Samples consist of one or more 16-byte blocks:
```
00h       Shift/Filter (reportedly same as for CD-XA) (see there)
01h       Flag Bits (see below)
02h       Compressed Data (LSBs=1st Sample, MSBs=2nd Sample)
03h       Compressed Data (LSBs=3rd Sample, MSBs=4th Sample)
04h       Compressed Data (LSBs=5th Sample, MSBs=6th Sample)
...       ...
0Fh       Compressed Data (LSBs=27th Sample, MSBs=28th Sample)
```

Flag Bits (in 2nd byte of ADPCM Header)
```
0   Loop End    (0=No change, 1=Set ENDX flag and Jump to [1F801C0Eh+N*10h])
1   Loop Repeat (0=Force Release and set ADSR Level to Zero; only if Bit0=1)
2   Loop Start  (0=No change, 1=Copy current address to [1F801C0Eh+N*10h])
3-7 Unknown    (usually 0)
```
Possible combinations for Bit0-1 are:
```
Code 0 = Normal     (continue at next 16-byte block)
Code 1 = End+Mute   (jump to Loop-address, set ENDX flag, Release, Env=0000h)
Code 2 = Ignored    (same as Code 0)
Code 3 = End+Repeat (jump to Loop-address, set ENDX flag)
```

Looped and One-shot Samples
The Loop Start/End flags in the ADPCM Header allow to play one or more sample block(s) in a loop, that can be either all block(s) endless repeated, or only the last some block(s) of the sample.
There's no way to stop the output, so a one-shot sample must be followed by dummy block (with Loop Start/End flags both set, and all data nibbles set to zero; so that the block gets endless repeated, but doesn't produce any sound).

SPU-ADPCM vs XA-ADPCM
The PSX supports two ADPCM formats: SPU-ADPCM (as described above), and XA-ADPCM. XA-ADPCM is decompressed by the CDROM Controller, and sent directly to the sound mixer, without needing to store the data in SPU RAM, nor needing to use a Voice channel.
The actual decompression algorithm is the same for both formats. However, the XA nibbles are arranged in different order, and XA uses 2x28 nibbles per block (instead of 2x14), XA blocks can contain mono or stereo data, XA supports only two sample rates, and, XA doesn't support looping.

SPU ADPCM Pitch

1F801C04h+N*10h - Voice 0..23 ADPCM Sample Rate (R/W) (VxPitch)
```
0-15  Sample rate (0=stop, 4000h=fastest, 4001h..FFFFh=usually same as 4000h)
```
Defines the ADPCM sample rate (1000h = 44100Hz). This register (and PMON) does affect only the ADPCM sample frequency (but not on the Noise frequency, which is defined - and shared for all voices - in the SPUCNT register).

1F801D90h - Voice 0..23 Pitch Modulation Enable Flags (PMON)
Pitch modulation allows to generate "Frequency Sweep" effects by mis-using the amplitude from channel (x-1) as pitch factor for channel (x).
```
0     Unknown... Unused?
1-23  Flags for Voice 1..23 (0=Normal, 1=Modulate by Voice 0..22)
24-31 Not used
```
For example, output a very loud 1Hz sine-wave on channel 4 (with ADSR volume 4000h, and with Left/Right volume=0; unless you actually want to output it to the speaker). Then additionally output a 2kHz sine wave on channel 5 with PMON.Bit5 set. The "2kHz" sound should then repeatedly sweep within 1kHz..3kHz range (or, for a more decent sweep in 1.8kHz..2.2kHz range, drop the ADSR volume of channel 4).

Pitch Counter
The pitch counter is adjusted at 44100Hz rate as follows:
```
Step = VxPitch                  ;range +0000h..+FFFFh (0...705.6 kHz)
IF PMON.Bit(x)=1 AND (x>0)      ;pitch modulation enable
  Factor = VxOUTX(x-1)          ;range -8000h..+7FFFh (prev voice amplitude)
  Factor = Factor+8000h         ;range +0000h..+FFFFh (factor = 0.00 .. 1.99)
  Step=SignExpand16to32(Step)   ;hardware glitch on VxPitch>7FFFh, make sign
  Step = (Step * Factor) SAR 15 ;range 0..1FFFFh (glitchy if VxPitch>7FFFh)
  Step=Step AND 0000FFFFh       ;hardware glitch on VxPitch>7FFFh, kill sign
IF Step>3FFFh then Step=4000h   ;range +0000h..+3FFFh (0.. 176.4kHz)
Counter = Counter + Step
```
Counter.Bit12 and up indicates the current sample (within a ADPCM block).
Counter.Bit3..11 are used as 8bit gaussian interpolation index.

Maximum Sound Frequency
The Mixer and DAC supports a 44.1kHz output rate (allowing to produce max 22.1kHz tones). The Reverb unit supports only half the frequency.
The pitch counter supports sample rates up to 176.4kHz. However, exceeding the 44.1kHz limit causes the hardware to skip samples (or actually: to apply incomplete interpolation on the 'skipped' samples).
VxPitch can be theoretically 0..FFFFh (max 705.6kHz), normally 4000h..FFFFh are simply clipped to max=4000h (176.4kHz). Except, 4000h..FFFFh could be used with pitch modulation (as they are multiplied by 0.00..1.99 before clipping; in practice this works only for 4000h..7FFFh; as values 8000h..FFFFh are mistaken as signed values).

4-Point Gaussian Interpolation
Interpolation is applied on the 4 most recent 16bit ADPCM samples (new,old,older,oldest), using bit4-11 of the pitch counter as 8bit interpolation index (i=00h..FFh):
```
out =       ((gauss[0FFh-i] * oldest) SAR 15)
out = out + ((gauss[1FFh-i] * older)  SAR 15)
out = out + ((gauss[100h+i] * old)    SAR 15)
out = out + ((gauss[000h+i] * new)    SAR 15)
```
The Gauss table contains the following values (in hex):
```
-001h,-001h,-001h,-001h,-001h,-001h,-001h,-001h ;\
-001h,-001h,-001h,-001h,-001h,-001h,-001h,-001h ;
0000h,0000h,0000h,0000h,0000h,0000h,0000h,0001h ;
```

```
0001h,0001h,0001h,0002h,0002h,0002h,0003h,0003h ;
0003h,0004h,0004h,0005h,0005h,0006h,0007h,0007h ;
0008h,0009h,0009h,000Ah,000Bh,000Ch,000Dh,000Eh ;
000Fh,0010h,0011h,0012h,0013h,0015h,0016h,0018h ; entry
0019h,001Bh,001Ch,001Eh,0020h,0021h,0023h,0025h ; 000h..07Fh
0027h,0029h,002Ch,002Eh,0030h,0033h,0035h,0038h ;
003Ah,003Dh,0040h,0043h,0046h,0049h,004Dh,0050h ;
0054h,0057h,005Bh,005Fh,0063h,0067h,006Bh,006Fh ;
0074h,0078h,007Dh,0082h,0087h,008Ch,0091h,0096h ;
009Ch,00A1h,00A7h,00ADh,00B3h,00BAh,00C0h,00C7h ;
00CDh,00D4h,00DBh,00E3h,00EAh,00F2h,00FAh,0101h ;
010Ah,0112h,011Bh,0123h,012Ch,0135h,013Fh,0148h ;
0152h,015Ch,0166h,0171h,017Bh,0186h,0191h,019Ch ;/
01A8h,01B4h,01C0h,01CCh,01D9h,01E5h,01F2h,0200h ;\
020Dh,021Bh,0229h,0237h,0246h,0255h,0264h,0273h ;
0283h,0293h,02A3h,02B4h,02C4h,02D6h,02E7h,02F9h ;
030Bh,031Dh,0330h,0343h,0356h,036Ah,037Eh,0392h ;
03A7h,03BCh,03D1h,03E7h,03FCh,0413h,042Ah,0441h ;
0458h,0470h,0488h,04A0h,04B9h,04D2h,04ECh,0506h ;
0520h,053Bh,0556h,0572h,058Eh,05AAh,05C7h,05E4h ; entry
0601h,061Fh,063Eh,065Ch,067Ch,069Bh,06BBh,06DCh ; 080h..0FFh
06FDh,071Eh,0740h,0762h,0784h,07A7h,07CBh,07EFh ;
0813h,0838h,085Dh,0883h,08A9h,08D0h,08F7h,091Eh ;
0946h,096Fh,0998h,09C1h,09EBh,0A16h,0A40h,0A6Ch ;
0A98h,0AC4h,0AF1h,0B1Eh,0B4Ch,0B7Ah,0BA9h,0BD8h ;
0C07h,0C38h,0C68h,0C99h,0CCBh,0CFDh,0D30h,0D63h ;
0D97h,0DCBh,0E00h,0E35h,0E6Bh,0EA1h,0ED7h,0F0Fh ;
0F46h,0F7Fh,0FB7h,0FF1h,102Ah,1065h,109Fh,10DBh ;
1116h,1153h,118Fh,11CDh,120Bh,1249h,1288h,12C7h ;/
1307h,1347h,1388h,13C9h,140Bh,144Dh,1490h,14D4h ;\
1517h,155Ch,15A0h,15E6h,162Ch,1672h,16B9h,1700h ;
1747h,1790h,17D8h,1821h,186Bh,18B5h,1900h,194Bh ;
1996h,19E2h,1A2Eh,1A7Bh,1AC8h,1B16h,1B64h,1BB3h ;
1C02h,1C51h,1CA1h,1CF1h,1D42h,1D93h,1DE5h,1E37h ;
1E89h,1EDCh,1F2Fh,1F82h,1FD6h,202Ah,207Fh,20D4h ;
2129h,217Fh,21D5h,222Ch,2282h,22DAh,2331h,2389h ; entry
23E1h,2439h,2492h,24EBh,2545h,259Eh,25F8h,2653h ; 100h..17Fh
26ADh,2708h,2763h,27BEh,281Ah,2876h,28D2h,292Eh ;
298Bh,29E7h,2A44h,2AA1h,2AFFh,2B5Ch,2BBAh,2C18h ;
2C76h,2CD4h,2D33h,2D91h,2DF0h,2E4Fh,2EAEh,2F0Dh ;
2F6Ch,2FCCh,302Bh,308Bh,30EAh,314Ah,31AAh,3209h ;
3269h,32C9h,3329h,3389h,33E9h,3449h,34A9h,3509h ;
3569h,35C9h,3629h,3689h,36E8h,3748h,37A8h,3807h ;
3867h,38C6h,3926h,3985h,39E4h,3A43h,3AA2h,3B00h ;
3B5Fh,3BBDh,3C1Bh,3C79h,3CD7h,3D35h,3D92h,3DEFh ;/
3E4Ch,3EA9h,3F05h,3F62h,3FBDh,4019h,4074h,40D0h ;\
412Ah,4185h,41DFh,4239h,4292h,42EBh,4344h,439Ch ;
43F4h,444Ch,44A3h,44FAh,4550h,45A6h,45FCh,4651h ;
46A6h,46FAh,474Eh,47A1h,47F4h,4846h,4898h,48E9h ;
493Ah,498Ah,49D9h,4A29h,4A77h,4AC5h,4B13h,4B5Fh ;
4BACh,4BF7h,4C42h,4C8Dh,4CD7h,4D20h,4D68h,4DB0h ;
4DF7h,4E3Eh,4E84h,4EC9h,4F0Eh,4F52h,4F95h,4FD7h ; entry
5019h,505Ah,509Ah,50DAh,5118h,5156h,5194h,51D0h ; 180h..1FFh
520Ch,5247h,5281h,52BAh,52F3h,532Ah,5361h,5397h ;
53CCh,5401h,5434h,5467h,5499h,54CAh,54FAh,5529h ;
5558h,5585h,55B2h,55DEh,5609h,5632h,565Bh,5684h ;
56ABh,56D1h,56F6h,571Bh,573Eh,5761h,5782h,57A3h ;
57C3h,57E2h,57FFh,581Ch,5838h,5853h,586Dh,5886h ;
589Eh,58B5h,58CBh,58E0h,58F4h,5907h,5919h,592Ah ;
593Ah,5949h,5958h,5965h,5971h,597Ch,5986h,598Fh ;
5997h,599Eh,59A4h,59A9h,59ADh,59B0h,59B2h,59B3h ;/
```

The PSX table is a bit different as the SNES table: Values up to 3569h are smaller as on SNES, the remaining values are bigger as on SNES, and the width of the PSX table entries is 4bit higher as on SNES.
The PSX table is slightly bugged: Theoretically, each four values (gauss[000h+i], gauss[0FFh-i], gauss[100h+i], gauss[1FFh-i]) should sum up to 8000h, but in practice they do sum up to 7F7Fh..7F81h (fortunately the PSX sum doesn't exceed the 8000h limit; meaning that the PSX interpolations won't overflow, which has been a hardware glitch on the SNES).

Waveform Examples

```
Incoming ADPCM Data ---> Interpolated Data
 _  _  _  _
| || || || ||          .   .   .   .    Nibbles=79797979, Filter=0
| || || || ||     ---> / \ / \ / \ / \   HALF-volume ZIGZAG-wave
| ||_|| ||_|| |_          '   '   '   '
 ___   ___
|   |   |   |             .'.     .'.     Nibbles=77997799, Filter=0
|   |   |   |      --->  /   \   /   \    FULL-volume SINE-wave
|   |___|   |___        '     '.'     '.
 _______                   ___
|       |                .'   '.         Nibbles=77779999, Filter=0
|       |          --->  /       \        SQUARE wave (with rounded edges)
|       |_______        '         '.____
 _____     _              __
|     |_  _|  _|          .' ''.    .'   Nibbles=7777CC44, Filter=0
|       |___|      --->  /       '..'    CUSTOM wave-form
|                       '
 ___   __
|   |___|  |    _        \ ! /  .  \ ! /  Nibbles=77DE9HZK, Filter=V
|_      ___|  _|   --->  - + -  +  - + -  SOLAR STORM wave-form
__|   |______|__        / ! \  '  / ! \
```

SPU Volume and ADSR Generator

1F801C08h+N*10h - Voice 0..23 Attack/Decay/Sustain/Release (ADSR) (32bit)

```
____lower 16bit (at 1F801C08h+N*10h)_____________________________________
15    Attack Mode       (0=Linear, 1=Exponential)
-     Attack Direction  (Fixed, always Increase) (until Level 7FFFh)
14-10 Attack Shift      (0..1Fh = Fast..Slow)
9-8   Attack Step       (0..3 = "+7,+6,+5,+4")
-     Decay Mode        (Fixed, always Exponential)
```

```
-     Decay Direction   (Fixed, always Decrease) (until Sustain Level)
7-4   Decay Shift       (0..0Fh = Fast..Slow)
-     Decay Step        (Fixed, always "-8")
3-0   Sustain Level     (0..0Fh)  ;Level=(N+1)*800h
____upper 16bit (at 1F801C0Ah+N*10h)_____________________________
31    Sustain Mode      (0=Linear, 1=Exponential)
30    Sustain Direction (0=Increase, 1=Decrease) (until Key OFF flag)
29    Not used?         (should be zero)
28-24 Sustain Shift     (0..1Fh = Fast..Slow)
23-22 Sustain Step      (0..3 = "+7,+6,+5,+4" or "-8,-7,-6,-5") (inc/dec)
21    Release Mode      (0=Linear, 1=Exponential)
-     Release Direction (Fixed, always Decrease) (until Level 0000h)
20-16 Release Shift     (0..1Fh = Fast..Slow)
-     Release Step      (Fixed, always "-8")
```

The Attack phase gets started when the software sets the voice ON flag (see below), the hardware does then automatically go through Attack/Decay/Sustain, and switches from Sustain to Release when the software sets the Key OFF flag.

1F801D80h - Mainvolume left
1F801D82h - Mainvolume right
1F801C00h+N*10h - Voice 0..23 Volume Left
1F801C02h+N*10h - Voice 0..23 Volume Right

Fixed Volume Mode (when Bit15=0):

```
15    Must be zero      (0=Volume Mode)
0-14  Voice volume/2    (-4000h..+3FFFh = Volume -8000h..+7FFEh)
```

Sweep Volume Mode (when Bit15=1):

```
15    Must be set       (1=Sweep Mode)
14    Sweep Mode        (0=Linear, 1=Exponential)
13    Sweep Direction   (0=Increase, 1=Decrease)
12    Sweep Phase       (0=Positive, 1=Negative)
7-11  Not used?         (should be zero)
6-2   Sweep Shift       (0..1Fh = Fast..Slow)
1-0   Sweep Step        (0..3 = "+7,+6,+5,+4" or "-8,-7,-6,-5") (inc/dec)
```

Sweep is another Volume envelope, additionally to the ADSR volume envelope (unlike ADSR, sweep can be used for stereo effects, such like blending from left to right).
Sweep starts at the current volume (which can be set via Bit15=0, however, caution - the Bit15=0 setting isn't applied until the next 44.1kHz cycle; so setting the initial level with Bit15=0, followed by the sweep parameter with Bit15=1 works only if there's a suitable delay between the two operations). Once when sweep is started, the current volume level increases to +7FFFh, or decreases to 0000h.
Sweep Phase should be equal to the sign of the current volume (not yet tested, in the negative mode it does probably "increase" to -7FFFh?). The Phase bit seems to have no effect in Exponential Decrease mode.

1F801DB0h - CD Audio Input Volume (for normal CD-DA, and compressed XA-ADPCM)
1F801DB4h - External Audio Input Volume

```
0-15  Volume Left   (-8000h..+7FFFh)
16-31 Volume Right  (-8000h..+7FFFh)
```

Note: The CDROM controller supports additional CD volume control (including ability to convert stereo CD output to mono, or to swap left/right channels).

Envelope Operation depending on Shift/Step/Mode/Direction

```
AdsrCycles = 1 SHL Max(0,ShiftValue-11)
AdsrStep = StepValue SHL Max(0,11-ShiftValue)
IF exponential AND increase AND AdsrLevel>6000h THEN AdsrCycles=AdsrCycles*4
IF exponential AND decrease THEN AdsrStep=AdsrStep*AdsrLevel/8000h
Wait(AdsrCycles)              ;cycles counted at 44.1kHz clock
AdsrLevel=AdsrLevel+AdsrStep  ;saturated to 0..+7FFFh
```

Exponential Increase is a fake (simply changes to a slower linear increase rate at higher volume levels).

1F801C0Ch+N*10h - Voice 0..23 Current ADSR volume (R/W)

```
15-0  Current ADSR Volume  (0..+7FFFh) (or -8000h..+7FFFh on manual write)
```

Reportedly Release can go down to -1 (FFFFh), but that isn't true; and release ends at 0... or does THAT depend on an END flag found in the sample-data?
The register is read/writeable, writing allows to let the ADSR generator to "jump" to a specific volume level. But, ACTUALLY, the ADSR generator does overwrite the setting (from another internal register) whenever applying a new Step?!

1F801DB8h - Current Main Volume Left/Right
1F801E00h+voice*04h - Voice 0..23 Current Volume Left/Right

```
0-15  Current Volume Left  (-8000h..+7FFFh)
16-31 Current Volume Right (-8000h..+7FFFh)
```

These are internal registers, normally not used by software (the Volume settings are usually set via Ports 1F801D80h and 1F801C00h+N*10h).

Note
Negative volumes are phase inverted, otherwise same as positive.

SPU Voice Flags

1F801D88h - Voice 0..23 Key ON (Start Attack/Decay/Sustain) (KON) (W)

```
0-23  Voice 0..23 On  (0=No change, 1=Start Attack/Decay/Sustain)
24-31 Not used
```

Starts the ADSR Envelope, and automatically initializes ADSR Volume to zero, and copies Voice Start Address to Voice Repeat Address.

1F801D8Ch - Voice 0..23 Key OFF (Start Release) (KOFF) (W)

```
0-23  Voice 0..23 Off (0=No change, 1=Start Release)
24-31 Not used
```

For a full ADSR pattern, OFF would be usually issued in the Sustain period, however, it can be issued at any time (eg. to abort Attack, skip the Decay and Sustain periods, and switch immediately to Release).

1F801D9Ch - Voice 0..23 ON/OFF (status) (ENDX) (R)

```
0-23  Voice 0..23 Status (0=Newly Keyed On, 1=Reached LOOP-END)
24-31 Not used
```

The bits get CLEARED when setting the corresponding KEY ON bits.
The bits get SET when reaching an LOOP-END flag in ADPCM header.bit0.

R/W
Key On and Key Off should be treated as write-only (although, reading returns the most recently 32bit value, this doesn't doesn't provide any status information about whether sound is on or off).

The on/off (status) (ENDX) register should be treated read-only (writing is possible in so far that the written value can be read-back for a short moment, however, thereafter the hardware is overwriting that value).

SPU Noise Generator

1F801D94h - Voice 0..23 Noise mode enable (NON)

```
0-23  Voice 0..23 Noise (0=ADPCM, 1=Noise)
24-31 Not used
```

SPU Noise Generator

The signed 16bit output Level is calculated as so (repeated at 44.1kHz clock):

```
Wait(1 cycle)          ;at 44.1kHz clock
Timer=Timer-NoiseStep  ;subtract Step (4..7)
ParityBit = NoiseLevel.Bit15 xor Bit12 xor Bit11 xor Bit10 xor 1
IF Timer<0 then NoiseLevel = NoiseLevel*2 + ParityBit
IF Timer<0 then Timer=Timer+(20000h SHR NoiseShift)  ;reload timer once
IF Timer<0 then Timer=Timer+(20000h SHR NoiseShift)  ;reload again if needed
```

Note that the Noise frequency is solely controlled by the Shift/Step values in SPUCNT register (the ADPCM Sample Rate has absolutely no effect on noise), so when using noise for multiple voices, all of them are forcefully having the same frequency; the only workaround is to store a random ADPCM pattern in SPU RAM, which can be then used with any desired sample rate(s).

SPU Control and Status Register

1F801DAAh - SPU Control Register (SPUCNT)

```
15    SPU Enable              (0=Off, 1=On)       (Don't care for CD Audio)
14    Mute SPU                (0=Mute, 1=Unmute)  (Don't care for CD Audio)
13-10 Noise Frequency Shift   (0..0Fh = Low .. High Frequency)
9-8   Noise Frequency Step    (0..03h = Step "4,5,6,7")
7     Reverb Master Enable    (0=Disabled, 1=Enabled)
6     IRQ9 Enable (0=Disabled/Acknowledge, 1=Enabled; only when Bit15=1)
5-4   Sound RAM Transfer Mode (0=Stop, 1=ManualWrite, 2=DMAwrite, 3=DMAread)
3     External Audio Reverb   (0=Off, 1=On)
2     CD Audio Reverb         (0=Off, 1=On) (for CD-DA and XA-ADPCM)
1     External Audio Enable   (0=Off, 1=On)
0     CD Audio Enable         (0=Off, 1=On) (for CD-DA and XA-ADPCM)
```

Changes to bit0-5 aren't applied immediately; after writing to SPUCNT, it'd be usually recommended to wait until the LSBs of SPUSTAT are updated accordingly. Before setting a new Transfer Mode, it'd be recommended first to set the "Stop" mode (and, again, wait until Stop is applied in SPUSTAT).

1F801DAEh - SPU Status Register (SPUSTAT) (R)

```
15-12 Unknown/Unused (seems to be usually zero)
11    Writing to First/Second half of Capture Buffers (0=First, 1=Second)
10    Data Transfer Busy Flag          (0=Ready, 1=Busy)
9     Data Transfer DMA Read Request   (0=No, 1=Yes)
8     Data Transfer DMA Write Request  (0=No, 1=Yes)
7     Data Transfer DMA Read/Write Request ;seems to be same as SPUCNT.Bit5
6     IRQ9 Flag                        (0=No, 1=Interrupt Request)
5-0   Current SPU Mode   (same as SPUCNT.Bit5-0, but, applied a bit delayed)
```

When switching SPUCNT to DMA-read mode, status bit9 and bit7 aren't set immediately (apparently the SPU is first internally collecting the data in the Fifo, before transferring it).

Bit11 indicates if data is currently written to the first or second half of the four 1K-byte capture buffers (for CD Audio left/right, and voice 1/3). Note: Bit11 works only if Bit2 and/or Bit3 of Port 1F801DACh are set.

The SPUSTAT register should be treated read-only (writing is possible in so far that the written value can be read-back for a short moment, however, thereafter the hardware is overwriting that value).

SPU Memory Access

1F801DA6h - Sound RAM Data Transfer Address

```
15-0  Address in sound buffer divided by eight
```

Used for manual write and DMA read/write SPU memory. Writing to this registers stores the written value in 1F801DA6h, and does additional store the value (multiplied by 8) in another internal "current address" register (that internal register does increment during transfers, whilst the 1F801DA6h value DOESN'T increment).

1F801DA8h - Sound RAM Data Transfer Fifo

```
15-0  Data (max 32 halfwords)
```

Used for manual-write. Not sure if it can be also used for manual read?

1F801DACh - Sound RAM Data Transfer Control (should be 0004h)

```
15-4   Unknown/no effect?                     (should be zero)
3-1    Sound RAM Data Transfer Type (see below) (should be 2)
0      Unknown/no effect?                     (should be zero)
```

The Transfer Type selects how data is forwarded from Fifo to SPU RAM:

```
__Transfer Type___Halfwords in Fifo________Halfwords written to SPU RAM__
0,1,6,7  Fill     A,B,C,D,E,F,G,H,...,X    X,X,X,X,X,X,X,X,...
2        Normal   A,B,C,D,E,F,G,H,...,X    A,B,C,D,E,F,G,H,...
3        Rep2     A,B,C,D,E,F,G,H,...,X    A,A,C,C,E,E,G,G,...
4        Rep4     A,B,C,D,E,F,G,H,...,X    A,A,A,A,E,E,E,E,...
5        Rep8     A,B,C,D,E,F,G,H,...,X    H,H,H,H,H,H,H,H,...
```

Rep2 skips the 2nd halfword, Rep4 skips 2nd..4th, Rep8 skips 1st..7th.

Fill uses only the LAST halfword in Fifo, that might be useful for memfill purposes, although, the length is probably determined by the number of writes to the Fifo (?) so one must still issue writes for ALL halfwords...?

Note:

The above rather bizarre results apply to WRITE mode. In READ mode, the register causes the same halfword to be read 2/4/8 times (for rep2/4/8).

SPU RAM Manual Write

- Be sure that [1F801DACh] is set to 0004h
- Set SPUCNT to "Stop" (and wait until it is applied in SPUSTAT)
- Set the transfer address
- Write 1..32 halfword(s) to the Fifo

- Set SPUCNT to "Manual Write" (and wait until it is applied in SPUSTAT)
- Wait until Transfer Busy in SPUSTAT goes off (that, AFTER above apply-wait)

For multi-block transfers: Repeat the above last three steps (that is rarely done by any games, but it is done by the BIOS intro; observe that waiting for SPUCNT writes being applied in SPUSTAT won't work in that case (since SPUCNT was already in manual write mode from previous block), so one must instead use some hardcoded delay of at least 300h cycles; the BIOS is using a much longer bizarre delay though).

SPU RAM DMA-Write

- Be sure that [1F801DACh] is set to 0004h
- Set SPUCNT to "Stop" (and wait until it is applied in SPUSTAT)
- Set the transfer address
- Set SPUCNT to "DMA Write" (and wait until it is applied in SPUSTAT)
- Start DMA4 at CPU Side (blocksize=10h, control=01000201h)
- Wait until DMA4 finishes (at CPU side)

SPU RAM Manual-Read

As by now, there's no known method for reading SPU RAM without using DMA.

SPU RAM DMA-Read (stable reading, with [1F801014h].bit24-27 = nonzero)

- Be sure that [1F801014h] is set to 220931E1h (bit24-27 MUST be nonzero)
- Be sure that [1F801DACh] is set to 0004h
- Set SPUCNT to "Stop" (and wait until it is applied in SPUSTAT)
- Set the transfer address
- Set SPUCNT to "DMA Read" (and wait until it is applied in SPUSTAT)
- Start DMA4 at CPU Side (blocksize=10h, control=01000200h)
- Wait until DMA4 finishes (at CPU side)

SPU RAM DMA-Read (unstable reading, with [1F801014h].bit24-27 = zero)

Below describes some dirt effects and some trickery to get around those dirt effects.

```
Below problems (and workarounds) apply ONLY if [1F801014h].bit24-27 = zero.
Ie. below info describes what happens when [1F801014h] is mis-initialized.
Normally one should set [1F801014h]=220931E1h (and can ignore below info).
```

With [1F801014h].bit24-27=zero, reading SPU RAM via DMA works glitchy:
The first received halfword within each block is FFFFh. So with a DMA blocksize of 10h words (=20h halfwords), the following is received:

```
1st block:   FFFFh, halfwords[00h..1Eh]
2nd block:   FFFFh, halfwords[20h..3Eh]
etc.
```

that'd theoretically match the SPU Fifo Size, but, because of the inserted FFFFh value, the last Fifo entry isn't received, ie. halfword[1Fh,3Fh] are lost. As a workaround, one can increase the DMA blocksize to 11h words, and then the following is received:

```
1st block:   FFFFh, halfwords[00h..1Eh], twice halfword[1Fh]
2nd block:   FFFFh, halfwords[20h..3Eh], twice halfword[3Fh]
etc.
```

this time, all data is received, but after the transfer one must still remove the FFFFh values, and the duplicated halfwords by software. Aside from the <inserted> FFFFh values there are occassionaly some unstable halfwords ORed by FFFFh (or ORed by other garbage values), this can be fixed by using "rep2" mode, which does then receive:

```
1st block:   FFFFh, halfwords[00h,00h,..0Eh,0Eh], triple halfword[0Fh]
2nd block:   FFFFh, halfwords[10h,10h,..1Eh,1Eh], triple halfword[1Fh]
etc.
```

again, remove the first halfword (FFFFh) and the last halfword, and, take the duplicated halfwords ANDed together. Unstable values occur only every 32 halfwords or so (probably when the SPU is simultaneously reading ADPCM data), but do never occur on two continous halfwords, so, even if one halfword was ORed by garbage, the other halfword is always correct, and the result of the ANDed halfwords is 100% stable.
Note: The unstable reading does NOT occur always, when resetting the PSX a couple of times it does occassionally boot-up with totally stable reading, since there is no known way to activate the stable "mode" via I/O ports, the stable/unstable behaviour does eventually depend on internal clock dividers/multipliers, and whether they are starting in sync with the CPU or not.
Caution: The "rep2" trick cannot be used in combination with reverb (reverb seems to be using the Port 1F801DACh Sound RAM Data Transfer Control, too).

SPU Interrupt

1F801DA4h - Sound RAM IRQ Address (IRQ9)

```
15-0  Address in sound buffer divided by eight
```

See also: SPUCNT (IRQ enable/disable/acknowledge) and SPUSTAT (IRQ flag).

Voice Interrupt

Triggers an IRQ when a voice reads ADPCM data from the IRQ address.
Mind that ADPCM cannot be stopped (uh, except, probably they CAN be stopped, by setting the sample rate to zero?), all voices are permanently reading data from SPU RAM - even in Noise mode, even if the Voice Volume is zero, and even if the ADSR pattern has finished the Release period - so even inaudible voices can trigger IRQs. To prevent unwanted IRQs, best set all unused voices to an endless looped dummy ADPCM block.
For stable IRQs, the IRQ address should be aligned to the 16-byte ADPCM blocks. If if the IRQ address is in the middle of a 16-byte ADPCM block, then the IRQ doesn't seem to trigger always (unknown why, but it seems to occassionally miss IRQs, even if the block gets repeated several times).

Capture Interrupt

Setting the IRQ address to 0000h..01FFh (aka byte address 00000h..00FFFh) will trigger IRQs on writes to the four capture buffers. Each of the four buffers contains 400h bytes (=200h samples), so the IRQ rate will be around 86.13Hz (44100Hz/200h).
CD-Audio capture is always active (even CD-Audio output is disabld in SPUCNT, and even if the drive door is open). Voice capture is (probably) also always active (even if the corresponding voice is off).
Capture IRQs do NOT occur if 1F801DACh.bit3-2 are both zero.

Reverb Interrupt

Reverb is also triggering interrupts if the IRQ address is located in the reverb buffer area. Unknown <which> of the various reverb read(s) and/or reverb write(s) are triggering interrupts.

Data Transfers

Data Transfers (usually via DMA4) to/from SPU-RAM do also trap SPU interrupts.

Note

IRQ Address is used by Metal Gear Solid, Legend of Mana, Tokimeki Memorial 2, Crash Team Racing, The Misadventures of Tron Bonne, and (somewhat?) by Need For Speed 3.

SPU Reverb Registers

Reverb Volume and Address Registers (R/W)

Port	Reg	Name	Type	Expl.
1F801D84h	spu	vLOUT	volume	Reverb Output Volume Left
1F801D86h	spu	vROUT	volume	Reverb Output Volume Right
1F801DA2h	spu	mBASE	base	Reverb Work Area Start Address in Sound RAM
1F801DC0h	rev00	dAPF1	disp	Reverb APF Offset 1
1F801DC2h	rev01	dAPF2	disp	Reverb APF Offset 2
1F801DC4h	rev02	vIIR	volume	Reverb Reflection Volume 1
1F801DC6h	rev03	vCOMB1	volume	Reverb Comb Volume 1
1F801DC8h	rev04	vCOMB2	volume	Reverb Comb Volume 2
1F801DCAh	rev05	vCOMB3	volume	Reverb Comb Volume 3
1F801DCCh	rev06	vCOMB4	volume	Reverb Comb Volume 4
1F801DCEh	rev07	vWALL	volume	Reverb Reflection Volume 2
1F801DD0h	rev08	vAPF1	volume	Reverb APF Volume 1
1F801DD2h	rev09	vAPF2	volume	Reverb APF Volume 2
1F801DD4h	rev0A	mLSAME	src/dst	Reverb Same Side Reflection Address 1 Left
1F801DD6h	rev0B	mRSAME	src/dst	Reverb Same Side Reflection Address 1 Right
1F801DD8h	rev0C	mLCOMB1	src	Reverb Comb Address 1 Left
1F801DDAh	rev0D	mRCOMB1	src	Reverb Comb Address 1 Right
1F801DDCh	rev0E	mLCOMB2	src	Reverb Comb Address 2 Left
1F801DDEh	rev0F	mRCOMB2	src	Reverb Comb Address 2 Right
1F801DE0h	rev10	dLSAME	src	Reverb Same Side Reflection Address 2 Left
1F801DE2h	rev11	dRSAME	src	Reverb Same Side Reflection Address 2 Right
1F801DE4h	rev12	mLDIFF	src/dst	Reverb Different Side Reflect Address 1 Left
1F801DE6h	rev13	mRDIFF	src/dst	Reverb Different Side Reflect Address 1 Right
1F801DE8h	rev14	mLCOMB3	src	Reverb Comb Address 3 Left
1F801DEAh	rev15	mRCOMB3	src	Reverb Comb Address 3 Right
1F801DECh	rev16	mLCOMB4	src	Reverb Comb Address 4 Left
1F801DEEh	rev17	mRCOMB4	src	Reverb Comb Address 4 Right
1F801DF0h	rev18	dLDIFF	src	Reverb Different Side Reflect Address 2 Left
1F801DF2h	rev19	dRDIFF	src	Reverb Different Side Reflect Address 2 Right
1F801DF4h	rev1A	mLAPF1	src/dst	Reverb APF Address 1 Left
1F801DF6h	rev1B	mRAPF1	src/dst	Reverb APF Address 1 Right
1F801DF8h	rev1C	mLAPF2	src/dst	Reverb APF Address 2 Left
1F801DFAh	rev1D	mRAPF2	src/dst	Reverb APF Address 2 Right
1F801DFCh	rev1E	vLIN	volume	Reverb Input Volume Left
1F801DFEh	rev1F	vRIN	volume	Reverb Input Volume Right

All volume registers are signed 16bit (range -8000h..+7FFFh).
All src/dst/disp/base registers are addresses in SPU memory (divided by 8), src/dst are relative to the current buffer address, the disp registers are relative to src registers, the base register defines the start address of the reverb buffer (the end address is fixed, at 7FFFEh). Writing a value to mBASE does additionally set the current buffer address to that value.

1F801D98h - Voice 0..23 Reverb mode aka Echo On (EON) (R/W)

```
0-23  Voice 0..23 Destination (0=To Mixer, 1=To Mixer and to Reverb)
24-31 Not used
```

Sets reverb for the channel. As soon as the sample ends, the reverb for that channel is turned off... that's fine, but WHEN does it end?
In Reverb mode, the voice seems to output BOTH normal (immediately) AND via Reverb (delayed).

Reverb Bits in SPUCNT Register (R/W)
The SPUCNT register contains a Reverb Master Enable flag, and Reverb Enable flags for External Audio input and CD Audio input.
When the Reverb Master Enable flag is cleared, the SPU stops to write any data to the Reverb buffer (that is useful when zero-filling the reverb buffer; ensuring that already-zero values aren't overwritten by still-nonzero values).
However, the Reverb Master Enable flag does not disable output from Reverb buffer to the speakers (that might be useful to output uncompressed 22050Hz samples) (otherwise, to disable the buffer output, set the Reverb Output volume to zero and/or zerofill the reverb buffer).

SPU Reverb Formula

Reverb Formula

```
___Input from Mixer (Input volume multiplied with incoming data)_____________
Lin = vLIN * LeftInput    ;from any channels that have Reverb enabled
Rin = vRIN * RightInput   ;from any channels that have Reverb enabled
____Same Side Reflection (left-to-left and right-to-right)__________________
[mLSAME] = (Lin + [dLSAME]*vWALL - [mLSAME-2])*vIIR + [mLSAME-2]  ;L-to-L
[mRSAME] = (Rin + [dRSAME]*vWALL - [mRSAME-2])*vIIR + [mRSAME-2]  ;R-to-R
___Different Side Reflection (left-to-right and right-to-left)_____________
[mLDIFF] = (Lin + [dRDIFF]*vWALL - [mLDIFF-2])*vIIR + [mLDIFF-2]  ;R-to-L
[mRDIFF] = (Rin + [dLDIFF]*vWALL - [mRDIFF-2])*vIIR + [mRDIFF-2]  ;L-to-R
___Early Echo (Comb Filter, with input from buffer)________________________
Lout=vCOMB1*[mLCOMB1]+vCOMB2*[mLCOMB2]+vCOMB3*[mLCOMB3]+vCOMB4*[mLCOMB4]
Rout=vCOMB1*[mRCOMB1]+vCOMB2*[mRCOMB2]+vCOMB3*[mRCOMB3]+vCOMB4*[mRCOMB4]
___Late Reverb APF1 (All Pass Filter 1, with input from COMB)______________
[mLAPF1]=Lout-vAPF1*[mLAPF1-dAPF1], Lout=[mLAPF1-dAPF1]+[mLAPF1]*vAPF1
[mRAPF1]=Rout-vAPF1*[mRAPF1-dAPF1], Rout=[mRAPF1-dAPF1]+[mRAPF1]*vAPF1
___Late Reverb APF2 (All Pass Filter 2, with input from APF1)_____________
[mLAPF2]=Lout-vAPF2*[mLAPF2-dAPF2], Lout=[mLAPF2-dAPF2]+[mLAPF2]*vAPF2
[mRAPF2]=Rout-vAPF2*[mRAPF2-dAPF2], Rout=[mRAPF2-dAPF2]+[mRAPF2]*vAPF2
___Output to Mixer (Output volume multiplied with input from APF2)__________
LeftOutput  = Lout*vLOUT
RightOutput = Rout*vROUT
___Finally, before repeating the above steps_______________________________
BufferAddress = MAX(mBASE, (BufferAddress+2) AND 7FFFEh)
Wait one 22050Hz cycle, then repeat the above stuff
```

Notes
The values written to memory are saturated to -8000h..+7FFFh.
The multiplication results are divided by +8000h, to fit them to 16bit range.
All memory addresses are relative to the current BufferAddress, and wrapped within mBASE..7FFFEh when exceeding that region.
All data in the Reverb buffer consists of signed 16bit samples. The Left and Right Reverb Buffer addresses should be choosen so that one half of the buffer contains Left samples, and the other half Right samples (ie. the data is L,L,L,L,... R,R,R,R,...; it is NOT interlaced like L,R,L,R,...), during operation, when the buffer address increases, the Left half will overwrite the older samples of the Right half, and vice-versa.
The reverb hardware spends one 44100h cycle on left calculations, and the next 44100h cycle on right calculations (unlike as shown in the above formula, where left/right are shown simultaneously at 22050Hz).

Bug
vIIR works only in range -7FFFh..+7FFFh. When set to -8000h, the multiplication by -8000h is still done correctly, but, the final result (the value written to memory) gets negated (this is a pretty strange feature, it is NOT a simple overflow bug, it does affect the "+[mLSAME-2]" addition; although that part normally shouldn't be affected by the "*vIIR" multiplication). Similar effects might (?) occur on some other volume registers when they are set to -8000h.

Speed of Sound
The speed of sound is circa 340 meters per second (in dry air, at room temperature). For example, a voice that travels to a wall at 17 meters distance, and back to its origin, should have a delay of 0.1 seconds.

SPU Reverb Examples

Reverb Examples
Below are some Reverb examples, showing the required memory size (ie. set Port 1F801DA2h to "(80000h-size)/8"), and the Reverb register settings for Port 1F801DC0h..1F801DFFh, ie. arranged like so:

```
dAPF1  dAPF2  vIIR   vCOMB1 vCOMB2  vCOMB3  vCOMB4  vWALL   ;1F801DC0h..CEh
vAPF1  vAPF2  mLSAME mRSAME mLCOMB1 mRCOMB1 mLCOMB2 mRCOMB2 ;1F801DD0h..DEh
dLSAME dRSAME mLDIFF mRDIFF mLCOMB3 mRCOMB3 mLCOMB4 mRCOMB4 ;1F801DE0h..EEh
dLDIFF dRDIFF mLAPF1 mRAPF1 mLAPF2  mRAPF2  vLIN    vRIN    ;1F801DF0h..FEh
```

Also, don't forget to initialize Port 1F801D84h, 1F801D86h, 1F801D98h, and SPUCNT, and to zerofill the Reverb Buffer (so that no garbage values are output when activating reverb). For whatever reason, one MUST also initialize Port 1F801DACh (otherwise reverb stays off).

Room (size=26C0h bytes)

```
007Dh,005Bh,6D80h,54B8h,BED0h,0000h,0000h,BA80h
5800h,5300h,04D6h,0333h,03F0h,0227h,0374h,01EFh
0334h,01B5h,0000h,0000h,0000h,0000h,0000h,0000h
0000h,0000h,01B4h,0136h,00B8h,005Ch,8000h,8000h
```

Studio Small (size=1F40h bytes)

```
0033h,0025h,70F0h,4FA8h,BCE0h,4410h,C0F0h,9C00h
5280h,4EC0h,03E4h,031Bh,03A4h,02AFh,0372h,0266h
031Ch,025Dh,025Ch,018Eh,022Fh,0135h,01D2h,00B7h
018Fh,00B5h,00B4h,0080h,004Ch,0026h,8000h,8000h
```

Studio Medium (size=4840h bytes)

```
00B1h,007Fh,70F0h,4FA8h,BCE0h,4510h,BEF0h,B4C0h
5280h,4EC0h,0904h,076Bh,0824h,065Fh,07A2h,0616h
076Ch,05EDh,05ECh,042Eh,050Fh,0305h,0462h,02B7h
042Fh,0265h,0264h,01B2h,0100h,0080h,8000h,8000h
```

Studio Large (size=6FE0h bytes)

```
00E3h,00A9h,6F60h,4FA8h,BCE0h,4510h,BEF0h,A680h
5680h,52C0h,0DFBh,0B58h,0D09h,0A3Ch,0BD9h,0973h
0B59h,08DAh,08D9h,05E9h,07ECh,04B0h,06EFh,03D2h
05EAh,031Dh,031Ch,0238h,0154h,00AAh,8000h,8000h
```

Hall (size=ADE0h bytes)

```
01A5h,0139h,6000h,5000h,4C00h,B800h,BC00h,C000h
6000h,5C00h,15BAh,11BBh,14C2h,10BDh,11BCh,0DC1h
11C0h,0DC3h,0DC0h,09C1h,0BC4h,07C1h,0A00h,06CDh
09C2h,05C1h,05C0h,041Ah,0274h,013Ah,8000h,8000h
```

Half Echo (size=3C00h bytes)

```
0017h,0013h,70F0h,4FA8h,BCE0h,4510h,BEF0h,8500h
5F80h,54C0h,0371h,02AFh,02E5h,01DFh,02B0h,01D7h
0358h,026Ah,01D6h,011Eh,012Dh,00B1h,011Fh,0059h
01A0h,00E3h,0058h,0040h,0028h,0014h,8000h,8000h
```

Space Echo (size=F6C0h bytes)

```
033Dh,0231h,7E00h,5000h,B400h,B000h,4C00h,B000h
6000h,5400h,1ED6h,1A31h,1D14h,183Bh,1BC2h,16B2h
1A32h,15EFh,15EEh,1055h,1334h,0F2Dh,11F6h,0C5Dh
1056h,0AE1h,0AE0h,07A2h,0464h,0232h,8000h,8000h
```

Chaos Echo (almost infinite) (size=18040h bytes)

```
0001h,0001h,7FFFh,7FFFh,0000h,0000h,0000h,8100h
0000h,0000h,1FFFh,0FFFh,1005h,0005h,0000h,0000h
1005h,0005h,0000h,0000h,0000h,0000h,0000h,0000h
0000h,0000h,1004h,1002h,0004h,0002h,8000h,8000h
```

Delay (one-shot echo) (size=18040h bytes)

```
0001h,0001h,7FFFh,7FFFh,0000h,0000h,0000h,0000h
0000h,0000h,1FFFh,0FFFh,1005h,0005h,0000h,0000h
1005h,0005h,0000h,0000h,0000h,0000h,0000h,0000h
0000h,0000h,1004h,1002h,0004h,0002h,8000h,8000h
```

Reverb off (size=10h dummy bytes)

```
0000h,0000h,0000h,0000h,0000h,0000h,0000h,0000h
0000h,0000h,0001h,0001h,0001h,0001h,0001h,0001h
0000h,0000h,0001h,0001h,0001h,0001h,0001h,0001h
0000h,0000h,0001h,0001h,0001h,0001h,0000h,0000h
```

Note that the memory offsets should be 0001h here (not 0000h), otherwise zerofilling the reverb buffer seems to fail (maybe because zero memory offsets somehow cause the fill-value to mixed with the old value or so; that appears even when reverb master enable is zero). Also, when not using reverb, Port 1F801D84h, 1F801D86h, 1F801D98h, and the SPUCNT reverb bits should be set to zero.

SPU Unknown Registers

1F801DA0h - Some kind of a read-only status register.. or just garbage..?

```
0-15 Unknown?
```

Usually 9D78h, occassionaly changes to 17DAh or 108Eh for a short moment.
Other day: Usually 9CF8h, or occassionally 9CFAh.

Another day: Usually 0000h, or occassionally 4000h.

1F801DBCh - 4 bytes - Unknown? (R/W)
```
80 21 4B DF
```
Other day (dots = same as above):
```
.. 31 .. ..
```

1F801E60h - 32 bytes - Unknown? (R/W)
```
7E 61 A9 96 47 39 F9 1E E1 E1 80 DD E8 17 7F FB
FB BF 1D 6C 8F EC F3 04 06 23 89 45 C1 6D 31 82
```
Other day (dots = same as above):
```
.. .. .. .. .. .. .. .. .. .. .. .. .. .. 7B ..
.. .. .. .. .. .. .. .. 04 .. .. .. .. .. .. 86
```

The bytes at 1F801DBCh and 1F801E60h usually have the above values on cold-boot. The registers are read/write-able, although writing any values to them doesn't seem to have any effect on sound output. Also, the SPU doesn't seem to modify the registers at any time during sound output, nor reverb calculations, nor activated external audio input... the registers seem to be just some kind of general-purpose RAM.

Interrupts

1F801070h I_STAT - Interrupt status register (R=Status, W=Acknowledge)
1F801074h I_MASK - Interrupt mask register (R/W)
Status: Read I_STAT (0=No IRQ, 1=IRQ)
Acknowledge: Write I_STAT (0=Clear Bit, 1=No change)
Mask: Read/Write I_MASK (0=Disabled, 1=Enabled)
```
0     IRQ0 VBLANK (PAL=50Hz, NTSC=60Hz)
1     IRQ1 GPU   Can be requested via GP0(1Fh) command (rarely used)
2     IRQ2 CDROM
3     IRQ3 DMA
4     IRQ4 TMR0  Timer 0 aka Root Counter 0 (Sysclk or Dotclk)
5     IRQ5 TMR1  Timer 1 aka Root Counter 1 (Sysclk or H-blank)
6     IRQ6 TMR2  Timer 2 aka Root Counter 2 (Sysclk/8)
7     IRQ7 Controller and Memory Card - Byte Received Interrupt
8     IRQ8 SIO
9     IRQ9 SPU
10    IRQ10 Controller - Lightpen Interrupt (reportedly also PIO...?)
11-15 Not used (always zero)
16-31 Garbage
```

Secondary IRQ10 Controller (Port 1F802030h)
EXP2 DTL-H2000 I/O Ports

Interrupt Request / Execution
The interrupt request bits in I_STAT are edge-triggered, ie. the get set ONLY if the corresponding interrupt source changes from "false to true".
If one or more interrupts are requested and enabled, ie. if "(I_STAT AND I_MASK)=nonzero", then cop0r13.bit10 gets set, and when cop0r12.bit10 and cop0r12.bit0 are set, too, then the interrupt gets executed.

Interrupt Acknowledge
To acknowledge an interrupt, write a "0" to the corresponding bit in I_STAT. Most interrupts (except IRQ0,4,5,6) must be additionally acknowledged at the I/O port that has caused them (eg. JOY_CTRL.bit4).
Observe that the I_STAT bits are edge-triggered (they get set only on High-to-Low, or False-to-True edges). The correct acknowledge order is:
```
First, acknowledge I_STAT                (eg. I_STAT.bit7=0)
Then, acknowledge corresponding I/O port (eg. JOY_CTRL.bit4=1)
```
When doing it vice-versa, the hardware may miss further IRQs (eg. when first setting JOY_CTRL.4=1, then a new IRQ may occur in JOY_STAT.4 within a single clock cycle, thereafter, setting I_STAT.7=0 would successfully reset I_STAT.7, but, since JOY_STAT.4 is already set, there'll be no further edge, so I_STAT.7 won't be ever set in future).

COP0 Interrupt Handling
Relevant COP0 registers are cop0r13 (CAUSE, reason flags), and cop0r12 (SR, control flags), and cop0r14 (EPC, return address), and, cop0cmd=10h (aka RFE opcode) is used to prepare the return from interrupts. For more info, see
COP0 - Exception Handling

PSX specific COP0 Notes
COP0 has six hardware interrupt bits, of which, the PSX uses only cop0r13.bit10 (the other ones, cop0r13.bit11-15 are always zero). cop0r13.bit10 is NOT a latch, ie. it gets automatically cleared as soon as "(I_STAT AND I_MASK)=zero", so there's no need to do an acknowledge at the cop0 side. COP0 additionally has two software interrupt bits, cop0r13.bit8-9, which do exist in the PSX, too, these bits are read/write-able latches which can be set/cleared manually to request/acknowledge exceptions by software.

Halt Function (Wait for Interrupt)
The PSX doesn't have a HALT opcode, so, even if the program is merely waiting for an interrupt to occur, the CPU is always running at full speed, which is resulting in high power consumption, and, in case of emulators, high CPU emulation load. To save energy, and to make emulation smoother on slower computers, I've added a Halt function for use in emulators:
EXP2 Nocash Emulation Expansion

DMA Channels

DMA Register Summary
```
1F80108xh DMA0 channel 0  MDECin  (RAM to MDEC)
1F80109xh DMA1 channel 1  MDECout (MDEC to RAM)
1F8010Axh DMA2 channel 2  GPU (lists + image data)
1F8010Bxh DMA3 channel 3  CDROM   (CDROM to RAM)
1F8010Cxh DMA4 channel 4  SPU
1F8010Dxh DMA5 channel 5  PIO (Expansion Port)
1F8010Exh DMA6 channel 6  OTC (reverse clear OT) (GPU related)
1F8010F0h DPCR - DMA Control register
1F8010F4h DICR - DMA Interrupt register
```
These ports control DMA at the CPU-side. In most cases, you'll additionally need to initialize an address (and transfer direction, transfer enabled, etc.) at the remote-side (eg. at the GPU-side for DMA2).

1F801080h+N*10h - D#_MADR - DMA base address (Channel 0..6) (R/W)
```
  0-23  Memory Address where the DMA will start reading from/writing to
  24-31 Not used (always zero)
```
In SyncMode=0, the hardware doesn't update the MADR registers (it will contain the start address even during and after the transfer) (unless Chopping is enabled, in that case it does update MADR, same does probably also happen when getting interrupted by a higher priority DMA channel).
In SyncMode=1 and SyncMode=2, the hardware does update MADR (it will contain the start address of the currently transferred block; at transfer end, it'll hold the end-address in SyncMode=1, or the 00FFFFFFh end-code in SyncMode=2)
Note: Address bit0-1 are writeable, but any updated current/end addresses are word-aligned with bit0-1 forced to zero.

1F801084h+N*10h - D#_BCR - DMA Block Control (Channel 0..6) (R/W)
For SyncMode=0 (ie. for OTC and CDROM):
```
  0-15  BC    Number of words (0001h..FFFFh) (or 0=10000h words)
  16-31 0     Not used (usually 0 for OTC, or 1 ("one block") for CDROM)
```
For SyncMode=1 (ie. for MDEC, SPU, and GPU-vram-data):
```
  0-15  BS    Blocksize (words) ;for GPU/SPU max 10h, for MDEC max 20h
  16-31 BA    Amount of blocks  ;ie. total length = BS*BA words
```
For SyncMode=2 (ie. for GPU-command-lists):
```
  0-31  0     Not used (should be zero) (transfer ends at END-CODE in list)
```
BC/BS/BA can be in range 0001h..FFFFh (or 0=10000h). For BS, take care not to set the blocksize larger than the buffer of the corresponding unit can hold. (GPU and SPU both have a 16-word buffer). A larger blocksize means faster transfer.
SyncMode=1 decrements BA to zero, SyncMode=0 with chopping enabled decrements BC to zero (aside from that two cases, D#_BCR isn't changed during/after transfer).

1F801088h+N*10h - D#_CHCR - DMA Channel Control (Channel 0..6) (R/W)
```
  0       Transfer Direction    (0=To Main RAM, 1=From Main RAM)
  1       Memory Address Step   (0=Forward;+4, 1=Backward;-4)
  2-7     Not used              (always zero)
  8       Chopping Enable       (0=Normal, 1=Chopping; run CPU during DMA gaps)
  9-10    SyncMode, Transfer Synchronisation/Mode (0-3):
            0  Start immediately and transfer all at once (used for CDROM, OTC)
            1  Sync blocks to DMA requests   (used for MDEC, SPU, and GPU-data)
            2  Linked-List mode              (used for GPU-command-lists)
            3  Reserved                      (not used)
  11-15   Not used              (always zero)
  16-18   Chopping DMA Window Size (1 SHL N words)
  19      Not used              (always zero)
  20-22   Chopping CPU Window Size (1 SHL N clks)
  23      Not used              (always zero)
  24      Start/Busy            (0=Stopped/Completed, 1=Start/Enable/Busy)
  25-27   Not used              (always zero)
  28      Start/Trigger         (0=Normal, 1=Manual Start; use for SyncMode=0)
  29      Unknown (R/W) Pause?  (0=No, 1=Pause?)     (For SyncMode=0 only?)
  30      Unknown (R/W)
  31      Not used              (always zero)
```
The Start/Trigger bit is automatically cleared upon BEGIN of the transfer, this bit needs to be set only in SyncMode=0 (setting it in other SyncModes would force the first block to be transferred instantly without DRQ, which isn't desired).
The Start/Busy bit is automatically cleared upon COMPLETION of the transfer, this bit must be always set for all SyncModes when starting a transfer.
For DMA6/OTC there are some restrictions, D6_CHCR has only three read/write-able bits: Bit24,28,30. All other bits are read-only: Bit1 is always 1 (step=backward), and the other bits are always 0.

1F8010F0h - DPCR - DMA Control Register (R/W)
```
  0-2   DMA0, MDECin  Priority      (0..7; 0=Highest, 7=Lowest)
  3     DMA0, MDECin  Master Enable (0=Disable, 1=Enable)
  4-6   DMA1, MDECout Priority      (0..7; 0=Highest, 7=Lowest)
  7     DMA1, MDECout Master Enable (0=Disable, 1=Enable)
  8-10  DMA2, GPU     Priority      (0..7; 0=Highest, 7=Lowest)
  11    DMA2, GPU     Master Enable (0=Disable, 1=Enable)
  12-14 DMA3, CDROM   Priority      (0..7; 0=Highest, 7=Lowest)
  15    DMA3, CDROM   Master Enable (0=Disable, 1=Enable)
  16-18 DMA4, SPU     Priority      (0..7; 0=Highest, 7=Lowest)
  19    DMA4, SPU     Master Enable (0=Disable, 1=Enable)
  20-22 DMA5, PIO     Priority      (0..7; 0=Highest, 7=Lowest)
  23    DMA5, PIO     Master Enable (0=Disable, 1=Enable)
  24-26 DMA6, OTC     Priority      (0..7; 0=Highest, 7=Lowest)
  27    DMA6, OTC     Master Enable (0=Disable, 1=Enable)
  28-30 Unknown, Priority Offset or so? (R/W)
  31    Unknown, no effect? (R/W)
```
Initial value on reset is 07654321h. If two or more channels have the same priority setting, then the priority is determined by the channel number (DMA0=Lowest, DMA6=Highest).

1F8010F4h - DICR - DMA Interrupt Register (R/W)
```
  0-5   Unknown  (read/write-able)
  6-14  Not used (always zero)
  15    Force IRQ (sets bit31)           (0=None, 1=Force Bit31=1)
  16-22 IRQ Enable for DMA0..DMA6        (0=None, 1=Enable)
  23    IRQ Master Enable for DMA0..DMA6 (0=None, 1=Enable)
  24-30 IRQ Flags for DMA0..DMA6         (0=None, 1=IRQ)    (Write 1 to reset)
  31    IRQ Master Flag                  (0=None, 1=IRQ)    (Read only)
```
IRQ flags in Bit(24+n) are set upon DMAn completion - but caution - they are set ONLY if enabled in Bit(16+n).
Bit31 is a simple readonly flag that follows the following rules:
```
  IF b15=1 OR (b23=1 AND (b16-22 AND b24-30)>0) THEN b31=1 ELSE b31=0
```
Upon 0-to-1 transition of Bit31, the IRQ3 flag (in Port 1F801070h) gets set.
Bit24-30 are acknowledged (reset to zero) when writing a "1" to that bits (and, additionally, IRQ3 must be acknowledged via Port 1F801070h).

1F8010F8h (usually 7FFAC68Bh? or 0BFAC688h)
```
    (changes to 7FE358D1h after DMA transfer)
```
1F8010FCh (usually 00FFFFF7h) (...maybe OTC fill-value)
```
    (stays so even after DMA transfer)
```
Contains strange read-only values (but not the usual "Garbage").
Not yet tested during transfer, might be remaining length and address?

Commonly used DMA Control Register values for starting DMA transfers
```
  DMA0 MDEC.IN  01000201h (always)
  DMA1 MDEC.OUT 01000200h (always)
  DMA2 GPU      01000200h (VramRead), 01000201h (VramWrite), 01000401h (List)
  DMA3 CDROM    11000000h (normal), 11400100h (chopped, rarely used)
```

```
DMA4 SPU      01000201h (write), 01000200h (read, rarely used)
DMA5 PIO      N/A       (not used by any known games)
DMA6 OTC      11000002h (always)
```

XXX: DMA2 values 01000201h (VramWrite), 01000401h (List) aren't 100% confirmed to be used by ALL existing games. All other values are always used as listed above.

DMA Transfer Rates

```
DMA0 MDEC.IN     1 clk/word   ;0110h clks per 100h words ;\plus whatever
DMA1 MDEC.OUT    1 clk/word   ;0110h clks per 100h words ;/decompression time
DMA2 GPU         1 clk/word   ;0110h clks per 100h words ;-plus ...
DMA3 CDROM/BIOS  24 clks/word ;1800h clks per 100h words ;\plus single/double
DMA3 CDROM/GAMES 40 clks/word ;2800h clks per 100h words ;/speed sector rate
DMA4 SPU         4 clks/word  ;0420h clks per 100h words ;-plus ...
DMA5 PIO         20 clks/word ;1400h clks per 100h words ;-not actually used
DMA6 OTC         1 clk/word   ;0110h clks per 100h words ;-plus nothing
```

MDEC decompression time is still unknown (may vary on RLE and color/mono).
GPU polygon rendering time is unknown (may be quite slow for large polys).
GPU vram read/write time is unknown (may vary on horizontal screen resolution).
CDROM BIOS default is 24 clks, for some reason most games change it to 40 clks.
SPU transfer is unknown (may have some extra delays).
XXX is SPU really only 4 clks (theoretically SPU access should be slower)?
PIO isn't used by any games (and if used: could be configured to other rates)
OTC is just writing to RAM without extra overload.
CDROM/SPU/PIO timings can be configured via Memory Control registers.

DRAM Hyper Page mode

DMA is using DRAM Hyper Page mode, allowing it to access DRAM rows at 1 clock cycle per word (effectively around 17 clks per 16 words, due to required row address loading, probably plus some further minimal overload due to refresh cycles). This is making DMA much faster than CPU memory accesses (CPU DRAM access takes 1 opcode cycle plus 6 waitstates, ie. 7 cycles in total)

CPU Operation during DMA

Basically, the CPU is stopped during DMA (theoretically, the CPU could be kept running when accessing only cache, scratchpad and on-chip I/O ports like DMA registers, and during the CDROM/SPU/PIO waitstates it could even access Main RAM, but these situations aren't supported).
However, the CPU operation resumes during periods when DMA gets interrupted (ie. after SyncMode 1 blocks, after SyncMode 2 list entries) (or in SyncMode 0 with Chopping enabled).

Timers

1F801100h+N*10h - Timer 0..2 Current Counter Value (R/W)

```
0-15  Current Counter value (incrementing)
16-31 Garbage
```

This register is automatically incrementing. It is write-able (allowing to set it to any value). It gets forcefully reset to 0000h on any write to the Counter Mode register, and on counter overflow (either when exceeding FFFFh, or when exceeding the selected target value).

1F801104h+N*10h - Timer 0..2 Counter Mode (R/W)

```
0     Synchronization Enable (0=Free Run, 1=Synchronize via Bit1-2)
1-2   Synchronization Mode   (0-3, see lists below)
       Synchronization Modes for Counter 0:
         0 = Pause counter during Hblank(s)
         1 = Reset counter to 0000h at Hblank(s)
         2 = Reset counter to 0000h at Hblank(s) and pause outside of Hblank
         3 = Pause until Hblank occurs once, then switch to Free Run
       Synchronization Modes for Counter 1:
         Same as above, but using Vblank instead of Hblank
       Synchronization Modes for Counter 2:
         0 or 3 = Stop counter at current value (forever, no h/v-blank start)
         1 or 2 = Free Run (same as when Synchronization Disabled)
3     Reset counter to 0000h  (0=After Counter=FFFFh, 1=After Counter=Target)
4     IRQ when Counter=Target (0=Disable, 1=Enable)
5     IRQ when Counter=FFFFh  (0=Disable, 1=Enable)
6     IRQ Once/Repeat Mode    (0=One-shot, 1=Repeatedly)
7     IRQ Pulse/Toggle Mode   (0=Short Bit10=0 Pulse, 1=Toggle Bit10 on/off)
8-9   Clock Source (0-3, see list below)
       Counter 0:  0 or 2 = System Clock,  1 or 3 = Dotclock
       Counter 1:  0 or 2 = System Clock,  1 or 3 = Hblank
       Counter 2:  0 or 1 = System Clock,  2 or 3 = System Clock/8
10    Interrupt Request       (0=Yes, 1=No) (Set after Writing)    (W=1) (R)
11    Reached Target Value    (0=No, 1=Yes) (Reset after Reading)        (R)
12    Reached FFFFh Value     (0=No, 1=Yes) (Reset after Reading)        (R)
13-15 Unknown (seems to be always zero)
16-31 Garbage (next opcode)
```

In one-shot mode, the IRQ is pulsed/toggled only once (one-shot mode doesn't stop the counter, it just suppresses any further IRQs until a new write to the Mode register occurs; if both IRQ conditions are enabled in Bit4-5, then one-shot mode triggers only one of those conditions; whichever occurs first).
Normally, Pulse mode should be used (Bit10 is permanently set, except for a few clock cycles when an IRQ occurs). In Toggle mode, Bit10 is set after writing to the Mode register, and becomes inverted on each IRQ (in one-shot mode, it remains zero after the IRQ) (in repeat mode it inverts Bit10 on each IRQ, so IRQ4/5/6 are triggered only each 2nd time, ie. when Bit10 changes from 1 to 0).

1F801108h+N*10h - Timer 0..2 Counter Target Value (R/W)

```
0-15  Counter Target value
16-31 Garbage
```

When the Target flag is set (Bit3 of the Control register), the counter increments up to (including) the selected target value, and does then restart at 0000h.

Dotclock/Hblank

For more info on dotclock and hblank timings, see:
GPU Timings
Caution: Reading the Current Counter Value can be a little unstable (when using dotclk or hblank as clock source); the GPU clock isn't in sync with the CPU clock, so the timer may get changed during the CPU read cycle. As a workaround: repeat reading the timer until the received value is the same (or slightly bigger) than the previous value.

CDROM Drive

Playstation CDROM I/O Ports
CDROM Controller I/O Ports

Playstation CDROM Commands
CDROM Controller Command Summary
CDROM - Control Commands
CDROM - Seek Commands
CDROM - Read Commands
CDROM - Status Commands
CDROM - CD Audio Commands
CDROM - Test Commands
CDROM - Secret Unlock Commands
CDROM - Mainloop/Responses
CDROM - Response Timings
CDROM - Response/Data Queueing

General CDROM Disk Format
CDROM Disk Format
CDROM Subchannels
CDROM Sector Encoding
CDROM XA Subheader, File, Channel, Interleave
CDROM XA Audio ADPCM Compression
CDROM ISO Volume Descriptors
CDROM ISO File and Directory Descriptors
CDROM ISO Misc
CDROM File Formats

Playstation CDROM Protection
CDROM Protection - SCEx Strings
CDROM Protection - Bypassing it
CDROM Protection - Modchips
CDROM Protection - LibCrypt

General CDROM Disk Images
CDROM Disk Images CCD/IMG/SUB (CloneCD)
CDROM Disk Images CDI (DiscJuggler)
CDROM Disk Images CUE/BIN/CDT (Cdrwin)
CDROM Disk Images MDS/MDF (Alcohol 120%)
CDROM Disk Images NRG (Nero)
CDROM Disk Image/Containers CDZ
CDROM Disk Image/Containers ECM
CDROM Subchannel Images
CDROM Disk Images Other Formats

Playstation CDROM Coprocessor
CDROM Internal Info on PSX CDROM Controller

CDROM Controller I/O Ports

1F801800h - Index/Status Register (Bit0-1 R/W) (Bit2-7 Read Only)
```
0-1 Port 1F801801h-1F801803h index  (0..3 = Index0..Index3)   (R/W)
2   XA-ADPCM fifo empty  (0=Empty)  (set when playing XA-ADPCM sound)
3   Parameter fifo empty (1=Empty)  (triggered before writing FIRST byte)
4   Parameter fifo full  (0=Full)   (triggered after writing 16 bytes)
5   Response fifo empty  (0=Empty)  (triggered after reading LAST byte)
6   Data fifo empty      (0=Empty)  (triggered after reading LAST byte)
7   Command/parameter transmission busy  (1=Busy)
```
Bit3,4,5 are bound to 5bit counters; ie. the bits become true at specified amount of reads/writes, and thereafter once on every further 32 reads/writes.
```
XXX  2 ADPBUSY
XXX  3 PRMEMPT
XXX  4 PRMWRDY
XXX  5 RSLRRDY
XXX  6 DRQSTS
XXX  7 BUSYSTS
```

1F801801h.Index0 - Command Register (W)
```
0-7  Command Byte
```
Writing to this address sends the command byte to the CDROM controller, which will then read-out any Parameter byte(s) which have been previously stored in the Parameter Fifo. It takes a while until the command/parameters are transferred to the controller, and until the response bytes are received; once when completed, interrupt INT3 is generated (or INT5 in case of invalid command/parameter values), and the response (or error code) can be then read from the Response Fifo. Some commands additionally have a second response, which is sent with another interrupt.

1F801802h.Index0 - Parameter Fifo (W)
```
0-7  Parameter Byte(s) to be used for next Command
```
Before sending a command, write any parameter byte(s) to this address.

1F801803h.Index0 - Request Register (W)
```
0-4  Not used (should be zero)
5    Want Command Start Interrupt on Next Command (0=No change, 1=Yes)
6    ...
7    Want Data         (0=No/Reset Data Fifo, 1=Yes/Load Data Fifo)
```
XXX bit5: SMEN
XXX bit6: BFWR
XXX bit7: BFRD

1F801802h.Index0..3 - Data Fifo - 8bit/16bit (R)

After ReadS/ReadN commands have generated INT1, software must set the Want Data bit (1F801803h.Index0.Bit7), then wait until Data Fifo becomes not empty (1F801800h.Bit6), the datablock (disk sector) can be then read from this register.

```
0-7  Data 8bit  (one byte), or alternately,
0-15 Data 16bit (LSB=First byte, MSB=Second byte)
```

The PSX hardware allows to read 800h-byte or 924h-byte sectors, indexed as [000h..7FFh] or [000h..923h], when trying to read further bytes, then the PSX will repeat the byte at index [800h-8] or [924h-4] as padding value.
Port 1F801802h can be accessed with 8bit or 16bit reads (ie. to read a 2048-byte sector, one can use 2048 load-byte opcodes, or 1024 load halfword opcodes, or, more conventionally, a 512 word DMA transfer; the actual CDROM databus is only 8bits wide, so CPU/DMA are apparently breaking 16bit/32bit reads into multiple 8bit reads from 1F801802h).

1F801801h.Index1 - Response Fifo (R)
1F801801h.Index0,2,3 - Response Fifo (R) (Mirrors)

```
0-7  Response Byte(s) received after sending a Command
```

The response Fifo is a 16-byte buffer, most or all responses are less than 16 bytes, after reading the last used byte (or before reading anything when the response is 0-byte long), Bit5 of the Index/Status register becomes zero to indicate that the last byte was received.
When reading further bytes: The buffer is padded with 00h's to the end of the 16-bytes, and does then restart at the first response byte (that, without receiving a new response, so it'll always return the same 16 bytes, until a new command/response has been sent/received).

1F801802h.Index1 - Interrupt Enable Register (W)
1F801803h.Index0 - Interrupt Enable Register (R)
1F801803h.Index2 - Interrupt Enable Register (R) (Mirror)

```
0-4  Interrupt Enable Bits (usually all set, ie. 1Fh=Enable All IRQs)
5-7  Unknown/unused (write: should be zero) (read: usually all bits set)
```

XXX WRITE: bit5-7 unused should be 0 // READ: bit5-7 unused

1F801803h.Index1 - Interrupt Flag Register (R/W)
1F801803h.Index3 - Interrupt Flag Register (R) (Mirror)

```
0-2   Read: Response Received   Write: 7=Acknowledge   ;INT1..INT7
3     Read: Unknown (usually 0) Write: 1=Acknowledge   ;INT8  ;XXX CLRBFEMPT
4     Read: Command Start       Write: 1=Acknowledge   ;INT10h;XXX CLRBFWRDY
5     Read: Always 1 ;XXX "_"   Write: 1=Unknown               ;XXX SMADPCLR
6     Read: Always 1 ;XXX "_"   Write: 1=Reset Parameter Fifo ;XXX CLRPRM
7     Read: Always 1 ;XXX "_"   Write: 1=Unknown               ;XXX CHPRST
```

Writing "1" bits to bit0-4 resets the corresponding IRQ flags; normally one should write 07h to reset the response bits, or 1Fh to reset all IRQ bits. Writing values like 01h is possible (eg. that would change INT3 to INT2, but doing that would be total nonsense). After acknowledge, the Response Fifo is made empty, and if there's been a pending command, then that command gets send to the controller.
The lower 3bit indicate the type of response received,

```
INT0   No response received (no interrupt request)
INT1   Received SECOND (or further) response to ReadS/ReadN (and Play+Report)
INT2   Received SECOND response (to various commands)
INT3   Received FIRST response (to any command)
INT4   DataEnd (when Play/Forward reaches end of disk) (maybe also for Read?)
INT5   Received error-code (in FIRST or SECOND response)
       INT5 also occurs on SECOND GetID response, on unlicensed disks
       INT5 also occurs when opening the drive door (even if no command
          was sent, ie. even if no read-command or other command is active)
INT6   N/A
INT7   N/A
```

The other 2bit indicate something else,

```
INT8   Unknown (never seen that bit set yet)
INT10h Command Start (when INT10h requested via 1F801803h.Index0.Bit5)
```

The response interrupts are queued, for example, if the 1st response is INT3, and the second INT5, then INT3 is delivered first, and INT5 is not delivered until INT3 is acknowledged (ie. the response interrupts are NOT ORed together to produce INT7 or so). The upper bits however can be ORed with the lower bits (ie. Command Start INT10h and 1st Response INT3 would give INT13h).

1F801802h.Index2 - Audio Volume for Left-CD-Out to Left-SPU-Input (W)
1F801803h.Index2 - Audio Volume for Left-CD-Out to Right-SPU-Input (W)
1F801801h.Index3 - Audio Volume for Right-CD-Out to Right-SPU-Input (W)
1F801802h.Index3 - Audio Volume for Right-CD-Out to Left-SPU-Input (W)
Allows to configure the CD for mono/stereo output (eg. values "80h,0,80h,0" produce normal stereo volume, values "40h,40h,40h,40h" produce mono output of equivalent volume).
When using bigger values, the hardware does have some incomplete saturation support; the saturation works up to double volume (eg. overflows that occur on "FFh,0,FFh,0" or "80h,80h,80h,80h" are clipped to min/max levels), however, the saturation does NOT work properly when exceeding double volume (eg. mono with quad-volume "FFh,FFh,FFh,FFh").

```
0-7  Volume Level (00h..FFh) (00h=Off, FFh=Max/Double, 80h=Default/Normal)
```

After changing these registers, write 20h to 1F801803h.Index3.
Unknown if any existing games are actually supporting mono output. Resident Evil 2 uses these ports to produce fade-in/fade-out effects (although, for that purpose, it should be much easier to use Port 1F801DB0h).

1F801803h.Index3 - Audio Volume Apply Changes (by writing bit5=1)

```
0    ADPMUTE Mute ADPCM               (0=Normal, 1=Mute)
1-4  -       Unused (should be zero)
5    CHNGATV Apply Audio Volume changes (0=No change, 1=Apply)
6-7  -       Unused (should be zero)
```

1F801801h.Index1 - Sound Map Data Out (W)

```
0-7  Data
```

This register seems to be restricted to 8bit bus, unknown if/how the PSX DMA controller can write to it (it might support only 16bit data for CDROM).

1F801801h.Index2 - Sound Map Coding Info (W)

```
0    Mono/Stereo     (0=Mono, 1=Stereo)
1    Reserved        (0)
2    Sample Rate     (0=37800Hz, 1=18900Hz)
3    Reserved        (0)
4    Bits per Sample (0=4bit, 1=8bit)
5    Reserved        (0)
6    Emphasis        (0=Off, 1=Emphasis)
7    Reserved        (0)
```

==========

Command Execution
Command/Parameter transmission is indicated by bit7 of 1F801800h.

When that bit gets zero, the response can be read immediately (immediately for MOST commands, but not ALL commands; so better wait for the IRQ).
Alternately, you can wait for an IRQ (which seems to take place MUCH later), and then read the response.
If there are any pending cdrom interrupts, these MUST be acknowledged before sending the command (otherwise bit7 of 1F801800h will stay set forever).

Command Busy Flag - 1F801800h.Bit7
Indicates ready-to-send-new-command,
```
0=Ready to send a new command
1=Busy sending a command/parameters
```
Trying to send a new command in the Busy-phase causes malfunction (the older command seems to get lost, the newer command executes and returns its results and triggers an interrupt, but, thereafter, the controller seems to hang). So, always wait until the Busy-bit goes off before sending a command.
When the Busy-flag goes off, a new command can be send immediately (even if the response from the previous command wasn't received yet), however, the new command stays in the Busy-phase until the IRQ from the previous command is acknowledged, at that point the actual transmission of the new command starts, and the Busy-flag goes off (once when the transmission completes).

Misc
Trying to do a 32bit read from 1F801800h returns the 8bit value at 1F801800h multiplied by 01010101h.

To init the CD
```
-Flush all IRQs
-1F801803h.Index0=0
-Com_Delay=4901 (=1325h) (Port 1F801020h) (means 16bit or 32bit write?)
   (the write seems to be 32bit, clearing the upper16bit of the register)
-Send two Getstat commands
-Send Command 0Ah (Init)
-Demute
```

Seek-Busy Phase
Warning: most or all of the info in the sentence below appear to incorrect (either that, or I didn't understand that rather confusing sentence).
REPORTEDLY:
"You should not send some commands while the CD is seeking (ie. Getstat returns with bit6 set). Thing is that stat only gets updated after a new command. I haven't tested this for other command, but for the play command (03h) you can just keep repeating the [which?] command and checking stat returned by that, for bit6 to go low (and bit7 to go high in this case). If you don't and try to do a getloc [GetlocP and/or GetlocL?] directly after the play command reports it's done [what done? meaning sending start-to-play was "done"? or meaning play reached end-of-disc?], the CD will stop. (I guess the CD can't get it's current location while it's seeking, so the logic stops the seek to get an exact fix, but never restarts..)"

Sound Map Flowchart
Sound Map mode allows to output XA-ADPCM from Main RAM (rather than from CDROM).
```
SPU: Init Master Volume Left/Right (Port 1F801D80h/1F801D82h)
SPU: Init CD Audio Volume Left/Right (Port 1F801DB0h/1F801DB2h)
SPU: Enable CD Audio (Port 1F801DAAh.Bit0=1)
CDROM/CMD: send Stop command (probably better to avoid conflicts)
CDROM/CMD: send Demute command (if muted) (but works only if disc inserted)
CDROM/HOST: init Codinginfo (Port 1F801801h.Index2)
CDROM/HOST: enable ADPCM (Port 1F801803h.Index3.Bit0=0)  ;probably needed?
... set dummy addr/len with DISHXFRC=1 ?  <-- NOT required !
... set SMEN ... and dummy BFWR?    <-- BOTH bits required ?
transfer 900h bytes (same format as ADPCM sectors) (Port 1F801801h.Index1)
Note: Before sending a byte, one should wait for DRQs (1F801801h.Bit6=1)
Note: ADPCM output doesn't start until the last (900h'th) byte is transferred
```
Sound Map mode may be very useful for testing XA-ADPCM directly from within an exe file (without needing a cdrom with ADPCM sectors). And, Sound Map supports both 4bit and 8bit compression (the SPU supports only 4bit).
Caution: If ADPCM wasn't playing, and one sends one 900h-byte block, then it will get stored in one of three 900h-byte slots in SRAM, and one would expect that that slot to be played when the ADPCM output starts - however, actually, the hardware will more or less randomly play one of the three slots; not necessarily the slot that was updated most recently.

CDROM Controller Command Summary

Command Summary
```
Command          Parameters      Response(s)
00h -            -               INT5(11h,40h)  ;reportedly "Sync" uh?
01h Getstat      -               INT3(stat)
02h Setloc     E amm,ass,asect   INT3(stat)
03h Play       E (track)         INT3(stat), optional INT1(report bytes)
04h Forward    E -               INT3(stat), optional INT1(report bytes)
05h Backward   E -               INT3(stat), optional INT1(report bytes)
06h ReadN      E -               INT3(stat), INT1(stat), datablock
07h MotorOn    E -               INT3(stat), INT2(stat)
08h Stop       E -               INT3(stat), INT2(stat)
09h Pause      E -               INT3(stat), INT2(stat)
0Ah Init         -               INT3(late-stat), INT2(stat)
0Bh Mute       E -               INT3(stat)
0Ch Demute     E -               INT3(stat)
0Dh Setfilter  E file,channel    INT3(stat)
0Eh Setmode      mode            INT3(stat)
0Fh Getparam     -               INT3(stat,mode,null,file,channel)
10h GetlocL    E -               INT3(amm,ass,asect,mode,file,channel,sm,ci)
11h GetlocP    E -               INT3(track,index,mm,ss,sect,amm,ass,asect)
12h SetSession E session         INT3(stat), INT2(stat)
13h GetTN      E -               INT3(stat,first,last)  ;BCD
14h GetTD      E track (BCD)     INT3(stat,mm,ss)       ;BCD
15h SeekL      E -               INT3(stat), INT2(stat)  ;\use prior Setloc
16h SeekP      E -               INT3(stat), INT2(stat)  ;/to set target
17h -            -               INT5(11h,40h)  ;reportedly "SetClock" uh?
18h -            -               INT5(11h,40h)  ;reportedly "GetClock" uh?
19h Test         sub_function    depends on sub_function (see below)
1Ah GetID      E -               INT3(stat), INT2/5(stat,flg,typ,atip,"SCEx")
1Bh ReadS      E?-               INT3(stat), INT1(stat), datablock
1Ch Reset        -               INT3(stat), Delay
1Dh GetQ       E adr,point       INT3(stat), INT2(10bytesSubQ,peak_lo) ;\not
1Eh ReadTOC      -               INT3(late-stat), INT2(stat)           ;/vC0
1Fh..4Fh -       -               INT5(11h,40h)  ;-Unused/invalid
50h Secret 1     -               INT5(11h,40h)  ;\
51h Secret 2     "Licensed by"   INT5(11h,40h)  ;
52h Secret 3     "Sony"          INT5(11h,40h)  ; Secret Unlock Commands
53h Secret 4     "Computer"      INT5(11h,40h)  ; (not in version vC0, and,
```

```
  54h Secret 5     "Entertainment" INT5(11h,40h)  ; nonfunctional in japan)
  55h Secret 6     "<region>"      INT5(11h,40h)  ;
  56h Secret 7     -               INT5(11h,40h)  ;/
  57h SecretLock   -               INT5(11h,40h)  ;-Secret Lock Command
  58h..5Fh Crash   -               Crashes the HC05 (jumps into a data area)
  6Fh..FFh -       -               INT5(11h,40h)  ;-Unused/invalid
```

E = Error 80h appears on some commands (02h..09h, 0Bh..0Dh, 10h..16h, 1Ah, 1Bh?, and 1Dh) when the disk is missing, or when the drive unit is disconnected from the mainboard.

sub_function numbers (for command 19h)

Test commands are invoked with command number 19h, followed by a sub_function number as first parameter byte. The Kernel seems to be using only sub_function 20h (to detect the CDROM Controller version).

```
  sub  params  response           ;Effect
  00h      -   INT3(stat)         ;Force motor on, clockwise, even if door open
  01h      -   INT3(stat)         ;Force motor on, anti-clockwise, super-fast
  02h      -   INT3(stat)         ;Force motor on, anti-clockwise, super-fast
  03h      -   INT3(stat)         ;Force motor off (ignored during spin-up)
  04h      -   INT3(stat)         ;Start SCEx reading and reset counters
  05h      -   INT3(total,success);Stop SCEx reading and get counters
  06h *    n   INT3(old)  ;\early ;Adjust balance in RAM, send CX(30+n XOR 7)
  07h *    n   INT3(old)  ; PSX   ;Adjust gain in RAM, send CX(38+n XOR 7)
  08h *    n   INT3(old)  ;/only  ;Adjust balance in RAM only
  06h..0Fh -   INT5(11h,10h)      ;N/A (11h,20h when NONZERO number of params)
  10h      -   INT3(stat) ;CX(..) ;Force motor on, anti-clockwise, super-fast
  11h      -   INT3(stat) ;CX(03) ;Move Lens Up (leave parking position)
  12h      -   INT3(stat) ;CX(02) ;Move Lens Down (enter parking position)
  13h      -   INT3(stat) ;CX(28) ;Move Lens Outwards
  14h      -   INT3(stat) ;CX(2C) ;Move Lens Inwards
  15h      -   INT3(stat) ;CX(22) ;If motor on: Move outwards,inwards,motor off
  16h      -   INT3(stat) ;CX(23) ;No effect?
  17h      -   INT3(stat) ;CX(E8) ;Force motor on, clockwise, super-fast
  18h      -   INT3(stat) ;CX(EA) ;Force motor on, anti-clockwise, super-fast
  19h      -   INT3(stat) ;CX(25) ;No effect?
  1Ah      -   INT3(stat) ;CX(21) ;No effect?
  1Bh..1Fh -   INT5(11h,10h)      ;N/A (11h,20h when NONZERO number of params)
  20h      -   INT3(yy,mm,dd,ver) ;Get cdrom BIOS date/version (yy,mm,dd,ver)
  21h      -   INT3(n)            ;Get Drive Switches (bit0=POS0, bit1=DOOR)
  22h ***  -   INT3("for ...")    ;Get Region ID String
  23h ***  -   INT3("CXD...")     ;Get Chip ID String for Servo Amplifier
  24h ***  -   INT3("CXD...")     ;Get Chip ID String for Signal Processor
  25h ***  -   INT3("CXD...")     ;Get Chip ID String for Decoder/FIFO
  26h..2Fh -   INT5(11h,10h)      ;N/A (11h,20h when NONZERO number of params)
  30h *    i,x,y     INT3(stat)       ;Prototype/Debug stuff   ;\supported on
  31h *    x,y       INT3(stat)       ;Prototype/Debug stuff   ; early PSX only
  4xh *    i         INT3(x,y)        ;Prototype/Debug stuff   ;/
  30h..4Fh ..        INT5(11h,10h)    ;N/A always 11h,10h (no matter of params)
  50h      a[,b[,c]] INT3(stat)       ;Servo/Signal send CX(a:b:c)
  51h **   39h,xx    INT3(stat,hi,lo) ;Servo/Signal send CX(39xx) with response
  51h..5Fh -         INT5(11h,10h)    ;N/A
  60h      lo,hi     INT3(databyte)   ;HC05 SUB-CPU read RAM and I/O ports
  61h..70h -         INT5(11h,10h)    ;N/A
  71h ***  adr       INT3(databyte)   ;Decoder Read one register
  72h ***  adr,dat   INT3(stat)       ;Decoder Write one register
  73h ***  adr,len   INT3(databytes..);Decoder Read multiple registers, bugged
  74h ***  adr,len,..INT3(stat)       ;Decoder Write multiple registers, bugged
  75h ***  -         INT3(lo,hi,lo,hi);Decoder Get Host Xfer Info Remain/Addr
  76h ***  a,b,c,d   INT3(stat)       ;Decoder Prepare Transfer to/from SRAM
  77h..FFh -         INT5(11h,10h)    ;N/A
```

* sub_functions 06h..08h, 30h..31h, and 4xh are supported only in vC0 and vC1.
** sub_function 51h is supported only in BIOS version vC2 and up.
*** sub_functions 22h..25h, 71h..76h supported only in BIOS version vC1 and up.

CDROM - Control Commands

Sync - Command 00h --> INTx(stat+1,40h) (?)

Reportedly "command does not succeed until all other commands complete. This can be used for synchronization - hence the name."
Uh, actually, returns error code 40h = Invalid Command...?

Setfilter - Command 0Dh,file,channel --> INT3(stat)

Automatic ADPCM (CD-ROM XA) filter ignores sectors except those which have the same channel and file numbers in their subheader. This is the mechanism used to select which of multiple songs in a single .XA file to play.
Setfilter does not affect actual reading (sector reads still occur for all sectors).
XXX err... that is... does not affect reading of non-ADPCM sectors (normal "data" sectors are kept received regardless of Setfilter).

Setmode - Command 0Eh,mode --> INT3(stat)

```
  7   Speed       (0=Normal speed, 1=Double speed)
  6   XA-ADPCM    (0=Off, 1=Send XA-ADPCM sectors to SPU Audio Input)
  5   Sector Size (0=800h=DataOnly, 1=924h=WholeSectorExceptSyncBytes)
  4   Ignore Bit  (0=Normal, 1=Ignore Sector Size and Setloc position)
  3   XA-Filter   (0=Off, 1=Process only XA-ADPCM sectors that match Setfilter)
  2   Report      (0=Off, 1=Enable Report-Interrupts for Audio Play)
  1   AutoPause   (0=Off, 1=Auto Pause upon End of Track) ;for Audio Play
  0   CDDA        (0=Off, 1=Allow to Read CD-DA Sectors; ignore missing EDC)
```

The "Ignore Bit" does reportedly force a sector size of 2328 bytes (918h), however, that doesn't seem to be true. Instead, Bit4 seems to cause the controller to ignore the sector size in Bit5 (instead, the size is kept from the most recent Setmode command which didn't have Bit4 set). Also, Bit4 seems to cause the controller to ignore the <exact> Setloc position (instead, data is randomly returned from the "Setloc position minus 0..3 sectors"). And, Bit4 causes INT1 to return status.Bit3=set (IdError). Purpose of Bit4 is unknown?

Init - Command 0Ah --> INT3(stat) --> INT2(stat)

Multiple effects at once. Sets mode=00h (or not ALL bits cleared?), activates drive motor, Standby, abort all commands.

Reset - Command 1Ch,(...) --> INT3(stat) --> Delay(1/8 seconds)

Resets the drive controller, reportedly, same as opening and closing the drive door. The command executes no matter if/how many parameters are used (tested with 0..7 params). INT3 indicates that the command was started, but there's no INT that would indicate when the command is finished, so, before sending any further commands, a delay of 1/8 seconds (or 400000h clock cycles) must be issued by software.

Note: Executing the command produces a click sound in the drive mechanics, maybe it's just a rapid motor on/off, but it might something more serious, like ignoring the /POS0 signal...?

MotorOn - Command 07h --> INT3(stat) --> INT2(stat)
Activates the drive motor, works ONLY if the motor was off (otherwise fails with INT5(stat,20h); that error code would normally indicate "wrong number of parameters", but means "motor already on" in this case).
Commands like Read, Seek, and Play are automatically starting the Motor when needed (which makes the MotorOn command rather useless, and it's rarely used by any games).
Myth: Older homebrew docs are referring to MotorOn as "Standby", claiming that it would work similar as "Pause", that is wrong: the command does NOT pause anything (if the motor is on, then it does simply trigger INT5, but without pausing reading or playing).
Note: The game "Nightmare Creatures 2" does actually attempt to use MotorOn to "pause" after reading files, but the hardware does simply ignore that attempt (aside from doing the INT5 thing).

Stop - Command 08h --> INT3(stat) --> INT2(stat)
Stops motor with magnetic brakes (stops within a second or so) (unlike power-off where it'd keep spinning for about 10 seconds), and moves the drive head to the begin of the first track. Official way to restart is command 0Ah, but almost any command will restart it.
The first response returns the current status (this already with bit5 cleared), the second response returns the new status (with bit1 cleared).

Pause - Command 09h --> INT3(stat) --> INT2(stat)
Aborts Reading and Playing, the motor is kept spinning, and the drive head maintains the current location within reasonable error.
The first response returns the current status (still with bit5 set if a Read command was active), the second response returns the new status (with bit5 cleared).

Data/ADPCM Sector Filtering/Delivery
The PSX CDROM BIOS is first trying to send sectors to the ADPCM decoder, and, if that didn't work out, then it's trying to send them to the main CPU (and if that didn't work out either, then it's silently ignoring the sector).

```
 try_deliver_as_adpcm_sector:
  reject if CD-DA AUDIO format
  reject if sector isn't MODE2 format
  reject if adpcm_disabled(setmode.6)
  reject if filter_enabled(setmode.3) AND selected file/channel doesn't match
  reject if submode isn't audio+realtime (bit2 and bit6 must be both set)
  deliver: send sector to xa-adpcm decoder when passing above cases
 try_deliver_as_data_sector:
  reject data-delivery if "try_deliver_as_adpcm_sector" did do adpcm-delivery
  reject if filter_enabled(setmode.3) AND submode is audio+realtime (bit2+bit6)
  1st delivery attempt: send INT1+data, unless there's another INT pending
  delay, and retry at later time... but this time with file/channel checking!
  reject if filter_enabled(setmode.3) AND selected file/channel doesn't match
  2nd delivery attempt: send INT1+data, unless there's another INT pending
```

BUG: Note that the data delivery is done in two different attempts: The first one regardless of file/channel, and the second one only on matching file/channel (if filtering is enabled).

CDROM - Seek Commands

Setloc - Command 02h,amm,ass,asect --> INT3(stat)
Sets the seek target - but without yet starting the seek operation. The actual seek is invoked by certain commands: SeekL (Data) and SeekP (Audio) are doing plain seeks (and do Pause after completion). ReadN/ReadS are similar to SeekL (and do start reading data after the seek operation). Play is similar to SeekP (and does start playing audio after the seek operation).
The amm,ass,asect parameters refer to the entire disk (not to the current track). To seek to a specific location within a specific track, use GetTD to get the start address of the track, and add the desired time offset to it.

SeekL - Command 15h --> INT3(stat) --> INT2(stat)
Seek to Setloc's location in data mode (using data sector header position data, which works/exists only on Data tracks, not on CD-DA Audio tracks).
After the seek, the disk stays on the seeked location forever (namely: when seeking sector N, it does stay at around N-8..N-0 in single speed mode, or at around N-5..N+2 in double speed mode).
Trying to use SeekL on Audio CDs passes okay on the first response, but (after two seconds or so) the second response will return an error (stat+4,04h), and stop the drive motor... that error doesn't appear ALWAYS though... works in some situations... such like when previously reading data sectors or so...?

SeekP - Command 16h --> INT3(stat) --> INT2(stat)
Seek to Setloc's location in audio mode (using the Subchannel Q position data, which works on both Audio on Data disks).
After the seek, the disk stays on the seeked location forever (namely: when seeking sector N, it does stay at around N-9..N-1 in single speed mode, or at around N-2..N in double speed mode).
Note: Some older docs claim that SeekP would recurse only "MM:SS" of the "MM:SS:FF" position from Setloc - that is wrong, it does seek to MM:SS:FF (verified on a PSone).
After the seek, status is stat.bit7=0 (ie. audio playback off), until sending a new Play command (without parameters) to start playback at the seeked location.

SetSession - Command 12h,session --> INT3(stat) --> INT2(stat)
Seeks to session (ie. moves the drive head to the session, with stat bit6 set during the seek phase).
When issued during active-play, the command returns error code 80h.
When issued during play-spin-up, play is aborted.

```
  ___Errors___
  session = 00h causes error code 10h.     ;INT5(03h,10h), no 2nd/3rd response
  ___On a non-multisession-disk___
  session = 01h passes okay.               ;INT3(stat), and once INT2(stat)
  session = 02h or higher cause seek error ;INT3(stat), and twice INT5(06h,40h)
  ___On a multisession-disk with N sessions___
  session = 01h..N+1 passes okay   ;where N+1 moves to the END of LAST session
  session = N+2 or higher cause seek error  ;2nd response = INT5(06h,20h)
```

after seek error --> disk stops spinning at 2nd response, then restarts spinning for 1 second or so, then stops spinning forever... and following gettn/gettd/getid/getlocl/getlocp fail with error 80h...
The command does automatically read the TOC of the new session.
There seems to be no way to determine the current sessions number (via Getparam or so), and more important, no way to determine if the disk is a multi-session disk or not... except by trial... which would stop the drive motor on seek errors on single-session disks...?
For setloc, one must probably specifiy minutes within the 1st track of the new session (the 1st track of 1st session usually/always starts at 00:02:00, but for other sessions one would need to use GetTD)...?

CDROM - Read Commands

ReadN - Command 06h --> INT3(stat) --> INT1(stat) --> datablock
Read with retry. The command responds once with "stat,INT3", and then it's repeatedly sending "stat,INT1 --> datablock", that is continued even after a successful read has occured; use the Pause command to terminate the repeated INT1 responses.
Unknown which responses are sent in case of read errors?
====
ReadN and ReadS cause errors if you're trying to read an unlicensed CD or CD-R without a mod chip. Sectors on Audio CDs can be read only when CDDA is enabled via Setmode (otherwise error code 40h is returned).
====
Actually, Read seems to work on unlicensed CD-R's, but the returned data is the whole sector or so (the 2048 data bytes preceeded by a 12byte header, and probably/maybe followed by error-correction info; in fact the total received data in the Data Fifo is 4096 bytes; the last some bytes probably being garbage) (however error correction is NOT performed by hardware, so the 2048 data bytes may be trashy) (however, if the error correction info IS received, then error correction could be performed by software) (also Setloc doesn't seem to work accurately on unlicensed CD-R's).
====
```
    ;Read occasionally returns 11h,40h ..? when TOC isn't loaded?
```
After receiving INT1, the Kernel does,
```
  [1F801800h]=00h
  00h=[1F801800h]
  [1F801803h]=00h
  00h=[1F801803h]
  [1F801800h]=00h
  [1F801803h]=80h
```
and then,
```
  [1F801018h]=00020943h  ;cdrom_delay
  [1F801020h]=0000132Ch  ;com_delay
```
then,
```
  x=[1F8010F4h] AND 00FFFFFFh   ;result is 00840000h
  [1F8010F4h] = x OR 00880000h
  [1F8010F0h] = [1F8010F0h] OR 00008000h
  [1F8010B0h] = A0010000h ;addr
  [1F8010B4h] = 00010200h ;LSBs=num words, MSBs=ignored/bullshit
  [1F8010B4h] = 11000000h ;DMA control
```
thereafter,
```
  [1F801800h]=01h
  [1F801803h]=40h    ;reset parameter fifo
  [0]=00000000h
  [0]=00000001h
  [0]=00000002h
  [0]=00000003h
  [1F801800h]=00h
  [1F801801h]=09h    ;command9 (pause)
```

ReadS - Command 1Bh --> INT3(stat) --> INT1(stat) --> datablock
Read without automatic retry. Not sure what that means... does WHAT on errors? Maybe intended for continous streaming video output (to skip bad frames, rather than to interrupt the stream by performing read-retrys).

ReadN/ReadS
Both ReadN/ReadS are reading data sequentially, starting at the sector specified with Setloc, and then automatically reading the following sectors.

CDROM Incoming Data / Buffer Overrun Timings
The Read commands are continously receiving 75 sectors per second (or 150 sectors at double speed), and, basically, the software must be fast enough to process that amount of incoming data. However, the PSX hardware includes a buffer that can hold up to a handful (exact number is unknown?) of sectors, so, occasional delays of more than 1/75 seconds between processing two sectors aren't causing lost sectors, unless the delay(s) are summing up too much. The relevant steps for receiving data are:
```
  Wait for Interrupt Request (INT1)         ;indicates that data is available
  Send Data Request (1F801803h.Index0.Bit7=1);accept data
  Acknowledge INT1                          ;
  Copy Data to Main RAM (via I/O or DMA)    ;read data
```
The Data Request accepts the data for the currently pending interrupt, it should be usually issued between receiving/acknowledging INT1 (however, it can be also issued shortly after the acknowledge; even if there are further sectors in the buffer, there seems to be a small delay between the acknowledge and the next interrupt, and Data Requests during that period are still treated to belong to the old interrupt).
If a buffer overrun has occured <before> issuing the Data Request, then wrong data will be received, ie. some sectors will be skipped (the hardware doesn't seem to support a buffer-overrun error flag? Anyways, see GetlocL description for a possible way to detect buffer-overruns).
If a buffer overrun occurs <after> issuing the Data Request, then the requested data can be still read via I/O or DMA intactly, ie. the requested data is "locked", and the overrun will affect only the following sectors.

ReadTOC - Command 1Eh --> INT3(stat) --> INT2(stat)
```
  Caution: Supported only in BIOS version vC1 and up. Not supported in vC0.
```
Reread the Table of Contents of current session without reset. The command is rather slow, the second response appears after about 1 second delay. The command itself returns only status information (to get the actual TOC info, use GetTD and GetTN commands).
Note: The TOC contains information about the tracks on the disk (not file names or so, that kind of information is obtained via Read commands). The TOC is read automatically on power-up, when opening/closing the drive door, and when changing sessions (so, normally, it isn't required to use this command).

Setloc, Read, Pause
A normal CDROM access (such like reading a file) consists of three commands:
```
  Setloc, Read, Pause
```
Normally one shouldn't mess up the ordering of those commands, but if one does, following rules do apply:
Setloc is memorizing the wanted target, and marks it as unprocessed, and has no other effect (it doesn't start reading or seeking, and doesn't interrupt or redirect any active reads).
If Read is issued with an unprocessed Setloc, then the drive is automatically seeking the Setloc location (and marks Setloc as processed).
If Read is issued without an unprocessed Setloc, the following happens: If reading is already in progress then it just continues reading. If Reading was Paused, then reading resumes at the most recently received sector (ie. returning that sector once another time).

CDROM - Status Commands

Status code (stat)
The 8bit status code is returned by Getstat command (and many other commands), the meaning of the separate stat bits is:
```
  7  Play          Playing CD-DA         ;\only ONE of these bits can be set
  6  Seek          Seeking               ; at a time (ie. Read/Play won't get   5  Read          Reading data sectors  ;/set until after Seek completi
  4  ShellOpen     Once shell open (0=Closed, 1=Is/was Open)
  3  IdError       (0=Okay, 1=GetID denied) (also set when Setmode.Bit4=1)
```

```
  2  SeekError     (0=Okay, 1=Seek error)     (followed by Error Byte)
  1  Spindle Motor (0=Motor off, or in spin-up phase, 1=Motor on)
  0  Error         Invalid Command/parameters (followed by Error Byte)
```

If the shell is closed, then bit4 is automatically reset to zero after reading stat with the Getstat command (most or all other commands do not reset that bit after reading). If stat bit0 or bit2 is set, then the normal respons(es) and interrupt(s) are not send, and, instead, INT5 occurs, and an error-byte is send as second response byte, with the following values:

```
  ___These values appear in the FIRST response; with stat.bit0 set___
  10h - Invalid Sub_function (for command 19h), or invalid parameter value
  20h - Wrong number of parameters
  40h - Invalid command
  80h - Cannot respond yet (eg. required info was not yet read from disk yet)
          (namely, TOC not-yet-read or so)
          (also appears if no disk inserted at all)
  ___These values appear in the SECOND response; with stat.bit2 set___
  04h - Seek failed (when trying to use SeekL on Audio CDs)
  ___These values appear even if no command was sent; with stat.bit2 set___
  08h - Drive door became opened
```

80h appears on some commands (02h..09h, 0Bh..0Dh, 10h..16h, 1Ah, 1Bh?, and 1Dh) when the disk is missing, or when the drive unit is disconnected from the mainboard.

Stat Seek/Play/Read bits

There's is only max ONE of the three Seek/Play/Read bits set at a time, ie. during Seek, ONLY the seek bit is set (and Read or Play doesn't get until seek completion), that is important for Gran Turismo 1, which checks for seek completion by waiting for READ getting set (rather than waiting for SEEK getting cleared).

Getstat - Command 01h --> INT3(stat)

Returns stat (like many other commands), and additionally does reset the shell open flag (for the following commands; unless the shell is still opened). This is different as for most or all other commands (which may return stat, but which do not reset the shell open flag).
In other docs, the command is eventually referred to as "Nop", believing that it does nothing than returning stat (ignoring the fact that it's having the special shell open reset feature).

Getparam - Command 0Fh --> INT3(stat,mode,null,file,channel)

Returns stat (see Getstat above), mode (see Setmode), a null byte (always 00h), and file/channel filter values (see Setfilter).

GetlocL - Command 10h --> INT3(amm,ass,asect,mode,file,channel,sm,ci)

Retrieves 4-byte sector header, plus 4-byte subheader of the current sector. GetlocL can be send during active Read commands (but, mind that the GetlocL-INT3-response can't be received until any pending Read-INT1's are acknowledged).
The PSX hardware can buffer a handful of sectors, the INT1 handler receives the <oldest> buffered sector, the GetlocL command returns the header and subheader of the <newest> buffered sector. Note: If the returned <newest> sector number is much bigger than the expected <oldest> sector number, then it's likely that a buffer overrun has occured.
GetlocL fails (with error code 80h) when playing Audio CDs (or Audio Tracks on Data CDs). These errors occur because Audio sectors don't have any header/subheader (instead, equivalent data is stored in Subchannel Q, which can be read with GetlocP).
GetlocL also fails (with error code 80h) when the drive is in Seek phase (such like shortly after a new ReadN/ReadS command). In that case one can retry issuing GetlocL (until it passes okay, ie. until the seek has completed). During Seek, the drive seems to decode only Subchannel position data (but no header/subheader data), accordingly GetlocL won't work during seek (however, GetlocP does work during Seek).

GetlocP - Command 11h - INT3(track,index,mm,ss,sect,amm,ass,asect)

Retrieves 8 bytes of position information from Subchannel Q with ADR=1. Mainly intended for displaying the current audio position during Play. All results are in BCD.

```
  track:  track number (AAh=Lead-out area) (FFh=unknown, toc, none?)
  index:  index number (Usually 01h)
  mm:     minute number within track (00h and up)
  ss:     second number within track (00h to 59h)
  sect:   sector number within track (00h to 74h)
  amm:    minute number on entire disk (00h and up)
  ass:    second number on entire disk (00h to 59h)
  asect:  sector number on entire disk (00h to 74h)
```

Note: GetlocP is also used for reading the LibCrypt protection data:
CDROM Protection - LibCrypt

GetTN - Command 13h --> INT3(stat,first,last) ;BCD

Get first track number, and last track number in the TOC of the current Session. The number of tracks in the current session can be calculated as (last-first+1). The first track number is usually 01h in the first (or only) session, and "last track of previous session plus 1" in further sessions.

GetTD - Command 14h,track --> INT3(stat,mm,ss) ;BCD

For a disk with NN tracks, parameter values 01h..NNh return the start of the specified track, parameter value 00h returns the end of the last track, and parameter values bigger than NNh return error code 10h.
The GetTD values are relative to Index=1 and are rounded down to second boundaries (eg. if track=N Index=0 starts at 12:34:56, and Track=N Index=1 starts at 12:36:56, then GetTD(N) will return 12:36, ie. the sector number is truncated, and the Index=0 region is skipped).

GetQ - Command 1Dh,adr,point --> INT3(stat) --> INT2(10bytesSubQ,peak_lo)

```
  Caution: Supported only in BIOS version vC1 and up. Not supported in vC0.
```

Allows to read 10 bytes from Subchannel Q in Lead-In (see CDROM Subchannels chapter for details). Unlike GetTD, this command allows to receive the exact MM:SS:FF address of the point'ed Track (GetTD reads a memorized MM:SS value from RAM, whilst GetQ reads the full MM:SS:FF from the disk, which is slower than GetTD, due to the disk-access).
With ADR=1, point can be a any point number for ADR=1 in Lead-in (eg. 01h..99h=Track N, A2h=Lead-Out). The returned 10 bytes are raw SubQ data (starting with the ADR/Control value; of which the lower 4bits are always ADR=1).
The 11th returned byte is the Peak LSB (similar as in Play+Report, but in this case only the LSB is transferred, which is apparently a bug in CDROM BIOS, the programmer probably wanted to send 10 bytes without peak, or 12 bytes with full peak; although peak wouldn't be too useful, as it should always zero during Lead-In... but some discs do seem return non-zero values for whatever reason).
Aside from ADR=1, a value of ADR=5 can be used on multisession disks (eg. with point B0h, C0h). Not sure if any other ADR values can be used (ADR=3, ISRC is usually not in the Lead-In, ADR=2, EAN may be in the lead-in, but one may need to specify point equal to the first EAN byte).
If the ADR/Point combination isn't found, then a timeout occurs after circa 6 seconds (to avoid this, use GetTN to see which tracks/points exist). After the timeout, the command starts playing track 1. If the controller wasn't already in audio mode before sending the command, then it does switch off the drive motor for a moment (that, after the timeout, and before starting playback).
In case of timeout, the normal INT3/INT2 responses are replaced by INT3/INT5/INT5 (INT3 at command start, 1st INT5 at timeout/stop, and 2nd INT5 at restart/play).
Note: GetQ sends scratch noise to the SPU while seeking to the Lead-In area.

GetID - Command 1Ah --> INT3(stat) --> INT2/5 (stat,flags,type,atip,"SCEx")

```
  Drive Status          1st Response   2nd Response
```

```
Door Open             INT5(11h,80h)  N/A
Spin-up               INT5(01h,80h)  N/A
Detect busy           INT5(03h,80h)  N/A
No Disk               INT3(stat)     INT5(08h,40h, 00h,00h, 00h,00h,00h,00h)
Audio Disk            INT3(stat)     INT5(0Ah,90h, 00h,00h, 00h,00h,00h,00h)
Unlicensed:Mode1      INT3(stat)     INT5(0Ah,80h, 00h,00h, 00h,00h,00h,00h)
Unlicensed:Mode2      INT3(stat)     INT5(0Ah,80h, 20h,00h, 00h,00h,00h,00h)
Unlicensed:Mode2+Audio INT3(stat)    INT5(0Ah,90h, 20h,00h, 00h,00h,00h,00h)
Licensed:Mode2        INT3(stat)     INT2(02h,00h, 20h,00h, 53h,43h,45h,4xh)
Modchip:Audio/Mode1   INT3(stat)     INT2(02h,00h, 00h,00h, 53h,43h,45h,4xh)
```

The status byte (ie. the first byte in the responses), may differ in some cases; values shown above are typically received when issuing GetID shortly after power-up; however, shortly after the detect-busy phase, seek-busy flag (bit6) bit may be set, and, after issuing commands like Play/Read/Stop, bit7,6,5,1 may differ. The meaning of the separate 2nd response bytes is:

```
1st byte: stat  (as usually, but with bit3 same as bit7 in 2nd byte)
2nd byte: flags (bit7=denied, bit4=audio... or reportedly import, uh?)
  bit7: Licensed (0=Licensed Data CD, 1=Denied Data CD or Audio CD)
  bit6: Missing  (0=Disk Present, 1=Disk Missing)
  bit4: Audio CD (0=Data CD, 1=Audio CD) (always 0 when Modchip installed)
3rd byte: Disk type (from TOC Point=A0h) (eg. 00h=Audio or Mode1, 20h=Mode2)
4th byte: Usually 00h (or 8bit ATIP from Point=C0h, if session info exists)
  that 8bit ATIP value is taken form the middle 8bit of the 24bit ATIP value
5th-8th byte: SCEx region (eg. ASCII "SCEE" = Europe) (0,0,0,0 = Unlicensed)
```

The fourth letter of the "SCEx" string contains region information: "SCEI" (Japan/NTSC), "SCEA" (America/NTSC), "SCEE" (Europe/PAL). The "SCEx" string is displayed in the intro, and the PSX refuses to boot if it doesn't match up for the local region.
With a modchip installed, the same response is sent for Mode1 and Audio disks; whether it is Audio or Mode1 can be checked by examining Subchannel Q ADR/Control.Bit6 (eg. via command 19h,60h,50h,00h).

CDROM - CD Audio Commands

To play CD-DA Audio CDs, init the following SPU Registers: CD Audio Volume, Main Volume, and SPU Control Bit0. Then send Demute command, and Play command.

Mute - Command 0Bh --> INT3(stat)
Turn off audio streaming to SPU (affects both CD-DA and XA-ADPCM).
Even when muted, the CDROM controller is internally processing audio sectors (as seen in 1F801800h.Bit2, which works as usually for XA-ADPCM), muting is just forcing the CD output volume to zero.
Mute is used by Dino Crisis 1 to mute noise during modchip detection.

Demute - Command 0Ch --> INT3(stat)
Turn on audio streaming to SPU (affects both CD-DA and XA-ADPCM). The Demute command is needed only if one has formerly used the Mute command (by default, the PSX is demuted after power-up (...and/or after Init command?), and is demuted after cdrom-booting).

Play - Command 03h (,track) --> INT3(stat) --> optional INT1(report bytes)
Starts CD Audio Playback. The parameter is optional, if there's no parameter given (or if it is 00h), then play either starts at Setloc position (if there was a pending unprocessed Setloc), or otherwise starts at the current location (eg. the last point seeked, or the current location of the current song; if it was already playing). For a disk with N songs, Parameters 1..N are starting the selected track. Parameters N+1..99h are restarting the begin of current track. The motor is switched off automatically when Play reaches the end of the disk, and INT4(stat) is generated (with stat.bit7 cleared).
The track parameter seems to be ignored when sending Play shortly after power-up (ie. when the drive hasn't yet read the TOC).
===
"Play is almost identical to CdlReadS, believe it or not. The main difference is that this does not trigger a completed read IRQ. CdlPlay may be used on data sectors. However, all sectors from data tracks are treated as 00, so no sound is played. As CdlPlay is reading, the audio data appears in the sector buffer, but is not reliable. Game Shark "enhancement CDs" for the 2.x and 3.x versions used this to get around the PSX copy protection."
Hmmm, what/where is the sector buffer... in the SPU?
And, what/who are the 2.x and 3.x versions?

Forward - Command 04h --> INT3(stat) --> optional INT1(report bytes)
Backward - Command 05h --> INT3(stat) --> optional INT1(report bytes)
After sending the command, the drive is in fast forward/backward mode, skipping every some sectors. The skipping rate is fixed (it doesn't increase after some seconds) (however, it increases when (as long as) sending the command again and again). The sound becomes (obviously) non-continous, and also rather very silent, muffled, and almost inaudible (that's making it rather useless; unless it's combined with a track/minute/second display). To terminate forward/backward, send a new Play command (with no parameters, so play starts at the "searched" location). Backward automatically switches to Play when reaching the begin of Track 1. Forward automatically Stops the drive motor with INT4(stat) when reaching the end of the last track.
Forward/Backwards work only if the drive was in Play state, and only if Play had already started (ie. not shortly/immediately after a Play command); if the drive was not in Play state, then INT5(stat+1,80h) occurs.

Setmode bits used for Play command
During Play, only bit 7,2,1 of Setmode are used, all other Setmode bits are ignored (that, including bit0, ie. during Play the drive is always in CD-DA mode, regardless of that bit).
Bit7 (double speed) should be usually off, although it can be used for a fast forward effect (with audible output). Bit2 (report) activates an optional interrupt for Play, Forward, and Backward commands (see below). Bit1 (autopause) pauses play at the end of the track.

Report --> INT1(stat,track,index,mm/amm,ss+80h/ass,sect/asect,peaklo,peakhi)
With report enabled via Setmode, the Play, Forward, and Backward commands do repeatedly generate INT1 interrupts, with eight bytes response length. The interrupt isn't generated on ALL sectors, and the response changes between absolute time, and time within current track (the latter one indicated by bit7 of ss):

```
amm/ass/asect are returned on asect=00h,20h,40h,60h   ;-absolute time
mm/ss+80h/sect are returned on asect=10h,30h,50h,70h  ;-within current track
(or, in case of read errors, report may be returned on other asect's)
```

The last two response bytes (peaklo,peakhi) contain the Peak value, as received from the CXD2510Q Signal Processor. That is: An unsigned absolute peak level in lower 15bit, and an L/R flag in upper bit. The L/R bit is toggled after each SUBQ read, however the PSX Report mode does usually forward SUBQ only every 10 frames (but does read SUBQ in <every> frame), so L/R will stay stuck in one setting (but may toggle after one second; ie. after 75 frames). And, peak is reset after each read, so 9 of the 10 frames are lost.
Note: Report mode affects only CD Audio (not Data, nor XA-ADPCM sectors).

AutoPause --> INT4(stat)
Autopause can be enabled/disabled via Setmode.bit1:

```
Setmode.bit1=1: AutoPause=On  --> Issue INT4(stat) and PAUSE at end of TRACK
Setmode.bit1=0: AutoPause=Off --> Issue INT4(stat) and STOP at end of DISC
```

End of Track is determined by sensing a track number transition in SubQ position info. After autopause, the disc stays at the <end> of the old track, NOT at the <begin> of the next track (so trying to resume playing by sending a new Play command without new Seek/Setloc command will instantly pause again).

Caution: SubQ track transitions may pause instantly when accidently starting to play at the end of the previous track rather than at begin of desired track (this <might> happen due to seek inaccuracies, for example, GetTD does round down TOC entries from MM:SS:FF to MM:SS:00, which may be off by 0.99 seconds, although this error should be usually compensated by the leading 2-second pregap/index0 region at the begin of each track, unfortunately there are a few .CUE sheet files that do lack both PREGAP and INDEX 00 entries on audio tracks, which might cause problems with autopause).
AutoPause is used by Rayman and Tactics Ogre.

Playing XA-ADPCM Sectors (compressed audio data)
Aside from normal uncompressed CD Audio disks, the PSX can also play XA-ADPCM compressed sectors. XA-ADPCM sectors are organized in Files (not in tracks), and are "played" with Read command (not Play command).
To play XA-ADPCM, initialize the SPU for CD Audio input (as described above), enable ADPCM via Setmode, then select the sector via Setloc, and issue a Read command (typically ReadS).
XA-ADPCM sectors are interleaved, ie. only each Nth sector should be played (where "N" depends on the Motor Speed, mono/stereo format, and sample rate). If the "other" sectors do contain XA-ADPCM data too, then the Setfilter command (and XA-Filter enable flag in Setmode) must be used to select the desired sectors. If the "other" sectors do contain code or data (eg. MDEC video data) which is wanted to be send to the CPU, then SetFilter isn't required to be enabled (although it shouldn't disturb reading even if it is enabled).
If XA-ADPCM (and/or XA-Filter) is enabled via Setmode, then INT1 is generated only for non-ADPCM sectors.
The Setmode sector-size selection is don't care for forwarding XA-ADPCM sectors to the SPU (the hardware does always decompress all 900h bytes).

CDROM - Test Commands

CDROM - Test Commands - Version, Switches, Region, Chipset, SCEx
CDROM - Test Commands - Test Drive Mechanics
CDROM - Test Commands - Prototype Debug Transmission
CDROM - Test Commands - Read/Write Decoder RAM and I/O Ports
CDROM - Test Commands - Read HC05 SUB-CPU RAM and I/O Ports

CDROM - Test Commands - Version, Switches, Region, Chipset, SCEx

19h,20h --> INT3(yy,mm,dd,ver)
Indicates the date (Year-month-day, in BCD format) and version of the HC05 CDROM controller BIOS. Known/existing values are:

```
(unknown)        ;DTL-H2000 (with SPC700 instead HC05)
94h,09h,19h,C0h  ;PSX (PU-7)               19 Sep 1994, version vC0 (a)
94h,11h,18h,C0h  ;PSX (PU-7)               18 Nov 1994, version vC0 (b)
95h,07h,24h,C1h  ;PSX (LATE-PU-8)          24 Jul 1995, version vC1
95h,07h,24h,D1h  ;PSX (LATE-PU-8,debug ver)24 Jul 1995, version vD1 (debug)
96h,09h,12h,C2h  ;PSX (PU-18) (japan)      12 Sep 1996, version vC2 (a.jap)
97h,01h,10h,C2h  ;PSX (PU-18) (us/eur)     10 Jan 1997, version vC2 (a)
97h,08h,14h,C2h  ;PSX (PU-20)              14 Aug 1997, version vC2 (b)
98h,06h,10h,C3h  ;PSX (PU-22)              10 Jul 1998, version vC3 (a)
99h,02h,01h,C3h  ;PSX/PSone (PU-23, PM-41) 01 Feb 1999, version vC3 (b)
(unknown)        ;PSone with newer PM-41(2) boards
(unknown)        ;PS2,   xx xxx xxxx, late PS2 models...?
```

19h,21h --> INT3(flags)
Returns the current status of the POS0 and DOOR switches. Bit0=HeadIsAtPos0, Bit1=DoorIsOpen, Bit2-7=AlwaysZero.

19h,22h --> INT3("for Europe")

```
Caution: Supported only in BIOS version vC1 and up. Not supported in vC0.
```

Indicates the region that console is to be used in:

```
"for Europe"  --> PAL, Europe               --> requires "SCEE" discs
"for U/C"     --> NTSC, North America       --> requires "SCEA" discs
"for Japan"   --> NTSC, Japan / NTSC, Asia  --> requires "SCEI" discs
"for US/AEP"  --> Region-free debug version --> accepts unlicensed CDRs
(unknown)     --> Yaroze home-use debug version
```

The CDROMs must contain a matching SCEx string accordingly.
The string "for Europe" does also suggest 50Hz PAL/SECAM video hardware.

19h,23h --> INT3("CXD2940Q/CXD1817Q/CXD2545Q/CXD1782BR") ;Servo Amplifier
19h,24h --> INT3("CXD2940Q/CXD1817Q/CXD2545Q/CXD2510Q") ;Signal Processor
19h,25h --> INT3("CXD2940Q/CXD1817Q/CXD1815Q/CXD1199BQ") ;Decoder/FIFO

```
Caution: Supported only in BIOS version vC1 and up. Not supported in vC0.
```

Indicates the chipset that the CDROM controller is intended to be used with. The strings aren't always precisely correct (CXD1782BR is actually CXA1782BR, ie. CXA, not CXD) (and CXD1199BQ chips exist on PU-7 boards, but later PU-8 boards do actually use CXD1815Q) (and CXD1817Q is actually CXD1817R) (and newer PSones are using CXD2938Q or possibly CXD2941R chips, but nothing called CXD2940Q).

19h,04h --> INT3(stat) ;Read SCEx string (and force motor on)
Resets the total/success counters to zero, and does then try to read the SCEx string from the current location (the SCEx is stored only in the Lead-In area, so, if the drive head is elsewhere, it will usually not find any strings, unless a modchip is permanently simulating SCEx strings).
This is a raw test command (the successful or unsuccessful results do not lock/unlock the disk). The results can be read with command 19h,05h (which will terminate the SCEx reading), or they can be read from RAM with command 19h,60h,lo,hi (which doesn't stop reading). Wait 1-2 seconds before expecting any results.
Note: Like 19h,00h, this command forces the drive motor to spin at standard speed (synchronized with the data on the disk), works even if the shell is open (but stops spinning after a while if the drive is empty).

19h,05h --> INT3(total,success) ;Get SCEx Counters
Returns the total number of "Sxxx" strings received (where at least the first byte did match), and the number of full "SCEx" strings (where all bytes did match). Typically, the values are "01h,01h" for Licensed PSX Data CDs, or "00h,00h" for disk missing, unlicensed data CDs, Audio CDs.
The counters are reset to zero, and SCEx receive mode is active for a few seconds after booting a new disk (on power up, on closing the drive door, on sending a Reset command, and on sub_function 04h). The disk is unlocked if the "success" counter is nonzero, the only exception is sub_function 04h which does update the counters, but does not lock/unlock the disk.

CDROM - Test Commands - Test Drive Mechanics

Signal Processor and Servo Amplifier

19h,50h,msb[,mid,[lsb[,xlo]]] --> INT3(stat)

Sends an 8bit/16bit/24bit command to the hardware, depending on number of parameters:

```
1 byte  --> send CX(Xx)       ;short 8bit command
2 bytes --> send CX(Xxxx)     ;longer 16bit command
3 bytes --> send CX(Xxxxxx)   ;full 24bit command
4 bytes --> send CX(Xxxxxxxx) ;extended 32bit command (BIOS vC3 only)
4..15 bytes: acts same as max (3 or 4 bytes) (extra bytes are ignored)
0 bytes or more than 15 bytes: generates an error
```

19h,51h,msb[,mid,[lsb]] --> INT3(stat,hi,lo) ;BIOS vC2/vC3 only
Supported by newer CDROM BIOSes only (such that use CXD2545Q or newer chips).
Works same as 19h,50h, but does additionally receive a response.
The command is always sending a 24bit CX(Xxxxxx) command, but it doesn't verify the number of parameter bytes (when using more than 3 bytes: extra bytes are ignored, when using less than 3 bytes: garbage is appended, which is somewhat valid because 8bit/16bit commands can be padded to 24bit size by appending "don't care" bits).
The command can be used to send any CX(..) command, but actually it does make sense only for the get-status commands, see below "19h,51h,39h,xxh" description.

19h,51h,39h,xxh --> INT3(stat,hi,lo) ;BIOS vC2/vC3 only
Supported by newer CDROM BIOSes only (such that use CXD2545Q or newer chips).
Sends CX(39xx) to the hardware, and receives a response (the response.hi byte is usually 00h for 8bit responses, or 00h..01h for 9bit responses). For example, this can be used to dump the Coefficient RAM.

19h,03h --> INT3(stat) ;force motor off
Forces the motor to stop spinning (ignored during spin-up phase).

19h,17h --> INT3(stat) ;force motor on, clockwise, super-fast
19h,01h --> INT3(stat) ;force motor on, anti-clockwise, super-fast
19h,02h --> INT3(stat) ;force motor on, anti-clockwise, super-fast
19h,10h --> INT3(stat) ;force motor on, anti-clockwise, super-fast
19h,18h --> INT3(stat) ;force motor on, anti-clockwise, super-fast
Forces the drive motor to spin at maximum speed (which is much faster than normal or double speed), in normal (clockwise), or reversed (anti-clockwise) direction. The commands work even if the shell is open. The commands do not try to synchronize the motor with the data on the disk (and do thus work even if no disk is inserted).

19h,00h --> INT3(stat) ;force motor on, clockwise (even if shell open)
This command seems to have effect only if the drive motor was off. If it was off, it does FFh-fills the TOC entries in RAM, and seek to the begin of the TOC at 98:30:00 or so (where minute=98 means minus two). From that location, it follows the spiral on the disk, although it does occassionally jump back some seconds. After clearing the TOC, the command does not write new data to the TOC buffer in RAM.
Note: Like 19h,04h, this command forces the drive motor to spin at standard speed (synchronized with the data on the disk), works even if the shell is open (but stops spinning after a while if the drive is empty).

19h,11h --> INT3(stat) ;Move Lens Up (leave parking position)
19h,12h --> INT3(stat) ;Move Lens Down (enter parking position)
19h,13h --> INT3(stat) ;Move Lens Outwards (away from center of disk)
19h,14h --> INT3(stat) ;Move Lens Inwards (towards center of disk)
Moves the laser lens. The inwards/outwards commands do move ONLY the lens (ie. unlike as for Seek commands, the overall-laser-unit remains in place, only the lens is moved).

19h,15h - if motor on: move head outwards + inwards + motor off
Moves the drive head to outer-most and inner-most position. Note that the drive doesn't have a switch that'd tell the controller when it has reached the outer-most position (so it'll forcefully hit against the outer edge) (ie. using this command too often may destroy the drive mechanics).
Note: The same destructive hit-outer-edge effect happens when using Setloc/Seek with too large values (like minute=99h).

19h,16h --> INT3(stat) ;Unknown / makes some noise if motor is on
19h,19h --> INT3(stat) ;Unknown / no effect
19h,1Ah --> INT3(stat) ;Unknown / makes some noise if motor is on
Seem to have no effect?
19h,16h seems to Move Lens Inwards, too.

19h,06h,new --> INT3(old) ;Adjust balance in RAM, and apply it via CX(30+n)
19h,07h,new --> INT3(old) ;Adjust gain in RAM, and apply it via CX(38+n)
19h,08h,new --> INT3(old) ;Adjust balance in RAM only
These commands are supported only by older CDROM BIOS versions (those with CXA1782BR Servo Amplifier).
Later BIOSes will respond with INT5(11h,20h) when trying to use these commands (because CXD2545Q and later Servo Amplifiers don't support the CX(30/38+n) commands).

CDROM - Test Commands - Prototype Debug Transmission

Serial Debug Messages
Older CDROM BIOSes are supporting debug message transmission via serial bus, using lower 3bit of the HC05 "databus" combined with the so-called "ROMSEL" pin (which apparently doesn't refer to Read-Only-Memory, but rather something like Runtime-Output-Message, or whatever).
Data is transferred in 24bit units (8bit command/index from HC05, followed by 16bit data to/from HC05), bigger messages are divided into multiple such 24bit snippets.
There are no connectors for external debug hardware on any PSX mainboards, so the whole stuff seems to be dating back to prototypes. And it seems to be removed from later BIOSes (which appear to use "ROMSEL" as "SCLK"; for receiving status info from the new CXD2545Q chips).

19h,30h,index,dat1,dat2 --> INT3(stat) ;Prototype/Debug stuff
19h,31h,dat1,dat2 --> INT3(stat) ;Prototype/Debug stuff
19h,4xh,index --> INT3(dat1,dat2) ;Prototype/Debug stuff
These functions are supported on older CDROM BIOS only; later BIOSes respond by INT5(11h,10h).
The functions do not affect the CDROM operation (they do simple allow to transfer data between Main CPU and external debug hardware).
Sub functions 30h and 31h may fail with INT5(11h,80h) when receiving wrong signals on the serial input line.
Sub function "4xh" value can be 40h..4Fh (don't care).

INT5 Debug Messages
Alongsides to INT5 errors, the BIOS is usually also sending information via the above serial bus (the error info is divided into multiple 8bit+16bit snippets, and contains stat, error code, mode, current SubQ position, and most recently issued command).

CDROM - Test Commands - Read/Write Decoder RAM and I/O Ports

Caution: Below commands 19h,71h..76h are supported only in BIOS version vC1 and up. Not supported in vC0.

19h,71h,index --> INT3(databyte) ;Read single register
index can be 00h..1Fh, bigger values seem to be mirrored to "index AND 1Fh", with one exception: index 13h in NOT mirrored, instead, index 33h, 53h, 93h, B3h, D3h, F3h return INT5(stat+1,10h), and index 73h returns INT5(stat+1,20h).
Aside from returning a value, the commands seem to DO something (like moving the drive head when a disk is inserted). Return values are usually:

```
index     value
00h       04h      ;04h=empty, 8Eh=licensed, 24h=audio
01h       [0B1h]   ;DCh=empty/licensed, DDh=audio
02h       00h
03h       00h          ;or variable when disk inserted
04h       00h
05h       80h          ;or 86h or 89h when disk inserted
06h       C0h
07h       02h
08h       8Ah
09h       C0h
0Ah       00h
0Bh       C0h
0Ch       [1F2h]
0Dh       [1F3h]
0Eh       00h          ;or 8Eh or E6h when disk inserted    ;D4h/audio
0Fh       00h          ;or sometimes 01h when disk inserted ;50h/audio
10h       C0h
11h       E0h
12h       71h
13h       stat
14h       FFh
15h..1Fh  C0h-filled       ;or 17h --> DEh
```

19h,72h,index,databyte --> INT3(stat) ;Write single register

```
;other response on param xx16h,xx18h with xx>00h
```

19h,73h,index,len --> INT3(databytes...) ;Read multiple registers (bugged)
19h,74h,index,len,databytes --> INT3(stat) ;Write multiple registers (bugged)
Same as read/write single register, but trying to transfer multiple registers at once. BUG: The transfer should range from 00h to len-1, but the loop counter is left uninitialized (set to X=48h aka "command number 19h-minus-1-mul-2" instead of X=00h). Causing to the function to read/write garbage at index 48h..FFh, it does then wrap to 00h and do the correct intended transfer, but the preceeding bugged part may have smashed RAM or I/O ports.

19h,75h --> INT3(remain.lo,remain.hi,addr.lo,addr.hi) ;Get Host Xfer Info
Returns a 4-byte value. In my early tests, on the first day it returned B1h,CEh,4Ch,01h, on the next day 2Ch,E4h,95h,D5h, and on all following days 00h,C0h,00h,00h (no idea why/where the earlier values came from).
The first byte seems to be always 00h; no matter of [1F0h].
The second byte seems to be always C0h; no matter of [1F1h].
The third,fourth bytes are [1F2h,1F3h].
That two bytes are 0Ch,08h after Read commands.

```
The first bytes are NOT affected by:
destroying [1F0h] via too-many-parameters in command-buffer,
changes to [1F1h] which may occur after read command (eg. may be 20h)
```

19h,76h,len_lo,len_hi,addr_lo,addr_hi --> INT3(stat) ;Prepare SRAM Transfer
Prepare Transfer to/from 32K SRAM.
After INT3, data can be read (same way as sector data after INT1).

CDROM - Test Commands - Read HC05 SUB-CPU RAM and I/O Ports

19h,60h,addr_lo,addr_hi --> INT3(data) ;Read one byte from Drive RAM or I/O
Reads one byte from the controller's RAM or I/O area, see the memory map below for more info. Among others, the command allows to read Subchannel Q data, eg. at [200h..209h], including ADR=2/UPC/EAN and ADR=3/ISRC values (which are suppressed by GetlocP). Eg. wait for ADR<>2, then for ADR=2, then read the remaining 9 bytes (because of the delayed IRQs, this works only at single speed) (at double speed one can read only 5 bytes before the values get overwritten by new data). Unknown if older boards (with 4.00MHz oscillators) are fast enough to read all 10 SubQ bytes.

CDROM Controller I/O Area and RAM Memory Map
First 40h bytes are I/O ports (as in MC68HC05 datasheet):

```
000h 4        FF 7B 00 FF  (other when disk inserted)
004h 5        11 00 20 20 0C
009h 1        00 (when disk inserted: changes between 00 or 80)
00Ah 2        71 00
00Ch 1        00 (when disk inserted: changes between 00 or 80)
00Dh 3        20 20 20
010h 8        02 80 00 60 00 00 99(orBB) 98
018h 4     changes randomly (even when no disk inserted)
01Ch 3        40 00 41
01Fh 1     changes randomly (even when no disk inserted)
020h 30    20h-filled
03Eh 2        82h 20h
```

Next 200h bytes are RAM:

```
040h 4        08 00 00 00  ;or 98 07 xx 0B when disk inserted ;[40].Bit1=MUTE
044h 4     00h-filled
048h 3        40 20 00     ;or 58 71 0F when disk inserted
04Bh 1     changes randomly (nodisk: 00 or 80 / disk: BFh)
04Ch 1     Zero (or C0h)
04Dh 3     MM:SS:FF (begin of current track MM:SS:00h) (or increasing addr)
050h 10    Subchannel Q (adjusted position values)
05Ah 2        ...
05Ch 1     00h (or 64h)
05Dh 3     MM:SS:FF (current read address) (sticky address during pause)
060h 1     increments at circa 16Hz or so (or other rate when spinning)
061h 12    00h-filled ;or else when disk inserted
```

```
  06Dh 1       01      ;or 0C when disk inserted
  06Eh 2     SetFilter setting (file,channel)
  070h 16    00h-filled ;or else when disk inserted
  080h 8     00h-filled
  088h 3       03:SS:FF (three, second, fraction)
  08Bh 3       03:SS:FF (three, second, fraction)
  08Eh 2       01 FF (or other values)
  090h 1       00h (or 91h when disk inserted + spinning)
  091h 13    Zero
  09Eh 1       00h (or 01h when disk inserted + spinning)
  09Fh 1     Zero
  0A0h 1     Always 23h
  0A1h 1       09h (5Dh when disk inserted)
  0A2h 7     00h-filled
  0A9h 1       40
  0AAh 4     00h-filled
  0AEh 1       00 (no disk) or 01 (disk) or so
  0AFh 1       00            ;or 06 when disk inserted
  0B0h 7       00 DC 00 02 00 E0 08             ;\or else when disk inserted
  0B7h 1       20         ;Bit6+7=MUTE          ;
  0B8h 3       DE 00 00                         ;/
  0BBh 1     SetMode setting (mode)
  0BCh 1     GetStat setting (stat)
  0BDh 3     00h-filled
  0C0h 6     FFh-filled            ;stack...                   ;\
  0C6h 1     Usually DFh           ;sometimes [0EBh and up] are non-FFh, too
  0C7h 15    FFh-filled            ;(depending on disk or commands or so)
  0D6h 1     Usually FDh (or FFh)  ;                           ;
  0D7h 24    FFh-filled                                        ; stack
  0EFh 4     on power-up FFh-filled, other once when disk read  ;
  0F3h 7     changes randomly (even when no disk inserted)      ;
  0FAh 6       2E 3C 2A D6 10 95                               ;/
  100h 2x99  TOC Entries for Start of Track 1..99 (MM:SS)
  1C6h 1     TOC First Track number (usually 01h)
  1C7h 1     TOC Last Track number (usually 01h or higher)
  1C8h 3     TOC Entry for Start of Lead-Out (MM:SS:FF)
  1CBh 2     Zero
  1CDh 1     Depends on disk (01 or 02 or 06) (or 00 when no disk)
  1CEh 1     Zero
  1CFh 1     Depends on disk (NULL minus N*6) (or 00 when no disk)
               (maybe reflection level / laser intensity or so)
                [1CDh..1CFh]
                01 00 E8 --> licensed/metalgear/kain
                01 00 EE --> licensed/alone2
                06 00 E2 or 00 00 02 00 E8 --> licensed/wipeout
                02 00 DC --> unlicensed/elo
                02 00 D6 --> unlicensed/driver
                00 00 EE --> audio/lola
                00 00 FA --> audio/marilyn
                00 00 F4 --> audio/westen
                00 00 00 --> disk missing
               last byte is always in steps of 6
  1D0h 4     SCEx String
  1D4h 4     Zero
  1D8h 2     SCEx Counters (total,success)  ;for command 19h,05h
  1DAh 6       00h-filled     (or ... SS:FF)
  1E0h 6     Command Buffer (usually 19h,60h,E2h,01h = Read RAM Command)
  1E6h 7       00h-filled (unless destroyed by more-than-6-byte-commands)
  1EDh 3     Setloc setting (MM:SS:FF)
  1F0h 1       00h          (unless destroyed by more-than-6-byte-commands)
  1F1h 3       C0h 00h 00h ;or 20h,0Ch,50h or C0h,0Ch,08h ;for command(19h,75h)
                           ;or 00h,00h,00h for audio
                           ;or 80h,00h,00h for disk missing
  1F4h 4       00h-filled ... or SCEx string
  1F8h 1       00h
  1F9h 1     Selected Target (parameter from Play and SetSession commands)
  1FAh 5       00h-filled       ;01 01 00 8B 00 00   ;or 01 02 8B 00 00
                       01 00 8B 00 00 -- audio/unlicensed
                       01 01 00 00 00 -- licensed
  1FFh 1       00h-on power up, changes when disk inserted  ;or 01 = Playing
    1FDh 3       MM:SS:FF (only during command 19h,00h) (MM=98..99=TOC)
  200h 10    Subchannel Q (real values)
  20Ah 2       whatever
  20Ch 1     Zero
  20Dh 1     Desired Session (from SetSession command)
  20Eh 1     Current Session (actual location of drive head)
  20Fh 1     Zero
  210h 10    Subchannel Q (adjusted position values)
  21Ah 6     00h-filled
  220h 4     Data Sector Header (MM:SS:FF:Mode)
  224h 4     Data Sector CD-XA Subheader (file,channel,sm,ci)
  228h 1         00h
  229h 1     Usually 00h (shortly other value on power-up, and maybe on seek)
  22Ah 1         10h (or 00h when no disk)
  22Bh 3     00h-filled
  22Eh 2         01,03 or 0A,00 or 03,01 (or else for other disk)
  230h 3     00h-filled (or other during spin-up / read-toc or so)
  233h 0Dh   00h-filled (unused RAM)
```

Other/invalid addresses are:

```
  240h..2FFh  - Invalid (00h-filled) (no ROM, RAM, or I/O mapped here)
  300h..3FFh  - Mirror of 200h..2FFh    ;\the BIOS is doing that
  400h..FFFFh - Mirrors of 000h..3FFh   ;/mirroring by software
```

Writing to RAM

There is no command for writing to RAM. Except that, one can write to the command/parameter buffer at 1E0h and up. Normally, the longest known command should have 6 bytes (19h,76h,a,b,c,d), and longer commands results in "bad number of parameters" response - however, despite of that error message, the controller does still store ALL parameter bytes in RAM (at address 1E1h..2E0h, then wrapping back to 1E1h). Whereas, writing more than 16 bytes (FIFO storage size) will mirror the FIFO content twice, and more than 32 bytes (FIFO counter size) will work only when feeding extra data into the FIFO during transmission. Anyways, writing to 1E1h and up doesn't allow to do interesting things (such like manipulating the stack and executing custom code on the CPU).

Subchannel Q Notes

The "adjusted position values" at 050h, 210h, 310h contain only position information (with ADR=1) (the PSX seems to check only the lower 2bit of the 4bit ADR

value, so it also treats ADR=5 as ADR=1, too). Additionally, during Lead-In, bytes 7..9 are overwritten by the position value from bytes 3..5. The "real values" contain unadjusted data, including ADR=2 and ADR=3 etc.

CDROM - Secret Unlock Commands

SecretUnlockPart1 - Command 50h --> INT5(11h,40h)
SecretUnlockPart2 - Command 51h,"Licensed by" --> INT5(11h,40h)
SecretUnlockPart3 - Command 52h,"Sony" --> INT5(11h,40h)
SecretUnlockPart4 - Command 53h,"Computer" --> INT5(11h,40h)
SecretUnlockPart5 - Command 54h,"Entertainment" --> INT5(11h,40h)
SecretUnlockPart6 - Command 55h,<region> --> INT5(11h,40h)
SecretUnlockPart7 - Command 56h --> INT5(11h,40h)

```
Caution: Supported only in BIOS version vC1 and up. Not supported in vC0.
Caution: Functional only in Europe/USA. Nonfunctional in Japan.
```

Sending these commands with the correct strings (in order 50h through 56h) does disable the "SCEx" protection. The region can be detected via test command 19h,22h, and must be translated to the following <region> string string:

```
"of America"    ;for NTSC/US          ;\handled, and actually working
"(Europe)"      ;for PAL/Europe       ;/
"Inc."          ;for NTSC/JP          ;-handled, but made non-functional
```

In the unlocked state, ReadN/ReadS are working for unlicensed CD-Rs, and for imported CDROMs from other regions (both without needing modchips). However there are some cases which may still cause problems: The GetID command (1Ah) does still identify the disc as being unlicensed, same for the Get SCEx Counters test command (19h,05h). And, if a game should happen to send the Reset command (1Ch) for some weird reason, then the BIOS would forget the unlocking, same for games that set the "HCRISD" I/O port bit. On the contrary, opening/closing the drive door does not affect the unlocking state.
The commands have been discovered in September 2013, and appear to be supported by all CDROM BIOS versions (from old PSXes up to later PSones).
Note that the commands do always respond with INT5 errors (even on successful unlocking).
Japanese consoles are internally processing the Secret Unlock commands, but they are ignoring the resulting unlocking flag, making the commands nonfunctional in Japan/Asia regions.

SecretLock - Command 57h --> INT5(11h,40h)
Undoes the unlocking and restores the normal locked state (same happens when sending the Unlocking commands in wrong order or with wrong parameters).

SecretCrash - Command 58h..5Fh --> Crash
Jumps to a data area and executes random code. Results are more or less unpredictable (as they involve executing undefined opcodes). Eventually the CPU might hit a RET opcode and recover from the crash.

CDROM - Mainloop/Responses

SUB-CPU Mainloop
The SUB-CPU is running a mainloop that is handling hardware events (by simple polling, not by IRQs):

```
check for incoming sectors (from CDROM decoder)
check for incoming commands (from Main CPU)
do maintenance stuff on the drive mechanics
```

There is no fixed priority: if both incoming sector and incoming command are present, then the SUB-CPU may handle either one, depending on which portion of the mainloop it is currently executing.
There is no fixed timing: if the mainloop is just checking for a specific event, then a new event may be processed immediately, otherwise it may take whole mainloop cycle until the SUB-CPU sees the event.
Whereas, the mainloop cycle execution time isn't constant: It may vary depending on various details. Especially, some maintenance stuff is only handled approximately around 15 times per second (so there are 15 slow mainloop cycles per second).

Responses
The PSX can deliver one INT after another. Instead of using a real queue, a it's merely using some flags that do indicate which INT(s) need to be delivered. Basically, there seem to be two flags: One for Second Response (INT2), and one for Data/Report Response (INT1). There is no flag for First Response (INT3); because that INT is generated immediately after executing a command.
The flag mechanism means that the SUB-CPU cannot hold more than undelivered INT1. Ie. the CDROM Decoder does notify that SUB-CPU about all newly received sectors, and it can hold up to eight sectors in the 32K SRAM, but the SUB-CPU BIOS merely sets a sector-delivery-needed flag (instead of memorizing which/how many sectors need to be delivered) (accordingly, the PSX can use only three of the available eight SRAM slots: One for currently pending INT1, one for undelivered INT1, and one for currently/incompletly received sector).

First Response (INT3) (or INT5 if failed)
The first response is sent immediately after processing a command. In detail:
The mainloop checks for incoming commands once every some clock cycles, and executes commands under following condition:

```
Main CPU has sent a command, AND, there is no INT pending
(if an INT is pending, then the command won't be executed yet, but will be
executed in following mainloop cycles; once when INT got acknowledged)
(even if no INT is pending, the mainloop may generate INT1/INT2 before
executing the command, if so, as said above, the command won't execute yet)
```

Once when the command gets executed it will sent the first response immediately after the command execution (which may only take a few clock cycles, or some more cycles, for example Init/ReadTOC do include some time consuming initializations). Anyways, there will be no other INTs generated during command execution, so once when the command execution has started, it's guaranteed that the next INT will contain the first response.

Second Responses (INT2) (or INT5 if failed)
Some commands do send a second response after actual command execution:

```
07h MotorOn    E -              INT3(stat), INT2(stat)
08h Stop       E -              INT3(stat), INT2(stat)
09h Pause      E -              INT3(stat), INT2(stat)
0Ah Init         -              INT3(late-stat), INT2(stat)
12h SetSession E session        INT3(stat), INT2(stat)
15h SeekL      E -              INT3(stat), INT2(stat)  ;\use prior Setloc
16h SeekP      E -              INT3(stat), INT2(stat)  ;/to set target
1Ah GetID      E -              INT3(stat), INT2/5(stat,flg,typ,atip,"SCEx")
1Dh GetQ       E adr,point      INT3(stat), INT2(10bytesSubQ,peak_lo)
1Eh ReadTOC      -              INT3(late-stat), INT2(stat)
```

In some cases (like seek or spin-up), it may take more than a second until the 2nd response is sent.
It should be highly recommended to WAIT until the second response is generated BEFORE sending a new command (it wouldn't make too much sense to send a new command between first and second response, and results would be unknown, and probably totally unpredictable).
Error Notes: If the command has been rejected (INT5 sent as 1st response) then the 2nd response isn't sent (eg. on wrong number of parameters, or if disc missing). If the command fails at a later stage (INT5 as 2nd response), then there are cases where another INT5 occurs as 3rd response (eg. on SetSession=02h on non-

multisession-disk).

Data/Report Responses (INT1)

```
  03h Play       E (track)         INT3(stat), optional INT1(report bytes)
  04h Forward    E -               INT3(stat), optional INT1(report bytes)
  05h Backward   E -               INT3(stat), optional INT1(report bytes)
  06h ReadN      E -               INT3(stat), INT1(stat), datablock
  1Bh ReadS      E?-               INT3(stat), INT1(stat), datablock
```

CDROM - Response Timings

Here are some response timings, measured in 33MHz units on a PAL PSone. The CDROM BIOSes mainloop is doing some maintenance stuff once and when, meaning that the response time will be higher in such mainloop cycles (max values), and less in normal cycles (min values). The maintenance timings do also depend on whether the motor is on or off (and probably on various other factors like seeking).

First Response

The First Response interrupt is sent almost immediately after processing the command (that is, when the mainloop sees a new command without any old interrupt pending). For GetStat, timings are as so:

```
  Command                Average   Min       Max
  GetStat (normal)       000c4e1h  0004a73h..003115bh
  GetStat (when stopped) 0005cf4h  000483bh..00093f2h
```

Timings for most other commands should be similar as above. One exception is the Init command, which is doing some initialization before sending the 1st response:

```
  Init                   0013cceh  000f820h..00xxxxxh
```

The ReadTOC command is doing similar initialization, and should have similar timing as Init command. Some (rarely used) Test commands include things like serial data transfers, which may be also quite slow.

Second Response

```
  Command                Average   Min       Max
  GetID                  0004a00h  0004922h..0004c2bh
  Pause (single speed)   021181ch  020eaefh..0216e3ch ;\time equal to
  Pause (double speed)   010bd93h  010477Ah..011B302h ;/about 5 sectors
  Pause (when paused)    0001df2h  0001d25h..0001f22h
  Stop (single speed)    0d38acah  0c3bc41h..0da554dh
  Stop (double speed)    18a6076h  184476bh..192b306h
  Stop (when stopped)    0001d7bh  0001ce8h..0001eefh
```

Moreover, Seek/Play/Read/SetSession/MotorOn/Init/ReadTOC are sending second responses which depend on seek time (and spin-up time if the motor was off).
The seek timings are still unknown, and they are probably quite complicated:
The CDROM BIOS seems to split seek distance somehow into coarse steps (eg. minutes) and fine steps (eg. seconds/sectors), so 1-minute seek distance may have completely different timings than 59-seconds distance.
The amount of data per spiral winding increases towards ends of the disc (so the drive head will need to be moved by shorter distance when moving from minute 59 to 60 as than moving from 00 to 01).
The CDROM BIOS contains some seek distance table, which is probably optimized for 72-minute discs (or whatever capacity is used on original PSX discs). 80-minute CDRs may have tighter spiral windings (the above seek table is probably causing the drive head to be moved too far on such discs, which will raise the seek time as the head needs to be moved backwards to compensate that error).

INT1 Rate

```
  Command                Average   Min       Max
  Read (single speed)    006e1cdh  00686dah..0072732h
  Read (double speed)    0036cd2h  00322dfh..003ab2bh
```

The INT1 rate needs to be precise for CD-DA and CD-XA Audio streaming, exact clock cycle values should be: SystemClock*930h/4/44100Hz for Single Speed (and half as much for Double Speed) (the "Average" values are AVERAGE values, not exact values).

CDROM - Response/Data Queueing

[Below are some older/outdated test cases]

Sector Buffer

The CDROM sector buffer is 32Kx8 SRAM (IC303). The buffer is apparently divided into 8 slots, theoretically allowing to buffer up to 8 sectors.
BUG: The drive controller seems to allow only 2 of those 8 sectors (the oldest sector, and the current/newest sector).
Ie. after processing the INT1 for the oldest sector, one would expect the controller to generate another INT1 for next newer sector - but instead it appears to jump directly to INT1 for the newest sector (skipping all other unprocessed sectors). There is no known way to get around that effect.
So far, the big 32Kbyte buffer is entirely useless (the two accessible sectors could have been as well stored in a 8Kbyte chip) (unless, maybe the 32Kbytes have been intended for some error-correction "read-ahead" purposes, rather than as "look-back" buffer for old sectors; one of the unused slots might be also used for XA-ADPCM sectors).
The bottom line is that one should process INT1's as soon as possible (ie. before the cdrom controller receives and skips further sectors). Otherwise sectors would be lost without notice (there appear to be absolutely no overrun status flags, nor overrun error interrupts).

Sector Buffer Test Cases

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Process INT1 --> receives sector header for 0:2:1
  Process INT1 --> receives sector header for 0:2:2
  Process INT1 --> receives sector header for 0:2:3
```

Above shows the normal flow when processing INT1's as they arise. Now, inserting delays (and not processing INT1's during that delays):

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  delay(1)
  Process INT1 --> receives sector header for 0:2:1 (oldest sector)
  Process INT1 --> receives sector header for 0:2:6 (newest sector)
  Process INT1 --> receives sector header for 0:2:7 (next sector)
```

Above suggests that the CDROM buffer can hold max 2 sectors (the oldest and current one). However, using a longer delay:

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  delay(2)
  Process INT1 --> receives sector header for 0:2:9  (oldest/overwritten)
  Process INT1 --> receives sector header for 0:2:11 (newest sector)
  Process INT1 --> receives sector header for 0:2:12 (next sector)
```

Above indicates that sector buffer can hold 8 sectors (as the sector 1 slot is overwritten by sector 9). And, another test with even longer delay:

```
  Setloc(0:2:0)+Read
```

```
Process INT1 --> receives sector header for 0:2:0
delay(3)
Process INT1 --> receives sector header for 0:2:17 (currently received)
Process INT1 --> receives sector header for 0:2:16 (newest full sector)
Process INT1 --> receives sector header for 0:2:17 (next sector)
Process INT1 --> receives sector header for 0:2:18 (next sector)
```

Above is a special case where sector 17 appears twice; the first one is the sector 1 slot (which was overwritten by sector 9, and apparently then half overwritten by sector 17).

Sector Buffer VS GetlocL Response Tests

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
GetlocL
Process INT3 --> receives getloc info for 0:2:0
Process INT1 --> receives sector header for 0:2:1
Process INT1 --> receives sector header for 0:2:2
Process INT1 --> receives sector header for 0:2:3
```

Another test, with Delay BEFORE Getloc:

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Delay(1)
GetlocL
Process INT1 --> receives sector header for 0:2:1
Process INT3 --> receives getloc info for 0:2:6
Process INT1 --> receives sector header for 0:2:6
Process INT1 --> receives sector header for 0:2:7
```

Another test, with Delay AFTER Getloc:

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
GetlocL
Delay(1)
Process INT3 --> receives getloc info for 0:2:0
Process INT1 --> receives sector header for 0:2:5
Process INT1 --> receives sector header for 0:2:6
Process INT1 --> receives sector header for 0:2:7
```

Another test, with Delay BEFORE and AFTER Getloc:

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Delay(1)
GetlocL
Delay(1)
Process INT1 --> receives sector header for 0:2:9
Process INT1 --> receives sector header for 0:2:11
Process INT3 --> receives getloc info for 0:2:12
Process INT1 --> receives sector header for 0:2:12
Process INT1 --> receives sector header for 0:2:13
```

Sector Buffer VS Pause Response Tests

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Pause
Process INT3 --> receives stat=22h (first pause response)
Process INT2 --> receives stat=02h (second pause response)
```

Another test, with Delay BEFORE Pause:

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Delay(1)
Pause
Process INT1 --> receives sector header for 0:2:1 (oldest)
Process INT3 --> receives stat=22h (first pause response)
Process INT2 --> receives stat=02h (second pause response)
```

Another test, with Delay AFTER Pause:

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Pause
Delay(1)
Process INT3 --> receives stat=22h (first pause response)
Process INT2 --> receives stat=02h (second pause response)
```

Another test, with Delay BEFORE and AFTER Pause:

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Delay(1)
Pause
Delay(1)
Process INT1 --> receives sector header for 0:2:9 (oldest/overwritten)
Process INT3 --> receives stat=22h (first pause response)
Process INT2 --> receives stat=02h (second pause response)
```

For above: Note that, despite of Pause, the CDROM is still writing to the internal buffer (and overwrites slot 1 by sector 9) (this might be because the Pause command isn't processed at all until INT1 is processed).

Double Commands (Getloc then Pause)

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
GetlocL
Pause
Process INT3 --> receives stat=22h (first pause response)
Process INT2 --> receives stat=02h (second pause response)
```

Another test,

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
Delay(1)
GetlocL
Pause
Process INT1 --> receives sector header for 0:2:1
Process INT1 --> receives sector header for 0:2:6
Process INT3 --> receives stat=22h (first pause response)
Process INT2 --> receives stat=02h (second pause response)
```

Another test,

```
Setloc(0:2:0)+Read
Process INT1 --> receives sector header for 0:2:0
```

```
  GetlocL
  Delay(1)
  Pause
  Process INT3 --> receives getloc info for 0:2:0 (first getloc response)
  Process INT3 --> receives stat=22h (first pause response)
  Process INT2 --> receives stat=02h (second pause response)
```

Another test,

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Delay(1)
  GetlocL
  Delay(1)
  Pause
  Process INT1 --> receives sector header for 0:2:9 (oldest/overwritten)
  Process INT3 --> receives stat=22h (first pause response)
  Process INT2 --> receives stat=02h (second pause response)
```

Double Commands (Pause then Getloc)

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Pause
  GetlocL
  Process INT3 --> receives getloc info for 0:2:0 (first getloc response)
  Process INT1 --> receives sector header for 0:2:1
  Process INT1 --> receives sector header for 0:2:2
  Process INT1 --> receives sector header for 0:2:3
```

Another test,

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Delay(1)
  Pause
  GetlocL
  Process INT1 --> receives sector header for 0:2:1
  Process INT3 --> receives getloc info for 0:2:6 (first getloc response)
  Process INT1 --> receives sector header for 0:2:6
  Process INT1 --> receives sector header for 0:2:7
```

Another test,

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Pause
  Delay(1)
  GetlocL
  Process INT3 --> receives stat=22h (first pause response)
  Process INT3 --> receives getloc info for 0:2:6 (first getloc response)
  (No further INT's, ie. read is paused, but second-pause-response is lost).
```

Another test,

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Pause
  Delay(1)
  GetlocL
  Delay(1)
  Process INT3 --> receives stat=22h (first pause response)
  Process INT3 --> receives getloc info for 0:2:6 (first getloc response)
  Process INT2 --> receives stat=02h (second pause response)
```

Another test,

```
  Setloc(0:2:0)+Read
  Process INT1 --> receives sector header for 0:2:0
  Delay(1)
  Pause
  Delay(1)
  GetlocL
  Process INT1 --> receives sector header for 0:2:9
  Process INT1 --> receives sector header for 0:2:11
  Process INT3 --> receives getloc info for 0:2:12 (first getloc response)
  Process INT1 --> receives sector header for 0:2:12
  Process INT1 --> receives sector header for 0:2:13
```

CDROM Disk Format

Overview

The PSX uses a ISO 9660 filesystem, with data stored on CD-XA (Mode2) Sectors. ISO 9660 is standard for CDROM disks, although newer CDROMs may use extended filesystems, allowing to use long filenames and lowercase filenames, the PSX Kernel doesn't support such stuff, and, in fact, it's putting some restrictions on the ISO standard: it's limiting files names to MSDOS-style 8.3 format, and it's allowing only a limited number of files and directories per disk.

CDROM Filesystem (ISO 9660 aka ECMA-119)

```
  Originally intended for Mode1 Sectors (but is also used for CD-XA Mode2)
  Supports "FILENAME.EXT;VERSION" filenames (version is usually "1")
  Supports all-uppercase filenames and directory names (0-9, A-Z, underscore)
  For PSX: Max 8-character filenames with max 3-character extensions
  For PSX: Max 8-character directory names, without extension
  For PSX: Max one sector per directory (?)
  For PSX: Max one sector (or less?) per path table (?)
```

CDROM Extended Architecture (CD-ROM XA aka CD-XA)

```
  Uses Mode2 Sectors (see Sector Encoding chapter)
  Allows 800h or 914h byte data per sector (with/without error correction)
  Allows to break interleaved data into separate files/channels
  Supports XA-ADPCM compressed audio data
  Stores "CD-XA001" at 400h Primary Volume Descriptor (?)
  Stores whatever 14 bytes in System Use area of Directory Entries ("UXA") (?)
```

Physical Audio/CDROM Disk Format (ISO/IEC 10149 aka ECMA-130)

```
  Defines physical metrics of the CDROM and Audio disks
  Defines Sub-channels and Track.Index and Minute.Second.Fraction numbering
  Defines 14bit-per-byte encoding, and splits sectors into frames
  Defines ECC and EDC (error correction and error detection codes)
```

Available Documentation
ISO documents are commercial standards (not available for download), however, they are based on ECMA standards (which are free for download, however, the ECMA stuff is in PDF format, so one may treat it as commercial bullshit, too). CD-ROM XA is commercial only (not available for download), and, CD-XA doesn't seem to have become very popular outside of the PSX-world, so there's very little information available, portions of CD-XA are also used in the CD-i standard (which may be a little better or worse documented).

Stuff
```
sessions  one or more sessions per disk
tracks    99 tracks per disk     (01h..99h) (usually only 01h on Data Disks)
index     99 indices per track   (01h..99h) (rarely used, usually always 01h)
minutes   74 minutes per disk    (00h..73h) (or more, with some restrictions)
seconds   60 seconds per minute  (00h..59h)
sectors   75 sectors per second  (00h..74h)
frames    98 frames per sector
bytes     33 bytes per frame (24+1+8 = data + subchannel + error correction)
bits      14 bits per byte   (256 valid combinations, and many invalid ones)
```

Track.Index (stored in subchannel, in BCD format)
Multiple Tracks are usually used only on Audio Disks (one track for each song, numbered 01h and up), a few Audio Disks may also split Tracks into separate fragments with different Index values (numbered 01h and up, but most tracks have only Index 01h). A simple Data Disk would usually contain only one Track (all sectors marked Track=01h and Index=01h), although some more complex Data Disks may have multiple Data tracks and/or Audio tracks.

Minute.Second.Sector (stored in subchannel, and in Data sectors, BCD format)
The sectors on CDROMs and CD Audio disks are numbered in Minutes, Seconds, and 1/75 fragments of a second (where a "second" is referring to single-speed drives, ie. the normal CD Audio playback speed).
Minute.Second.Sector is stored twice in the subchannel (once the "absolute" time, and once the "local" time).
The "absolute" sector number (counted from the begin of the disk) is mainly relevant for Seek purposes (telling the controller if the drive head is on the desired location, or if it needs to move the head backwards or forwards).
The "local" sector number (counted from the begin of the track) is mainly relevant for Audio Players, allowing to pass the data directly to the Minute:Second display, without needing to subtract the start address of the track.
Data disks are additionally storing the "absolute" values in their Data Areas, basically that's just the subchannel data duplicated, but more precisely assigned - the problem with the subchannel data is that the CD Audio standard seems to lack a clear definition that would assign the begin of the sub-channel block to the exact begin of a sector; so, when using only the subchannel data, some Drive Controllers may assign the begin of a new sector to another location as than other Controllers do, for Audio Disks that isn't too much of a problem, but for Data Disks it'd be fatal.

Subchannels
Each frame contains 8 subchannel bits (named P,Q,R,S,T,U,V,W). So, a sector (with 98 frames) contains 98 bits (12.25 bytes) for each subchannel.
CDROM Subchannels

Error Correction
Each Frame contains 8 bytes Error Correction information, which is mainly used for Audio Disks, but it isn't 100% fail-proof, for that reason, Data Disks are containing additional Error Correction in the 930h-byte data area (the audio correction is probably focusing on repairing the MSBs of the 16bit samples, and gives less priority on the LSBs). Error Correction is some kind of a huge complex checksum, which allows to detect the location of faulty bytes, and to fix them.

930h-Byte Sectors
The "user" area for each sector is 930h bytes (2352 bytes). That region is combined of the 24-byte data per frame (and excludes the 8-byte audio error correction info, and the 1-byte subchannel data).
Most CDROM Controllers are only giving access to this 930h-byte region (ie. there's no way to read the audio error correction info by software, and only limited access to the subchannel data, such like allowing to read only the Q-channel for showing track/minute/second in audio playback mode).
On Audio disks, the 930h bytes are plain data, on Data disks that bytes are containing headers, error correction, and usually only 800h bytes user data (for more info see Sector Encoding chapter).

Sessions
Multi-Sessions are mainly used on CDR's, allowing to append newer data at the end of the disk at a later time. First of, the old session must contain a flag indicating that there may be a newer session, telling the CDROM Controller to search if one such exists (and if that is equally flagged, to search for an even newer session, and so on until reaching the last and newest session).
Each session contains a complete new ISO Volume Descriptor, and may additionally contain new Path Tables, new Directories, and new Files. The Driver Controller is usually recursing only the Volume Descriptor of the newest session. However, the various Directory Records of the new session may refer to old files or old directories from previous sessions, allowing to "import" the older files, or to "rename" or "delete" them by assigning new names to that files, or by removing them from the directory.
The PSX is reportedly not supporting multi-session disks, but that doesn't seem to be correct, namely, the Setsession command is apparently intended for that purpose... though not sure if the PSX Kernel is automatically searching the newest session... otherwise the boot executable in the first session would need to do that manually by software, and redirect control to the boot executable in the last session.

CDROM Subchannels

Subchannel P
Subchannel P contains some kind of a Pause flag (to indicate muted areas between Audio Tracks). This subchannel doesn't have any checksum, so the data cannot be trusted to be intact (unless when sensing a longer stream of all-one's, or all zero's). Theoretically, the 98 pause bits are somehow associated to the 98 audio frames (with 24 audio bytes each) of the sector. However, reportedly, Subchannel P does contain two sync bits, if that is true, then there'd be only 96 pause flags for 98 audio frames. Strange.
Note: Another way to indicate "paused" regions is to set Subchannel Q to ADR=1 and Index=00h.

Subchannel Q
contains the following information:
```
Bits Expl.
2    Sub-channel synchronization field
8    ADR/Control (see below)
72   Data (content depends on ADR)
16   CRC-16-CCITT error detection code (big-endian: bytes ordered MSB, LSB)
```
Possible values for the ADR/Control field are:
```
Bit0-3 ADR (0=No data, 1..3=see below, 4..0Fh=Reserved)
Bit4   Audio Preemphasis  (0=No, 1=Yes)      (Audio only, must be 0 for Data)
Bit5   Digital Copy       (0=Prohibited, 1=Allowed)
Bit6   Data               (0=Audio, 1=Data)
Bit7   Four-Channel Audio (0=Stereo, 1=Quad) (Audio only, must be 0 for Data)
```
The 72bit data regions are, depending on the ADR value...

Subchannel Q with ADR=1 during Lead-In -- Table of Contents (TOC)

```
  8    Track number (fixed, must be 00h=Lead-in)
  8    Point (01h..99h or A0h..A2h, see last three bytes for more info)
  24   MSF address (incrementing address within the Lead-in area)
         Note: On some disks, these values are choosen so that the lead-in
         <starts> at 00:00:00, on other disks so that it <ends> at 99:59:74.
  8    Reserved (00h)
```

When Point=01h..99h (Track 1..99) or Point=A2h (Lead-Out):

```
  24   MSF address (absolute address, start address of the "Point" track)
```

When Point=A0h (First Track Number):

```
  8    First Track number (BCD)
  8    Disk Type Byte (00h=CD-DA or CD-ROM, 10h=CD-I, 20h=CD-ROM-XA)
  8    Reserved (00h)
```

When Point=A1h (Last Track Number):

```
  8    Last Track number (BCD)
  16   Reserved (0000h)
```

ADR=1 should exist in 3 consecutive lead-in sectors.

Subchannel Q with ADR=1 in Data region -- Position

```
  8    Track number (01h..99h=Track 1..99)
  8    Index number (00h=Pause, 01h..99h=Index within Track)
  24   Track relative MSF address (decreasing during Pause)
  8    Reserved (00h)
  24   Absolute MSF address
```

ADR=1 is required to exist in at least 9 out of 10 consecutive data sectors.

Subchannel Q with ADR=1 during Lead-Out -- Position

```
  8    Track number (fixed, must be AAh=Lead-Out)
  8    Index number (fixed, must be 01h) (there's no Index=00h in Lead-Out)
  24   Track relative MSF address (increasing, 00:00:00 and up)
  8    Reserved (00h)
  24   Absolute MSF address
```

ADR=1 should exist in 3 consecutive lead-out sectors (and may be then followed by ADR=5 on multisession disks).

Subchannel Q with ADR=2 -- Catalogue number of the disc (UPC/EAN barcode)

```
  52   EAN-13 barcode number (13-digit BCD)
  12   Reserved (000h)
  8    Absolute Sector number (BCD, 00h..74h) (always 00h during Lead-in)
```

If the first digit of the EAN-13 number is "0", then the remaining digits are a UPC-A barcode number. Either the 13-digit EAN-13 number, or the 12-digit UPC-A number should be printed as barcode on the rear-side of the CD package.
The first some digits contain a country code (EAN only, not UPC), followed by a manufacturer code, followed by a serial number. The last digit contains a checksum, which can be calculated as 250 minus the sum of the first 12 digits, minus twice the sum of each second digit, modulated by 10.
ADR=2 isn't included on all CDs, and, many CDs do have ADR=2, but the 13 digits are all zero. Most CDROM drives do not allow to read EAN/UPC numbers.
If present, ADR=2 should exist in at least 1 out of 100 consecutive sectors. ADR=2 may occur also in Lead-in.

Subchannel Q with ADR=3 -- ISRC number of the current track

(ISO 3901 and DIN-31-621):

```
  12   Country Code      (two 6bit characters)   (ASCII minus 30h) ;eg. "US"
  18   Owner Code        (three 6bit characters) (ASCII minus 30h)
  2    Reserved          (zero)
  8    Year of recording (2-digit BCD) ;eg. 82h for 1982
  20   Serial number     (5-digit BCD) ;usually increments by 1 or 10 per track
  4    Reserved          (zero)
  8    Absolute Sector number (BCD, 00h..74h) (always 00h during Lead-in)
```

Most CDROM drives for PC's do not allow to read ISRC numbers (or even worse, they may accidently return the same ISRC number on every two tracks).
If present, ADR=3 should exist in at least 1 out of 100 consecutive sectors. However, reportedly, ADR=3 should not occur in Lead-in.

Subchannel Q with ADR=5 in Lead-in -- Multisession Lead-In Info

When Point=B0h:

```
  8    Track number (fixed, must be 00h=Lead-in)
  8    POINT = B0h (multi-session disc)
  24   MM:SS:FF = the start time for the next possible session's program area,
       a final session is indicated by FFh:FFh:FFh,
       or when the ADR=5 / Point=B0h is absent.
  8    Number of different Mode-5 pointers present.
  24   MM:SS:FF = the maximum possible start time of the outermost Lead-out
```

When Point=C0h:

```
  8    Track number (fixed, must be 00h=Lead-in)
  8    POINT = C0h (Identifies a Multisession disc, together with POINT=B0h)
  24   ATIP values from Special Information 1, ID=101
  8    Reserved (must be 00h)
  24   MM:SS:FF = Start time of the first Lead-in area of the disc
```

And, optionally, when Point=C1h:

```
  8    Track number (fixed, must be 00h=Lead-in)
  8    POINT=C1h
  8x7  Copy of information from A1 point in ATIP
```

Subchannel Q with ADR=5 in Lead-Out -- Multisession Lead-Out Info

```
  8    Track number (fixed, must be AAh=Lead-out)
  8    POINT = D1h (Identifies a Multisession lead-out)
  24   Usually zero (or maybe ATIP as in Lead-In with Point=C0h...?)
  8    Seems to be the session number?
  24   MM:SS:FF = Absolute address of the First data sector of the session
```

Present in 3 consequtive sectors (3x ADR=1, 3x ADR=5, 3x ADR=1, 3x ADR=5, etc).

Subchannel Q with ADR=5 in Lead-in -- CDR/CDRW Skip Info (Audio Only)

When Point=01h..40h:

```
  8    Track number (fixed, must be 00h=Lead-in)
  8    POINT=01h..40h (This identifies a specific playback skip interval)
  24   MM:SS:FF Skip interval stop time in 6 BCD digits
  8    Reserved (must be 00h)
  24   MM:SS:FF Skip interval start time in 6 BCD digits
```

When Point=B1h:

```
  8    Track number (fixed, must be 00h=Lead-in)
  8    POINT=B1h (Audio only: This identifies the presence of skip intervals)
  8x4  Reserved (must be 00h,00h,00h,00h)
  8    the number of skip interval pointers in POINT=01h..40h
```

```
8    the number of skip track assignments in POINT=B2h..B4h
8    Reserved (must be 00h)
```
When Point=B2h,B3h,B4h:
```
8    Track number (fixed, must be 00h=Lead-in)
8    POINT=B2h,B3h,B4h (This identifies tracks that should be skipped)
8    1st Track number to skip upon playback (01h..99h, must be nonzero)
8    2nd Track number to skip upon playback (01h..99h, or 00h=None)
8    3rd Track number to skip upon playback (01h..99h, or 00h=None)
8    Reserved (must be 00h)... unclear... OR... 4th (of 7) skip info's...?
8    4th Track number to skip upon playback (01h..99h, or 00h=None)
8    5th Track number to skip upon playback (01h..99h, or 00h=None)
8    6th Track number to skip upon playback (01h..99h, or 00h=None)
```
Note: Skip intervals are seldom written by recorders and typically ignored by readers.

Subchannel R..W

Subchannels R..W are usually unused, except for some extended formats:
```
CD-TEXT in the Lead-In area (see below)
CD-TEXT in the Data area    (rarely used)
CD plus Graphics (CD+G)     (rarely used)
```
Most CDROM drives do not allow to read these subchannels. CD-TEXT was designed by Sony and Philips in 1997, so it should be found only on (some) newer discs. Most CD/DVD players don't support it (the only exception is that CD-TEXT seems to be popular for car hifi equipment). Most record labels don't support CD-TEXT, even Sony seems to have discontinued it on their own records after some years (so CD-TEXT is very rare on original disks, however, CDR software does often allow to write CD-TEXT on CDRs).

Subchannel R..W, when used for CD-TEXT in the Lead-In area

CD-TEXT is stored in the six Subchannels R..W. Of the 12.25 bytes (98 bits) per subchannel, only 12 bytes are used. Together, all 6 subchannels have a capacity of 72 bytes (6x12 bytes) per sector. These 72 bytes are divided into four CD-TEXT fragments (of 18 bytes each). The format of these 18 bytes is:
```
00h 1  Header Field ID1: Pack Type Indicator
01h 1  Header Field ID2: Track Number
02h 1  Header Field ID3: Sequence Number
03h 1  Header Field ID4: Block Number and Character Position Indicator
04h 12 Text/Data Field
10h 2  CRC-16-CCITT (big-endian) (across bytes 00h..0Fh)
```
ID1 - Pack Type Indicator:
```
80h   Titel      (TEXT)
81h   Performer  (TEXT)
82h   Songwriter (TEXT)
83h   Composer   (TEXT)
84h   Arranger   (TEXT)
85h   Message    (TEXT)
86h   Disc ID    (TEXT?)  (content/format/purpose unknown?)
87h   Genre      (BINARY) (ID codes unknown?)
88h   TOC        (BINARY) (content/format/purpose unknown?)
89h   TOC2       (BINARY) (content/format/purpose unknown?)
8Ah   Reserved for future
8Bh   Reserved for future
8Ch   Reserved for future
8Dh   Reserved for "content provider" aka "closed information"
8Eh   UPC/EAN and ISRC Codes (TEXT) (content/format/purpose unknown?)
8Fh   Blocksize (BINARY) (see below)
```
ID2 - Track Number:
```
00h       Title/Performer/etc. for the Disc
01h..63h  Title/Performer/etc. for Track 1..99 (Non-BCD) (Bit7=Extension)
```
ID3 - Sequence Number:
```
00h..FFh  Incrementing Number (00h=First 18-byte fragment, 01h=Second, etc.)
```
ID4 - Block Number and Character Position Indicator:
```
Bit7      Character Set      (0=8bit, 1=16bit)
Bit6-4    Block Number       (0..7 = Language number, as set by "Blocksize")
Bit3-0    Character Position (0..0Eh=Position, 0Fh=Append to prev fragment)
```
Example Data (generated with CDRWIN):
```
ID TR SQ CH <------------Text/Data------------> -CRC-  <---Text--->
80 00 00 00 54 65 73 74 44 69 73 6B 54 69 74 6C E2 22  TestDiskTitl
80 00 01 0C 65 00 54 65 73 74 54 72 61 63 6B 54 C9 1B  e.TestTrackT
80 01 02 0A 69 74 6C 65 31 00 54 65 73 74 54 72 40 3A  itle1.TestTr
80 02 03 06 61 63 6B 54 69 74 6C 65 32 00 00 00 80 E3  ackTitle2...
81 00 04 00 54 65 73 74 44 69 73 6B 50 65 72 66 03 DF  TestDiskPerf
81 00 05 0C 6F 72 6D 65 72 00 54 65 73 74 54 72 12 A5  ormer.TestTr
81 01 06 06 61 63 6B 50 65 72 66 6F 72 6D 65 72 BC 5B  ackPerformer
81 01 07 0F 31 00 54 65 73 74 54 72 61 63 6B 50 AC 41  1.TestTrackP
81 02 08 0A 65 72 66 6F 72 6D 65 72 32 00 00 00 64 1A  erformer2...
8F 00 09 00 01 01 02 00 04 05 00 00 00 00 00 00 6D E2  ............
8F 01 0A 00 00 00 00 00 00 00 00 03 0B 00 00 00 CD 0C  ............
8F 02 0B 00 00 00 00 00 09 00 00 00 00 00 00 00 FC 8C  ............
00   ;<--- for some reason, CDRWIN stores an ending 00h byte in .CDT files
```
Each Text string is terminated by a 00h byte (or 0000h for 16bit character set). If there's still room in the 12-byte data region, then first characters for the next Text string (for the next track) are appended after the 00h byte (if there's no further track, then the remaining bytes should be padded with 00h).
The "Blocksize" (ID1=8Fh) consists three packs with 24h bytes of data (first 0Ch bytes stored with ID2=00h, next 0Ch bytes with ID2=01h, and last 0Ch bytes with ID2=02h):
```
00h  1   Character set (00h,01h,80h,81h,82h = see below)
01h  1   First track number (usually/always 01h)
02h  1   Last track number (01h..63h)
03h  1   1bit-cd-text-in-data-area-flag, 7bit-copy-protection-flags
04h  16  Number of 18-byte packs for ID1=80h..8Fh
14h  8   Last sequence number of block 0..7 (or 00h=none)
1Ch  8   Language codes for block 0..7 (definitions are unknown)
```
Character Set values (for ID1=8Fh, ID2=00h, DATA[0]=charset):
```
00h ISO 8859-1
01h ISO 646, ASCII
80h MS-JIS
81h Korean character code
82h Mandarin (standard) Chinese character code
Other = reserved
```
"In case the same character stings is used for consecutive tracks, character 09h (or 0909h for 16bit charset) may be used to indicate the same as previous track. It shall not used for the first track."

adjust_crc_16_ccitt(addr_len) ;for CD-TEXT and Subchannel Q
```
lsb=00h, msb=00h      ;-initial value (zero for both CD-TEXT and Sub-Q)
for i=0 to len-1      ;-len (10h for CD-TEXT, 0Ah for Sub-Q)
  x = [addr+i] xor msb
```

```
  x = x xor (x shr 4)
  msb = lsb xor (x shr 3) xor (x shl 4)
  lsb = x xor (x shl 5)
next i
[addr+len+0]=msb xor FFh, [addr+len+1]=lsb xor FFh   ;inverted / big-endian
```

CDROM Sector Encoding

Audio

```
000h 930h Audio Data (2352 bytes) (LeftLsb,LeftMsb,RightLsb,RightMsb)
```

Mode0 (Empty)

```
000h 0Ch  Sync
00Ch 4    Header (Minute,Second,Sector,Mode=00h)
010h 920h Zerofilled
```

Mode1 (Original CDROM)

```
000h 0Ch  Sync
00Ch 4    Header (Minute,Second,Sector,Mode=01h)
010h 800h Data (2048 bytes)
810h 4    EDC (checksum accross [000h..80Fh])
814h 8    Zerofilled
81Ch 114h ECC (error correction codes)
```

Mode2/Form1 (CD-XA)

```
000h 0Ch  Sync
00Ch 4    Header (Minute,Second,Sector,Mode=02h)
010h 4    Sub-Header (File, Channel, Submode AND DFh, Codinginfo)
014h 4    Copy of Sub-Header
018h 800h Data (2048 bytes)
818h 4    EDC (checksum accross [010h..817h])
81Ch 114h ECC (error correction codes)
```

Mode2/Form2 (CD-XA)

```
000h 0Ch  Sync
00Ch 4    Header (Minute,Second,Sector,Mode=02h)
010h 4    Sub-Header (File, Channel, Submode OR 20h, Codinginfo)
014h 4    Copy of Sub-Header
018h 914h Data (2324 bytes)
92Ch 4    EDC (checksum accross [010h..92Bh]) (or 00000000h if no EDC)
```

encode_sector

```
sector[000h]=00h,FFh,FFh,FFh,FFh,FFh,FFh,FFh,FFh,FFh,FFh,00h
sector[00ch]=bcd(adr/75/60)      ;0..7x
sector[00dh]=bcd(adr/75 MOD 60)  ;0..59
sector[00eh]=bcd(adr MOD 75)     ;0..74
sector[00fh]=mode
if mode=00h then
  sector[010h..92Fh]=zerofilled
if mode=01h then
  adjust_edc(sector+0, 800h+10h)
  sector[814h..817h]=00h,00h,00h,00h,00h,00h,00h,00h
  calc_p_parity(sector)
  calc_q_parity(sector)
if mode=02h and form=1
  sector[012h]=sector[012h] AND (NOT 20h)  ;indicate not form2
  sector[014h..017h]=sector[010h..013h]    ;copy of sub-header
  adjust_edc(sector+10h,800h+8)
  push sector[00ch]           ;\temporarily clear header
  sector[00ch]=00000000h      ;/
  calc_p_parity(sector)
  calc_q_parity(sector)
  pop sector[00ch]            ;-restore header
if mode=02h and form=2
  sector[012h]=sector[012h] OR 20h         ;indicate form2
  sector[014h..017h]=sector[010h..013h]    ;copy of sub-header
  adjust_edc(sector+10h,914h+8)            ;edc is optional for form2
```

calc_parity(sector,offs,len,j0,step1,step2)

```
src=00ch, dst=81ch+offs, srcmax=dst
for i=0 to len-1
  base=src, x=0000h, y=0000h
  for j=j0 to 42
    x=x xor GF8_PRODUCT[j,sector[src+0]]
    y=y xor GF8_PRODUCT[j,sector[src+1]]
    src=src+step1, if (step1=2*44) and (src>=srcmax) then src=src-2*1118
  sector[dst+2*len+0]=x AND 0FFh, [dst+0]=x SHR 8
  sector[dst+2*len+1]=y AND 0FFh, [dst+1]=y SHR 8
  dst=dst+2, src=base+step2
```

calc_p_parity(sector) = calc_parity(sector,0,43,19,2*43,2)
calc_q_parity(sector) = calc_parity(sector,43*4,26,0,2*44,2*43)

adjust_edc(addr,len)

```
x=00000000h
for i=0 to len-1
  x=x xor byte[addr+i], x=(x shr 8) xor edc_table[x and FFh]
word[addr+len]=x  ;append EDC value (little endian)
```

init_tables

```
for i=0 to FFh
  x=i, for j=0 to 7, x=x shr 1, if carry then x=x xor D8018001h
  edc_table[i]=x
GF8_LOG[00h]=00h, GF8_ILOG[FFh]=00h, x=01h
for i=00h to FEh
  GF8_LOG[x]=i, GF8_ILOG[i]=x
  x=x SHL 1, if carry8bit then x=x xor 1dh
for j=0 to 42
  xx=GF8_ILOG[44-j],  yy=subfunc(xx xor 1,19h)
  xx=subfunc(xx,01h), xx=subfunc(xx xor 1,18h)
  xx=GF8_LOG[xx], yy = GF8_LOG[yy]
  GF8_PRODUCT[j,0]=0000h
  for i=01h to FFh
    x=xx+GF8_LOG[i], if x>=255 then x=x-255
```

```
    y=yy+GF8_LOG[i], if y>=255 then y=y-255
    GF8_PRODUCT[j,i]=GF8_ILOG[x]+(GF8_ILOG[y] shl 8)
```

subfunc(a,b)

```
  if a>0 then
    a=GF8_LOG[a]-b, if a<0 then a=a+255
    a=GF8_ILOG[a]
  return(a)
```

CDROM XA Subheader, File, Channel, Interleave

The Sub-Header for normal data sectors is usually 00h,00h,08h,00h (some PSX sectors have 09h instead 08h, indicating the end of "something" or so?

1st Subheader byte - File Number (FN)

```
  0-7 File Number    (00h..FFh) (for Audio/Video Interleave, see below)
```

2nd Subheader byte - Channel Number (CN)

```
  0-4 Channel Number (00h..1Fh) (for Audio/Video Interleave, see below)
  5-7 Should be always zero
```

3rd Subheader byte - Submode (SM)

```
  0   End of Record (EOR) (all Volume Descriptors, and all sectors with EOF)
  1   Video     ;\Sector Type (usually ONE of these bits should be set)
  2   Audio     ; Note: PSX .STR files are declared as Data (not as Video)
  3   Data      ;/
  4   Trigger           (for application use)
  5   Form2             (0=Form1/800h-byte data, 1=Form2, 914h-byte data)
  6   Real Time (RT)
  7   End of File (EOF) (or end of Directory/PathTable/VolumeTerminator)
```

The EOR bit is set in all Volume Descriptor sectors, the last sector (ie. the Volume Descriptor Terminator) additionally has the EOF bit set. Moreover, EOR and EOF are set in the last sector of each Path Table, and last sector of each Directory, and last sector of each File.

4th Subheader byte - Codinginfo (CI)

When used for Data sectors:

```
  0-7 Reserved (00h)
```

When used for XA-ADPCM audio sectors:

```
  0-1 Mono/Stereo     (0=Mono, 1=Stereo, 2-3=Reserved)
  2-2 Sample Rate     (0=37800Hz, 1=18900Hz, 2-3=Reserved)
  4-5 Bits per Sample (0=Normal/4bit, 1=8bit, 2-3=Reserved)
  6   Emphasis        (0=Normal/Off, 1=Emphasis)
  7   Reserved        (0)
```

Audio/Video Interleave (Multiple Files/Channels)

The CDROM drive mechanics are working best when continously following the data spiral on the disk, that works fine for uncompressed Audio Data at normal speed, but compressed Audio Data the disk is spinning much too fast. To avoid the drive to need to pause reading or to do permanent backwards seeking, CD-XA allows to store data interleaved in separate files/channels. With common interleave values like so:

```
  Interleave   Data Format
  1/1 (none)   44100Hz Stereo CD Audio at normal speed
  1/8          37800Hz Stereo ADPCM compressed Audio at double speed
  1/16         18900Hz Stereo ADPCM compressed Audio at double speed
  1/16         37800Hz Mono   ADPCM compressed Audio at double speed
  1/32         18900Hz Mono   ADPCM compressed Audio at double speed
  7/8          15fps 320x224 pixel MDEC compressed Videos at double speed
  Unknown if 1/16 and 1/32 interleaves are actually possible (the PSX cdrom
  controller seems to overwrite the IC303 sector buffer entries once every
  eight sectors, so ADPCM data may get destroyed on interleaves above 1/8).
  (Crash Team Racing uses 37800Hz Mono at Double speed, so 1/16 must work).
```

For example, 1/8 means that the controller processes only each 8th sector (each having the same File Number and Channel Number), and ignores the next 7 sectors (which must have other File Number and/or other Channel Number). There are various ways to arrange multiple files or channels, for example,

```
  one file with eight 1/8 audio channels
  one file with one 1/8 audio channels, plus one 7/8 video channel (*)
  one file with one 1/8 audio channels, plus 7 unused channels
  eight different files with one 1/8 audio channel each
  etc.
```

(*) If the Audio and Video data belongs together then both should use the SAME channel.
Note: Above interleave values are assuming that PSX Game Disks are always running at double speed (that's fastest for normal data files, and ADPCM files are usually using the same speed; otherwise it'd be neccessary to change the drive speed everytime when switching between Data to ADPCM modes).
Note: The file/channel numbers can be somehow selected with the Setfilter command. No idea if the controller is automatically switching to the next channel or so when reaching the end of the file?

Real Time Streaming

With the above Interleave, files can be played continously at real time - that, unless read-errors do occur. In that case the drive controller would usually perform time-consuming error-correction and/or read-retries. For video/audio streaming the resulting delay would be tendencially more annoying as than processing or skipping the incorrect data.
In such cases the drive controller is allowed to ignore read errors; that probably on sectors that have the Real Time (RT) flag set in their subheaders. The controller is probably doing some read-ahead buffering (so, if it has buffered enough data, then it may still perform read retries and/or error correction, as long as it doesn't affect real time playback).

CDROM XA Audio ADPCM Compression

CD-ROM XA ADPCM is used for Audio data compression. Each 16bit sample is encoded in 4bit nibbles; so the compression rate is almost 1:4 (only almost 1:4 because there are 16 header bytes within each 128-byte portion). The data is usually/always stored on 914h-byte sectors (without error correction).

Subheader

The Subheader (see previous chapter) contains important info for ADPCM: The file/channel numbers for Interleaved data, and the codinginfo flags: mono/stereo flag, 37800Hz/18900Hz sampling rate, 4bit/8bit format, and emphasis.

ADPCM Sectors

Each sector consists of 12h 128-byte portions (=900h bytes) (the remaining 14h bytes of the sectors 914h-byte data region are 00h filled).
The separate 128-byte portions consist of a 16-byte header,

```
  00h..03h  Copy of below 4 bytes (at 04h..07h)
  04h       Header for 1st Block/Mono, or 1st Block/Left
  05h       Header for 2nd Block/Mono, or 1st Block/Right
  06h       Header for 3rd Block/Mono, or 2nd Block/Left
  07h       Header for 4th Block/Mono, or 2nd Block/Right
  08h       Header for 5th Block/Mono, or 3rd Block/Left  ;\unknown/unused
  09h       Header for 6th Block/Mono, or 3rd Block/Right ; for 8bit ADPCM
  0Ah       Header for 7th Block/Mono, or 4th Block/Left  ; (maybe 0, or maybe
  0Bh       Header for 8th Block/Mono, or 4th Block/Right ;/copy of above)
  0Ch..0Fh  Copy of above 4 bytes (at 08h..0Bh)
```

followed by twentyeight data words (4x28-bytes),

```
  10h..13h  1st Data Word (packed 1st samples for 2-8 blocks)
  14h..17h  2nd Data Word (packed 2nd samples for 2-8 blocks)
  18h..1Bh  3rd Data Word (packed 3rd samples for 2-8 blocks)
  ...       Nth Data Word (packed Nth samples for 2-8 blocks)
  7Ch..7Fh  28th Data Word (packed 28th samples for 2-8 blocks)
```

and then followed by the next 128-byte portion.
The "Copy" bytes are allowing to repair faulty headers (ie. if the CDROM controller has sensed a read-error in the header then it can eventually replace it by the copy of the header).

XA-ADPCM Header Bytes

```
  0-3   Shift  (0..12) (0=Loudest) (13..15=Reserved/Same as 9)
  4-5   Filter (0..3) (only four filters, unlike SPU-ADPCM which has five)
  6-7   Unused (should be 0)
```

Note: The 4bit (or 8bit) samples are expanded to 16bit by left-shifting them by 12 (or 8), that 16bit value is then right-shifted by the selected 'shift' amount. For 8bit ADPCM shift should be 0..8 (values 9..12 will cut-off the LSB(s) of the 8bit value, this works, but isn't useful). For both 4bit and 8bit ADPCM, reserved shift values 13..15 will act same as shift=9).

XA-ADPCM Data Words (32bit, little endian)

```
  0-3   Nibble for 1st Block/Mono, or 1st Block/Left  (-8h..+7h)
  4-7   Nibble for 2nd Block/Mono, or 1st Block/Right (-8h..+7h)
  8-11  Nibble for 3rd Block/Mono, or 2nd Block/Left  (-8h..+7h)
  12-15 Nibble for 4th Block/Mono, or 2nd Block/Right (-8h..+7h)
  16-19 Nibble for 5th Block/Mono, or 3rd Block/Left  (-8h..+7h)
  20-23 Nibble for 6th Block/Mono, or 3rd Block/Right (-8h..+7h)
  24-27 Nibble for 7th Block/Mono, or 4th Block/Left  (-8h..+7h)
  28-31 Nibble for 8th Block/Mono, or 4th Block/Right (-8h..+7h)
```

or, for 8bit ADPCM format:

```
  0-7   Byte for 1st Block/Mono, or 1st Block/Left    (-80h..+7Fh)
  8-15  Byte for 2nd Block/Mono, or 1st Block/Right   (-80h..+7Fh)
  16-23 Byte for 3rd Block/Mono, or 2nd Block/Left    (-80h..+7Fh)
  24-31 Byte for 4th Block/Mono, or 2nd Block/Right   (-80h..+7Fh)
```

decode_sector(src)

```
  src=src+12+4+8   ;skip sync,header,subheader
  for i=0 to 11h
   for blk=0 to 3
    IF stereo ;left-samples (LO-nibbles), plus right-samples (HI-nibbles)
      decode_28_nibbles(src,blk,0,dst_left,old_left,older_left)
      decode_28_nibbles(src,blk,1,dst_right,old_right,older_right)
    ELSE      ;first 28 samples (LO-nibbles), plus next 28 samples (HI-nibbles)
      decode_28_nibbles(src,blk,0,dst_mono,old_mono,older_mono)
      decode_28_nibbles(src,blk,1,dst_mono,old_mono,older_mono)
    ENDIF
   next blk
   src=src+128
  next i
  src=src+14h+4    ;skip padding,edc
```

decode_28_nibbles(src,blk,nibble,dst,old,older)

```
  shift  = 12 - (src[4+blk*2+nibble] AND 0Fh)
  filter =      (src[4+blk*2+nibble] AND 30h) SHR 4
  f0 = pos_xa_adpcm_table[filter]
  f1 = neg_xa_adpcm_table[filter]
  for j=0 to 27
    t = signed4bit((src[16+blk+j*4] SHR (nibble*4)) AND 0Fh)
    s = (t SHL shift) + ((old*f0 + older*f1+32)/64);
    s = MinMax(s,-8000h,+7FFFh)
    halfword[dst]=s, dst=dst+2, older=old, old=s
  next j
```

Pos/neg Tables

```
  pos_xa_adpcm_table[0..4] = (0, +60, +115, +98, +122)
  neg_xa_adpcm_table[0..4] = (0,   0,  -52, -55,  -60)
```

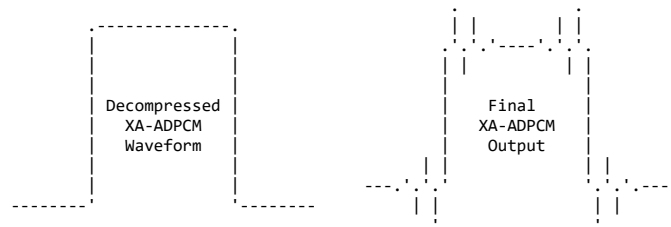
Note: XA-ADPCM supports only four filters (0..3), unlike SPU-ADPCM which supports five filters (0..4).

Old/Older Values

The incoming old/older values are usually that from the previous part, or garbage (in case of decoding errors in the previous part), or whatever (in case there was no previous part) (ie. maybe zero on power-up?) (and maybe there's also a way to reset the values to zero at the begin of a new file, or *maybe* it's silently done automatically when issuing seek commands?).

25-point Zigzag Interpolation

The CDROM decoder is applying some weird 25-point zigzag interpolation when resampling the 37800Hz XA-ADPCM output to 44100Hz. This part is different from SPU-ADPCM (which uses 4-point gaussian pitch interpolations). For example, XA-ADPCM interpolation applied to a square wave looks like this:



The zigzagging does produce some (inaudible) 22050Hz noise, and does produce some low-pass (?) filtering. The effect can be reproduced somewhat like so:

```
  Output37800Hz(sample):
```

```
    ringbuf[p AND 1Fh]=sample, p=p+1, sixstep=sixstep-1
    if sixstep=0
      sixstep=6
      Ouput44100Hz(ZigZagInterpolate(p,Table1))
      Ouput44100Hz(ZigZagInterpolate(p,Table2))
      Ouput44100Hz(ZigZagInterpolate(p,Table3))
      Ouput44100Hz(ZigZagInterpolate(p,Table4))
      Ouput44100Hz(ZigZagInterpolate(p,Table5))
      Ouput44100Hz(ZigZagInterpolate(p,Table6))
      Ouput44100Hz(ZigZagInterpolate(p,Table7))
    endif
  ZigZagInterpolate(p,TableX):
    sum=0
    for i=1 to 29, sum=sum+(ringbuf[(p-i) AND 1Fh]*TableX[i])/8000h, next i
    return MinMax(sum,-8000h,+7FFFh)
  Table1, Table2, Table3, Table4, Table5, Table6, Table7  ;Index
    0     , 0     , 0     , 0     , -0001h, +0002h, -0005h  ;1
    0     , 0     , 0     , -0001h, +0003h, -0008h, +0011h  ;2
    0     , 0     , -0001h, +0003h, -0008h, +0010h, -0023h  ;3
    0     , -0002h, +0003h, -0008h, +0011h, -0023h, +0046h  ;4
    0     , 0     , -0002h, +0006h, -0010h, +002Bh, -0017h  ;5
    -0002h, +0003h, -0005h, +0005h, +000Ah, +001Ah, -0044h  ;6
    +000Ah, -0013h, +001Fh, -001Bh, +006Bh, -00EBh, +015Bh  ;7
    -0022h, +003Ch, -004Ah, +00A6h, -016Dh, +027Bh, -0347h  ;8
    +0041h, -004Bh, +00B3h, -01A8h, +0350h, -0548h, +080Eh  ;9
    -0054h, +00A2h, -0192h, +0372h, -0623h, +0AFAh, -1249h  ;10
    +0034h, -00E3h, +02B1h, -05BFh, +0BCDh, -16FAh, +3C07h  ;11
    +0009h, +0132h, -039Eh, +09B8h, -1780h, +53E0h, +53E0h  ;12
    -010Ah, -0043h, +04F8h, -11B4h, +6794h, +3C07h, -16FAh  ;13
    +0400h, -0267h, -05A6h, +74BBh, +234Ch, -1249h, +0AFAh  ;14
    -0A78h, +0C9Dh, +7939h, +0C9Dh, -0A78h, +080Eh, -0548h  ;15
    +234Ch, +74BBh, -05A6h, -0267h, +0400h, -0347h, +027Bh  ;16
    +6794h, -11B4h, +04F8h, -0043h, -010Ah, +015Bh, -00EBh  ;17
    -1780h, +09B8h, -039Eh, +0132h, +0009h, -0044h, +001Ah  ;18
    +0BCDh, -05BFh, +02B1h, -00E3h, +0034h, -0017h, +002Bh  ;19
    -0623h, +0372h, -0192h, +00A2h, -0054h, +0046h, -0023h  ;20
    +0350h, -01A8h, +00B3h, -004Bh, +0041h, -0023h, +0010h  ;21
    -016Dh, +00A6h, -004Ah, +003Ch, -0022h, +0011h, -0008h  ;22
    +006Bh, -001Bh, +001Fh, -0013h, +000Ah, -0005h, +0002h  ;23
    +000Ah, +0005h, -0005h, +0003h, -0001h, 0     , 0       ;24
    -0010h, +0006h, -0002h, 0     , 0     , 0     , 0       ;25
    +0011h, -0008h, +0003h, -0002h, +0001h, 0     , 0       ;26
    -0008h, +0003h, -0001h, 0     , 0     , 0     , 0       ;27
    +0003h, -0001h, 0     , 0     , 0     , 0     , 0       ;28
    -0001h, 0     , 0     , 0     , 0     , 0     , 0       ;29
```

The above formula/table gives nearly correct results, but with small rounding errors in some cases - possibly due to actual rounding issues, or due to factors with bigger fractional portions, or due to a completely different formula...
Probably, the hardware does actually do the above stuff in two steps: first, applying a zig-zag filter (with only around 21-points) to the 37800Hz output, and then doing 44100Hz interpolation (2-point linear or 4-point gaussian or whatever) in a second step.
That two-step theory would also match well for 18900Hz resampling (which has lower-pitch zigzag, and gets spread accross about fifty 44100Hz samples).

XA-ADPCM Emphasis
With XA-Emphasis enabled in Sub-header, output will appear as so:



The exact XA-Emphasis formula is unknown (maybe it's just same as for CD-DA's SUBQ emphasis). Additionally, zig-zag interpolation is applied (somewhere before or after applying the emphasis stuff).
Note: The Emphasis feature isn't used by any known PSX games.

Uninitialized Six-step Counter
The hardware does contain some six-step counter (for interpolating 37800Hz to 44100Hz, ie. to insert one extra sample after each six samples). The 900h-byte sectors contain a multiple of six samples, so the counter will be always same before & after playing a sector. However, the initial counter value on power-up is uninitialized random (and the counter will fallback to that initial random setting after each 900h-byte sector).

RIFF Headers (on PCs)
When reading files that consist of 914h-byte sectors on a PC, the PC seems to automatically insert a 2Ch-byte RIFF fileheader. Like so, for ADPCM audio files:

```
  00h 4   "RIFF"
  04h 4   Total Filesize (minus 8)
  08h 8   "CDXAfmt "
  10h 4   Size of below stuff (10h)
  14h 14  Stuff (looks like the "LEN_SU" region from XA-Directory Record)
  22h 2   Zero  (probably just dummy padding for 32bit alignment)
  24h 4   "data"
  28h 4   Size of following data (usually N*930h)
```

That RIFF stuff isn't stored on the CDROM (at least not in the file area) (however, some of that info, like the "=UXA" stuff, is stored in the directory area of the CDROM).
After the RIFF header, the normal sector data is appended, that, with the full 930h bytes per sector (ie. the 914h data bytes preceeded by sync bytes, header, subheader, and followed by the EDC value).
The Channel Interleave doesn't seem to be resolved, ie. the Channels are kept arranged as how they are stored on the CDROM. However, File Interleave <should> be resolved, ie. other Files that "overlap" the file shouldn't be included in the file.

CDROM ISO Volume Descriptors

System Area (prior to Volume Descriptors)
The first 16 sectors on the first track are the system area, for a Playstation disk, it contains the following:

```
  Sector 0..3   - Zerofilled (Mode2/Form1, 4x800h bytes, plus ECC/EDC)
  Sector 4      - Licence String
  Sector 5..11  - Playstation Logo (3278h bytes) (remaining bytes FFh-filled)
  Sector 12..15 - Zerofilled (Mode2/Form2, 4x914h bytes, plus EDC)
```

Of which, the Licence String in sector 4 is,

```
  000h 32    Line 1      ("          Licensed  by          ")
  020h 32+6  Line 2 (EU) ("Sony Computer Entertainment Euro"," pe   ") ;\either
```

```
  020h 32+1  Line 2 (JP) ( Sony Computer Entertainment Inc. ",0Ah)      ; one of
  020h 32+6  Line 2 (US) ("Sony Computer Entertainment Amer","  ica ") ;/these
  041h 1983  Empty (JP)    (filled by repeating pattern 62x30h,1x0Ah, 1x30h)
  046h 1978  Empty (EU/US) (filled by 00h-bytes)
```

The Playstation Logo in sectors 5..11 contains data like so,

```
  0000h ..   41h,00h,00h,00h,00h,00h,00h,01h,00h,00h,00h,1Ch,23h,00h,00h
  0010h ..   51h,01h,00h,00h,A4h,2Dh,00h,00h,99h,00h,00h,00h,1Ch,00h,00h,00h
  0020h ..   ...
  3278h 588h FF-filled (remaining bytes on sector 11)
```

that region contains a header, polygons, vertices and normals for the "PS" logo (which is displayed when booting from CDROM). The BIOS compares these 3278h bytes against an identical copy in ROM, and refuses to boot if the data isn't 1:1 the same.

Volume Descriptors (Sector 16 and up)

Playstation disks usually have only two Volume Descriptors,

```
  Sector 16    - Primary Volume Descriptor
  Sector 17    - Volume Descriptor Set Terminator
```

Primary Volume Descriptor (sector 16 on PSX disks)

```
  000h 1    Volume Descriptor Type        (01h=Primary Volume Descriptor)
  001h 5    Standard Identifier           ("CD001")
  006h 1    Volume Descriptor Version     (01h=Standard)
  007h 1    Reserved                      (00h)
  008h 32   System Identifier             (a-characters) ("PLAYSTATION")
  028h 32   Volume Identifier             (d-characters) (max 8 chars for PSX?)
  048h 8    Reserved                      (00h)
  050h 8    Volume Space Size             (2x32bit, number of logical blocks)
  058h 32   Reserved                      (00h)
  078h 4    Volume Set Size               (2x16bit) (usually 0001h)
  07Ch 4    Volume Sequence Number        (2x16bit) (usually 0001h)
  080h 4    Logical Block Size in Bytes   (2x16bit) (usually 0800h) (1 sector)
  084h 8    Path Table Size in Bytes      (2x32bit) (max 800h for PSX)
  08Ch 4    Path Table 1 Block Number     (32bit little-endian)
  090h 4    Path Table 2 Block Number     (32bit little-endian) (or 0=None)
  094h 4    Path Table 3 Block Number     (32bit big-endian)
  098h 4    Path Table 4 Block Number     (32bit big-endian) (or 0=None)
  09Ch 34   Root Directory Record         (see next chapter)
  0BEh 128  Volume Set Identifier         (d-characters) (usually empty)
  13Eh 128  Publisher Identifier          (a-characters) (company name)
  1BEh 128  Data Preparer Identifier      (a-characters) (empty or other)
  23Eh 128  Application Identifier        (a-characters) ("PLAYSTATION")
  2BEh 37   Copyright Filename            ("FILENAME.EXT;VER") (empty or text)
  2E3h 37   Abstract Filename             ("FILENAME.EXT;VER") (empty)
  308h 37   Bibliographic Filename        ("FILENAME.EXT;VER") (empty)
  32Dh 17   Volume Creation Timestamp     ("YYYYMMDDHHMMSSFF",timezone)
  33Eh 17   Volume Modification Timestamp ("0000000000000000",00h)
  34Fh 17   Volume Expiration Timestamp   ("0000000000000000",00h)
  360h 17   Volume Effective Timestamp    ("0000000000000000",00h)
  371h 1    File Structure Version        (01h=Standard)
  372h 1    Reserved for future           (00h-filled)
  373h 141  Application Use Area          (00h-filled for PSX)
  400h 8    CD-XA Identifying Signature   ("CD-XA001" for PSX)
  408h 2    CD-XA Flags (unknown purpose) (00h-filled for PSX)
  40Ah 8    CD-XA Startup Directory       (00h-filled for PSX)
  412h 8    CD-XA Reserved                (00h-filled for PSX)
  41Ah 345  Application Use Area          (00h-filled for PSX)
  573h 653  Reserved for future           (00h-filled)
```

Volume Descriptor Set Terminator (sector 17 on PSX disks)

```
  000h 1    Volume Descriptor Type    (FFh=Terminator)
  001h 5    Standard Identifier       ("CD001")
  006h 1    Terminator Version        (01h=Standard)
  007h 2041 Reserved                  (00h-filled)
```

Boot Record (none such on PSX disks)

```
  000h 1    Volume Descriptor Type    (00h=Boot Record)
  001h 5    Standard Identifier       ("CD001")
  006h 1    Boot Record Version       (01h=Standard)
  007h 32   Boot System Identifier    (a-characters)
  027h 32   Boot Identifier           (a-characters)
  047h 1977 Boot System Use           (not specified content)
```

Supplementary Volume Descriptor (none such on PSX disks)

```
  000h 1    Volume Descriptor Type (02h=Supplementary Volume Descriptor)
  001h ..   Same as for Primary Volume Descriptor (see there)
  007h 1    Volume Flags           (8bit)
  008h ..   Same as for Primary Volume Descriptor (see there)
  058h 32   Escape Sequences       (32 bytes)
  078h ..   Same as for Primary Volume Descriptor (see there)
```

Volume Partition Descriptor (none such on PSX disks)

```
  000h 1    Volume Descriptor Type      (03h=Volume Partition Descriptor)
  001h 5    Standard Identifier         ("CD001")
  006h 1    Volume Partition Version    (01h=Standard)
  007h 1    Reserved                    (00h)
  008h 32   System Identifier           (a-characters) (32 bytes)
  028h 32   Volume Partition Identifier (d-characters) (32 bytes)
  048h 8    Volume Partition Location   (2x32bit) Logical Block Number
  050h 8    Volume Partition Size       (2x32bit) Number of Logical Blocks
  058h 1960 System Use                  (not specified content)
```

Reserved Volume Descriptors (none such on PSX disks)

```
  000h 1    Volume Descriptor Type    (04h..FEh=Reserved, don't use)
  001h 2047 Reserved                  (don't use)
```

CDROM ISO File and Directory Descriptors

The location of the Root Directory is described by a 34-byte Directory Record being located in Primary Volume Descriptor entries 09Ch..0BDh. The data therein is: Block Number (usually 22 on PSX disks), LEN_FI=01h, Name=00h, and, LEN_SU=00h (due to the 34-byte limit).

Format of a Directory Record

```
00h 1      Length of Directory Record (LEN_DR) (33+LEN_FI+pad+LEN_SU)
01h 1      Extended Attribute Record Length (usually 00h)
02h 8      Data Logical Block Number (2x32bit)
0Ah 8      Data Size in Bytes        (2x32bit)
12h 7      Recording Timestamp       (yy-1900,mm,dd,hh,mm,ss,timezone)
19h 1      File Flags 8 bits         (usually 00h=File, or 02h=Directory)
1Ah 1      File Unit Size            (usually 00h)
1Bh 1      Interleave Gap Size       (usually 00h)
1Ch 4      Volume Sequence Number    (2x16bit, usually 0001h)
20h 1      Length of Name            (LEN_FI)
21h LEN_FI File/Directory Name ("FILENAME.EXT;1" or "DIR_NAME" or 00h or 01h)
xxh 0..1   Padding Field (00h) (only if LEN_FI is even)
xxh LEN_SU System Use (LEN_SU bytes) (see below for CD-XA disks)
```

LEN_SU can be calculated as "LEN_DR-(33+LEN_FI+Padding)". For CD-XA disks (as used in the PSX), LEN_SU is 14 bytes:

```
00h 2      Owner ID Group  (whatever, usually 0000h, big endian)
02h 2      Owner ID User   (whatever, usually 0000h, big endian)
04h 2      File Attributes (big endian):
             0   Owner Read    (usually 1)
             1   Reserved      (0)
             2   Owner Execute (usually 1)
             3   Reserved      (0)
             4   Group Read    (usually 1)
             5   Reserved      (0)
             6   Group Execute (usually 1)
             7   Reserved      (0)
             8   World Read    (usually 1)
             9   Reserved      (0)
             10  World Execute (usually 1)
             11  IS_MODE2        (0=MODE1 or CD-DA, 1=MODE2)
             12  IS_MODE2_FORM2  (0=FORM1, 1=FORM2)
             13  IS_INTERLEAVED  (0=No, 1=Yes...?) (by file and/or channel?)
             14  IS_CDDA         (0=Data or ADPCM, 1=CD-DA Audio Track)
             15  IS_DIRECTORY    (0=File or CD-DA, 1=Directory Record)
           Commonly used Attributes are:
             0D55h=Normal Binary File (with 800h-byte sectors)
             2555h=Unknown            (wipeout .AV files) (MODE1 ??)
             4555h=CD-DA Audio Track  (wipeout .SWP files, alone .WAV file)
             3D55h=Streaming File     (ADPCM and/or MDEC or so)
             8D55h=Directory Record   (parent-, current-, or sub-directory)
06h 2      Signature     ("XA")
08h 1      File Number   (Must match Subheader's File Number)
09h 5      Reserved      (00h-filled)
```

The names are alphabetically sorted, no matter if the names refer to files or directories (ie. SUBDIR would be inserted between STRFILE.EXT and SYSFILE.EXT).
The first two entries (with non-ascii names 00h and 01h) are referring to current and parent directory.

Path Tables

The Path Table contain a summary of the directory names (the same information is also stored in the directory records, so programs may either use path tables or directory records; the path tables are allowing to read the whole directory tree quickly at once, without neeeding to seek from directory to directory).
Path Table 1 is in Little-Endian format, Path Table 3 contains the same data in Big-Endian format. Path Table 2 and 4 are optional copies of Table 1 and 3. The size and location of the tables is stored in Volume Descriptor entries 084h..09Bh. The format of the separate entries within a Path Table is,

```
00h 1       Length of Directory Name (LEN_DI) (01h..08h for PSX)
01h 1       Extended Attribute Record Length  (usually 00h)
02h 4       Directory Logical Block Number
06h 2       Parent Directory Number           (0001h and up)
08h LEN_DI  Directory Name (d-characters, d1-characters) (or 00h for Root)
xxh 0..1    Padding Field (00h) (only if LEN_FI is odd)
```

The first entry (directory number 0001h) is the root directory, the root doesn't have a name, nor a parent (the name field contains a 00h byte, rather than ASCII text, LEN_DI is 01h, and parent is 0001h, making the root it's own parent; ignoring the fact that incest is forbidden in many countries).
The next entries (directory number 0002h and up) (if any) are sub-directories within the root (sorted in alphabetical order, and all having parent=0001h). The next entries are sub-directories (if any) of the first sub-directory (also sorted in alphabetical order, and all having parent=0002h). And so on.
PSX disks usually contain all four tables (usually on sectors 18,19,20,21).

Format of an Extended Attribute Record (none such on PSX disks)

If present, an Extended Attribute Record shall be recorded over at least one Logical Block. It shall have the following contents.

```
00h 4       Owner Identification (numerical value)  ;\used only if
04h 4       Group Identification (numerical value)  ; File Flags Bit4=1
08h 2       Permission Flags (16bit, little-endian) ;/
0Ah 17      File Creation Timestamp      ("YYYYMMDDHHMMSSFF",timezone)
1Bh 17      File Modification Timestamp  ("0000000000000000",00h)
2Ch 17      File Expiration Timestamp    ("0000000000000000",00h)
3Dh 17      File Effective Timestamp     ("0000000000000000",00h)
4Eh 1       Record Format                (numerical value)
4Fh 1       Record Attributes            (numerical value)
50h 4       Record Length                (numerical value)
54h 32      System Identifier            (a-characters, a1-characters)
74h 64      System Use                   (not specified content)
B4h 1       Extended Attribute Record Version (numerical value)
B5h 1       Length of Escape Sequences   (LEN_ESC)
B6h 64      Reserved for future standardization (00h-filled)
F6h 4       Length of Application Use    (LEN_AU)
FAh LEN_AU  Application Use
xxh LEN_ESC Escape Sequences
```

Unknown WHERE that data is located... the Directory Records can specify the Extended Attribute Length, but not the location... maybe it's meant to be located in the first some bytes or blocks of the File or Directory...?

CDROM ISO Misc

Both Byte Order

All 16bit and 32bit numbers in the ISO region are stored twice, once in Little-Endian order, and then in Big-Endian Order. For example,

```
2x16bit value 1234h     ---> stored as 34h,12h,12h,34h
2x32bit value 12345678h ---> stored as 78h,56h,34h,12h,12h,34h,56h,78h
```

Exceptions are the 16bit Permission Flags which are stored only in Little-Endian format (although the flags are four 4bit groups, so that isn't a real 16bit number), and, the Path Tables are stored in both formats, but separately, ie. one table contains only Little-Endian numbers, and the other only Big-Endian numbers.

d-characters (Filenames)
```
"0..9", "A..Z", and "_"
```

a-characters
```
"0..9", "A..Z", SPACE, "!"%&'()*+,-./:;<=>?_"
```
Ie. all ASCII characters from 20h..5Fh except "#$@[\]^"

SEPARATOR 1 = 2Eh (aka ".") (extension; eg. "EXT")
SEPARATOR 2 = 3Bh (aka ";") (file version; "1".."32767")

Fixed Length Strings/Filenames
The Volume Descriptors contain a number fixed-length string/filename fields (unlike the Directory Records and Path Tables which have variable lengths). These fields should be padded with SPACE characters if they are empty, or if the string is shorter than the maximum length.
Filename fields in Volume Descriptors are referring to files in the Root Directory. On PSX disks, the filename fields are usually empty, but some disks are mis-using the Copyright Filename to store the Company Name (although no such file exists on the disk).

Volume Descriptor Timestamps
The various timestamps occupy 17 bytes each, in form of
```
"YYYYMMDDHHMMSSFF",timezone
"0000000000000000",00h         ;empty timestamp
```
The first 16 bytes are ASCII Date and Time digits (Year, Month, Day, Hour, Minute, Second, and 1/100 Seconds. The last byte is Offset from Greenwich Mean Time in number of 15-minute steps from -48 (West) to +52 (East); or actually: to +56 when recursing Kiribati's new timezone.
Note: PSX games manufactured in year 2000 were accidently marked to be created in year 0000.

Recording Timestamps
Occupy only 7 bytes, in non-ascii format
```
year-1900,month,day,hour,minute,second,timezone
00h,00h,00h,00h,00h,00h,00h    ;empty timestamp
```
The year ranges from 1900+0 to 1900+255.

File Flags
If this Directory Record identifies a directory then bit 2,3,7 shall be set to ZERO.
If no Extended Attribute Record is associated with the File Section identified by this Directory Record then bit positions 3 and 4 shall be set to ZERO.
```
0  Existence       (0=Normal, 1=Hidden)
1  Directory       (0=File, 1=Directory)
2  Associated File (0=Not an Associated File, 1=Associated File)
3  Record
      If set to ZERO, shall mean that the structure of the information in
      the file is not specified by the Record Format field of any associated
      Extended Attribute Record (see 9.5.8).
      If set to ONE, shall mean that the structure of the information in
      the file has a record format specified by a number other than zero in
      the Record Format Field of the Extended Attribute Record (see 9.5.8).
4  Restrictions    (0=None, 1=Restricted via Permission Flags)
5  Reserved        (0)
6  Reserved        (0)
7  Multi-Extent    (0=Final Directory Record for the file, 1=Not final)
```

Permission Flags (in Extended Attribute Records)
```
0-3   Permissions for upper-class owners
4-7   Permissions for normal owners
8-11  Permissions for upper-class users
12-15 Permissions for normal users
```
This is a bit bizarre, an upper-class owner is "an owner who is a member of a group of the System class of user". An upper-class user is "any user who is a member of the group specified by the Group Identification field". The separate 4bit permission codes are:
```
Bit0  Permission to read the file    (0=Yes, 1=No)
Bit1  Must be set (1)
Bit2  Permission to execute the file (0=Yes, 1=No)
Bit3  Must be set (1)
```

CDROM File Formats

FILENAME.EXT
The BIOS seems to support only (max) 8-letter filenames with 3-letter extension, typically all uppercase, eg. "FILENAME.EXT". Eventually, once when the executable has started, some programs might install drivers for long filenames(?)

SYSTEM.CNF
Contains boot info in ASCII/TXT format, similar to the CONFIG.SYS or AUTOEXEC.BAT files for MSDOS. A typical SYSTEM.CNF would look like so:
```
BOOT = cdrom:\abcd_123.45;1 arg ;boot exe (drive:\path\name.ext;version)
TCB = 4                         ;HEX (=4 decimal)   ;max number of threads
EVENT = 10                      ;HEX (=16 decimal)  ;max number of events
STACK = 801FFF00                ;HEX (=memtop-256)
```
The first line specifies the executable to load, from the "cdrom:" drive, "\" root directory, filename "abcd_123.45" (case-insensitive, the real name in the disk directory would be uppercase, ie. "ABCD_123.45"), and, finally ";1" is the file's version number (a rather strange ISO-filesystem specific feature) (the version number should be usually/always 1). Additionally, "arg" may contain an optional 128-byte command line argument string, which is copied to address 00000180h, where it may be interpreted by the executable (most or all games don't use that feature).
Each line in the file should be terminated by 0Dh,0Ah characters... not sure if it's also working with only 0Dh, or only 0Ah...?

A note on the "ABCD_123.45" file:
This is a normal executable (exactly as for the .EXE files, described below), however, the filename/extension is taken from the game code (the "ABCD-12345" text that is printed on the CD cover), but, with the minus replaced by an underscore, and due to the 8-letter filename limit, the last two characters are stored in the extension region.
That "XXXX_NNN.NN" naming convention seems to apply for all official licensed PSX games, not sure if it's possible to specify something like "FILENAME.EXE" as boot-file.

XXXX_NNN.NN (Boot-Executable) (filename specified in SYSTEM.CNF)
FILENAME.EXE (General-Purpose Executable)
PSX executables are having an 800h-byte header, followed by the code/data.
```
000h-007h ASCII ID "PS-X EXE"
008h-00Fh Zerofilled
010h      Initial PC                   (usually 80010000h, or higher)
```

```
  014h      Initial GP/R28              (usually 0)
  018h      Destination Address in RAM  (usually 80010000h, or higher)
  01Ch      Filesize (must be N*800h)   (excluding 800h-byte header)
  020h      Unknown/Unused              (usually 0)
  024h      Unknown/Unused              (usually 0)
  028h      Memfill Start Address       (usually 0) (when below Size=None)
  02Ch      Memfill Size in bytes       (usually 0) (0=None)
  030h      Initial SP/R29 & FP/R30 Base (usually 801FFFF0h) (or 0=None)
  034h      Initial SP/R29 & FP/R30 Offs (usually 0, added to above Base)
  038h-04Bh Reserved for A(43h) Function (should be zerofilled in exefile)
  04Ch-xxxh ASCII marker
             "Sony Computer Entertainment Inc. for Japan area"
             "Sony Computer Entertainment Inc. for Europe area"
             "Sony Computer Entertainment Inc. for North America area"
             (or often zerofilled in some homebrew files)
             (the BIOS doesn't verify this string, and boots fine without it)
  xxxh-7FFh Zerofilled
  800h...   Code/Data                 (loaded to entry[018h] and up)
```

The code/data is simply loaded to the specified destination address, ie. unlike as in MSDOS .EXE files, there is no relocation info in the header.
Note: In bootfiles, SP is usually 801FFFF0h (ie. not 801FFF00h as in system.cnf). When SP is 0, the unmodified caller's stack is used. In most cases (except when manually calling DoExecute), the stack values in the exeheader seem to be ignored though (eg. replaced by the SYSTEM.CNF value).
The memfill region is zerofilled by a "relative" fast word-by-word fill (so address and size must be multiples of 4) (despite of the word-by-word filling, still it's SLOW because the memfill executes in uncached slow ROM).
The reserved region at [038h-04Bh] is internally used by the BIOS to memorize the caller's RA,SP,R30,R28,R16 registers (for some bizarre reason, this information is saved in the exe header, rather than on the caller's stack).
Additionally to the initial PC,R28,SP,R30 values that are contained in the header, two parameter values are passed to the executable (in R4 and R5 registers) (however, usually that values are simply R4=1 and R5=0).
Like normal functions, the executable can return control to the caller by jumping to the incoming RA address (provided that it hasn't destroyed the stack or other important memory locations, and that it has pushed/popped all registers) (returning works only for non-boot executables; if the boot executable returns to the BIOS, then the BIOS will simply lockup itself by calling the "SystemErrorBootOrDiskFailure" function.

The PSX uses the standard CDROM ISO9660 filesystem without any encryption (ie. you can put an original PSX CDROM into a DOS/Windows computer, and view the content of the files in text or hex editors without problems).

PSX.EXE
Aside from SYSTEM.CNF, the Kernel seems to be also checking for a file named PSX.EXE, purpose is unknown, maybe this is the default executable name when SYSTEM.CNF is not found?

CDROM Protection - SCEx Strings

SCEx String
The heart of the PSX copy-protection is the four-letter "SCEx" string, encoded in the wobble signal of original PSX disks, which cannot reproduced by normal CD writers. The last letter varies depending on the region, "SCEI" for Japan, "SCEA" for America (and all other NTSC countries except Japan), "SCEE" for Europe (and all other PAL countries like Australia). If the string is missing (or if it doesn't match up for the local region) then the PSX refuses to boot. The verification is done by the Firmware inside of the CDROM Controller (not by the PSX BIOS, so there's no way to bypass it by patching the BIOS ROM chip).

Wobble Groove and Absolute Time in Pregroove (ATIP) on CD-R's
A "blank" CDR contains a pre-formatted spiral on it. The number of windings in the spiral varies depending on the number of minutes that can be recorded on the disk. The spiral isn't made of a straight line (------), but rather a wobbled line (/\/\/\), which is used to adjust the rotation speed during recording; at normal drive speed, wobble should produce a 22050Hz sine wave.
Additionally, the CDR wobble is modulated to provide ATIP information, ATIP is used for locating and positioning during recording, and contains information about the approximate laser power necessary for recording, the last possible time location that lead out can start, and the disc application code.
Wobble is commonly used only on (recordable) CDRs, ie. usually NOT on (readonly) CDROMs and Audio Disks. The copyprotected PSX CDROMs are having a short CDR-style wobble period in the first some seconds, which seems to contain the "SCEx" string instead of ATIP information.

Other Protections
Aside from the SCEx string, PSX disks are required to contain region and licence strings (in the ISO System Area, and in the .EXE file headers), and the "PS" logo (in the System Area, too). This data can be reproduced with normal CD writers, although it may be illegal to distribute unlicensed disks with licence strings.

CDROM Protection - Bypassing it

Modchips
A modchip is a small microcontroller which injects the "SCEx" signal to the mainboard, so the PSX can be booted even from CDRs which don't contain the "SCEx" string. Some modchips are additionally patching region checks contained in the BIOS ROM.
Note: Although regular PSX disks are black, the hardware doesn't verify the color of the disks, and works also with normal silver disks.

Disk-Swap-Trick
Once when the PSX has recognized a disk with the "SCEx" signal, it'll be satisfied until a new disk is inserted, which is sensed by the SHELL_OPEN switch. When having that switch blocked, it is possible to insert a CDR without the PSX noticing that the disk was changed.
Additionally, the trick requires some boot software that stops the drive motor (so the new disk can be inserted, despite of the PSX thinking that the drive door is still closed), and that does then start the boot executable on the new disk.
The boot software can be stored on a special boot-disk (that do have the "SCEx" string on it). Alternately, a regular PSX game disk could be used, with the boot software stored somewhere else (eg. on Expansion ROM, or BIOS ROM replacement, or Memory Card).

Booting via BIOS ROM or Expansion ROM
The PSX can be quite easily booted via Expansion ROM, or BIOS ROM replacements, allowing to execute code that is stored in the ROM, or that is received via whatever serial or parallel cable connection from a PC.
However, even with a BIOS replacement, the protection in the CDROM controller is still active, so the ROM can't read "clean" data from the CDROM Drive (unless the Disk-Swap trick is used).
Whereas, no "clean" data doens't mean no data at all. The CDROM controller does still seem to output "raw" data (without removing the sector header, and without handling error correction, and with only limited accuracy on the sector position). So, eventually, a customized BIOS could convert the "raw" data to "clean" data.

Secret Unlock Commands
There is an "official" backdoor that allows to disable the SCEx protection by software via secret commands (for example, sending those commands can be done via BIOS patches, nocash BIOS clone, or Expansion ROMs).
CDROM - Secret Unlock Commands

Booting via Memory Card
Some games that load data from memory cards may get confused if the save data isn't formatted as how they expect it - with some fine tuning you can get them to "crash" in a manner that they do accidently execute bootcode stored on the memory card.
Requires a tools to write to the memory card (eg. parallel port cable), and the memory card data customized for a specific game, and an original CDROM with that specific game. Once when the memory card code is booted, the Disk-Swap trick can be used.

CDROM Protection - Modchips

Modchip Source Code
The Old Crow mod chip source code works like so:

```
entrypoint:                  ;at power_up
  gate=input/highz
  data=input/highz
  wait 50 ms
  data=output/low
  wait 850 ms
  gate=output/low
  wait 314 ms
loop:
  wait 72 ms                 ;pause (eighteen "1=low" bits)
  sendbyte("S")              ;1st letter
  sendbyte("C")              ;2nd letter
  sendbyte("E")              ;3rd letter
  sendbyte(...)              ;4th letter (A, E, or I, depending on region)
  goto loop
sendbyte(char):
  sendbit(0)                 ;one start bit (0=highz)
  for i=0 to 7
    sendbit(char AND 1)      ;output data (LSB first)
    char=char/2
  next i
  sendbit(1)                 ;1st stop bit (1=low)
  sendbit(1)                 ;2nd stop bit (1=low)
  return
sendbit(bit):
  if bit=1 then data=output/low elseif bit=0 then data=input/highz
  wait 4 ms          ;4ms per bit = 250 bits per second
  return
```

Connection for the data/gate/sync signals:
For older PSX boards (data/gate):

```
Board        data             gate
PU-xx        unknown?         unknown?        ;older PSX boards
```

For newer PSX and PSone boards (data/sync):

```
Board        data             sync
PU-23, PM-41 CXD2938Q.Pin42   CXD2938Q.Pin5   ;newer PSX and older PSone
PM-41(2)     CXD2941R.Pin36   CXD2941R.Pin76  ;newer PSone boards
```

On the mainboard should be a big SMD capacitor (connected to the "data" pin), and a big testpoint (connected to the "sync" pin); it's easier to connect the signals to that locations than to the tiny CXD-chip pins.
gate and data must be tristate outputs, or open-collector outputs (or normal high/low outputs passed through a diode).

Note on "data" pin (all boards)
Transfers the "SCEx" data. Note that the signal produced by the modchip is looking entirly different than the signal produced by original disks, the real signal would be modulated 22050Hz wobble, while the modchip is simply dragging the signal permanently LOW throughout "1" bits, and leaves it floating for "0" bits. Anyways the "faked" signal seems to be accurate enough to work.

Note on "gate" pin (older PSX boards only)
The "gate" pin needs to be LOW only for use with original licensed disks (reportedly otherwise the SCEx string on that disks would conflict with the SCEx string from the modchip).
At the mainboard side, the "gate" signal is an input, and "data" is an inverted output of the gate signal (so dragging gate to low, would cause data to go high).

Note on "sync" pin (newer PSX and PSone boards only)
The "sync" pin is a testpoint on the mainboard, which does (at single speed) output a frequency of circa 44.1kHz/6 (of which some clock pulses seem to be longer or shorter, probably to indicate adjustments to the rotation speed).
Some modchips are connected directly to "sync" (so they are apparently synchronizing the data output with that signal; which is not implemented in the above source code).
Anyways, other modchips are using a more simplified connection: The modchip itself connects only to the "data" pin, and "sync" is required to be wired to IC723.Pin17.

Note on Multi-Region chips
Modchips that are designed to work in different regions are sending a different string (SCEA, SCEE, SCEI) in each loop cycle. Due to the slow 250bps transfer rate, it may take a while until the PSX has received the correct string, so this multi-region technique may cause a noticeable boot-delay.

Stealth (hidden modchip)
The Stealth connection is required for some newer games with anti-modchip protection, ie. games that refuse to run if they detect a modchip. The detection relies on the fact that the SCEx signal is normally received only when booting the disk, whilst older modchips were sending that signal permanently. Stealth modchips are sending the signal only on power-up (and when inserting a new disk, which can be sensed via SHELL_OPEN signal).
Modchip detection reportedly works like so (not too sure if all commands are required, some seem to be rather offtopic):

```
1.  Com 19h,20h   ;Retrieve CDROM Controller timestamp
2.  Com 01h       ;CdlNop: Get CD status
3.  Com 07h       ;CdlMotorOn: Make CD-ROM drive ready (blah?)
4.  Com 02h,1,1,1 ;CdlSetloc(01:01:01) (sector that does NOT have SCEx data)
5.  Com 0Eh,1     ;CdlSetmode: Turn on CD-DA read mode
6.  Short Delay
7.  Com 16h       ;CdlSeekP: Seek to Setloc's parameters (4426)
8.  Com 0Bh       ;CdlMute: Turn off sound so CdlPlay is inaudible
9.  Com 03h       ;CdlPlay: Start playing CD-DA.
10. Com 19h,04h   ;ResetSCExInfo (reset GetSCExInfo response to 0,0)
11. Long Delay    ;wait until the modchip (if any) has output SCEx data
12. Com 19h,05h   ;GetSCExInfo (returns total,success counters)
13. Com 09h       ;CdlPause: Stop command 19h.
```

If GetSCExInfo returns nonzero values, then the console is equipped with a modchip, and if so, anti-modchip games would refuse to work (no matter if the disk is an illegal copy, or not).

NTSC-Boot BIOS Patch
Typically connects to two or three BIOS address/data lines, apparently watching that signals, and dragging a data line LOW at certain time, to skip software based region checks (eg. allowing to play NTSC games on PAL consoles).
Aside from the modchip connection, that additionally requires to adjust the video signal (in 60Hz NTSC mode, the PSX defaults to generate a NTSC video signal) (whilst most PAL screens can handle 60Hz refresh, they can't handle NTSC colors) (on PSone boards, this can be fixed simply by grounding the /PAL pin; IC502.Pin13) (on older PSX boards it seems to be required to install an external color clock generator).

MODCHIP Connection Example
Connection for 8pin "12C508" mod chip from fatcat.co.nz for a PAL PSone with PM-41 board (ie. with 208pin SPU CXD2938Q, and 52pin IC304 "C 3060, SC430943PB"):
```
1 3.5V        (supply)
2 IC304.Pin44 (unknown?) (XLAT)
3 BIOS.Pin15  (D2)
4 BIOS.Pin31  (A18)
5 SPU.Pin5    ("sync")
6 SPU.Pin42   ("data")
7 IC304.Pin19 (SHELL_OPEN)
8 GND         (supply)
```
The chip can be used in a Basic connection (with only pin1,5,6,8 connected), or Stealth and NTSC-Boot connection (additionally pin2,3,4,7 connected). Some other modchips (such without internal oscillator) are additionally connected to a 4MHz or 4.3MHz signal on the mainboard. Some early modchips also connected to a bunch of additional pins that were reportedly for power-on timings (whilst newer chips use hardcoded power-on delays).

Nocash BIOS "Modchip" Feature
The nocash PSX bios outputs the "data" signal on the A20 address line, so (aside from the BIOS chip) one only needs to install a 1N4148 diode and two wires to unlock the CDROM:
```
SPU.Pin42 "data" -------|>|------ CPU.Pin149 (A20)
SPU.Pin5  "sync" ---------------- IC723.Pin17
```
With the "sync" connection, the SCEx signal from the disk is disabled (ie. even original licensed disks are no longer recognized, unless SCEx is output via A20 by software). The circuit works stable with original disks, but not with ALL unlicensed disks (apparently some signals are still passed despite of the "sync" connection, eg. my "Leila K. ca plane pour moi" audio disk is still treated as unlicensed, which is no problem since audio disks don't need to be licensed.)

CDROM Protection - LibCrypt

LibCrypt is an additional copy-protection, used by about 100 PSX games. The protection uses a 16bit decryption key, which is stored as bad position data in Subchannel Q. The 16bit key is then used for a simple XOR-decryption on certain 800h-byte sectors.

Protected sectors generation schemas
There are some variants on how the Subchannel Q data is modified:
```
1. 2 bits from both MSFs are modified,
   CRC-16 is recalculated and XORed with 0x0080.
   Games: MediEvil (E).
2. 2 bits from both MSFs are modified,
   original CRC-16 is XORed with 0x8001.
   Games: CTR: Crash Team Racing (E) (No EDC), CTR: Crash Team Racing (E)
   (EDC), Dino Crisis (E), Eagle One: Harrier Attack (E) et al.
3. Either 2 bits or none from both MSFs are modified,
   CRC-16 is recalculated and XORed with 0x0080.
   Games: Ape Escape (S) et al.
```
Anyways, the relevant part is that the modified sectors have wrong CRCs (which means that the PSX cdrom controller will ignore them, and the GetlocP command will keep returning position data from the previous sector).

LibCrypt sectors
The modified sectors could be theoretically located anywhere on the disc, however, all known protected games are having them located on the same sectors:
```
No.    <------- Minute=03/Normal ------->  <------- Minute=09/Backup ------->
Bit15  14105 (03:08:05)  14110 (03:08:10)  42045 (09:20:45)  42050 (09:20:50)
Bit14  14231 (03:09:56)  14236 (03:09:61)  42166 (09:22:16)  42171 (09:22:21)
Bit13  14485 (03:13:10)  14490 (03:13:15)  42432 (09:25:57)  42437 (09:25:62)
Bit12  14579 (03:14:29)  14584 (03:14:34)  42580 (09:27:55)  42585 (09:27:60)
Bit11  14649 (03:15:24)  14654 (03:15:29)  42671 (09:28:71)  42676 (09:29:01)
Bit10  14899 (03:18:49)  14904 (03:18:54)  42813 (09:30:63)  42818 (09:30:68)
Bit9   15056 (03:20:56)  15061 (03:20:61)  43012 (09:33:37)  43017 (09:33:42)
Bit8   15130 (03:21:55)  15135 (03:21:60)  43177 (09:35:52)  43182 (09:35:57)
Bit7   15242 (03:23:17)  15247 (03:23:22)  43289 (09:37:14)  43294 (09:37:19)
Bit6   15312 (03:24:12)  15317 (03:24:17)  43354 (09:38:04)  43359 (09:38:09)
Bit5   15378 (03:25:03)  15383 (03:25:08)  43408 (09:38:58)  43413 (09:38:63)
Bit4   15628 (03:28:28)  15633 (03:28:33)  43634 (09:41:59)  43639 (09:41:64)
Bit3   15919 (03:32:19)  15924 (03:32:24)  43963 (09:46:13)  43968 (09:46:18)
Bit2   16031 (03:33:56)  16036 (03:33:61)  44054 (09:47:29)  44059 (09:47:34)
Bit1   16101 (03:34:51)  16106 (03:34:56)  44159 (09:48:59)  44164 (09:48:64)
Bit0   16167 (03:35:42)  16172 (03:35:47)  44312 (09:50:62)  44317 (09:50:67)
```
Each bit is stored twice on Minute=03 (five sectors apart). For some reason, there is also a "backup copy" on Minute=09 (however, the libcrypt software doesn't actually support using that backup stuff, and, some discs don't have the backup at all (namely, discs with less than 10 minutes on track 1?)).
A modified sector means a "1" bit, an unmodified means a "0" bit. The 16bit keys of the existing games are always having eight "0" bits, and eight "1" bits (meaning that there are 16 modified sectors on Minute=03, and, if present, another 16 ones one Minute=09).

Example (Legacy of Kain)
Legacy of Kain (PAL) is reading the LibCrypt data during the title screen, and does then display GOT KEY!!! on TTY terminal (this, no matter if the correct 16bit key was received).
The actual protection jumps in a bit later (shortly after learning to glide, the game will hang when the first enemies appear if the key isn't okay). Thereafter, the 16bit key is kept used once and when to decrypt 800h-byte sector data via simple XORing.
The 16bit key (and some other related counters/variables) aren't stored in RAM, but rather in COP0 debug registers (which are mis-used as general-purpose storage in this case), for example, the 16bit key is stored in LSBs of the "cop0r3" register.

CDROM Disk Images CCD/IMG/SUB (CloneCD)

File.IMG - 2352 (930h) bytes per sector
Contains the sector data, recorded at 930h bytes per sector. Unknown if other sizes are also used/supported (like 800h bytes/sector, or even images with mixed sizes of 800h and 930h for different tracks).

File.SUB - 96 (60h) bytes per sector (subchannel P..W with 96 bits each)
Contains subchannel data, recorded at 60h bytes per sector.
```
00h..0Bh 12 Subchannel P (Pause-bits, usually all set, or all cleared)
0Ch..17h 12 Subchannel Q (ADR/Control, custom info, CRC-16-CCITT)
18h..5Fh .. Subchannel R..W (usually zero) (can be used for CD-TEXT)
```
Optionally, the .SUB file can be omitted (it's needed only for discs with non-standard subchannel data, such like copy-protected games).

File.CCD - Lead-in info in text format
Contains Lead-in info in ASCII text format. Lines should be terminated by 0Dh,0Ah. The overall CCD filestructure is:
```
[CloneCD]     ;File ID and version
[Disc]        ;Overall Disc info
[CDText]      ;CD-TEXT (included only if present)
[Session N]   ;Session(s) (numbered 1 and up)
[Entry N]     ;Lead-in entries (numbered 0..."TocEntries-1")
[TRACK N]     ;Track info (numbered 1 and up)
```
Read on below for details on the separate sections.

[CloneCD]
```
Version=3             ;-version (usually 3) (rarely 2)
```

[Disc]
```
TocEntries=4          ;-number of [Entry N] fields (lead-in info blocks)
Sessions=1            ;-number of sessions (usually 1)
DataTracksScrambled=0 ;-unknown purpose (usually 0)
CDTextLength=0        ;-total size of 18-byte CD-TEXT chunks (usually 0)
CATALOG=NNNNNNNNNNNNN ;-13-digit EAN-13 barcode (included only if present)
```

[CDText]
```
Entries=N       ;number of following entries (CDTextLength/18) (not /16)
Entry 0=80 00 NN NN NN NN NN NN NN NN NN NN NN NN   ;entry 0
Entry 1=80 NN NN NN NN NN NN NN NN NN NN NN NN NN   ;entry 1
...
Entry XX=8f NN NN NN NN NN NN NN NN NN NN NN NN NN  ;entry N-1
Note: Each entry contains 16 bytes (ie. "18-byte CD-TEXT" with CRC excluded)
"NN NN NN.." consists of 2-digit lowercase HEX numbers (without leading "0x")
```

[Session 1]
```
PreGapMode=2          ;-unknown purpose (usually 1 or 2)
PreGapSubC=1          ;-unknown purpose (usually 0 or 1)
```

[Entry 0]
[Entry 0..2] are usually containing Point A0h..A2h info. [Entry 3..N] are usually TOC info for Track 1 and up.
```
Session=1             ;-session number that this entry belongs to (usually 1)
Point=0xa0            ;-point (0..63h=Track, non-BCD!) (A0h..XXh=specials) Q2
ADR=0x01              ;-lower 4bit of ADR/Control (usually 1)           Q0.lo
Control=0x04          ;-upper 4bit of ADR/Control (eg. 0=audio, 4=data) Q0.hi
TrackNo=0             ;-usually/always 0 (as [Entry N]'s are in Lead-in)   Q1
AMin=0                ;\current MSF address                               Q3
ASec=0                ; (dummy zero values) (actual content               Q4
AFrame=0              ; would be current lead-in position)                Q5
ALBA=-150             ;/ALBA=((AMin*60+ASec)*75+AFrame)-PreGapSize
Zero=0                ;-probably reserved byte from Q channel             Q6
PMin=1                ;\referenced MSF address (non-BCD!), for certain    Q7
PSec=32               ; Point's, PMin may contain a Track number, and PSec Q8
PFrame=0              ; the disc type value (that without non-BCD-glitch)  Q9
PLBA=6750             ;/PLBA=((PMin*60+PSec)*75+PFrame)-PreGapSize
```

[TRACK 1] ;-track number (non-BCD) (1..99)
```
MODE=2                ;-mode (0=Audio, 1=Mode1, 2=Mode2)
ISRC=XXXXXNNNNNNN     ;-12-letter/digit ISRC code (included only if present)
INDEX 0=N             ;-1st sector with index 0, missing EVEN if any?
INDEX 1=N             ;-1st sector with index 1, usually same as track's PLBA
INDEX 2=N             ;-1st sector with index 2, if any
etc.
```

Missing Sectors & Sector Size
The .CCD file doesn't define the "PreGapSize" (the number of missing sectors at begin of first track). It seems to be simply constant " PreGapSize=150". Unless one is supposed to calculate it as "PreGapSize=((PMin*60+PSec)*75+PFrame)-PLBA".
The SectorSize seems to be also constant, "SectorSize=930h".

Non-BCD Caution
All Min/Sec/Frame/Track/Index values are expressed in non-BCD, ie. they must be converted to BCD to get the correct values (as how they are stored on real CDs). Exceptions are cases where those bytes have other meanings: For example, "PSec=32" does normally mean BcdSecond=32h, but for Point A0h it would mean DiscType=20h=CD-ROM-XA).
The Point value is also special, it is expressed in hex (0xNN), but nonetheless it is non-BCD, ie. Point 1..99 are specified as 0x01..0x63, whilst, Point A0h..FFh are specified as such (ie. as 0xA0..0xFF).

Versions
Version=1 doesn't seem to exist (or it is very rare). Version=2 is quite rare, and it seems to lack the [TRACK N] entries (meaning that there is no MODE and INDEX information, except that the INDEX 1 location can be assumed to be same as PLBA). Version=3 is most common, this version includes [TRACK N] entries, but often only with INDEX=1 (and up, if more indices), but without INDEX 0 (on Track 1 it's probably missing due to pregap, on further Tracks it's missing without reason) (so, only ways to reproduce INDEX=0 would be to guess it being located 2 seconds before INDEX=1, or, to use the information from the separate .SUB file, if that file is present; note: presence of index 0 is absolutely required for some games like PSX Tomb Raider 2).

Entry & Points & Sessions
The [Entry N] fields are usually containing Point A0h,A1h,A2h, followed by Point 1..N (for N tracks). For multiple sessions: The session is terminated by Point B0h,C0h. The next session does then contain Point A0h,A1h,A2h, and Point N+1..X (for further tracks). The INDEX values in the [TRACK N] entries are originated at the begin of the corresponding session, whilst PLBA values in [Entry N] entries are always originated at the begin of the disk.

CDROM Disk Images CDI (DiscJuggler)

Overall Format
```
Sector Data (sector 00:00:00 and up)          ;-body
```

```
Number of Sessions (1 byte)     <--- located at "Filesize-Footersize"
Session Block for 1st session (15 bytes)      ;\
nnn-byte info for 1st track                   ; 1st session
nnn-byte info for 2nd track (if any)          ;
etc.                                          ;/
Session Block for 2nd session (15 bytes)      ;\
nnn-byte info for 1st track                   ; 2nd session (if any)
nnn-byte info for 2nd track (if any)          ;
etc.                                          ;/
etc.                                          ;-further sessions (if any)
Session Block for no-more-sessions (15 bytes) ;-end marker
nnn-byte Disc Info Block                      ;-general disc info
Entrypoint (4 bytes)            <--- located at "Filesize-4"
```

Sector Data

Contains Sector Data for sector 00:00:00 and up (ie. all sectors are stored in the file, there are no missing "pregap" sectors).
Sector Size can be 800h..990h bytes/sector (sector size may vary per track).

Number of Sessions (1 byte)

```
00h   1   Number of Sessions (usually 1)
```

Session Block (15-bytes)

```
00h   1   Unknown (00h)
01h   1   Number of Tracks in session (01h..63h) (or 00h=No More Sessions)
02h   7   Unknown (00h-filled)
09h   1   Unknown (01h)
0Ah   3   Unknown (00h-filled)
0Dh   2   Unknown (FFh,FFh)
```

Track/Disc Header (30h+F bytes) (used in Track Blocks and Disc Info Block)

```
00h   12  Unknown (FFh,FFh,00h,00h,01h,00h,00h,00h,FFh,FFh,FFh,FFh)
0Ch   3   Unknown (DAh,0Ah,D5h or 64h,05h,2Ah) (random/id/chksum?)
0Fh   1   Total Number of Tracks on Disc (00h..63h) (non-BCD)
10h   1   Length of below Path/Filename (F)
11h   (F) Full Path/Filename (eg. "C:\folder\file.cdi")
11h+F 11  Unknown (00h-filled)
1Ch+F 1   Unknown (02h)
1Dh+F 10  Unknown (00h-filled)
27h+F 1   Unknown (80h)
28h+F 4   Unknown (00057E40h) (=360000 decimal) (disc capacity 80 minutes?)
2Ch+F 2   Unknown (00h,00h)
2Eh+F 2   Medium Type (0098h=CD-ROM, 0038h=DVD-ROM)
```

Track Block (E4h+F+I+T bytes)

```
00h     30h+F Track/Disc Header (see above)
30h+F   02h   Number of Indices (usually 0002h) (I=Num*4)
32h+F   (I)   32bit Lengths (per index) (eg. 00000096h,00007044h)
32h+FI  04h   Number of CD-Text blocks (usually 0) (T=Num*18+VariableLen's)
36h+FI  (T)   CD-Text (if any) (see "mirage_parser_cdi_parse_cdtext")
36h+FIT 02h   Unknown (00h,00h)
38h+FIT 01h   Track Mode (0=Audio, 1=Mode1, 2=Mode2/Mixed)
39h+FIT 07h   Unknown (00h,00h,00h,00h,00h,00h,00h)
40h+FIT 04h   Session Number (starting at 0) (usually 00h)
44h+FIT 04h   Track Number   (non-BCD, starting at 0) (00h..62h)
48h+FIT 04h   Track Start Address (eg. 00000000h)
4Ch+FIT 04h   Track Length        (eg. 000070DAh)
50h+FIT 0Ch   Unknown (00h-filled)
5Ch+FIT 04h   Unknown (00000000h or 00000001h)
60h+FIT 04h   read_mode (0..4)
                0: Mode1,        800h, 2048
                1: Mode2,        920h, 2336
                2: Audio,        930h, 2352
                3: Raw+PQ,       940h, 2352+16 non-interleaved (P=only 1bit)
                4: Raw+PQRSTUVW, 990h, 2352+96 interleaved
64h+FIT 4     Control (Upper 4bit of ADR/Control, eg. 00000004h=Data)
68h+FIT 1     Unknown (00h)
69h+FIT 4     Track Length        (eg. 000070DAh) (same as above)
6Dh+FIT 4     Unknown (00h,00h,00h,00h)
71h+FIT 12    ISRC Code 12-letter/digit (ASCII?) string (00h-filled if none)
7Dh+FIT 4     ISRC Valid Flag (0=None, Other?=Yes?)
81h+FIT 1     Unknown (00h)
82h+FIT 8     Unknown (FFh,FFh,FFh,FFh,FFh,FFh,FFh,FFh)
8Ah+FIT 4     Unknown (00000001h)
8Eh+FIT 4     Unknown (00000080h)
92h+FIT 4     Unknown (00000002h)     (guess: maybe audio num channels??)
96h+FIT 4     Unknown (00000010h)      (guess: maybe audio bits/sample??)
9Ah+FIT 4     Unknown (0000AC44h) (44100 decimal, ie. audio sample rate?)
9Eh+FIT 2Ah   Unknown (00h-filled)
C8h+FIT 4     Unknown (FFh,FFh,FFh,FFh)
CCh+FIT 12    Unknown (00h-filled)
D8h+FIT 1       session_type  ONLY if last track of a session (else 0)
                 (0=Audio/CD-DA, 1=Mode1/CD-ROM, 2=Mode2/CD-XA)
D9h+FIT 5     Unknown (00h-filled)
DEh+FIT 1     Not Last Track of Session Flag (0=Last Track, 1=Not Last)
DFh+FIT 1     Unknown (00h)
E0h+FIT 4        address for last track of a session? (otherwise 00,00,FF,FF)
```

Disc Info Block (5Fh+F+V+T bytes)

```
00h     30h+F Track/Disc Header (see above)
30h+F   4     Disc Size (total number of sectors)
34h+F   1     Volume ID Length (V) ;\from Primary Volume Descriptor[28h..47h]
35h+F   (V)   Volume ID String     ;/(ISO Data discs) (unknown for Audio)
35h+FV  1     Unknown (00h)
36h+FV  4     Unknown (01h,00h,00h,00h)
3Ah+FV  4     Unknown (01h,00h,00h,00h)
3Eh+FV  13    EAN-13 Code 13-digit (ASCII?) string (00h-filled if none)
4Bh+FV  4     EAN-13 Valid Flag (0=None, Other?=Yes?)
4Fh+FV  4     CD-Text Length in bytes (T=Num*1)
53h+FV  (T)   CD-Text (for Lead-in) (probably 18-byte units?)
53h+FVT 8     Unknown (00h-filled)
5Bh+FVT 4     Unknown (06h,00h,00h,80h)
```

Entrypoint (4 bytes) (located at "Filesize-4")
```
00h     4     Footer Size in bytes
```

CDROM Disk Images CUE/BIN/CDT (Cdrwin)

.CUE/.BIN (CDRWIN)
CDRWIN stores disk images in two separate files. The .BIN file contains the raw disk image, starting at sector 00:02:00, with 930h bytes per sector, but without any TOC or subchannel information. The .CUE file contains additional information about the separate track(s) on the disk, in ASCII format, for example:
```
FILE "PATH\FILENAME.BIN" BINARY
  TRACK 01 MODE2/2352
    INDEX 01 00:00:00          ;real address = 00:02:00  (+2 seconds)
  TRACK 02 AUDIO
    PREGAP 00:02:00            ;two missing seconds      (NOT stored in .BIN)
    INDEX 01 08:09:29          ;real address = 08:13:29  (+2 seconds +pregap)
  TRACK 03 AUDIO
    INDEX 00 14:00:29          ;real address = 14:04:29  (+2 seconds +pregap)
    INDEX 01 14:02:29          ;real address = 14:06:29  (+2 seconds +pregap)
  TRACK 04 AUDIO
    INDEX 00 18:30:20          ;real address = 18:34:20  (+2 seconds +pregap)
    INDEX 01 18:32:20          ;real address = 18:36:20  (+2 seconds +pregap)
```
The .BIN file does not contain ALL sectors, as said above, the first 2 seconds are not stored in the .BIN file. Moreover, there may be missing sectors somewhere in the middle of the file (indicated as PREGAP in the .CUE file; PREGAPs are usually found between Data and Audio Tracks).
The MM:SS:FF values in the .CUE file are logical addresses in the .BIN file, rather than physical addresses on real CDROMs. To convert the .CUE values back to real addresses, add 2 seconds to all MM:SS:FF addresses (to compensate the missing first 2 seconds), and, if the .CUE contains a PREGAP, then the pregap value must be additionally added to all following MM:SS:FF addresses.
The end address of the last track is not stored in the .CUE, instead, it can be only calculated by converting the .BIN filesize to MM:SS:FF format and adding 2 seconds (plus any PREGAP values) to it.

FILE <filename> BINARY|MOTOTOLA..or..MOTOROLA?|AIFF|WAVE|MP3
```
(must appear before any other commands, except CATALOG)
(uh, may also appear before further tracks)
```

FLAGS DCP 4CH PRE SCMS

INDEX NN MM:SS:FF

TRACK NN datatype
```
AUDIO          ;930h  ;bytes 000h..92Fh
CDG            ;?     ;?
MODE1/2048     ;800h  ;bytes 010h..80Fh
MODE1/2352     ;930h  ;bytes 000h..92Fh
MODE2/2336     ;920h  ;bytes 010h..92Fh
MODE2/2352     ;930h  ;bytes 000h..92Fh
CDI/2336       ;920h  ;?
CDI/2352       ;930h  ;bytes 000h..92Fh
```

PREGAP MM:SS:FF
POSTGAP MM:SS:FF
Duration of silence at the begin (PREGAP) or end (POSTGAP) of a track. Even if it isn't specified, the first track will always have a 2-second pregap.
The gaps are NOT stored in the BIN file.

REM comment
Allows to insert comments/remarks (which are usually ignored). Some third-party tools are mis-using REM to define additional information.

CATALOG 1234567890123
ISRC ABCDE1234567
```
(ISRC must be after TRACK, and before INDEX)
```

PERFORMER "The Band"
SONGWRITER "The Writer"
TITLE "The Title"
These entries allow to defined basic CD-Text info directly in the .CUE file.
Some third-party utilites allow to define additional CD-Text info via REM lines, eg. "REM GENRE Rock".
Alternately, more complex CD-Text data can be stored in a separate .CDT file.

CDTEXTFILE "C:\LONG FILENAME.CDT"
Specifies an optional file which may contain CD-TEXT. The .CDT file consists of raw 18-byte CD-TEXT fragments (which may include any type of information, including exotic one's like a "Message" from the producer), for whatever reason, there's a 00h-byte appended at the end of the file. Alternately to the .CDT file, the less exotic types of CD-TEXT can be defined by PERFORMER, TITLE, and SONGWRITER commands in the .CUE file.

Missing
Unknown if newer CUE/BIN versions do also support subchannel data.

CDROM Disk Images MDS/MDF (Alcohol 120%)

File.MDF - Contains sector data (optionally with sub-channel data)
Contains the sector data, recorded at 800h..930h bytes per sector, optionally followed by 60h bytes subchannel data (appended at the end of each sector). The stuff seems to be start on 00:02:00 (ie. the first 150 sectors are missing; at least it is like so when "Session Start Sector" is -150).
The subchannel data (if present) consists of 8 subchannels, stored in 96 bytes (each byte containing one bit per subchannel).
```
Bit7..0 = Subchannel P..W (in that order, eg. Bit6=Subchannel Q)
```
The 96 bits (per subchannel) can be translated to bytes, as so:
```
1st..8th bit  = Bit7..Bit0 of 1st byte (in that order, ie. MSB/Bit7 first)
9st..16th bit = Bit7..Bit0 of 2nd byte ("")
17th..        = etc.
```

File.MDS - Contains disc/lead-in info (in binary format)
An MDS file's structure consists of the following stuff ...

```
Header             (58h bytes)
Session block(s)   (usually one 18h byte entry)
Data blocks        (N*50h bytes)
Index blocks       (usually N*8 bytes)
Filename blocks(s) (usually one 10h byte entry)
Filename string(s) (usually one 6 byte string)
```

Header (58h bytes)

```
00h 16  File ID ("MEDIA DESCRIPTOR")
10h 2   Unknown (01h,03h or 01h,04h or 01h,05h) (Fileformat version?)
12h 2   Media Type (0=CD-ROM, 1=CD-R, 2=CD-RW, 10h=DVD-ROM, 12h=DCD-R)
14h 2   Number of sessions (usually 1)
16h 4   Unknown (02h,00h,00h,00h)
1Ah 2   Zero (for DVD: Length of BCA data)
1Ch 8   Zero
24h 4   Zero (for DVD: Offset to BCA data)
28h 18h Zero
40h 4   Zero (for DVD: Offset to Disc Structures)   (from begin of .MDS file)
44h 0Ch Zero
50h 4   Offset to First Session-Block (usually 58h) (from begin of .MDS file)
54h 4   Zero (for DVD?: Offset to DPM data blocks)  (from begin of .MDS file)
```

Session-Blocks (18h bytes)

```
00h 4   Session Start Sector (starting at FFFFFF6Ah=-150 in first session)
04h 4   Session End Sector     (XXX plus 150 ?)
08h 2   Session number (starting at 1) (non-BCD)
0Ah 1   Number of Data Blocks with any Point value (Total Data Blocks)
0Bh 1   Number of Data Blocks with Point>=A0h      (Special Lead-In info)
0Ch 2   First Track Number in Session (01h..63h, non-BCD!)
0Eh 2   Last Track Number in Session  (01h..63h, non-BCD!)
10h 4   Zero
14h 4   Offset to First Data-Block (usually 70h) (from begin of .MDS file)
```

Data-Blocks (50h bytes)

Block 0..2 are usually containing Point A0h..A2h info. Block 3..N are usually TOC info for Track 1 and up.

```
00h 1   Track mode (see below for details)
01h 1   Number of subchannels in .MDF file (0=None, 8=Sector has +60h bytes)
02h 1   ADR/Control (but with upper/lower 4bit swapped, ie. MSBs=ADR!)    Q0
03h 1   TrackNo (usually/always 00h; as this info is in Lead-in area)     Q1
04h 1   Point (Track 01h..63h, non-BCD!) (or A0h and up=Lead-in info)     Q2
05h 4   Zero (probably dummy MSF and reserved byte from Q channel)   Q3..Q6?
09h 1   Minute (Non-BCD!) (if track >= 0xA0 -> info about track ###)      Q7
                          (if track = 0xA2 -> min. @ lead-out)
0Ah 1   Second (Non-BCD!) (if track = 0xA2 -> sec. @ lead-out)            Q8
0Bh 1   Frame  (Non-BCD!) (if track = 0xA2 -> frame @ lead-out)           Q9
```

For Point>=A0h, below 44h bytes at [0Ch..4Fh] are zero-filled

```
0Ch 4   Offset to Index-block for this track    (from begin of .MDS file)
10h 2   Sector size (800h..930h) (or 860h..990h if with subchannels)
12h 1   Unknown (02h) (maybe number of indices?)
13h 11h Zero
24h 4   Track start sector, PLBA (00000000h=00:02:00)
28h 8   Track start offset                      (from begin of .MDF file)
30h 4   Number of Filenames for this track (usually 1)
34h 4   Offset to Filename Block for this track (from begin of .MDS file)
38h 18h Zero
```

Trackmode:

```
(upper 4bit seem to be meaningless?)
00h=None (used for entries with Point=A0h..FF)
A9h=AUDIO       ;sector size = 2352    930h  ;bytes 000h..92Fh
AAh=MODE1       ;sector size = 2048    800h  ;bytes 010h..80Fh
ABh=MODE2       ;sector size = 2336    920h  ;bytes 010h..92Fh
ACh=MODE2_FORM1 ;sector size = 2048    800h  ;bytes 018h..817h (incomplete!)
ADh=MODE2_FORM2 ;sector size = 2324+0? 914h  ;bytes 018h..91Bh (incomplete!)
ADh=MODE2_FORM2 ;sector size = 2324+4? 918h  ;bytes ??..?? (contains what?)
ECh=MODE2       ;sector size = 2448    990h  ;(930h+60h) (with subchannels)
```

Index Blocks (usually 8 bytes per track)

```
00h 4  Number of sectors with Index 0 (usually 96h or zero)
04h 4  Number of sectors with Index 1 (usually size of main-track area)
```

Index blocks are usually 8 bytes in size (two indices per track). Maybe this block is/was intended to allow to contain more indices (although the Alcohol 120% does always store only 2 indices, even when recording a CD with more than 2 indices per track).
The MDS file does usually contain Index blocks for <all> Data Blocks (ie. including unused dummy Index Blocks for Data Blocks with Point>=A0h).

Filename Blocks (10h bytes)

```
00h 4  Offset to Filename (from begin of .MDS file)
04h 1  Filename format (0=8bit, 1=16bit characters)
05h 11 Zero
```

Normally all tracks are sharing the same filename block (although theoretically the tracks could use separate filename blocks; with different filenames).

Filename Strings (usually 6 bytes)

```
00h 6  Filename, terminated by zero (usually "*.mdf",00h)
```

Contains the filename of the of the sector data (usually "*.mdf", indicating to use the same name as for the .mds file, but with .mdf extension).

Missing

Unknown if/how this format supports EAN-13, ISRC, CD-TEXT.
Unknown if Track/Point/Index are BCD or non-BCD.

CDROM Disk Images NRG (Nero)

.NRG (NERO)

Nero is probably the most bloated and most popular CD recording software. The first part of the file contains the disk image, starting at sector 00:00:00, with 800h..930h bytes per sector. Additional chunk-based information is appended at the end of the file, usually consisting of only four chunks: CUES,DAOI,END!,NERO (in that order).

Chunk Entrypoint (in last 8/12 bytes of file)

```
4   File ID "NERO"/"NER5"
```

```
  4/8 FileOffset of first chunk
```

Cue Sheet (summary of the Table of Contents, TOC)

```
  4   Chunk ID "CUES"/"CUEX"
  4   Chunk size (bytes)
```

below EIGHT bytes repeated for each track/index,
of which, first FOUR bytes are same for both CUES and CUEX,

```
  1   ADR/Control from TOC (usually LSBs=ADR=1=fixed, MSBs=Control=Variable)
  1   Track  (BCD) (00h=Lead-in, 01h..99h=Track N, AAh=Lead-out)
  1   Index  (BCD) (usually 00h=pregap, 01h=actual track)
  1   Zero
```

next FOUR bytes for CUES,

```
  1   Zero
  1   Minute (BCD) ;starting at 00:00:00 = 2 seconds before ISO vol. descr.
  1   Second (BCD)
  1   Sector (BCD)
```

or, next FOUR four bytes for CUEX,

```
  4   Logical Sector Number (HEX) ;starting at FFFFFF6Ah (=00:00:00)
```

Caution: Above may contain two position 00:00:00 entries: one nonsense entry for Track 00 (lead-in), followed by a reasonable entry for Track 01, Index 00.

Disc at Once Information

```
  4   Chunk ID "DAOI"/"DAOX"
  4   Chunk size (bytes)
  4   Garbage (usually same as above Chunk size)
  13  EAN-13 Catalog Number (13-digit ASCII) (or 00h-filled if none/unknown)
  1   Zero
  1   Disk type (00h=Mode1 or Audio, 20h=XA/Mode2) (and probaby 10h=CD-I?)
  1   Unknown (01h)
  1   First track (Non-BCD) (01h..63h)
  1   Last track  (Non-BCD) (01h..63h)
```

below repeated for each track,

```
  12  ISRC in ASCII (eg. "USXYZ9912345") (or 00h-filled if none/unknown)
  2   Sector size (usually 800h, 920h, or 930h) (see Mode entry for more info)
  1   Mode:
        0=Mode1/800h ;raw mode1 data (excluding sync+header+edc+errorinfo)
        3=Mode2/920h ;almost full sector (exluding first 16 bytes; sync+header)
        6=Mode2/930h ;full sector (including first 16 bytes; sync+header)
        7=Audio/930h ;full sector (plain audio data)
      Mode values from wikipedia:
        00h for data                                  Mode1/800h
        02h
        03h for Mode 2 Form 1 data   eh? FORM1???     Mode2/920h
        05h for raw data                              Mode1?/930h
        06h for raw Mode 2/form 1 data                Mode2/930h
        07h for audio                                 Audio/930h
        0Fh for raw data with sub-channel             Mode1?/930h+WHAT?
        10h for audio with sub-channel                Audio/930h+WHAT?
        11h for raw Mode 2/form 1 data with sub-channel  Mode2/WHAT?+WHAT?
       Note: Some newer files do actually use different sector sizes for each
       track (eg. 920h for the data track, and 930h for any following audio
       tracks), older files were using the same sector size for all tracks
       (eg. if the disk contained 930-byte Audio tracks, then Data tracks
       were stored at the same size, rather than at 800h or 920h bytes).
  3   Unknown (always 00h,00h,01h)
  4/8 Fileoffset 1 (Start of Track's Pregap) (with Index=00h)
  4/8 Fileoffset 2 (Start of actual Track) (with Index=01h and up)
  4/8 Fileoffset 3 (End of Track+1) (aka begin of next track's pregap)
```

End of chain

```
  4   Chunk ID "END!"
  4   Chunk size (always zero)
```

Track Information (contained only in Track at Once images)

```
  4     Chunk ID "TINF"/"ETNF"/"ETN2"
  4     Chunk size (bytes)
```

below repeated for each track,

```
  4/4/8 Track fileoffset        ;\32bit in TINF/ETNF chunks,
  4/4/8 Track length (bytes)    ;/64bit in ETN2 chunks
  4     Mode (should be same as in DAO chunks, see there) (implies sector size)
  0/4/4 Start lba on disc       ;\only in ETNF/ETN2 chunks,
  0/4/4 Unknown?                ;/not in TINF chunks
```

Unknown 1 (contained only in Track at Once images)

```
  4   Chunk ID "RELO"
  4   Chunk size (bytes)
  4   Zero
```

Unknown 2 (contained only in Track at Once images)

```
  4   Chunk ID "TOCT"
  4   Chunk size (bytes)
  1   Disk type (00h=Mode1 or Audio, 20h=XA/Mode2) (and probaby 10h=CD-I?)
  1   Zero (00h)
```

Session Info (begin of a session) (contained only in multi-session images)

```
  4   Chunk ID "SINF"
  4   Chunk size (bytes)
  4   Number of tracks in session
```

CD-Text (contained only in whatever images)

```
  4   Chunk ID None/"CDTX"
  4   Chunk size (bytes) (must be a multiple of 18 bytes)
```

below repeated for each fragment,

```
  18  Raw 18-byte CD-text data fragments
```

Media Type? (contained only in whatever images)

```
  4   Chunk ID "MTYP"
  4   Chunk size (bytes)
  4   Unknown? (00000001h for CDROM) (maybe other value for DVD)
```

Notes
Newer/older .NRG files may contain 32bit/64bit values (and use "OLD"/"NEW" chunk names) (as indicated by the "/" slashes).
CAUTION: All 16bit/32bit/64bit values are in big endian byte-order.

Missing
Unknown if newer NRG versions do also support subchannel data.

CDROM Disk Image/Containers CDZ

.CDZ is a compressed disk image container format (developed by pSX Author, and used only by the pSX emulator). The disk is split into 64kbyte blocks, which allows fast random access (without needing to decompress all preceeding sectors).
However, the compression ratio is surprisingly bad (despite of being specifically designed for cdrom compression, the format doesn't remove redundant sector headers, error correction information, and EDC checksums).

.CDZ File Structure
```
FileID ("CDZ",00h for cdztool v0/v1, or "CDZ",01h for cdztool v2 and up)
One or two Chunk(s)
```

.CDZ Chunk Format
Chunk Header in v0 (unreleased prototype):
```
4    32bit Decompressed Size (of all blocks) (must be other than "ZLIB")
```
Chunk Header in v1 (first released version):
```
4    ZLIB ID ("ZLIB")
8    64bit Decompressed Size (of all blocks)
```
Chunk Header in v2 and up (later versions):
```
4    Chunk ID (eg. "CUE",00h)
8    Chunk Size in bytes (starting at "ZLIB" up to including Footer, if any)
4    ZLIB ID ("ZLIB")
8    64bit Decompressed Size (of all blocks)
```
Chunk Body (same in all versions):
```
4    Number of Blocks (N)
4    Block 1 Compressed Size (CS.1)
4    Block 1 Decompressed Size (always 00010000h, except last block)
CS.1 Block 1 Compressed ZLIB Data (starting with 78h,9Ch)
...  ...                                ;\
4    Block N Compressed Size (CS.N)     ; further block(s)
4    Block N Decompressed Size          ; (if any)
CS.N Block N Compressed ZLIB Data       ;/
```
Chunk Footer in v0 (when above header didn't have the "ZLIB" ID):
```
4*N       Directory Entries for N blocks     ;-this ONLY for BIN chunk
```
Chunk Footer in v1 and up:
```
BPD*(N-1) Directory Entries for N-1 blocks   ;\this ONLY for BIN chunk
1         Bytes per Directory Entry (BPD)    ;/(not for CUE/CCD/MDS)
```
The "Compressed ZLIB Data" parts contain Deflate'd data (starting with 2-byte ZLIB header, and ending with 4-byte ZLIB/ADLER checksum), for details see:
Inflate - Core Functions
Inflate - Initialization & Tree Creation
Inflate - Headers and Checksums

.CDZ Chunks / Content
The chunk(s) have following content:
```
noname+noname       --> .CUE+.BIN (cdztool v1 and below)
"BIN",0             --> .ISO      (cdztool v2? and up)
"CUE",0+"BIN",0     --> .CUE+.BIN (cdztool v2 and up)
"CCD",0+"BIN",0     --> .CCD+.IMG (cdztool v2 and up)
"CCD",0+"BIN",01h   --> .CCD+.IMG+.SUB (930h sectors, plus 60h subchannels)
"MDS",0+"BIN",0     --> .MDS+.MDF (cdztool v5 only)
```
Note: cdztool doesn't actually recognize files with .ISO extension (however, one can rename them to .BIN, and then compress them as CUE-less .BIN file).

Cdztool.exe Versions
```
cdztool.exe v0, unrelased prototype
cdztool.exe v1, 22 May 2005, CRC32=620dbb08, 102400 bytes, pSX v1.0-5
cdztool.exe v2, 02 Jul 2006, CRC32=bcb29c1e, 110592 bytes, pSX v1.6
cdztool.exe v3, 22 Jul 2006, CRC32=4062ba82, 110592 bytes, pSX v1.7
cdztool.exe v4, 13 Aug 2006, CRC32=7388dd3d, 118784 bytes, pSX v1.8-11
cdztool.exe v5, 22 Jul 2007, CRC32=f25c1659, 155648 bytes, pSX v1.12-13
```
Note: v0 wasn't ever released (it's only noteworthy because later versions do have backwards compatibility for decompressing old v0 files). v1 didn't work with all operating systems (on Win98 it just says "Error: Couldn't create <output>" no matter what one is doing, however, v1 does work on later windows versions like WinXP or so?).

CDROM Disk Image/Containers ECM

ECM (Error Code Modeler by Neill Corlett) is a utility that removes unneccessary ECC error correction and EDC error detection values from CDROM-images. This is making the images a bit smaller, but the real size reduction isn't gained until subsequently compressing the images via tools like ZIP. Accordingly, these files are extremly uncomfortable to use: One most first UNZIP them, and then UNECM them.

.EXT.ECM - Double extension
ECM can be applied to various CDROM-image formats (like .BIN, .CDI, .IMG, .ISO, .MDF, .NRG), as indicated by the double-extension. Most commonly it's applied to .BIN files (hence using extension .BIN.ECM).

Example / File Structure
```
45 43 4D 00                                     ;FileID "ECM",00h
3C                                              ;Type 0, Len=10h (aka 0Fh+1)
00 FF FF FF FF FF FF FF FF FF 00 00 02 00 02    ;16 data bytes
02                                              ;Type 2, Len=1 (aka 00h+1)
00 00 08 00 00 00 00 00 00 00 ..... 00 00 00    ;804h data bytes
3C                                              ;Type 0, Len=10h (aka 0Fh+1)
00 FF FF FF FF FF FF FF FF FF 00 00 02 01 02    ;16 data bytes
02                                              ;Type 2, Len=1 (aka 00h+1)
00 00 08 00 00 00 00 00 00 00 ..... 00 00 00    ;804h data bytes
...
FC FF FF FF 3F                                  ;End Code (Len=FFFFFFFFh+1)
```

```
NN NN NN NN                                     ;EDC (on decompressed data)
```

Type/Length Byte(s)
Type/Length is encoded in 1..5 byte(s), with "More=1" indicating that further length byte(s) follow:
```
1st Byte: Bit7=More, Bit6-2=LengthBit4-0, Bit1-0=Type(0..3)
2nd Byte: Bit7=More, Bit6-0=LengthBit5-11
3rd Byte: Bit7=More, Bit6-0=LengthBit12-18
4th Byte: Bit7=More, Bit6-0=LengthBit19-25
5th Byte: Bit7-6=Reserved/Zero, Bit5-0=LengthBit26-31
```
Length=FFFFFFFFh=End Indicator
The actual decompression LEN is: "LEN=Length+1"

ECM Decompression
Below is repeated LEN times (with LEN being the Length value plus 1):
```
Type 0: load 1 byte, save 1 byte
Type 1: load 803h bytes [0Ch..0Eh,10h..80Fh], save 930h bytes [0..92Fh]
Type 2: load 804h bytes [14h..817h], save 920h bytes [10h..92Fh]
Type 3: load 918h bytes [14h..91Bh], save 920h bytes [10h..92Fh]
```
Type 1-3 are reconstructing the missing bytes before saving. Type 2-3 are saving only 920h bytes, so (if the original image contained full 930h byte sectors) the missing 10h bytes must be inserted via Type 0. Type 0 can be also used for copying whole sectors as-is (eg. Audio sectors, or Data sectors with invalid Sync/Header/ECC/EDC values). And, Type 0 can be used to store non-sector data (such like the chunks at the end of .NRG or .CDI files).

Central Mistakes
There's a lot of wrong with the ECM format. The two central problems are that it doesn't support data-compression (and needs external compression tools like zip/rar), and, that it doesn't contain a sector look-up table (meaning that random access isn't possible unless when scanning the whole file until reaching the desired sector).

Worst-case Scenario
As if ECM as such wouldn't be uncomfortable enough, you may expect typical ECM users to get more things messed up. For example:
```
A RAR file containing a 7Z file containing a ECM file containing a BIN file.
The BIN containing only Track 1, other tracks stored in APE files.
And, of course, the whole mess without including the required CUE file.
```

CDROM Subchannel Images

SBI (redump.org)
SBI Files start with a 4-byte FileID:
```
4 bytes FileID ("SBI",00h)
```
The followed by entries as so:
```
3 bytes real absolute MM:SS:FF address where the sub q data was bad
1 byte Format: the format can be 1, 2 or 3:
Format 1: complete 10 bytes sub q data           (Q0..Q9)
Format 2: 3 bytes wrong relative MM:SS:FF address (Q3..Q5)
Format 3: 3 bytes wrong absolute MM:SS:FF address (Q7..Q9)
```
Note: The PSX libcrypt protection relies on bad checksums (Q10..Q11), which will cause the PSX cdrom controller to ignore Q0..Q9 (and to keep returning position data from most recent sector with intact checksum).
Ironically, the SBI format cannot store the required Q10..Q11 checksum. The trick for using SBI files with libcrypted PSX discs is to ignore the useless Q0..Q9 data, and to assume that all sectors in the SBI file have wrong Q10..Q11 checksums.

M3S (Subchannel Q Data for Minute 3) (ePSXe)
M3S files are containing Subchannel Q data for all sectors on Minute=03 (the region where PSX libcrypt data is located) (there is no support for storing the (unused) libcrypt backup copy on Minute=09). The .M3S filesize is 72000 bytes (60 seconds * 75 sectors * 16 bytes). The 16 bytes per sector are:
```
Q0..Q9   Subchannel Q data (normally position data)
Q10..Q11 Subchannel Q checksum
Q12..Q15 Dummy/garbage/padding (usually 00000000h or FFFFFFFFh)
```
Unfortunately, there are at least 3 variants of the format:
```
1. With CRC (Q0..Q11 intact) (and Q12..Q15 randomly 00000000h or FFFFFFFFh)
2. Without CRC (only Q0..Q9 intact, but Q10..Q15 zerofilled)
3. Without anything (only Q0 intact, but Q1..Q15 zerofilled)
```
The third variant is definetly corrupt (and one should ignore such zerofilled entries). The second variant is corrupt, too (but one might attempt to repair them by guessing the missing checksum: if it contains normal position values assume correct crc, if it contains uncommon values assume a libcrypted sector with bad crc).
The M3S format is intended for libcrypted PSX games, but, people seem to have also recorded (corrupted) M3S files for unprotected PSX games (in so far, more than often, the M3S files might cause problems, instead of solving them).
Note: The odd 16-byte format with 4-byte padding does somehow resemble the "P and Q Sub-Channel" format 'defined' in MMC-drafts; if the .M3S format was based on the MMC stuff: then the 16th byte might contain a Subchannel P "pause" flag in bit7.

CDROM Images with Subchannel Data
Most CDROM-Image formats can (optionally) contain subchannel recordings. The downsides are: Storing all 8 subchannels for a full CDROM takes up about 20MBytes. And, some entries may contain 'wrong' data (read errors caused by scratches cannot be automatically repaired since subchannels do not contain error correction info).
If present, the subchannel data is usually appended at the end of each sector in the main binary file (one exception is CloneCD, which stores it in a separate .SUB file instead of in the .IMG file).
```
CCD/IMG/SUB (CloneCD)  P-W  60h-bytes Non-interleaved (in separate .SUB file)
CDI (DiscJuggler)      P-Q  10h-bytes Non-interleaved (in .CDI file)
""                     P-W  60h-bytes Interleaved (in .CDI file)
CUE/BIN/CDT (Cdrwin)        N/A
ISO (single-track)          N/A
MDS/MDF (Alcohol 120%) P-W  60h-bytes Interleaved (in .MDF file)
NRG (Nero)             P-W  60h-bytes Interleaved (in .NRG file)
```
Interleaved Subchannel format (eg. .MDF files):
```
00h-07h   80 C0 80 80 80 80 80 C0   ;P=FFh, Q=41h=ADR/Control, R..W=00h
08h-0Fh   80 80 80 80 80 80 80 C0   ;P=FFh, Q=01h=Track,       R..W=00h
10h-17h   80 80 80 80 80 80 80 C0   ;P=FFh, Q=01h=Index,       R..W=00h
18h-1Fh   80 80 80 80 80 80 80 80   ;P=FFh, Q=00h=RelMinute,   R..W=00h
20h-27h   80 80 80 80 80 80 80 80   ;P=FFh, Q=00h=RelSecond,   R..W=00h
28h-2Fh   80 80 80 80 80 80 80 80   ;P=FFh, Q=00h=RelSector,   R..W=00h
30h-37h   80 80 80 80 80 80 80 80   ;P=FFh, Q=00h=Reserved,    R..W=00h
38h-3Fh   80 80 80 80 80 80 80 80   ;P=FFh, Q=00h=AbsMinute,   R..W=00h
40h-47h   80 80 80 80 80 80 C0 80   ;P=FFh, Q=02h=AbsSecond,   R..W=00h
48h-4Fh   80 80 80 80 80 80 80 80   ;P=FFh, Q=00h=AbsSector,   R..W=00h
50h-57h   80 80 C0 80 C0 80 80 80   ;P=FFh, Q=28h=ChecksumMsb, R..W=00h
58h-5Fh   80 80 C0 C0 80 80 C0 80   ;P=FFh, Q=32h=ChecksumLsb, R..W=00h
```

Non-Interleaved Subchannel format (eg. CloneCD .SUB files):

```
  00h-0Bh   FF FF FF FF FF FF FF FF FF FF FF FF  ;Subchannel P (Pause)
  0Ch-17h   41 01 01 00 00 00 00 00 02 00 28 32  ;Subchannel Q (Position)
  18h-23h   00 00 00 00 00 00 00 00 00 00 00 00  ;Subchannel R
  24h-2Fh   00 00 00 00 00 00 00 00 00 00 00 00  ;Subchannel S
  30h-3Bh   00 00 00 00 00 00 00 00 00 00 00 00  ;Subchannel T
  3Ch-47h   00 00 00 00 00 00 00 00 00 00 00 00  ;Subchannel U
  48h-53h   00 00 00 00 00 00 00 00 00 00 00 00  ;Subchannel V
  54h-5Fh   00 00 00 00 00 00 00 00 00 00 00 00  ;Subchannel W
```

Non-Interleaved P-Q 10h-byte Subchannel format:

```
  This is probably based on MMC protocol, which would be as crude as so:
  The 96 pause bits are summarized in 1 bit. Pause/Checksum are optional.
  00h-09h   41 01 01 00 00 00 00 00 02 00        ;Subchannel Q (Position)
  0Ah-0Bh   28 32    ;<-- OPTIONAL, can be zero! ;Subchannel Q (Checksum)
  0Ch-0Eh   00 00 00                             ;Unused padding (zero)
  0F        80       ;<-- OPTIONAL, can be zero! ;Subchannel P (Bit7=Pause)
```

CDROM Disk Images Other Formats

.ISO - A raw ISO9660 image (can contain a single data track only)
Contains raw sectors without any sub-channel information (and thus it's restricted to the ISO filesystem region only, and cannot contain extras like additional audio tracks or additional sessions). The image should start at 00:02:00 (although I wouldn't be surprised if some <might> start at 00:00:00 or so). Obviously, all sectors must have the same size, either 800h or 930h bytes (if the image contains only Mode1 or Mode2/Form1 sectors then 800h bytes would usually enough; if it contains one or more Mode2/Form2 sectors then all sectors should be 930h bytes).
Handling .ISO files does thus require to detect the image's sector size, and to search the sector that contains the first ISO Volume Descriptor. In case of 800h byte sectors it may be additionally required to detect if it is a Mode1 or Mode2/Form1 image; for PSX images (and any CD-XA images) it'd be Mode2.

.CHD
Something used by MAME/MESS. Originally intended for compressed ROM-images, but does also support compressed CDROM-images. Fileformat and compression ratio are unknown. Also unknown if it allows random-access.
Some info can be found in MAME source code (looking at CHDMAN tools source code may be a good starting point... although the actual file structure may be hidden in other source files).

.C2D
Something. Can contain compressed or uncompressed CDROM-images. Fileformat and compression ratio are unknown. Also unknown if it allows random-access.
Some info (on uncompressed) .C2D files can found in libmirage source code.

.ISZ
This is reportedly simply a zipped .ISO file, apparently with the .ZIP extension renamed to .ISZ extension.

.MDX
Reportedly a compressed MDS/MDF file, supported by Daemon Tools. Fileformat and compression ratio are unknown. Also unknown if it allows random-access.

CD Image File Format (Xe - Multi System Emulator)
This is a rather crude file format, used only by the Xe Emulator. The files are meant to be generated by a utility called CDR (CD Image Ripper), which, in practice merely displays an "Unable to read TOC." error message.
The overall file structure is, according to "Xe User's Manual":

```
  header: 200h bytes header (see below)
  data:   990h bytes per sector (2352 Main, 96 Sub), 00:00:00->Lead Out
```

The header "definition" from the "Xe User's Manual" is as unclear as this:

```
  000h   00
  001h   00
  002h   First Track
  003h   Last Track
  004h   Track 1 (ADR << 4) | CTRL              ;\
  005h   Track 1 Start Minutes                  ; Track 1
  006h   Track 1 Start Seconds                  ;
  007h   Track 1 Start Frames                   ;/
  ...     ...                                   ;-Probably Further Tracks (?)
  n+0    Last Track Start Minutes               ;\
  n+1    Last Track Start Seconds               ; Last Track
  n+2    Last Track Start Frames                ;
  n+3    Last Track (ADR << 4) | CTRL           ;/
  n+4    Lead-Out Track Start Minutes           ;\
  n+5    Lead-Out Track Start Seconds           ; Lead-Out
  n+6    Lead-Out Track Start Frames            ;
  n+7    Lead-Out Track (ADR << 4) | CTRL       ;/
  ...    00
  1FFh   00
```

Unknown if MM:SS:FF values and/or First+Last Track numbers are BCD or non-BCD.
Unknown if Last track is separately defined even if there is only ONE track.
Unknown if Track 2 and up include ADR/Control (and if yes: where?).
Unknown if ADR/Control is really meant to be <before> MM:SS:FF on Track 1.
Unknown if ADR/Control is really meant to be <after> MM:SS:FF on Last+Lead-Out.
Unknown if this format does have a file extension (if yes: which?).
Unknown if subchannel data is meant to be interleaved or not.
The format supports only around max 62 tracks (in case each track is 4 bytes).
There is no support for "special" features like multi-sessions, cd-text.

CDROM Internal Info on PSX CDROM Controller

PSX software can access the CDROM via Port 1F801800h..1F801803h (as described in the previous chapters). The following chapters describe the inner workings of the PSX CDROM controller - this information is here for curiosity only - normally PSX software cannot gain control of those lower-level stuff (although some low level registers can be manipulated via Test commands, but that will usually conflict with normal operation).

Motorola MC68HC05 (8bit single-chip CPU)
The Playstation CDROM drive is controlled by a MC68HC05 8bit CPU with on-chip I/O ports and on-chip BIOS ROM. There is no way to reprogram that BIOS, nor to tweak it to execute custom code in RAM.
CDROM Internal HC05 Instruction Set

CDROM Internal HC05 On-Chip I/O Ports
CDROM Internal HC05 I/O Port Usage in PSX
CDROM Internal HC05 Motorola Selftest Mode
The PSX can read HC05 I/O Ports and RAM via Test Commands:
CDROM - Test Commands - Read HC05 SUB-CPU RAM and I/O Ports

Decoder/FIFO (CXD1199BQ or CXD1815Q)
This chip handles error correction and ADPCM decoding, and acts as some sort of FIFO interface between main/sub CPUs and incoming cdrom sector data. On the MIPS Main CPU it is controlled via Port 1F801800h..1F801803h.
CDROM Controller I/O Ports
On the HC05 Sub CPU it is controlled via Port A (data in/out), Port E (address/index), and Port D (read/write/select signals); the HC05 doesn't have external address/data bus, so one must manually access the CXD1815Q via those ports.
CDROM Internal CXD1815Q Sub-CPU Configuration Registers
CDROM Internal CXD1815Q Sub-CPU Sector Status Registers
CDROM Internal CXD1815Q Sub-CPU Address Registers
CDROM Internal CXD1815Q Sub-CPU Misc Registers
The PSX can read/write the Decoder I/O Ports and SRAM via Test commands:
CDROM - Test Commands - Read/Write Decoder RAM and I/O Ports
The sector buffer used in the PSX is 32Kx8 SRAM. Old PU-7 boards are using CXD1199BQ chips, later boards are using CXD1815Q, and even later boards have the stuff intergrated in the SPU. Note: The CXD1199BQ/CXD1815Q are about 99% same as described in CXD1199AQ datasheet.

Signal Processor and Servo Amplifier
Older PSX mainboards are using two separate chips:
CDROM Internal Commands CX(0x..3x) - CXA1782BR Servo Amplifier
CDROM Internal Commands CX(4x..Ex) - CXD2510Q Signal Processor
Later PSX mainboards have the above intergrated in a single chip, with some extended features:
CDROM Internal Commands CX(0x..Ex) - CXD2545Q Servo/Signal Combo
Later version is CXD1817R (Servo/Signal/Decoder Combo).
Even later PSX mainboards have it integrated in the Sound Chip: CXD2938Q (SPU+CDROM) with some changed bits and New SCEx transfer:
CDROM Internal Commands CX(0x..Ex) - CXD2938Q Servo/Signal/SPU Combo
Finally, PM-41(2) boards are using a CXD2941R chip (SPU+CDROM+SPU_RAM), unknown if/how far the CDROM part of that chip differs from CXD2938Q.
Some general notes:
CDROM Internal Commands CX(xx) - Notes
CDROM Internal Commands CX(xx) - Summary of Used CX(xx) Commands
The PSX can manipulate the CX(..) registers via some test commands:
CDROM - Test Commands - Test Drive Mechanics
Note: Datasheets for CXD2510Q/CXA1782BR/CXD2545Q do exist.

CDROM Pinouts
Pinouts - DRV Pinouts
Pinouts - HC05 Pinouts

CDROM Internal HC05 Instruction Set

ALU, Load/Store, Jump/Call
```
  Opcode      Clk HINZC Name Syntax
  x6 ...      2-5 --NZ- LDA  MOV  A,<op>      ;A=op
  xE ...      2-5 --NZ- LDX  MOV  X,<op>      ;X=op
  x7 ...      4-6 --NZ- STA  MOV  <op>,A      ;op=A
  xF ...      4-6 --NZ- STX  MOV  <op>,X      ;op=X
  xC ...      2-4 ----- JMP  JMP  <op>        ;PC=op
  xD ...      5-7 ----- JSR  CALL <op>        ;[SP]=PC, PC=op
  xB ...      2-5 H-NZC ADD  ADD  A,<op>      ;A=A+op
  x9 ...      2-5 H-NZC ADC  ADC  A,<op>      ;A=A+op+C
  x0 ...      2-5 --NZC SUB  SUB  A,<op>      ;A=A-op
  x2 ...      2-5 --NZC SBC  SBC  A,<op>      ;A=A-op-C
  x4 ...      2-5 --NZ- AND  AND  A,<op>      ;A=A AND op
  xA ...      2-5 --NZ- ORA  OR   A,<op>      ;A=A OR op
  x8 ...      2-5 --NZ- EOR  XOR  A,<op>      ;A=A XOR op
  x1 ...      2-5 --NZC CMP  CMP  A,<op>      ;A-op
  x3 ...      2-5 --NZC CPX  CMP  X,<op>      ;X-op
  x5 ...      2-5 --NZ- BIT  TEST A,<op>      ;A AND op
  A7,AF,AC = Reserved (no STA/STX/JMP with immediate operand)
```
Operands can be...
```
  Opcode      Clk ALU/LDA/LDX      Clk STA/STX          Clk JMP/CALL
  Ax nn         2 cmd r,nn           - N/A              -/6 call relative (BSR)
  Bx nn         3 cmd r,[nn]         4 mov [nn],r       2/5 cmd nn
  Cx nn mm      4 cmd r,[nnmm]       5 mov [nnmm],r     3/6 cmd nnmm
  Dx nn mm      5 cmd r,[X+nnmm]     6 mov [X+nnmm],r   4/7 cmd X+nnmm
  Ex nn         4 cmd r,[X+nn]       5 mov [X+nn],r     3/6 cmd X+nn
  Fx            3 cmd r,[X]          4 mov [X],r        2/5 cmd X
```

Read-Modify-Write
```
  Opcode      Clk HINZC Name Syntax
  xC ...      3-6 --NZ- INC  INC op      ;increment   ;op=op+1
  xA ...      3-6 --NZ- DEC  DEC op      ;decrement   ;op=op-1
  xF ...      3-6 --01- CLR  ??  op,00h  ;clear       ;op=op AND 00h
  x3 ...      3-6 --NZ1 COM  NOT op      ;complement  ;op=op XOR FFh
  x0 ...      3-6 --NZC NEG  NEG op      ;negate      ;op=00h-op
  x9 ...      3-6 --NZC ROL  RCL op      ;rotate left through carry
  x6 ...      3-6 --NZC ROR  RCR op      ;rotate right through carry
  x8 ...      3-6 --NZC LSL  SHL op      ;shift left logical
  x4 ...      3-6 --0ZC LSR  SHR op      ;shift right logical
  x7 ...      3-6 --NZC ASR  SAR op      ;shift right arithmetic
  xD ...      3-5 --NZ- TST  TEST op,FFh ;test for negative or zero (AND FFh?)
  x1,x2,x5,xB,xE = Reserved (except for: 42 = MUL)
```
Operands can be...
```
  Opcode      Clk RMW          Clk CLR               Clk TST
  3x nn         5 cmd [nn]       5 MOV [nn],00h        4 TEST [nn],0FFh
  4x            3 cmd A          3 MOV A,00h,slow      3 TEST A,0FFh,slow
  5x            3 cmd X          3 MOV X,00h,slow      3 TEST X,0FFh
  6x nn         6 cmd [X+nn]     6 MOV [X+nn],00h      5 TEST [X+nn],0FFh
```

```
7x            5 cmd [x]       5 mov [x],00h         4 test [x],0ffh
```

CLR includes a dummy-read-cycle, whilst TST does omit the dummy-write cycle.
The ",slow" RMW opcodes are smaller, but slower than equivalent ALU opcodes.

Bit Manipulation and Bit Test with Relative Jump (to $+3+/-dd)

```
Opcode      Clk HINZC Name  Syntax
00h+i*2 nn dd 5 ----C BRSET JNZ [nn].i,dest  ;C=[nn].i, branch if set
01h+i*2 nn dd 5 ----C BRCLR JZ  [nn].i,dest  ;C=[nn].i, branch if clear
10h+i*2 nn    5 ----- BSET  SET [nn].i       ;set [nn].i
11h+i*2 nn    5 ----- BCLR  RES [nn].i       ;clear [nn].i
```

Branch (Relative jump to $+2+/-nn)

```
Opcode      Clk HINZC Name    Syntax
20 nn         3 ----- BRA     JR  nn      ;branch always
21 nn         3 ----- BRN     NUL nn      ;branch never
22 nn         3 ----- BHI     JA  nn      ;if C=0 and Z=0, higher        ?
23 nn         3 ----- BLS     JBE nn      ;if C=1 or Z=1, lower or same  ?
24 nn         3 ----- BCC/BHS JNC/JAE nn  ;if C=0, carry clear, higher.same
25 nn         3 ----- BCS/BLO JC/JB nn    ;if C=1, carry set, lower
26 nn         3 ----- BNE     JNZ/JNE nn  ;if Z=0, not equal / not zero
27 nn         3 ----- BEQ     JZ/JE nn    ;if Z=1, equal / zero
28 nn         3 ----- BHCC    JNH nn      ;if H=0, half-carry clear
29 nn         3 ----- BHCS    JH  nn      ;if H=1, half-carry set
2A nn         3 ----- BPL     JNS nn      ;if S=0, plus / not signed
2B nn         3 ----- BMI     JS  nn      ;if S=1, minus / signed
2C nn         3 ----- BMC     JEI nn      ;if I=0, interrupt mask clear
2D nn         3 ----- BMS     JDI nn      ;if I=1, interrupt mask set
2E nn         3 ----- BIL     JIL nn      ;if XX=LO, interrupt line low
2F nn         3 ----- BIH     JIH nn      ;if XX=HI, interrupt line high
AD nn         6 ----- BSR     CALL relative nn  ;branch to subroutine always
```

Control/Misc

```
Opcode      Clk HINZC Name Syntax
9D            2 ----- NOP  NOP           ;no operation
97            2 ----- TAX  MOV X,A       ;transfer A to X
9F            2 ----- TXA  MOV A,X       ;transfer X to A
9C            2 ----- RSP  MOV SP,00FFh  ;reset stack pointer (SP=00FFh)
42           11 0---0 MUL  MUL X,A       ;X:A=X*A (unsigned multiply)
81            6 ----- RTS  RET           ;return from subroutine
80            9 xxxxx RTI  RETI          ;return from interrupt
99            2 ----1 SEC  STC           ;set carry flag
98            2 ----0 CLC  CLC           ;clear carry flag
9B            2 -1--- SEI  DI            ;set interrupt mask (disable ints)
9A            2 -0--- CLI  EI            ;clear interrupt mask (enable ints)
8E           ..2 -0--- STOP STOP         ;?
8F           ..2 -0--- WAIT WAIT         ;?
83            10 -1--- SWI  SWI          ;software interrupt ...? PC=[FFFCh]
<IRQ>          ? ????? Interrupt         ;?                       PC=[FFFxh]
<RESET>        ? ????? Reset             ;?                       PC=[FFFEh]
82,84..8D,90..96,9E = Reserved
```

MUL isn't supported in original "M146805 CMOS" family (MUL is used/supported in PSX cdrom controller).

Registers

```
A   8bit  accumulator
X   8bit  index register
SP  6bit  stack pointer (range 00C0h..00FFh)
PC  16bit program pointer (range 0000h..FFFFh)
CCR 5bit  condition code register (flags) (111HINZC)
```

Pushed on IRQ are:

```
SP.highest PC.lo
           PC.hi
           X
           A
SP.lowest  Flags (CCR, 5bit condition code register) (111HINZC)
```

Addressing Modes

```
nn       immediate             ;00h..FFh
[nn]     direct address        ;[0000h..00FFh]
[nnmm]   extended address      ;[0000h..FFFFh]
[X]      indexed, no offset    ;[0000h..00FFh]
[X+nn]   indexed, 8bit offset  ;[0000h..01FEh]
[X+nnmm] indexed, 16bit offset ;[0000h..FFFFh]
[nn].i   bit                   ;[0000h..00FFh].bit0..7
dd       relative              ;$+2..3+(-80h..+7Fh)
```

Notes:

```
operand "X+nn" performs an unsigned addition, and can address 0000h..01FEh.
16bit operands (nnmm) are encoded in BIG-ENDIAN format (same for pushed PC).
```

Exception Vectors

Exception vectors are 16bit BIG-ENDIAN values at FFF0h-FFFFh (or at FFE0h-FFEFh when running in Motorola Bootstrap mode).

```
Vector Prio Usage
FFF0h  7=lo TBI Vector (Timebase)
FFF2h  6    SSPI Vector (SPI bus)     (SPI1 and SPI2)
FFF4h  5    Timer 2 Interrupt Vector  (Timer 2 Input/Compare)
FFF6h  4    Timer 1 Interrupt Vector  (Timer 1 Input/Compare/Overflow)
FFF8h  3    KWI Vector (Key Wakeup)   (KWI0..7 pins)
FFFAh  2    External Interrupt Vector (/IRQ1 and /IRQ2 pins)
FFFCh  none Software Interrupt Vector (SWI opcode)           ;\regardless of
FFFEh  1=hi Reset Vector              (/RESET signal and COP) ;/CPU's "I"
```

Directives/Pseudos (used by a22i assembler; in no$psx utility menu)

```
.hc05         select HC05 instruction set (default would be .mips)
.nocash       select nocash syntax (default would be .native opcode names)
db ...        define 8bit byte(s), or quoted ascii strings
dw ...        define 16bit word(s) in BIG ENDIAN (for HC05 exception vectors)
org nnnn      change origin for following opcodes
end           end of file
mov c,[nn].i  alias for "jnz [nn].i,$+3" (dummy jump & set carry=[nn].i)
```

CDROM Internal HC05 On-Chip I/O Ports

HC05 Port 3Eh - MISC - Miscellaneous Register (R/W)
```
0    OPTM  Option Map Select (bank-switching for Port 00h..0Fh)
1    FOSCE Fast (Main) Oscillator Enable (0=Disable OSC, 1=Normal)
2-3  SYS   System Clock Select (0=OSC/2, 1=OSC/4, 2=OSC/64, 3=XOSC/2)
4-5  -     Not used (0)
6    STUP  XOSC Time Up Flag   (R)
7    FTUP  OSC Time Up Flag    (R)   (0=Busy, 1=Ready/Good/Stable)
```
Note: For PSX, OSC is 4.0000MHz (PU-7/PU-8), 4.2336MHz (PU-18 and up). SysClk is usually set to OSC/2, ie. around 2MHz.

HC05 Port OPTM=0:00h - PORTA - Port A Data Register (R/W)
HC05 Port OPTM=0:01h - PORTB - Port B Data Register (R)
HC05 Port OPTM=0:02h - PORTC - Port C Data Register (R/W)
HC05 Port OPTM=0:03h - PORTD - Port D Data Register (R/W)
HC05 Port OPTM=0:04h - PORTE - Port E Data Register (R/W)
HC05 Port OPTM=0:05h - PORTF - Port F Data Register (R) (undoc: R/W)
These are general purpose I/O ports (controlling external pins). Some ports are Input-only, and some can be optionally used for special things (like IRQs, SPI-bus, or as Timer input/output).
```
PA.0-7  PAn   Port A Bit0..7 Input/Output           (0=Low, 1=High) (R/W)
PB.0-7  PBn   Port B Bit0..7 Input        /KWI0..7  (0=Low, 1=High) (R)
PC.0    PC0   Port C Bit0    Input/Output /SDI1 (SPI)(0=Low, 1=High) (R/W)
PC.1    PC1   Port C Bit1    Input/Output /SDO1 (SPI)(0=Low, 1=High) (R/W)
PC.2    PC2   Port C Bit2    Input/Output /SCK1 (SPI)(0=Low, 1=High) (R/W)
PC.3    PC3   Port C Bit3    Input/Output /TCAP (T1) (0=Low, 1=High) (R/W)
PC.4    PC4   Port C Bit4    Input/Output /EVI  (T2) (0=Low, 1=High) (R/W)
PC.5    PC5   Port C Bit5    Input/Output /EVO  (T2) (0=Low, 1=High) (R/W)
PC.6    PC6   Port C Bit6    Input/Output /IRQ2      (0=Low, 1=High) (R/W)
PC.7    PC7   Port C Bit7    Input/Output /IRQ1      (0=Low, 1=High) (R/W)
PD.0-7  PDn   Port D Bit0..7 Input/Output            (0=Low, 1=High) (R/W)
PE.0-7  PEn   Port E Bit0..7 Input/Output            (0=Low, 1=High) (R/W)
PF.0-7  PFn   Port F Bit0..7 Input/Undoc  A/D-input  (0=Low, 1=High) (R)(R/W)
```

HC05 Port OPTM=1:00h - DDRA - Port A Data Direction Register (R/W)
HC05 Port OPTM=1:02h - DDRC - Port C Data Direction Register (R/W)
HC05 Port OPTM=1:03h - DDRD - Port D Data Direction Register (R/W)
HC05 Port OPTM=1:04h - DDRE - Port E Data Direction Register (R/W)
HC05 Port OPTM=1:05h - DDRF - Port F Data Direction Register (undoc)
```
DDRX.0-7  DDRXn Port X Data Direction Bit0..7 (0=Input, 1=Output) (R/W)
```
Officially, there are no DDRB and DDRF registers (Port B and F are always Inputs). Although, actually, Motorola's Bootstrap RAM <does> manipulate DDRF.

HC05 Port OPTM=1:08h - RCR1 - Resistor Control Register 1 (R/W)
HC05 Port OPTM=1:09h - RCR2 - Resistor Control Register 2 (R/W)
```
RCR1.0    RAL   Port A.Bit0-3 Pullup Resistors (0=Off, 1=On)
RCR1.1    RAH   Port A.Bit4-7 Pullup Resistors (0=Off, 1=On)
RCR1.2    RBL   Port B.Bit0-3 Pullup Resistors (0=Off, 1=On)
RCR1.3    RBH   Port B.Bit4-7 Pullup Resistors (0=Off, 1=On)
RCR1.4    RGL   Port G.Bit0-3 Pullup Resistors (0=Off, 1=On) ;\
RCR1.5    RGH   Port G.Bit4-7 Pullup Resistors (0=Off, 1=On) ; on chips
RCR1.6    RHL   Port H.Bit0-3 Pullup Resistors (0=Off, 1=On) ; with Port G,H
RCR1.7    RHH   Port H.Bit4-7 Pullup Resistors (0=Off, 1=On) ;/
RCR2.0-7  RCn   Port C.Bit0-7 Pullup Resistors (0=Off, 1=On)
```

HC05 Port OPTM=1:0Ah - WOM1 - Open Drain Output Control Register 1 (R/W)
HC05 Port OPTM=1:0Bh - WOM2 - Open Drain Output Control Register 2 (R/W)
```
WOM1.0   AWOML Port A.Bit0-3 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM1.1   AWOMH Port A.Bit4-5 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM1.2   GWOML Port G.Bit0-3 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM1.3   GWOMH Port G.Bit4-5 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM1.4   HWOML Port H.Bit0-3 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM1.5   HWOMH Port H.Bit4-5 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM1.6-7 -     Not used (0)
WOM2.0-5 CWOMn Port C.Bit0..5 Open Drain Mode when DDR=1 (0=No, 1=Open Drain)
WOM2.6-7 -     Not used (always both bits set)
```

==== Interrupts =====

HC05 Port OPTM=0:08h - INTCR - Interrupt Control Register (R/W)
```
0-1  -     Not used (0)
2    IRQ2S IRQ2 Select Edge-Sensitive Only (0=LowLevelAndNegEdge, 1=NegEdge)
3    IRQ1S IRQ1 Select Edge-Sensitive Only (0=LowLevelAndNegEdge, 1=NegEdge)
4    KWIE  Key Wakeup Interrupt Enable (0=Disable, 1=Enable)
5    -     Not used (0)
6    IRQ2E IRQ2 Interrupt Enable (0=Disable, 1=Enable)
7    IRQ1E IRQ1 Interrupt Enable (0=Disable, 1=Enable)
```

HC05 Port OPTM=0:09h - INTSR - Interrupt Status Register (R and W)
```
0    RKWIF Reset Key Wakeup Interrupt Flag (0=No Change, 1=Reset) (W)
1    -     Not used (0)
2    RIRQ2 Reset IRQ2 Interrupt Flag       (0=No Change, 1=Reset) (W)
3    RIRQ1 Reset IRQ1 Interrupt Flag       (0=No Change, 1=Reset) (W)
4    KWIF  Key Wakeup Interrupt Flag (PB/KWI)       (0=No, 1=IRQ) (R)
5    -     Not used (0)
6    IRQ2F IRQ2 Interrupt Flag (PC6)                (0=No, 1=IRQ) (R)
7    IRQ1F IRQ1 Interrupt Flag (PC7)                (0=No, 1=IRQ) (R)
```

HC05 Port OPTM=1:0Eh - KWIE - Key Wakeup Interrupt Enable Register (R/W)
```
0-7  KWIEn Port B.Bit0..7 Key Wakeup Interrupt Enable (0=Disable, 1=Enable)
```

==== SPI Bus ====

HC05 Port OPTM=0:0Ah - SPCR1 - Serial Peripheral Control Register 1 (R/W)
```
0    SPRn  SPI Clock Rate (0=ProcessorClock/2, 1=ProcessorClock/16)
1-3  -     Not used (0)
4    MSTRn SPI Master Mode Select      (0=Slave/SCK.In, 1=Master/SCK.Out)
```

```
  5    DORDn SPI Data Transmission Order         (0=MSB First, 1=LSB First)
  6    SPEn  SPI Enable (SPI1:PortC, SPI2:PortG) (0=Disable, 1=Enable)
  7    SPIEn SPI Interrupt Enable (... ack HOW?) (0=Disable, 1=Enable)
```

HC05 Port OPTM=0:0Bh - SPSR1 - Serial Peripheral Status Register 1 (R)

```
  0-5  -     Not used (0)
  6    DCOLn SPI Data Collision Occurred         (0=No, 1=Collision)
  7    SPIFn SPI Transfer Complete Flag          (0=Busy, 1=Complete) (R)
```

Note: SPSR1.7 appears to be reset after reading SPSR1 (probably same for SPSR1.6, and maybe also same for whatever SPI IRQ signal).

HC05 Port OPTM=0:0Ch - SPDR1 - Serial Peripheral Data Register 1 (R/W)

```
  0-7  BITn  Data to be sent / being received
```

==== Time Base / Config ====

HC05 Port 10h - TBCR1 - Time Base Control Register 1 (R/W)

```
  0-1  T2R   Timer2 Prescaler (0=SysClk, 1=SysClk/4, 2=SysClk/32, 3=SysClk/256)
  2-3  T3R   PWM Prescaler    (0=CLK3, 1=CLK3/2, 2=CLK3/8, 3=Timer2compare)
  4-6  -     Not used (0)
  7    TBCLK Time Base Clock (0=XOSC, 1=OSC/128) ;<-- write-able only ONCE
```

HC05 Port 11h - TBCR2 - Time Base Control Register 2 (R/W, some bits R or W)

```
  0    COPC  COP Clear 2bit COP timeout divider (0=No Change, 1=Clear) (W)
  1    COPE  COP Enable                         ;<-- write-able only ONCE
  2    -     Not used (0)
  3    RTBIF Reset Time Base Interrupt Flag (0=No Change, 1=Clear TBIF) (W)
  4-5  TBR   Time Base Interrupt Rate (0=TBCLK/128, 1=/4096, 2=/8192, 3=/16384)
  6    TBIE  Time Base Interrupt Enable (0=Disable, 1=Enable)
  7    TBIF  Time Base Interrupt Flag   (0=No, 1=IRQ)        (R)
```

HC05 Port OPTM=1:0Fh - MOSR - Mask Option Status Register (R)

```
  0-4  -     Not used (0)
  5    XOSCR XOSC Feedback Resistor (0=None, 1=Implemented)
  6    OSCR  OSC Feedback Resistor  (0=None, 1=Implemented)
  7    RSTR  /RESET Pullup Resistor (0=None, 1=Implemented)
```

Reading this register returns A0h (on PSX/PSone with 52pin chips).

==== Timer 1 ====

HC05 Port 12h - TCR - Timer 1 Control Register (R/W)

```
  0    OLVL  Output Level on TCMP pin on Compare Match? (0=Low, 1=High)
  1    IEDG  Input Edge on TCAP pin (0=NegativeEdge, 1=PositiveEdge)
  2-4  -     Not used (0)
  5    TOIE  Timer Overflow Interrupt Enable (0=Disable, 1=Enable)
  6    OC1IE Output Compare Interrupt Enable (0=Disable, 1=Enable)
  7    ICIE  Input Capture Interrupt Enable  (0=Disable, 1=Enable)
```

HC05 Port 13h - TSR - Timer 1 Status Register (R)

```
  0-4  -     Not used (0)
  5    TOF   Timer Overflow Flag (0=No, 1=Yes) (R) ;clear by Port 19h access
  6    OC1F  Output Compare Flag (0=No, 1=Yes) (R) ;clear by Port 17h access
  7    ICF   Input Capture Flag  (0=No, 1=Yes) (R) ;clear by Port 15h access
```

HC05 Port 14h - ICH - Timer 1 Input Capture High (undoc)
HC05 Port 15h - ICL - Timer 1 Input Capture Low (undoc)

```
  0-15 Capture Value
```

HC05 Port 16h - OC1H - Timer 1 Output Compare 1 High (undoc)
HC05 Port 17h - OC1H - Timer 1 Output Compare 1 Low (undoc)

```
  0-15 Compare Value
```

HC05 Port 18h - TCNTH - Timer 1 Counter 1 High (undoc)
HC05 Port 19h - TCNTL - Timer 1 Counter 1 Low (undoc)

```
  0-15 Counter
```

HC05 Port 1Ah - ACNTH - Alternate Counter High (undoc)
HC05 Port 1Bh - ACNTL - Alternate Counter Low (undoc)

```
  0-15 Alternate Counter (uh, what?)
```

==== Timer 2 ====

HC05 Port 1Ch - TCR2 - Timer 2 Control Register (R/W)

```
  0    OL2   Timer Output 2 Edge (0=Falling, 1=Rising)
  1    OE2   Timer Output 2 Enable (EVO) (0=Disable, 1=Enable)
  2    IL2   Timer Input 2 Edge/Level (0=Low/Falling, 1=High/Rising)
  3    IM2   Timer Input 2 Mode Select for EVI (0=EventMode, 1=GatedByCLK2)
  4    T2CLK Timer 2 Clock Select (0=CLK2 from Prescaler, 1=EXCLK from EVI)
  5    -     Not used (0)
  6    OC2IE Output Compare 2 Interrupt Enable    (0=Disable, 1=Enable)
  7    TI2IE Timer Input 2 Interrupt Enable (EVI) (0=Disable, 1=Enable)
```

HC05 Port 1Dh - TSR2 - Timer 2 Status Register (R/W)

```
  0-1  -     Not used (0)
  2    ROC2F Reset Output Compare 2 Interrupt Flag (0=No Change, 1=Clear) (W)
  3    RTI2F Reset Timer Input 2 Interrupt Flag    (0=No Change, 1=Clear) (W)
  4-5  -     Not used (0)
  6    OC2F  Output Compare 2 Interrupt Flag    (0=No, 1=Yes) (R)
  7    TI2F  Timer Input 2 Interrupt Flag (EVI) (0=No, 1=Yes) (R)
```

HC05 Port 1Eh - OC2 - Timer 2 Output Compare Register (R/W)

```
  0-7  Compare Value ("Transferred to buffer on certain events?")
```

HC05 Port 1Fh - TCNT2 - Timer 2 Counter Register (R) (W=Set Counter to 01h)

```
  0-7  Counter Value, incremented at T2R (set to 01h on Compare Matches)
```

==== Reserved ====

HC05 Port 3Fh - Unknown/Unused
Reading this port via Sony's test command returns 20h (same as openbus), but reading it via Motorola's selftest function returns 00h (unlike openbus), so it seems to have some unknown/undocumented function; bit5 might indicate selftest mode, or it might reflect initialization of whatever other ports.

HC05 Port OPTM=0:06h..07h,0Dh..0Fh - Reserved
HC05 Port OPTM=1:01h,06h..07h,0Ch..0Dh - Reserved
HC05 Port 20h..3Dh - Reserved
These ports are unused/reserved. Trying to read them on a PSone does return 20h (possibly the prefetched next opcode value from the RAM test command). Other HC05 variants contain some extra features in these ports:
CDROM Internal HC05 On-Chip I/O Ports - Extras
The PSX CDROM BIOS doesn't use any of these ports - execpt, it is writing [20h]=2Eh (possibly to disable unused LCD hardware; which might be actually present in the huge 80pin HC05 chips on old PU-7 mainboards).

HC05 Openbus
Openbus values can be read from invalid memory locations, on PSX with 52pin chips:
```
I/O bank 0:    0:06h..07h, 0:0Dh..0Fh
I/O bank 1:    1:01h, 1:06h..07h, 1:0Ch..0Dh, and upper 4bit of 1:05h
Unbanked I/O:  20h..3Dh
Unused Memory: 0240h..0FFFh, 5000h..FDFFh
```
The returned openbus value depends on the opcode's memory operand:
```
[nn],[mmnn],[nn+x],[mmnn+x] --> returns LAST byte of current opcode (=nn)
[x]                         --> returns FIRST byte of following opcode
```

CDROM Internal HC05 On-Chip I/O Ports - Extras

HC05 Port OPTM=0:0Dh - SPCR2 - Serial Peripheral Control Register 2 (R/W)
HC05 Port OPTM=0:0Eh - SPSR2 - Serial Peripheral Status Register 2 (R)
HC05 Port OPTM=0:0Fh - SPDR2 - Serial Peripheral Data Register 2 (R/W)
This is a second SPI channel, works same as first SPI channel, but using the lower 3bits of Port G (instead of Port C) for the SPI signals.

HC05 Port OPTM=0:06h - PORTG - Port G Data Register (R/W)
HC05 Port OPTM=0:07h - PORTH - Port H Data Register (R/W)
HC05 Port 3Ch - PORTJ - Port J Data Register (R/W)
```
PG.0    PG0   Port G Bit0    Input/Output /SDI2      (0=Low, 1=High) (R/W)
PG.1    PG1   Port G Bit1    Input/Output /SDO2      (0=Low, 1=High) (R/W)
PG.2    PG2   Port G Bit2    Input/Output /SCK2      (0=Low, 1=High) (R/W)
PG.3    PG3   Port G Bit3    Input/Output /TCMP      (0=Low, 1=High) (R/W)
PG.4    PG4   Port G Bit4    Input/Output /PWM0      (0=Low, 1=High) (R/W)
PG.5    PG5   Port G Bit5    Input/Output /PWM1      (0=Low, 1=High) (R/W)
PG.6    PG6   Port G Bit6    Input/Output /PWM2      (0=Low, 1=High) (R/W)
PG.7    PG7   Port G Bit7    Input/Output /PWM3      (0=Low, 1=High) (R/W)
PH.0-7  PHn   Port H Bit0..7 Input/Output            (0=Low, 1=High) (R/W)
PJ.0-3  PJn   Port J Bit0..3 Output                  (0=Low, 1=High) (R/W)
PJ.4-7  -     Not used (0)
```

HC05 Port OPTM=1:06h - DDRG - Port G Data Direction Register (R/W)
HC05 Port OPTM=1:07h - DDRH - Port H Data Direction Register (R/W)
```
0-7  DDRXn Port X Data Direction Bit0..7 (0=Input, 1=Output) (R/W)
```

HC05 Port 20h - LCDCR - LCD Control Register (R/W)
```
0    -     Not used (0)
1    PDH   Select Port D (H) (0=FP35-FP38 pins, 1=PD7-PD4 pins)
2    PEL   Select Port E (L) (0=FP31-FP34 pins, 1=PE3-PE0 pins)
3    PEH   Select Port E (H) (0=FP27-FP30 pins, 1=PE7-PE4 pins)
4    -     Not used (0)
5-6  DUTY  LCD Duty Select (...)
7    LCDE  LCD Output Enable BP and FP pins (0=Disable, 1=Enable)
```

HC05 Port 21h..34h - LCDDR1..20 - LCD Data Register 1..20 (R/W)
```
0-3  First Data Unit  ;\Fourty 4bit LCD values (in the twenty registers)
4-7  Second Data Unit ;/(some duties use only the LSBs of that 4bit values)
```

HC05 Port 34h - PWMCR - PWM Pulse Width Modulation Control Register (R/W)
```
0-3  CH0-3 PWM Channel 0..3 on Port G.Bit4-7 Enable (0=Disable, 1=Enable)
4-7  -     Not used (0)
```

HC05 Port 35h - PWMCNT - PWM Counter Register (R) (W=Set Counter to FFh)
```
0-7  PWM Counter, incremented at PHI2 (range 01h..FFh)
```

HC05 Port 36h - PWMDR0 - PWM Duty Register 0 (R/W)
HC05 Port 37h - PWMDR1 - PWM Duty Register 1 (R/W)
HC05 Port 38h - PWMDR2 - PWM Duty Register 2 (R/W)
HC05 Port 39h - PWMDR3 - PWM Duty Register 3 (R/W)
```
0-7  Duty (N cycles High, 255-N cycles Low)
```

HC05 Port 3Ah - ADR - A/D Data Register (R)
```
0-3  A/D Conversion result (probably unsigned, 00h=Lowest, FFh=Max voltage?)
```

HC05 Port 3Bh - ADSCR - A/D Status and Control Register (R/W)
```
0-3  CH0-3 A/D Channel (0..7=PortF.Bit0-7, 8..0Fh=Reserved/Vref/FactorTest)
4    -     Not used (0)
5    ADON  A/D Charge Pump enable (0=Disable, 1=Enable)
6    ADRC  A/D RC Oscillator On (0=Normal/Use CPU Clock, 1=Use RC Clock)
7    COCO  A/D Conversion Complete (0=Busy, 1=Complete) (R)
```

HC05 Port 3Dh - PCR - Program Control Register (R/W) (for EPROM version)
```
0    PGM   EPROM Program Command (0=Normal, 1=Apply Programming Power)
1    ELAT  EPROM Latch Control (0=Normal/Read, 1=Latch/Write)
2-7  RES   Reserved for Factory Testing (always 0 in user mode)
```

CDROM Internal HC05 I/O Port Usage in PSX

Port A - Data (indexed via Port E)
```
porta.0-7 i/o  CXD1815Q.Data (indexed via Port E)
porta.0   in   debug.dta.serial.in  ;\normally unused (exists in early bios)
porta.1   out  debug.dta.serial.out ; (prototype/debug_status stuff)
porta.2   out  debug.clk.serial.out ;/(with portc.5 = debug.select)
```

Port B - Inputs
```
portb.0   in   F-BIAS  ;unused
portb.1   in   SCEx input (serial 250 baud, received via 1000Hz timer2 irq)
portb.2   in   LMTSW  aka /POS0        ;\pos0 and door switches
portb.3   in   DOOR   aka SHELL_OPEN   ;/
portb.4   in   TEST2
portb.5   in   TEST1 (CL316) enter test mode (instead of mainloop)
portb.6   in   COUT   ;<-- unused, extra pin, not "SENSE=COUT"
portb.7   in   CXD2510Q.SENSE ;-from CXD2510Q (and forwarded from CXA1782BR)
```

Port C - Inputs/Outputs
```
portc.0   in   CXD2510Q.SUBQ  ;\
portc.1   in?  NC (SPI.OUT)   ; used via SPDR1 to receive SPI bus SUBQ data
portc.2   out  CXD2510Q.SQCK  ;/
portc.3   out  SPEED
portc.4   out    ="SPEED XOR 1"  ... AL/TE ... or CG ... or MIRR ?
portc.5   out  ROMSEL: debug.select   (or "SCLK" on later boards???)
portc.6   in   CXD1815Q.XINT/IRQ2 ;unused (instead INTSTS bits are polled)
portc.7   in   CXD2510Q.SCOR/IRQ1 ;used via polling INTSR.7 (not as irq)
```

Port D - Outputs
```
portd.0   out  NC             ;-unused (always 1)
portd.1   out  CXD2510Q.DATA  ;\serial bus for CXD2510Q
portd.2   out  CXD2510Q.XLAT  ; (and also forwarded to CXA1782BR)
portd.3   out  CXD2510Q.CLOK  ;/
portd.4   out  CXD1815Q.DECCS ;\
portd.5   out  CXD1815Q.DECWR ; control for data/index on Port A/E
portd.6   out  CXD1815Q.DECRD ;/
portd.7   out  LDON  ... IC723.Pin11 ... maybe "laser on" ?
```

Port E - Index (for data on Port A)
```
porte.0-4 out  CXD1815Q.Index (for data on Port A)
porte.5   out  NC, not used
porte.6   out  NC, see "idx_4xh" maybe test signal ???
porte.7   out? NC, TEST? configured as OUTPUT... but used as INPUT?
```

Port F - Motorola Bootstrap Serial I/O (not used in cdrom bios)
```
portf.0   out  NC, TX         ;\
portf.1   in   NC, RX         ; not used by sony's cdrom bios
portf.2   out  NC, RTS        ; (but used by motorola's bootstrap rom)
portf.3   out  NC, DTR        ;/
portf.4-7 -    NC, not used (probably pins don't even exist)
```

Other HC05 I/O Ports
```
SPI 1   - used for receiving SUBQ (via Port C)
IRQ 1   - used for latching/polling SUBQ's "SCOR" (not used as interrupt)
IRQ 2   - connects to CXD1815Q.XINT, but isn't actually used at all
Timer 1 - unused
Timer 2 - generates 1000Hz interrupts (for 250 baud "SCEx" string transfers)
DDRx    - data directions for Port A-F (as listed above)
```
Note: The PSX has the HC05 clocked via 4.00MHz oscillator (older boards), or via 4.3MHz signal from SPU (newer boards); internally, the HC05 is clocked at half of those frequencies (ie. around 2 MHz).

CDROM Internal HC05 Motorola Selftest Mode

52-pin HC05 chips (newer psx cdrom controllers)
52-pin chips are used on LATE-PU-8 boards, and on later boards ranging from PU-18 up to PM-41(2).
CDROM Internal HC05 Motorola Selftest Mode (52pin chips)

80-pin HC05 chips (older psx cdrom controllers)
80-pin chips are used PU-7, EARLY-PU-8, and PU-9 boards.
CDROM Internal HC05 Motorola Selftest Mode (80pin chips)

32-pin HC05 chips (joypad/mouse)
Sony's Digital Joypad and Mouse are using 32pin chips (with TQFP-32 package), which are probably containing Motorola HC05 CPUs, too. Unknown if/how those chips can be switched into bootstrap/dumping modes.

Pinouts
Pinouts - HC05 Pinouts

CDROM Internal HC05 Motorola Selftest Mode (52pin chips)

Motorola Bootstrap ROM
The Motorola MC68HC05 chips are including a small bootstrap ROM which gets activated upon /RESET when having two pins strapped to following levels:
```
Pin30 PortC.6 (/IRQ2) (/XINT) ----> wire to 3.5V (VCC)
Pin31 PortC.7 (/IRQ1) (SCOR)  ----> wire to 7V (2xVCC)
```
Moreover, two pins are needed on /RESET for selecting a specific test mode:
```
Pin16 PortB.0 ----> ModeSelectBit0 (0=GND, 1=3.5V)
Pin17 PortB.1 ----> ModeSelectBit1 (0=GND, 1=3.5V)
```
The selectable four modes are:
```
Mode0: Jump to RAM Address 0040h (useless when RAM is empty)
Mode1: Semifunctional Selftest (useless)
```

```
Mode2: Upload 200h bytes to RAM & jump to 0040h (allows fast/custom dumping)
Mode3: Download ROM as ASCII hexdump (nice, but very slow)
```

The upload/download functions are using following additional pins:

```
Pin50 PortF.0 ----> TX output (11bytes: 0Dh,0Ah," AAAA DD ")
Pin51 PortF.1 <---- RX input  (1byte: "!" to request next 11 bytes)
Pin52 PortF.2 ----> RTS output or so (not needed)
Pin1  PortF.3 ----> DTR output or so (not needed)
Ground ------------ GND for RX/TX
```

RX/TX are RS232-like serial signals (but using other voltages, 0=0V and 1=3.5V). Transfer format is 8-N-1, ie. one startbit(0), 8 databits LSB first, no parity, one stopbit(1). Baudrate is OSC/2/208 (ie. 9616 bps for 4.000MHz, or 10176 bps for 4.2336MHz clock derived from CXD2545Q/CXD2938Q).
Note: Above pins may vary on some chips (namely on chips that don't have PortF). The pins for entering bootstrap mode (PortC in this case) should be described in datasheets; but transfer protocol and mode selection (PortB) and transmission (PortF) aren't officially documented.

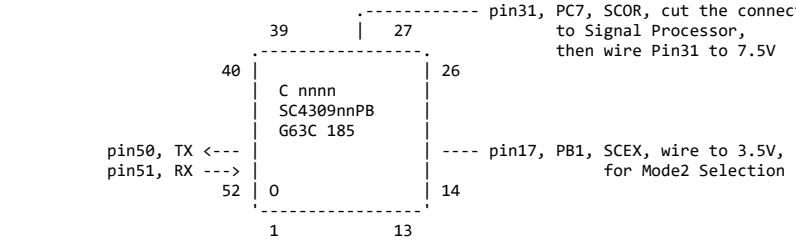
Mode2: Upload 200h bytes to RAM & jump to 0040h

This mode is very simple and powerful: After /RESET, you send 200h bytes to the RX input (without any response on TX output), the bytes are stored at 0040h..023Fh in RAM, and the chip jumps to 0040h after transferring the last byte. The uploaded program can contain a custum highspeed dumping function, or perform hardware tests, etc. A custom dumping function for PSX/PSone can be found at:

```
http://www.psxdev.net/forum/viewtopic.php?f=70&t=557
```

After uploading the 200h-byte dumping function it will respond by send 4540h bytes (containing some ASCII string, the 16.5Kbyte ROM image, plus dumps for RAM and (banked) I/O port region, plus openbus tests for unused memory and I/O regions.

Wiring for Mode2 on PSX/PSone consoles with 52-pin HC05 chips



Good places to pick 3.5V and 7.5V from nice solder pads are:

```
CN602.Pin1  = 7.5V     ;\on PSX boards (with either 5pin or
CN602.Pin3  = 3.5V     ;/               7pin CN602 connectors)
IC601.Pin1  = 7.5V     ;-on PSone boards (3pin 78M05 voltage regulator)
IC102.Pin32 = 3.5V     ;-on PSone boards (32pin Main BIOS ROM chip)
```

The SCOR trace on Pin31, connects to Signal Processor...

```
CXD2510Q.Pin63  (eg. on PU-8 boards)   ;\
CXD2545Q.Pin74  (eg. on PU-18 boards)  ; either one of these, depending
CXD2938Q.Pin77  (eg. on PM-41 boards)  ; on which chipset you have
CXD2941R.Pin85  (eg. PM-41(2) boards)  ;/
```

cut that trace (preferably on the PCB between two vias or test points, so you can later repair it easily) (better don't try to lift Pin31, it breaks off easily)
Note: Mode2 also requires Pin16=Low, and Pin30=High (but PSX/PSone boards should have those pins at that voltages anyways).

Mode3: Download ROM as ASCII hexdump

This mode is very slow and not too powerful. But it may useful if you can't get Mode2 working for whatever reason. Wiring for Mode3 is same as above, plus PortB.0=3.5V. In this mode, the chip will send one 0Dh,0Ah," AAAA DD " string immediately after /RESET (with 16bit address "AAAA" (initially 1000h), and 8bit data "DD"). Thereafter the chip will wait for incoming commands:

```
4-digit ASCII HEX address --> change address, and return 0Dh,0Ah," AAAA DD "
chr(00h) --> increment address, and return 0Dh,0Ah," AAAA DD "
chr(07h) --> jump to current address (not so useful)
other characters --> same as chr(00h)
All digits/characters sent to RX input will produce an echo on TX output.
```

Basic setup would be wiring RX to GND (the chip will treat that as infinite stream of start bits with chr(00h), so it will respond by sending data from increasing addresses automatically; the increment wraps from 4FFFh to FE00h (skipping the gap between Main ROM and Bootstrap ROM), and also wraps from FFFFh to 0000h; transfer is ultraslow: 13 characters needed per dumped byte: chr(00h) to chip, chr(00h) echo from chip, and 0Dh,0Ah," AAAA DD " from chip.

CDROM Internal HC05 Motorola Selftest Mode (80pin chips)

80pin Sony 4246xx chips

And for anyone else planning to try this, these are the connections:

```
Pin PortC
46  PC7/IRQ1 (SCOR) disconnect from PCB, then wire the pin to Vtst (7.6V)
45  PC6/IRQ2 (/XINT) wire to Vdd (3.5V) (you have to solder to the pin)
```

In bootstrap mode, Port A is used as follows:

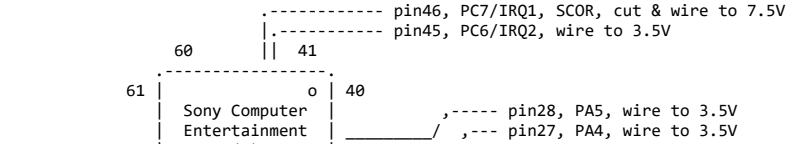
```
Pin PortA DDRA Usage
23  PA0   in   RXD
24  PA1   out  TXD
25  PA2   in   -
26  PA3   in   Testmode.bit0 (GND=0, 3.5V=1)
27  PA4   in   Testmode.bit1 (GND=0, 3.5V=1)
28  PA5   in   Testmode.bit2 (GND=0, 3.5V=1)
29  PA6   out  RTS (don't care)
30  PA7   out  -
```

The selectable testmodes are:

```
PA5 PA4 PA3  Effect
0   x   x    Jump to 0040h      ;\
1   0   0    Test (complex)     ; not so useful
1   0   1    Test (simple loop) ;/
1   1   0    ROM Dump 4200h bytes (plain binary, non-ASCII)
1   1   1    RAM Upload 100h bytes to 0040h..013Fh, then jump to 0040h
```

RX/TX are plain binary (non-ASCII), baudrate is 9600 (when using 4.000MHz oscillator), transfer format is 8,N,2 (aka 8,N,1 with an extra pause bit).

Wiring for Upload/Download on PSX consoles with 80-pin HC05 chips



```
      |     4246xx 183  | ----> pin24, PA1, TXD (for ROM dump)
      |                 | <---- pin23, PA0, RXD (for RAM upload)
   80 | 0               | 21
      '-----------------'
        1             20
```

Good places to pick 3.5V and 7.5V from nice solder pads are:

```
CN602.Pin1  = 7.5V     ;\on PSX boards (with 7pin CN602 connectors)
CN602.Pin3  = 3.5V     ;/
```

Credits to TriMesh for finding the 80pin chip's bootstrap signals.

Other 80pin chips

DTL-H100x uses 80pin chip with onchip PROM (chip text "(M) MC68HC705L15", instead of "Sony [...] 4246xx"), wiring for serial dumping on that is unknown (the bootstrap ROM may be a little different because it should contain PROM burning functions). PU-9 boards boards seem to use a similar PROM (with some sticker on it).

DTL-H2000 uses 80pin CXP82300 chip with socketed piggyback 32pin EPROM - that chip is a Sony SPC700 CPU, not a Motorola HC05 CPU. Accordingly there's no Motorola Bootstrap mode in it, but of course one could simply dump the EPROM with standard eprom utilities, but nobody did do so yet).

CDROM Internal CXD1815Q Sub-CPU Configuration Registers

00h - DRVIF - Drive Interface (W)

```
0-1 "L"     Reserved (should be 0)
2   LSB 1st CD DSP DATA order               (0=MSB first, 1=LSB first)
3-4 BCK MD  CD DSP Number of BCLKs per WCLK (0=16, 1=24, 2=32, 3=Reserved)
5   BCK RED Strobe DATA on BLCK Edge        (0=Falling Edge, 1=Rising Edge)
6   LCH LOW Channel on LRCK=Low             (0=Right, 1=Left)
7   C2PL1st      ... C2PO lower byte 1st
```

01h - CONFIG 1 - Configuration 1 (W)

```
0   HCLKDIS  HCLK Pin Output (0=8.4672MHz, 1=Disable; Fixed Low)
1   CLKDIS   CLK Pin Output (0=16.9344MHz, 1=Disable; Fixed Low)
2   9BITRAM  SRAM Databus width (0=8bit/normal, 1=9bit)
3-4 RAM SZ   SRMA Address bus (0=32K, 1=64K, 2=128K, 3=Reserved)
5   PRTYCTL      ... Priority Control
6   XSLOW    Number of clock cycles per DMA cycle (0=12, 1=4) (for SRAM)
7   "L"      Reserved (should be 0)
```

02h - CONFIG 2 - Configuration 2 (W)

```
0   "L"      Reserved (should be 0)
1   DACODIS        .... DAC Out Disable
2   DAMIXEN  Digital Audio Mixer Enable (0=Attentuator/Mixer for CD-DA, 1=No)
3   SMBF2    Number of Sound Map Buffer Surfaces (0=Three, 1=Two)
4   SPMCTL   Sound Parameter Majority Control (0=?)  ;\for ADPCM params
5   SPECTL   Sound Parameter Error Control    (0=?)  ;/
6-7 "L"      Reserved (should be 0)
```

03h - DECCTL - Decoder Control (W)

```
0-2 DECMD    Decoder Mode (0-7)
               0 or 1  Decoder Disable            ;-disable sector reading
               2 or 3  Monitor-only Mode          ;\no error correction
               4       Write-only Mode            ;/
               5       Real-time Correction Mode  ;\with error correction
               6       Repeat Correction Mode     ;/
               7       CD-DA Mode                 ;-audio
3   AUTODIST Auto Distinction (0=Use MODESEL/FORMSEL bits, 1=Use Sector Hdr)
             (Error Correction is done according to above MODE/FORM values)
4   FORMSEL  Form Select (0=FORM1, 1=FORM2) (must be 0 for MODE1)
5   MODESEL  Mode Select (0=MODE1, 1=MODE2)
6   ECCSTR   ECC Strategy (0=Normal, 1=Use Error Flags; requires 9bit SRAM)
7   ENDLADR  Enable Drive Last Address    ...
```

07h - CHPCTL - Chip Control (W)

```
0   "L"      Reserved (should be 0)
1   DBLSPD   Double Speed Mode (0=Normal, 1=Double) (init CD DSP first)
2   RPSTART  Repeat Correction Start (0=No, 1=Start) (automatically cleared)
3   SWOPEN   Sync Window Open (0=SyncControlledByIC, 1=OpenDetectionWindow)
4   CD-DA    CD-DA Play     (0=No, 1=Playback CD-DA as audio)
5   CDDAMUTE CD-DA Mute     (0=Normal, 1=Mute CD-DA Audio)
6   RTMUTE   Real-time Mute (0=Normal, 1=Mute CDROM ADPCM)
7   SMMUTE   Sound Map Mute (0=Normal, 1=Mute Sound Map ADPCM)
```

CDROM Internal CXD1815Q Sub-CPU Sector Status Registers

00h - ECCSTS - ECC Status (R)

```
0 CFORM   FORM assumed by Error Correction (0=FORM1, 1=FORM2)
1 CMODE   MODE assumed by Error Correction (0=MODE1, 1=MODE2)
2 ECCOK   ECC Okay (0=Bad, 1=Okay)
3 EDCOK   EDC Okay (0=Bad, 1=Okay)
4 CORDONE Correction Done (0=None, 1=Error occurred and was corrected)
5 CORINH  Correction Inhibit (0=Okay,1=AUTODIST couldn't determine MODE/FORM)
6 ERINBLK Erasure in Block (0=Okay, 1=At least 1 byte is wrong & uncorrected)
7 EDCALL0 EDC all-zero  (0=No/EDC Exists, 1=Yes/All four EDC bytes are 00h)
```

01h - DECSTS - Decoder Status (R)

```
0   NOSYNC   No Sync       (0=Okay, 1=Sector Sync Mark Missing)
1   SHRTSCT  Short Sector  (0=Okay, 1=Sector Sync Mark within Sector Data)
2-4 -        Reserved (undefined)
5   RTADPBSY Real-time ADPCM Busy (0=No, 1=Busy/playback)
6-7 -        Reserved (undefined)
```

02h - HDRFLG - Header/Subheader Error Flags for HDR/SHDR registers (R)

```
0 CI      Error in 4th Subheader byte (Coding Info) (0=Okay, 1=Error)
1 SUBMODE Error in 3rd Subheader byte (Submode)     (0=Okay, 1=Error)
2 CHANNEL Error in 2nd Subheader byte (Channel)     (0=Okay, 1=Error)
3 FILE    Error in 1st Subheader byte (File)        (0=Okay, 1=Error)
4 MODE    Error in 4th Header byte (MODE)           (0=Okay, 1=Error)
```

```
  5 BLOCK   Error in 3rd Header byte (FF)            (0=Okay, 1=Error)
  6 SEC     Error in 2nd Header byte (SS)            (0=Okay, 1=Error)
  7 MIN     Error in 1st Header byte (MM)            (0=Okay, 1=Error)
```

Error flags for current sector are probably stored straight in this register (ie. these flags are probably available even without using 9bit SRAM).
Or maybe not... if the chip supports receiving newer sectors during time-consuming error corrections... then those newer would need to be stored in SRAM, and would thus require 9bit SRAM for the error flags?

03h - HDR - Header Bytes (R)

```
  1st read: 1st Header byte (MM)
  2nd read: 2nd Header byte (SS)
  3rd read: 3rd Header byte (FF)
  4th read: 4th Header byte (MODE)
```

04h - SHDR - Subheader Bytes (R)

```
  1st read: 1st Subheader byte (File)
  2nd read: 2nd Subheader byte (Channel)
  3rd read: 3rd Subheader byte (Submode) (SM)
  4th read: 4th Subheader byte (Coding Info) (CI)
```

The contents of the HDRFLG, HDR, SHDR registers indicate:

```
     (1) The corrected value in the real-time correction or
         repeat correction mode
     (2) Value of the raw data from the drive in the monitor-only
         or write-only mode
   The CMOME? and CMODE bits (bits 1, 0) of ECCSTS indicate the FORM and MODE
   of the sector the decoder has discriminated by the raw data from the drive.
   Due to erroneous correction, the values of these bits may be at variance
   with those of the HDR register MODE byte and SHDR register submode byte
   bit5.
```

Unknown when 1st..4th read indices are reset for HDR and SHDR (maybe on access to certain I/O ports, or maybe anytime when receiving a new sector), also unknown what happens on 5th read and up.

CDROM Internal CXD1815Q Sub-CPU Address Registers

Drive Address -- used for storing incoming CDROM sectors in Buffer RAM
Host Address -- used for transferring Buffer RAM to (or from) Main CPU
ADPCM Address -- used for Real-time ADPCM audio output from Buffer RAM

05h - CMADR - Drive Current Minute Address (R)

```
  0-6   CMADR    Address bit10-16 (in 1Kbyte steps)
  7     -        Reserved (undefined)
```

Indicates the start address of the most recently decoded sector (called "Minute Address" because it points to the MM byte of the MM:SS:FF:MODE sector header).
Normally, CMADR should be forwarded to Host:

```
  HADR = (CMADR AND 7Fh)*400h+offset
  HXFR = length OR 4000h
```

Whereas, offset would be usually 00h, 04h, or 0Ch (to start reading from the begin of the sector, or to skip 4-byte MODE1 header, or 12-byte MODE2 header). And length would be usually 800h (normal data sector), or 924h (entire sector, excluding the leading 12 sync-bytes). Length bit14 is undocumented/reserved, but the PSX CDROM BIOS does set that bit for whatever reason.
Alternately, the sector can be forwarded to the Real-time ADPCM decoder:

```
  ADPMNT = (CMADR AND 7Fh) OR 80h
```

19h - ADPMNT - ADPCM "MNT" Address (W)

```
  0-6   ADPxxx  ADPCM source Address bit10-16 (in 1Kbyte-steps)
  7     RTADPEN Real-time ADPCM Enable (0=Disable, 1=Enable Real-time ADPCM)
```

04h - DLADR-L, Drive Last Address, bit0-7 (W)
05h - DLADR-M, Drive Last Address, bit8-15 (W)
06h - DLADR-H, Drive Last Address, bit16 (W)

```
  0-16  DLADR  Addr. bit0-16     ...
  17-23 "L"    Reserved (should be 0)
```

10h - DADRC-L - Drive Address Counter, bit0-7 (W)
11h - DADRC-M - Drive Address Counter, bit8-15 (W)
12h - DADRC-H - Drive Address Counter, bit16 (W)

```
  0-16  DADRC    Incrementing Drive-to-Buffer Write Address, bit0-16
  17-23 "L"      Reserved (should be 0)
```

0Eh - DADRC-L - Drive Address Counter, Bit0-7 (R)
0Fh - DADRC-M - Drive Address Counter, Bit8-15 (R)

```
  0-15  DADRC Address bit0-15  ;bit16 is in Port 0Bh            ...
```

0Ch - HXFR-L - Host Transfer Length, bit0-7 (W)
0Dh - HXFR-H - Host Transfer Length, bit8-11 and stuff (W)

```
  0-11  HXFR     number of data bytes, bit0-11 (0..FFFh)       ...
  12    HADR.16  HADR  bit16
  13    "L"      Reserved (should be 0)
  14    "L" ??   Reserved (should be 0)   ;<-- XXX but on PSX: Always 1 !?!
                                          ;    seems to Disable INT8 ?!!!
  15    DISHXFRC Disable HXFRC (0=Use HXFRC, 1=Disable, Infinite-or-Zero Len?)
```

0Eh - HADR-L - Host Transfer Address, bit0-7 (W)
0Fh - HADR-M - Host Transfer Address, bit8-15 (W)

```
  0-15  HADR     Addr. bit0-15  ;bit16 in Port 0Dh    ...
```

0Ah - HXFRC-L - Host Transfer Remain Counter, bit0-7 (R)
0Bh - HXFRC-H - Host Transfer Remain Counter, bit8-11, and other bits (R)

```
  0-11  HXFRC Host Transfer Counter bit0-11 (number of remaining bytes)
  12    HADRC bit16  ;MSB of Port 0Ch/0Dh
  13    DADRC bit16  ;MSB of Port 0Eh/0Fh
  14-15 -     Reserved (undefined) (usually both bits set)
```

0Ch - HADRC-L - Host Transfer Address Counter, bit0-7 (R)

0Dh - HADRC-M - Host Transfer Address Counter, bit8-15 (R)
```
  0-15  HADRC Address bit0-15  ;bit16 is in Port 0Bh
```
"This counter keeps the addresses which write or read the data with host into/from the buffer.
When data from the host is written into the buffer or data to the host is read from the buffer, the HADRC value is output from MA0 to 16. HADRC is incremented each time one byte of data from the drive is read from the buffer (BFRD is high) or written into the buffer (BFWR is high)."

Note
When reading from SRAM, data seems to go through a 8-byte data fifo, the HXFRC (remain) and HADRC (addr) values aren't aware of that FIFO (ie. if there's data in the fifo, then addr will be 8 bigger and remain 8 smaller than what has arrived at the host).

Unclear Notes
"If sound map data is to be transferred before the data is transferred (immediately after the host has set the BFRD and BFWR bits (bits 7 and 6) of the HCHPCTL register high)":
```
  900h is loaded into HXFRC
  and 600Ch, 6A0Ch, or 740Ch is loaded into HADRC
  (at least, supposedly, above addresses , for cases when using 32K SRAM)
```
"At any other time":
```
  HADR and HXFR are loaded into HADRC and HXFRC
```
Unknown what the above crap is trying to say exactly.
"At any other time" does apparently refer to cases when transfers get started (whilst during transfer, the address/remain values are obviously increasing/decreasing).
For sound map, theoretically, the SMEN bit should be set, but above does somewhat suggest that BFRD or BFWR (or actually: both BFRD and BFWR) need to be set...?

Sector Buffer Memory Map (32Kx8 SRAM)
```
  0000h 1st Sector (at 0000h..0923h) (unused gap at 0924h..0BFFh)
  0C00h 2nd Sector (at 0C00h..1523h) (unused gap at 1524h..17FFh)
  1800h 3rd Sector (at 1800h..2123h) (unused gap at 2124h..23FFh)
  2400h 4th Sector (at 2400h..2D23h) (unused gap at 2D24h..2FFFh)
  3000h 5th Sector (at 3000h..3923h) (unused gap at 3924h..3BFFh)
  3C00h 6th Sector (at 3C00h..4523h) (unused gap at 4524h..47FFh)
  4800h 7th Sector (at 4800h..5123h) (unused gap at 5124h..53FFh)
  5400h 8th Sector (at 5400h..5D23h) (unused gap at 5D24h..5FFFh)
  6000h Soundmap ADPCM Buffer (at 600Ch..690Bh) (gaps at 6000h and 690Ch)
  6A00h Soundmap ADPCM Buffer (at 6A0Ch..730Bh) (gaps at 6A00h and 730Ch)
  7400h Soundmap ADPCM Buffer (at 740Ch..7D0Bh) (gaps at 7400h and 7D0Ch)
  7E00h Unknown/Unused
```

CDROM Internal CXD1815Q Sub-CPU Misc Registers

16h - HIFCTL - Host Interface Control (W)
```
  0-2  HINT    Request Host Interrupt (INT1..INT7, or 0=None/No change)
  3-7  "L"     Reserved (should be 0)
```

11h - HIFSTS - Host Interface Status (R)
```
  0-2 HINTSTS  Pending Host Interrupt (INT1..INT7, or 0=None/All acknowledged)
  3   DMABUSY  DMA Busy (0=Data FIFO Empty and HXFRC=0, 1=Data Transfer Busy)
  4   PRMRRDY  Paramter Read Ready (0=Parameter FIFO Empty, 1=Ready/Not Empty)
  5   RSLEMPT  Result Empty        (0=Response FIFO Not Empty, 1=Empty)
  6   RSLWRDY  Result Write Ready  (0=Response FIFO Full, 1=Ready/Not Full)
  7   BUSYSTS  Command Busy Status (0=Command Not Empty, 1=Ack'ed by CLRBUSY)
```

0Ah - CLRCTL - Clear Control (W)
```
  0   RESYNC   Sync with CD DSP (0=No change, 1=Resync, eg. after speed change)
  1-3 "L"      Reserved (should be 0)
  4   RTADPCLR Abort Real-time ADPCM (0=No Change, 1=Abort; when ADPMNT.7=0)
  5   CLRRSLT  Clear Reply FIFO    (0=No change, 1=Acknowledge; clear FIFO)
  6   CLRBUSY  Acknowledge Command (0=No change, 1=Acknowledge; clear BUSYSTS)
  7   CHPRST   Chip Reset          (0=No change, 1=Do Chip Initialization)
```

07h - INTSTS - Interrupt Status (R) - (0=No, 1=IRQ)
09h - INTMSK - Interrupt Mask (W) - (0=Disable, 1=Enable)
0Bh - CLRINT - Clear Interrupt Status (W) - (0=No change, 1=Clear/Ack)
```
  0   HCRISD   Host Chip Reset Issued
  1   HSTCMND  Host Command                ...
  2   DECINT   Decoder Interrupt           ..
  3   HDMACMP  Host DMA Complete           .   <-- PSX: used for retry ?!?!!!
  4   RTADPEND Real-time ADPCM end
  5   RSLTEMPT Result Empty
  6   DECTOUT  Decoder Time Out
  7   DRVOVRN  Drive Overrun
```

12h - HSTPRM - Host Parameter (R)
```
  0-7  Param FIFO (check HIFSTS.4 to see if the FIFO is empty)
```
HIFSTS.4 goes off when all bytes read.
Said to have 8-byte FIFO in CXD1199AQ datasheet.
But, PSX has 16-byte Parameter FIFO...!?!

13h - HSTCMD - Host Command (R)
```
  0-7  Command (check INTSTS.1 or HIFSTS.7 to see if a command was sent)
```
Command should be ack'ed via CLRINT.1 and CLRCTL.6.

17h - RESULT - Response FIFO (W)
```
  0-7  Data (has 8-byte FIFO)
```
Said to have 8-byte FIFO in CXD1199AQ datasheet.
But, PSX has 16-byte Response FIFO...!?!

08h - ADPCI - ADPCM Coding Information (R)
```
  0   S/M      ADPCM Stereo/Mono        (0=Mono, 1=Stereo)
  1   -        Reserved (undefined)
  2   FS       ADPCM Sampling Frequency (0=37.8kHz, 1=18.9kHz)
  3   -        Reserved (undefined)
  4   BITLNGTH ADPCM Sample Bit Length  (0=Normal/4bit, 1=8bit)
  5   ADPBUSY  ADPCM Decoding           (0=No, 1=Yes/Busy)
```

```
  6   EMPHASIS ADPCM Emphasis          (0=Normal/Off, 1=On)
  7   MUTE     DA Data is Muted (uh?)  (0=No, 1=Yes/Muted)
```

Unknown if ADPCI is affected by configurations by Main-CPU's Sound Map ADPCM or by Sub-CPU's Real-time ADPCM (or by both)?
Note: Bit5,7 are semi-undocumented in the datasheet (mentioned in the ADPCI description, but missing in overall register summary).

1Bh - RTCI - Real-time ADPCM Coding Information (W)

```
  0    S/M      ADPCM Stereo/Mono        (0=Mono, 1=Stereo)
  1    "L"      Reserved (should be 0)
  2    FS       ADPCM Sampling Frequency (0=37.8kHz, 1=18.9kHz)
  3    "L"      Reserved (should be 0)
  4    BITLNGTH ADPCM Sample Bit Length  (0=Normal/4bit, 1=8bit)
  5    "L"      Reserved (should be 0)
  6    EMPHASIS ADPCM Emphasis           (0=Normal/Off, 1=On)
  7    "L"      Reserved (should be 0)
```

06h,09h,10h,14h..1Fh - Reserved (R)

```
  0-7  Reserved (undefined)
```

Of these, 09h and 10h are officially unused/reserved. And addresses 06h and 14h..1Fh aren't officially mentioned to exist at all.
Trying to read these registers on a PSone returns Data=C0h for 06h, 09h, 10h, 15h-16h, 18h-1Fh, and Data=FFh for 14h, and Data=DEh for 17h.

08h,13h-15h,18h,1Ah,1Ch-1Fh - Reserved (W)

```
  0-7  Reserved (should be 00h) (or don't write at all)
```

Of these, 09h,13h-15h,18h,1Ah are officially unused/reserved. And addresses 1Ch-1Fh aren't officially mentioned to exist at all.

CDROM Internal Commands CX(0x..3x) - CXA1782BR Servo Amplifier

CXA1782BR - CX(0x) - Focus Servo Control - "FZC" FocusZeroCross at SENS pin

```
  23-20 4bit  Command (00h)
  19    1bit  FS4 Focus Servo (0=Off, 1=On)
  18    1bit  FS3 DEFECT
  17    1bit  FS2 Enable Focus Search Voltage (0=Off, 1=On)
  16    1bit  FS1 Select Focus Search Voltage (0=Falling, 1=Rising)
  15-0  16bit Unused (don't care)
```

For Focus Search: Keep FS1=on, and toggle FS2 on and off (this will generate a waveform, and SENS will indicate when reaching a good focus voltage).

CXA1782BR - CX(1x) - Tracking/Brake/Sled - "DEFECT" at SENS pin

```
  23-20 4bit  Command (01h)
  19    1bit  TG1,TG2 ON/OFF Tracking Servo Gain Up/Normal (hmmm?)
  18    1bit  Brake Circuit ON/OFF
  17-16 2bit  PS Sled Kick Height (0=+/-1, 1=+/-2, 2=Undoc, 3="Don't use"?)
  15-0  16bit Unused (don't care)
```

Note: The PSX CDROM BIOS does use the "Undoc" setting (ie. bit17=1), but the effect is undoc/unknown?
Note: CX(1x) works different on CXD2545Q (some bits are moved around, and the "SledKickHeight" bits are renamed to "SledKickLevel" and moved to different/new CX(3X) commands.

CXA1782BR - CX(2x) - Tracking and Sled Servo Control - "TZC" at SENS pin

```
  23-20 4bit  Command (02h)
  19-18 2bit  Tracking Control (0=Off, 1=Servo On, 2=F-Jump, 3=R-Jump) ;TM1,3,4
  17-16 2bit  Sled Control     (0=Off, 1=Servo On, 2=F-Fast, 3=R-Fast) ;TM2,5,6
  15-0  16bit Unused (don't care)
```

CXA1782BR - CX(3x) - "Automatic Adjustment Comparator Output" at SENS pin

```
  23-20 4bit  Command (03h)
  19    1bit  Value to be adjusted (0=Balance, 1=Gain)
  18-16 3bit  New Balance or Gain value (depending on above bit)
  15-0  16bit Unused (don't care)
```

Note: CX(3x) is extended and works very different on CXD2545Q.

CXA1782BR Command 4x..7x - "HIGH-Z" at SENS pin

```
  N-N   4bit  Command (04h..07h)
```

CXA1782BR Command 8x..Fx - "UNSPECIFIED???" at SENS pin

```
  N-N   4bit  Command (08h..0Fh)
```

Note

The Servo Amplifier isn't directly connected to the CPU. Instead, it's connected as a slave device to the Signal Processor. There are two ways to access the Servo Amplifier:
1) The CPU can send CX(0X..3X) commands to the Signal Processor (which will then forward them to the Servo Amplifier).
2) The Signal Processor can send CX(0X..3X) commands to the Servo Amplifier (this happens during CX(4xxx) Auto Sequence command).

CDROM Internal Commands CX(4x..Ex) - CXD2510Q Signal Processor

CXD2510Q - CX(4xxx) - Auto Sequence

```
  23-20 4bit  Command (4)
  19-16 4bit  AS3-0 Auto Sequence Command (see below)
  15-12 4bit  MT3-0 Max Timer Value (N timer units, or 0=Invalidate Timer)
  11    1bit  LSSL  Timer Units     (0=2.9ms, 1=186ms) (for above MT value)
  10-8  3bit  Unused (zero)
  7-0   8bit  Unused (don't care)
```

Values for AS (Auto Sequence Command):

```
  00h     Cancel
  04h/05h Forward/Reverse Fine Search   ;<--sends CX(25h) ;\these do internally
  07h     Focus-On                      ;<--sends CX(02h) ; send commands to
  08h/09h Forward/Reverse 1 Track Jump  ;\                ; CXA1782BR
  0Ah/0Bh Forward/Reverse 10 Track Jump ;   sends CX(25h) ; and, auto sequence
  0Ch/0Dh Forward/Reverse 2N Track Jump ;/                ;/is interrupted?
  0Eh/0Fh Forward/Reverse 1N Track Move ;<--CXD2545Q only(Reserved on CXD2510Q)
  01h..03h,06h = Reserved
```

CXD2510Q - CX(5x) - Blind,Brake,Overflow Timer

```
  23-20 4bit  Command (5)
  19-16 4bit  TR3-0 Timer (N*0.022ms for Blind/Overflow, N*0.045ms for Brake)
```

```
  15-8  8bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

CXD2510Q - CX(6xx) - SledKick,Brake,Kick Timer

```
  23-20 4bit  Command (6)
  19-16 4bit  SD3-0 Timer KICK.D (N*2.9ms for Fine Search? else N*1.45ms?)
  15-12 4bit  KF3-0 Timer KICK.F (N*0.09ms)
  11-8  4bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

CXD2510Q - CX(7xxxx) - Track jump count setting (for Auto Sequence Command)

```
  23-20 4bit  Command (7)
  19-4  16bit Track Jump Count Setting (0..65535) for Command 4x
  3-0   4bit  Unused (don't care)
```

CXD2510Q - CX(8xx) - MODE Specification

```
  23-20 4bit  Command (8)
  19    1bit  CDROM        (0=Audio, 1=CDROM; no average and pre-value stuff)
  18    1bit  DOUT Mute    (0=Not muted, 1=Mute DOUT)
  17    1bit  D.out Mute-F (0=Not muted, 1=Mute DA)
  16    1bit  WSEL         (0=Enhanced Sync Window, 1=Enhanced Anti-Rolling)
  15    1bit  VCO SEL      (0=Double Correction, 1=Quadruple Correction)
  14    1bit  ASHS         (0=Double Correction, 1=Quadruple Correction)
  13    1bit  SOCT         (0=Output SubQ to SQSO, 1=Output Each? to SQSO)
  12    1bit  Unused (zero)
  11-8  4bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

CXD2510Q - CX(9xx) - Function Specification

```
  23-20 4bit  Command (9)
  19    1bit  DCLV ON-OFF (complex stuff, related to gain and frequencies)
  18    1bit  DSPB ON-OFF (0=Normal Speed, 1=Double Speed; fixed pitch)
  17    1bit  ASEQ ON-OFF (select output on SENS pin)
  16    1bit  DPLL ON-OFF (0=Analog RFPLL, 1=Digital RFPLL)
  15-14 1bit  Bilingual Audio (0=Stereo, 1=RightOnly, 2=LeftOnly, 3=Mute)
  13    1bit  FLFC (normally 0)
  12    1bit  Unused (zero)
  11-8  4bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

CXD2510Q - CX(Axx) - Audio Control

```
  23-20 4bit  Command (0Ah)
  19    1bit  Vari Up   (write 1-then-0 to increase pitch by +0.1%)
  18    1bit  Vari Down (write 1-then-0 to decrease pitch by -0.1%)
  17    1bit  Mute (0=Not muted; unless muted elsewhere, 1=Mute & Peak=0)
  16    1bit  ATT  (0=Attentuation off, 1=Minus 12 dB)
  15-14 2bit  PCT  (0=Normal, 1=LevelMeter, 2=PeakMeter, 3=Normal) (0-1=QuadC2)
  13-12 2bit  Unused (zero)
  11-8  4bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

Normal: SQSO outputs... WHAT?
PeakMeter: SQSO outputs highest peak ever on any channel (bit15: usually 0)
LevelMeter: SQSO outputs recent peak (with bit15 toggled: 0=Right, 1=Left)

CXD2510Q - CX(Bxxxx) - Traverse Monitor Counter Setting

```
  23-20 4bit  Command (0Bh)
  19-4  16bit Traverse Monitor Count (used when monitored by COMP and COUT) (?)
  3-0   4bit  Unused (don't care)
```

CXD2510Q - CX(Cxx) - Spindle Servo Coefficient Setting

```
  23-20 4bit  Command (0Ch)
  19-18 2bit  Gain MDP for CLVP mode (0=-6db, 1=0dB, 1=+6dB, 3=Reserved)
  17-16 2bit  Gain MDS for CLVS/CLVP (0=-12dB, 1=-6dB, 2=0dB, 3=Reserved)
  15    1bit  Zero (zero)
  14    1bit  Gain DCLV0 overall gain (0=0dB, 1=+6dB
  13-12 2bit  Unused (zero)
  11-8  4bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

CXD2510Q - CX(Dx) - CLV Control

```
  23-20 4bit  Command (0Dh)
  19    1bit  DCLV PWM MD Digital CLV PWM mode  (0=Use MDS+MDP, 1=Ternary MDP)
  18    1bit  TB Bottom Hold in CLVS/CLVH modes (0=At cycle RFCK/32, 1=RFCK/16)
  17    1bit  TP Peak Hold in CLVS mode         (0=At cycle RFCK/4, 1=RFCK/2)
  16    1bit  Gain CLVS for CLVS mode (0=0dB, 1=+6dB)(always +6dB in CLVP mode)
  15-8  8bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

CXD2510Q - CX(Ex) - CLV Mode

```
  23-20 4bit  Command (0Eh)
  19-16 4bit  CM3-0
  15-8  8bit  Unused (don't care on CXD2510Q, zero on CXD2545Q)
  7-0   8bit  Unused (don't care)
```

Values for CM (CLV Mode):

```
  00h Stop  Spindle Motor Stop mode
  06h CLVA  Automatic CLVS/CLVP switching mode, normally used for playback
  08h Kick  Spindle Motor Forward rotation mode
  0Ah Brake Spindle Motor Reverse rotation mode
  0Ch CLVH  Peak hold at 34kHz
  0Eh CLVS  Rough Servo Mode, RF-PLL related
  0Fh CLVP  PLL Servo mode
```

N/A - CX(F) - Reserved

```
  23-0  N/A  Don't use
```

SUBQ Output

```
 80bit subq
 15bit peak level (lsb first) (absolute/unsigned value)
 1bit  peak l/r flag (aka appears as "MSB" of peak level)
```

L/R is toggled after each SUBQ reading, however the PSX Report mode does usually forward SUBQ only every 10 frames, so it stays stuck in one setting (but may toggle after one second; ie. every 75 frames). And, peak is reset after each read, so 9 of the 10 frames are lost.

CXD2510Q - SENS output

```
  Index          ASEQ=0  ASEQ=1   ;<-- ASEQ can be set via CX(9xx)
  0X             HighZ   SEIN (FZC)                   ;\aka SENS output
  1X             HighZ   SEIN (A.S)    ... aka DEFECT   ; from CXA1782BR
  2X             HighZ   SEIN (T.Z.C)  ... aka TZC      ; forwarded through
  3X             HighZ   SEIN (SSTOP)  ... aka Gain/Bal ;/CXD2510Q
  4X             HighZ   XBUSY
  5X             HighZ   FOK
  6X             HighZ   SEIN (HighZ)
  AX             GFS     GFS
  BX             COMP    COMP
  CX             COUT    COUT
  EX             /OV64   /OV64
  7X-9X,DX,FX    HighZ   0
```

Whereas,

```
 FZC    Focus Zero Cross
 DEFECT Defect?
 TZC    Tracking Zero Cross
 SSTOP  Gain or Balance adjust reached wanted level
 XBUSY  Low while the auto sequencer operation is busy
 FOK    High for "focus OK" (same as FOK pin)
 GFS    High when the played back frame sync is obtained with correct timing
 COMP   Measures the number of tracks set with Reg B. High when Reg B is
          latched, low when the initial Reg B number is input by CNIN
 COUT   Measures the number of tracks set with Reg B. High when Reg B is
          latched, toggles each time the Reg B number is input by CNIN. While
          $44 and $45 are being executed, toggles with each CNIN 8-count
          instead of the Reg B number
 OV64   Low when filtered EFM signal is lengthened by 64 channel clock
          pulses or more
```

CDROM Internal Commands CX(0x..Ex) - CXD2545Q Servo/Signal Combo

CXD2545Q - CX(0x) and CX(2x) - same as CXA1782BR Servo Amplifier
CDROM Internal Commands CX(0x..3x) - CXA1782BR Servo Amplifier

CXD2545Q - CX(4x..Ex) - same as CXD2510Q Signal Processor
CDROM Internal Commands CX(4x..Ex) - CXD2510Q Signal Processor
One small difference is that the CXD2545Q supports a new "M Track Move" function as part of the CX(4xxx) command. And, some "don't care" bits are now reserved (ie. some commands need to be padded with additional leading "0" bits).

CXD2545Q - CX(1x) - Anti Shock/Brake/Tracking Gain/Filter

```
  23-20 4bit  Command (01h)
  19    1bit  Anti Shock (0=Off, 1=On)
  18    1bit  Brake      (0=Off, 1=On)
  17    1bit  Tracking Gain (0=Normal, 1=Up)
  16    1bit  Tracking Gain Filter (0=Select 1, 1=Select 2)
  15-0  16bit Unused (don't care)
```

CXD2545Q - CX(30..33) - Sled Kick Level

```
  23-20 4bit  Command (03h)
  19-18 2bit  Subcommand (0)
  17-16 2bit  Sled Kick Level (0=+/-1, 1=+/-2, 2=+/-3, 3=+/-4)
  15-0  16bit Unused (don't care)
```

CXD2545Q - CX(34xxxx) - Write to Coefficient RAM

```
  23-16 8bit  Command (34h)
  15    1bit  Zero (0)
  14-8  7bit  Address (00h..4Fh = Select Coefficient K00..K4F)
  7-0   8bit  Data    (00h..FFh = New value)
  PLUS  8bit  Eight more bits on PSone (!)
```

Allows to change the default preset coefficient values,
CDROM Internal Coefficients (for CXD2545Q)

CXD2545Q - CX(34Fxxx) - Write to Special Register

```
  23-12 12bit Command (34Fh)
  11-10 2bit  Index (0=TRVSC, 1=FBIAS, 2=?, 3=?)
  9-0   10bit Data (for FBIAS: bit0=don't care)
```

CXD2545Q - CX(35xxxx) - FOCUS SEARCH SPEED/VOLTAGE/AUTO GAIN

```
  23-16 8bit  Command (35h)
  15-14 2bit  FT  Focus Search-up speed 1
  13-8  6bit  FS  Focus Search limit voltage (default 011000b) (+/-1.875V)
  7     1bit  FTZ Focus Search-up speed 2
  6-0   7bit  FG  AGF Convergence Gain Setting (default 0101101b)
```

CXD2545Q - CX(36xxxx) - DTZC/TRACK JUMP VOLTAGE/AUTO GAIN

```
  23-16 8bit  Command (36h)
  15    1bit  Zero (0)
  14    1bit  DTZC DTZC Delay (0=4.25us/Default, 1=8.5us)
  13-8  6bit  TJ   Track Jump voltage (default 001110b) (+/-1.09V)
  7     1bit  Zero (0)
  6-0   7bit  TG   AGT Convergence Gain Setting (default 0101110b)
```

CXD2545Q - CX(37xxxx) - FZSL/SLED MOVE/Voltage/AUTO GAIN

```
  23-16 8bit  Command (37h)
  15-14 2bit  FZS              XXX pg. 84
  13-8  6bit  SM    Sled Move Voltage
  7     1bit  AGS
  6     1bit  AGJ
  5     1bit  AGGF
  4     1bit  AGGT
  3     1bit  AGV1
```

```
2     1bit  AGV2
1     1bit  AGHS
0     1bit  AGHT
```

CXD2545Q - CX(38xxxx) - Level/Auto Gain/DFSW (Initialize)

```
23-16 8bit  Command (38h)
15    1bit  VCLM VC level measurement on/off
14    1bit  VCLC VC level compensation for FCS_In Register on/off
13    1bit  FLM  Focus zero level measurement on/off
12    1bit  FLC0 Focus zero level compensation for FZC Register on/off
11    1bit  RFLM RF zero level measurement on/off
10    1bit  RFLC RF zero level compensation on/off
9     1bit  AGF  Focus automatic gain adjustment on/off
8     1bit  AGT  Tracking automatic gain adjustment on/off
7     1bit  DFSW Defect switch disable (1=disable defect measurement)
6     1bit  LKSW Lock switch (1=disable sled free-running prevention)
5     1bit  TBLM Traverse center measurement on/off
4     1bit  TCLM Tracking zero level measurement on/off
3     1bit  FLC1 Focus zero level compensation for FCS_In Register on/off
2     1bit  TLC2 Traverse center compensation on/off
1     1bit  TLC1 Tracking zero level compensation on/off
0     1bit  TLC0 VC level compensation for TRK/SLD_In register on/off
```

VCLM,FLM,RFLM,TCLM are accepted every 2.9ms.

CXD2545Q - CX(39xx) - Select internal RAM/Registers for serial readout

```
23-16 8bit  Command (39h)
15    1bit  DAC Serial data readout DAC mode on/off
14-8  7bit  SD  Serial readout data select (see below)
7-0   8bit  Unused (don't care)
```

Serial Readout Addresses:

```
Addr    Data  Content
00h     8bit  VC input signal
01h     8bit  SE input signal
02h     8bit  TE input signal
03h     8bit  FE input signal
04h-07h 9bit  TE AVRG register (mirrored to 04h-07h)
08h-0Bh 9bit  FE AVRG register (mirrored to 08h-0Bh)
0Ch-0Fh 9bit  VC AVRG register (mirrored to 0Ch-0Fh)
12h     8bit  RFDC envelope (peak)
13h     8bit  RFDC envelope (bottom)
1Ch     9bit  TRVSC register
1Dh     9bit  FBIAS register
1Eh     8bit  RFDC input signal
1Fh     8bit  RF AVRG register
20h-3Fh 16bit Data RAM        (M00-M1F)
40h-7Fh 8bit  Coefficient RAM (K00-K3F) (note: K40-K4F cannot be read out)
```

CXD2545Q - CX(3Ax000) - Focus BIAS addition enable

```
23-16 8bit  Command (3Ah)
15    1bit  Zero (0)
14    1bit  FBON: FBIAS register addition (0=off, 1=Add FBIAS to FCS)
13-0  14bit Zero (0)
```

CXD2545Q - CX(3Bxxxx) - Operation for MIRR/DFCT/FOK

```
23-16 8bit  Command (3Bh)
15-14 2bit  SFO  FOK Slice Level (...depends on SFOX)
13-12 2bit  SDF  DFCT Slice Level (0=89mV, 1=134mV, 2=179mV, 3=224mV)
11-10 2bit  MAX  DFCT Maximum Time (0=No Limit, 1=2ms, 2=2.36ms, 3=2.72ms)
9     1bit  SFOX FOK Slice Level (...depends on SFO)
8     1bit  BTF  Bottom Hold Double-Speed Count-Up mode for MIRR (0=off)
7-6   2bit  D2V  Peak Hold 2 for DFCT (0=22.05kHz, 1=44.1, 2=88.2, 3=176.4)
5-4   2bit  D1V  Peak Hold 1 for DFCT (0=176.4kHz, 1=352.8, 2=705.6, 3=1411)
3-0   4bit  Zero (0)
```

CXD2545Q - CX(3Cxxxx) - TZC for COUT SLCT HPTZC (Default)

```
23-16 8bit  Command (3Ch)
15-0  16bit Unused (don't care)
```

CXD2545Q - CX(3Dxxxx) - TZC for COUT SLCT DTZC

```
23-16 8bit  Command (3Dh)
15-0  16bit Unused (don't care)
```

CXD2545Q - CX(3Exxxx) - Filter

```
23-16 8bit  Command (3Eh)
15-14 2bit  F1NDM FCS servo filter 1st stage (1=normal, 2=down)
13-12 2bit  F3NUM FCS servo filter 3rd stage (1=normal, 2=up)
11-10 2bit  T1NDM TRK servo filter 1st stage (1=normal, 2=down)
9-8   2bit  T3NUM TRK servo filter 3rd stage (1=normal, 2=up)
7     1bit  DFIS  FCS hold filter input extraction node (0=M05, 1=M04)
6     1bit  TLCD  Mask TLC2 set by D2 of CX(38) only when FOK low
5     1bit  RFLP  Pass signal from RFDC pin through low-pass-filter
4-0   5bit  Zero (0)
```

CXD2545Q - CX(3Fxxxx) - Others

```
23-16 8bit  Command (3Fh)          ... XXX pg. 89
15-14 2bit  Unused (0)
13-12 2bit  XTD
11    1bit  Unused (0)
10-8  3bit  DRR
7     1bit  Unused (0)
6     1bit  ASFG
5     1bit  Unused (0)
4     1bit  LPAS
3-2   2bit  SRO
1-0   2bit  Unused (0)
```

CXD2545Q feedback on 39xx: see pg. 77 (eg. 390C = VC AVRG)

XXX

CXD2545Q - SENS output

Index	ASEQ=0	ASEQ=1	Length
$0X	Z	FZC	-
$1X	Z	AS	-
$2X	Z	TZC	-
$38	Z	AGOK*1	-
$38	Z	XAVEBSY*1	-
$30-37,$3A-3F	Z	SSTP	-
$3904	Z	TE Avrg Reg.	9 bit
$3908	Z	FE Avrg Reg.	9 bit
$390C	Z	VC Avrg Reg.	9 bit
$391C	Z	TRVSC Reg.	9 bit
$391D	Z	FB Reg.	9 bit
$391F	Z	RFDC Avrg Reg.	8 bit
$4X	Z	XBUSY	-
$5X	Z	FOK	-
$6X	Z	0	-
$AX	GFS	GFS	-
$BX	COMP	COMP	-
$CX	COUT	COUT	-
$EX	OV64	OV64	-
$7X-9X,DX,FX	Z	0	-

*1 $38 outputs AGOK during AGT and AGF command settings, and XAVEBSY during AVRG measurement.
SSTP is output in all other cases.

CDROM Internal Commands CX(0x..Ex) - CXD2938Q Servo/Signal/SPU Combo

Most commands are same as on CXD2545Q. New/changed commands are:

CXD2938Q - CX(349xxxxx) - New SCEx
Older PSX consoles have received the "SCEx" string at 250 baud via HC05 PortB.bit1, which allowed modchips to injected faked "SCEx" signals to that pin. To prevent that, the CXD2938Q contains some new 32bit commands that allow to receive somewhat encrypted "SCEx" strings via SPI bus. The used commands are:

```
CX(34910000) NewScexStopReading
CX(3491xy80) NewScexRandomKey(xy)
CX(34920000) NewScexFlushResyncOrSo
CX(34944A00) NewScexInitValue1
CX(3498C000) NewScexInitValue2
CX(349C1000) NewScexThis   ;\inverse  ;\use CX(3C2080) for COUT selection
CX(349D1000) NewScexThat   ;/of COUT  ;/
```

The relevant command is NewScexRandomKey(xy) which does send a random value (x=01h..0Fh, and y=01h), and does then receive a 12-byte response via SPI bus (which is normally used to receive SUB-Q data).

```
1st byte: Unknown/unused (normally ADR/Control) ;\these should be probably
2nd byte: Unknown/unused (normally Track)       ; set to some invalid values
3rd byte: Unknown/unused (normally Index/Point) ;/to avoid SUB-Q confusion
4th..10th byte: SCEx data or Dummy bytes (depending on xy.bit7..1)
11th..12th byte: Unknown/unused (normally Audio Peak level)
```

The 12-byte packet does contain one SCEx character encoded in 4th..10th byte corresponding to "xy" flags bits 7..1 (in that order):
All bytes with Flag=1 are ORed together to compute a Character byte (those bytes could be all set to 53h for "S", or if more than one flag is set, it could be theoretically split to something like 41h and 12h).
All bytes with Flag=0 are ORed together to compute a Dummy byte. If the Character byte is same as the Dummy byte, then it gets destroyed by setting Character=00h (to avoid that, one could set all dummies to 00h, or set one or more dummies to FFh, for example).
Finally, "xy" bit0=0 indicates that the resulting character byte is inverted (XORed by FFh), however, the CDROM BIOS does always use bit0=1, so the inversion feature is never used.
For the whole SCEx string, there must be at least one 00h byte inserted between each character (or some Char=Dummy mismatch, which results in Char=00h either), and there should be a few more 00h bytes preceeding the first character ("S").
Note: Modchips didn't bother to reproduce that new SCEx transfers, instead they have simply bypassed it by injecting the 250 baud SCEx string to some analog lower level signal.

CXD2938Q - CX(3Bxxxx) - Some Changed Bits
Same as in older version, but initialized slightly differently: CXD2545Q used CX(3B2250) whilst CXD2938Q is using CX(3B7250).

CXD2938Q - CX(3Cxxxx) - TzcCoutSelect with New/Changed Extra Bits
The CXD2545Q used two 8bit commands, CX(3C) and CX(3D), for TzcOut selection, which are now replaced by a single 24bit command, CX(3Cxxxx), and which do include a new mode related to New SCEx.

```
CXD2545Q   CXD2938Q
CX(3C)     CX(3C0080) TzcCoutSelectHPTZC;\ <--formerly CX(3C)
-          CX(3C2080) TzcCoutSelectSCEX ;  <--special NewScex mode
CX(3D)     CX(3C4080) TzcCoutSelectDTZC ;/ <--formerly CX(3D)
```

CXD2938Q - CX(8xxxxx) - Disc Mode with New/Changed Extra Bits
Command CX(8xx) has been 12bit wide on CXD2545Q, and is now expanded 24bit width (with some changed/unknown bits).

```
CXD2545Q   CXD2938Q
CX(8180)   CX(810408) MODE = Audio (CD-DA)
CX(8120)   CX(812400) MODE = Audio (CD-DA) (manual SPI bus access)
CX(8980)   CX(890408) MODE = CDROM (Data)
-          CX(898000) MODE = CDROM (Data) (used on RESET)
```

CXD2938Q - CX(9xx000) - Normal/Double Speed with New Extra Bits
Command CX(9xx) has been 12bit wide on CXD2545Q (with bit12=reserved/zero), and is now expanded 24bit width (with bit12=unknown/one and bit11-0=unknown/zero).

CXD2938Q - CX(Dx0000) and CX(Ex0000) - New Zero Padding
Commands CX(Dx) and CX(Ex) have been 8bit wide on CXD2545Q, and are now zeropadded to 24bit width, ie. CX(Dx0000) and CX(Ex0000). Unknown if the extra bits are hiding any extra features. In practice, the CDROM BIOS is always setting them zero (except in some test commands which are accidently still using the old 8bit form, resulting in garbage in lower 16bits).

CDROM Internal Commands CX(xx) - Notes

Serial Command Transmission (for Signal Processor and Servo Amplifier)
Commands are sent serially LSB first via DATA,CLOK,XLAT pins: DATA+CLOK are used to send commands to the chip, command execution is then applied by dragging XLAT low for a moment.

Commands can be up to 24bits in length, but unused LSBs (the "don't care" bits) can be omitted; the PSX BIOS clips the length to 8bit/16bit where possible (due to the LSB-first transfer order, the chip does treat the most recently transferred bit as MSB=bit23, and there's no need to transfer the LSBs if they aren't used).
Aside from being used as command number, the four most recently transferred bits are also used to select SENS status feedback (for the SENS selection it doesn't matter if the four bits were terminated by XLAT or not).

Sled Motor / Track Jumps / Tracking
The Sled motor moves the drive head to the current read position. On a Compact Disc, the term "Track" does normally refer to Audio tracks (songs). But the drive hardware uses the terms "Track" and "Tracking" for different purposes:
Tracking appears to refer to moving the Optics via magnets (ie. moving only the laser/lens, without actually moving the whole sled) (this is done for fine adjustments, and it seems to happen more or less automatically; emulators can just return increasing sectors without needing to handle special tracking commands).
Track jumps refer to moving the entire Sled, one "track" is equal to one spiral winding (1.6 micrometers). One winding contains between 9 sectors on innermost windings, and 22.5 sectors on outermost windings (the PSX cdrom bios is translating the sector-distance to non-linear track-distance, and emulators must undo that translation; otherwise the sled doesn't arrive at the intended location; the cdrom bios will retry seeking a couple of times and eventually settle down at the desired location - but it will timeout if the sled emulation is too inaccurate).
The PSX hardware uses two mechanisms for moving the Sled:
Command CX(4xxx) Forward/Reverse Track Jump: allows to move the sled by 1..131070 tracks (ie. max 210 millimeters), and the hardware does stop automatically after reaching the desired distance.
Command CX(2x) Forward/Reverse Fast Sled: moves the sled continously until it gets stopped by another command (in this mode, software can watch the COUT signal, which gets toggled each "N" tracks; whereas "N" can be selected via Command CX(Bxxxx), which is configured to N=100h in PSX).
The PSX cdrom bios is issuing another Fast Sled command (in opposite direction) after Fast Sled commands, emulators must somehow interprete this as "sled slowdown" (rather than as actually moving the sled in opposite direction, which could move the sled miles away from the target). For some reason vC1 BIOS is using a relative short slowdown period, whilst vC2/vC3 are using much longer slowdown periods (probably related to different SledKickHeight aka SledKickLevel settings and/or to different Sled Move Voltage settings).

Focus / Gain / Balance
The hardware includes commands for adjusting & measuring focus/gain/balance. Emulators can just omit that stuff, and can always signalize good operation (except that one should emulate failures for Disc Missing; and eventually also for cases like Laser=Off, or Spindle=Stopped).
Focus does obviously refer to moving the lens up/down. Gain does probably refer to reflection level/laser intensity. Balance might refer to tracking adjustments or so.

CDROM Internal Commands CX(xx) - Summary of Used CX(xx) Commands

The PSX CDROM BIOS versions vC1, vC2, and vC3 are using following CX() commands:

```
<--vC1-->  <--vC2-->  <--vC3-->
CXD2510Q   CXD2545Q   CXD2938Q
CX(00)     CX(00)     CX(00)     AllFocusSwitchesOff
CX(02)     CX(02)     CX(02)     FocusSearchVoltageFalling
CX(03)     CX(03)     CX(03)     FocusSearchVoltageRising ;ForTestOnly
CX(08)     CX(08)     CX(08)     FocusServoOn
CX(0C)     CX(0C)     CX(0C)     FocusServoOnAndDefectOn ;diff.usage vC# ?
-----
CX(11)     -          -          SledKickHeight2
CX(12)     -          -          SledKickHeightInvalid
CX(19)     -          -          TrackingGainAndSledKickHeight2
CX(1D)     -          -          TrackingGainBrakeAndSledKickHeight2
CX(1E)     -          -          TrackingGainBrakeAndSledKickHeightInvalid
-----
-          CX(11)     CX(11)     AntiShockOff            ;\
-          CX(13)     CX(13)     AntiShockOffGainUp      ;
-          CX(17)     CX(17)     AntiShockOffGainUpBrake ;/
-----
CX(20)     CX(20)     CX(20)     SledAndTrackingOff
CX(21)     CX(21)     CX(21)     SledServoOn          ;ForTestOnly
CX(22)     CX(22)     CX(22)     SledFastForward
CX(23)     CX(23)     CX(23)     SledFastReverse
CX(24)     -          -          TrackingServoOn
CX(25)     CX(25)     CX(25)     SledAndTrackingServoOn
-          CX(26)     CX(26)     SledFastForwardAndTrackingServoOn
CX(28)     CX(28)     CX(28)     TrackingForwardJump  ;ForTestOnly
CX(2C)     CX(2C)     CX(2C)     TrackingReverseJump  ;ForTestOnly
-----
CX(30+n)   -          -          BalanceAdjust(0..7)
CX(38+n)   -          -          GainAdjust(0..7)
-----
-          CX(30)     CX(30)     SetSledKickLevel1       ;\
-          CX(31)     CX(31)     SetSledKickLevel2       ;
-          CX(32)     CX(32)     SetSledKickLevel3       ;/
-----
-          CX(3400E6) CX(3400E6) SetK00toE6hSledInputGain            ;def=E0h
-          CX(340730) CX(340730) SetK07to30hSledAutoGain       ;blah ;def=30h
-          CX(34114A) CX(34114A) SetK11to4AhFocusOutputGain          ;def=32h
-          CX(341330) CX(341330) SetK13to30hFocusAutoGain      ;blah ;def=30h
-          CX(341D6F) CX(341D6F) SetK1Dto6FhTrackingLowBoostFilterAL ;def=44h
-          CX(341F64) CX(341F64) SetK1Fto64hTrackingLowBoostFilterBL ;def=5Eh
-          CX(342220) CX(342220) SetK22to20hTrackingOutputGain       ;def=18h
-          CX(342330) CX(342330) SetK23to30hTrackingAutoGain   ;blah ;def=30h
-          CX(342D28) CX(342D28) SetK2Dto28hFocusGainDownOutputGain  ;def=1Bh
-          CX(343E70) CX(343E70) SetK3Eto70hTrackingGainUpOutputGain ;def=57h
-          -          CX(34910000) NewScexStopReading      ;\
-          -          CX(3491x180) NewScexRandomKey(x)     ;
-          -          CX(34920000) NewScexFlushResyncOrSo  ; SCEX SPECIAL
-          -          CX(34944A00) NewScexInitValue1       ; see also:
-          -          CX(3498C000) NewScexInitValue2       ; CX(3C2080)
-          -          CX(349C1000) NewScexThis   ;\inverse ;
-          -          CX(349D1000) NewScexThat   ;/of COUT ;/
-          CX(34F000) CX(34F000) SetTRVSCto000h
-          CX(34Fxxx) CX(34Fxxx) SetFBIAStoNNNh
-----
-          CX(3740AA) CX(3740AA) SetSMto00h      ;\set SM to 0,6,7,9
-          CX(3746AA) CX(3746AA) SetSMto06h      ; (sled move voltage)
-          CX(3747AA) CX(3747AA) SetSMto07h      ; (and init several
-          CX(3749AA) CX(3749AA) SetSMto09h      ;/fixed settings)
-----
-          CX(380010) CX(380010) ModeMeasureTrackingZeroLevel ;\Measure modes
-          CX(380800) CX(380800) ModeMeasureRfZeroLevel       ; (accepted
-          CX(382000) CX(382000) ModeMeasureFocusZeroLevel    ; every 2.9ms)
```

```
-          CX(388000) CX(388000) ModeMeasureLevel           ;/
-          CX(38140A) CX(38140A) ModeCompensate
-          CX(38140E) CX(38140E) ModeCompensateAndTraverseCenter
-          CX(38148A) CX(38148A) ModeCompensateAndDefectOff
-          CX(38148E) CX(38148E) ModeCompensateAndDefectOffTraverseCenter
-          CX(3814AA) CX(3814AA) ModeCompensateAndStuffAndMeasureTraverse ;!
-          CX(38150A) CX(38150A) ModeCompensateAndTrackingAutoGain
-          CX(38150E) CX(38150E) ModeCompensateAndTrackingAutoGain
-          CX(38160A) CX(38160A) ModeCompensateAndFocusAutoGain
-----------
-          CX(391E)   -          SenseRFDCinputSignalWithoutDAC  ;\rather
-          CX(3983)   -          SenseFEinputSignalWithDAC       ;/unused
-          CX(399C)   -          SenseTRVSCregisterWithDAC       ;\only if
-          CX(399D)   -          SenseFBIASregisterWithDAC       ;/TEST1=LOW
-----------
-          CX(3A0000) CX(3A0000) FocusBiasAdditionOff            ;\
-          CX(3A4000) CX(3A4000) FocusBiasAdditionOn             ;/
-          CX(3B2250)!CX(3B7250)!InitOperationForMirrDfctFok <-- vC2/vC3 DIFF
-          CX(3C)  !!!CX(3C0080) TzcCoutSelectHPTZC;\ <--formerly CX(3C)
-          -       !!!CX(3C2080) TzcCoutSelectSCEX ;  <--special NewScex mode
-          CX(3D)  !!!CX(3C4080) TzcCoutSelectDTZC ;/ <--formerly CX(3D)
-          CX(3E0000) CX(3E0000) InitFilterBits                  ;\
-          CX(3E0008) CX(3E0008) InitFilterBitsInvalid           ;/
-          CX(3F0008) CX(3F0008) InitOtherStuff                  ;-
-----------
CX(4000)   CX(4000)   CX(4000)   AutoSeqCancel
CX(4700)   CX(4700)   CX(4700)   AutoSeqFocusOn
CX(4800)   CX(4800)   CX(4800)   Forward1track
CX(4900)   CX(4900)   CX(4900)   Reverse1track
CX(4C00)   CX(4C00)   CX(4C00)   Forward2Ntrack
CX(4D00)   CX(4D00)   CX(4D00)   Reverse2Ntrack
-----------
CX(54)     CX(54)     CX(54)     BlindBrakeOverflowTimer=4
CX(5A)     CX(5A)     CX(5A)     BlindBrakeOverflowTimer=A
CX(6100)   CX(6100)   CX(6100)   SledKickBrakeKickTimer
CX(70xxx0) CX(70xxx0) CX(70xxx0) TrackJumpCountSetting
CX(8180)   CX(8180)!!!CX(810408) MODE = Audio (CD-DA)
-          CX(8120)!!!CX(812400) MODE = Audio (CD-DA) (manual SPI bus access)
-          -          CX(810000/UNUSED)
-          -          CX(812000/UNUSED)
CX(8980)   CX(8980)   CX(890408) MODE = CDROM (Data)
-          -          CX(898000) MODE = CDROM (Data) (used on RESET)
CX(9B00)   CX(9B00)!!!CX(9B1000) NormalSpeed
CX(9F00)   CX(9F00)!!!CX(9F1000) DoubleSpeed
CX(A040)   CX(A040)   CX(A040)   Attentuation Off
CX(A140)   CX(A140)   CX(A140)   Attentuation -12 dB
CX(B01000) CX(B01000) CX(B01000) TraverseMonitorCounterSetting
CX(C600)   CX(C600)   CX(C600)   SpindleServoCoefficientSetting
CX(D7)     CX(D7)     CX(D70000) ClvControl (fixed)
CX(E0)     CX(E0)     CX(E00000) SpindleMotorStop
-          -          CX(E02000) <-- aka bugged CX(E0) with CRAP=2000h
CX(E6)     CX(E6)     CX(E60000) AutomaticNormal
CX(E8)     CX(E8)     CX(E80000) SpindleMotorForward
-          -          CX(E8crap) <-- aka bugged CX(E8) with CRAP=xxxxh
CX(EA)     CX(EA)     CX(EA0000) SpindleMotorReverse
-          -          CX(EAcrap) <-- aka bugged CX(EA) with CRAP=xxxxh
CX(EE)     CX(EE)     CX(EE0000) RoughServo
-----------
CX(F)      CX(F)      CX(F)      Unused (N/A)
-----------
CX(Xx)     CX(Xx)     CX(Xx)       ;\
CX(Xxxx)   CX(Xxxx)   CX(Xxxx)     ; TestCommand (cmd_19h_50h)
CX(Xxxxxx) CX(Xxxxxx) CX(Xxxxxx)   ;
-          -          CX(Xxxxxxxx) ;/
-          CX(Xxxxxx) CX(Xxxxxx) SerialSense, CX(Xxxx) with extra 8bit junk
```

Note: for vC2, some CX(38xxxx) values may differ depending on "set_mid_lsb_to_140Eh".
For vC2, CX(Dx) and CX(Ex) should be officially zero-padded to CX(Dx00) and CX(Ex00), but the vC2 BIOS doesn't do that, it still uses short 8bit form.
For vC2, CX(Dx) and CX(Ex) should be apparently zero-padded to CX(Dx0000) and CX(Ex0000), at least, the vC3 BIOS is doing so (except on some test comannds that do still use the CX(Ex) short form).

Used Sense Values

```
sense(30)  SEIN.BAL       ;vC2: SSTP
sense(38)  SEIN.GAIN      ;vC2: AGOK(AGT/AGF) or XAVEBSY(AVRG) or SSTP(else?)
sense(40)  XBUSY (low=AutoSeqBusy)
sense(50)  FOK   (high=FokusOkay)
sense(A0)  GFS   (high=GoodFrameSync, ie. CorrectPlaybackSpeed)
sense(C5)  COUT  (toggles each 100h 'tracks') (100h=selected via CX(B01000))
sense(EA)  /OV64 (low=EFM too long?)
```

CDROM Internal Coefficients (for CXD2545Q)

The CXD2545Q contains Preset Coefficients in internal ROM, which are copied to internal Coefficient RAM shortly after Reset. CX(34xxxx) allows to change those RAM settings, and CX(39xxxx) allows to readout some of those values serially.

CXD2545Q - Coefficient Preset Values

```
Addr Val Expl.
K00  E0  Sled input gain
K01  81  Sled low boost filter A-H
K02  23  Sled low boost filter A-L
K03  7F  Sled low boost filter B-H
K04  6A  Sled low boost filter B-L
K05  10  Sled output gain
K06  14  Focus input gain
K07  30  Sled auto gain
K08  7F  Focus high cut filter A
K09  46  Focus high cut filter B
K0A  81  Focus low boost filter A-H
K0B  1C  Focus low boost filter A-L
K0C  7F  Focus low boost filter B-H
K0D  58  Focus low boost filter B-L
```

```
 K0E  82  Focus phase compensate filter A
 K0F  7F  Focus defect hold gain
 K10  4E  Focus phase compensate filter B
 K11  32  Focus output gain
 K12  20  Anti shock input gain
 K13  30  Focus auto gain
 K14  80  HPTZC / Auto Gain High pass filter A
 K15  77  HPTZC / Auto Gain High pass filter B
 K16  80  Anti shock high pass filter A
 K17  77  HPTZC / Auto Gain low pass filter B
 K18  00  Fix (should not change this preset value)
 K19  F1  Tracking input gain
 K1A  7F  Tracking high cut filter A
 K1B  3B  Tracking high cut filter B
 K1C  81  Tracking low boost filter A-H
 K1D  44  Tracking low boost filter A-L
 K1E  7F  Tracking low boost filter B-H
 K1F  5E  Tracking low boost filter B-L
 K20  82  Tracking phase compensate filter A
 K21  44  Tracking phase compensate filter B
 K22  18  Tracking output gain
 K23  30  Tracking auto gain
 K24  7F  Focus gain down high cut filter A
 K25  46  Focus gain down high cut filter B
 K26  81  Focus gain down low boost filter A-H
 K27  3A  Focus gain down low boost filter A-L
 K28  7F  Focus gain down low boost filter B-H
 K29  66  Focus gain down low boost filter B-L
 K2A  82  Focus gain down phase compensate filter A
 K2B  44  Focus gain down defect hold gain
 K2C  4E  Focus gain down phase compensate filter B
 K2D  1B  Focus gain down output gain
 K2E  00  Not used
 K2F  00  Not used
 K30  80  Fix (should not change this preset value)
 K31  66  Anti shock low pass filter B
 K32  00  Not used
 K33  7F  Anti shock high pass filter B-H
 K34  6E  Anti shock high pass filter B-L
 K35  20  Anti shock filter comparate gain
 K36  7F  Tracking gain up2 high cut filter A
 K37  3B  Tracking gain up2 high cut filter B
 K38  80  Tracking gain up2 low boost filter A-H
 K39  44  Tracking gain up2 low boost filter A-L
 K3A  7F  Tracking gain up2 low boost filter B-H
 K3B  77  Tracking gain up2 low boost filter B-L
 K3C  86  Tracking gain up phase compensate filter A
 K3D  0D  Tracking gain up phase compensate filter B
 K3E  57  Tracking gain up output gain
 K3F  00  Not used
 K40  04  Tracking hold filter input gain
 K41  7F  Tracking hold filter A-H
 K42  7F  Tracking hold filter A-L
 K43  79  Tracking hold filter B-H
 K44  17  Tracking hold filter B-L
 K45  6D  Tracking hold filter output gain
 K46  00  Not used
 K47  00  Not used
 K48  02  Focus hold filter input gain
 K49  7F  Focus hold filter A-H
 K4A  7F  Focus hold filter A-L
 K4B  79  Focus hold filter B-H
 K4C  17  Focus hold filter B-L
 K4D  54  Focus hold filter output gain
 K4E  00  Not used
 K4F  00  Not used
```

Inflate

Inflate/Deflate is a common (de-)compression algorithm. In the PSX world, it's used by the .CDZ cdrom-image format.

Inflate - Core Functions
Inflate - Initialization & Tree Creation
Inflate - Headers and Checksums

Inflate - Core Functions

tinf_uncompress(dst,src)

```
 tinf_init()                    ;init constants (needed to be done only once)
 tinf_align_src_to_byte_boundary()
 repeat
  bfinal=tinf_getbit()          ;read final block flag (1 bit)
  btype=tinf_read_bits(2)       ;read block type (2 bits)
  if btype=0 then tinf_inflate_uncompressed_block()
  if btype=1 then tinf_build_fixed_trees(), tinf_inflate_compressed_block()
  if btype=2 then tinf_decode_dynamic_trees(), tinf_inflate_compressed_block()
  if btype=3 then ERROR         ;reserved
 until bfinal=1
 tinf_align_src_to_byte_boundary()
 ret
```

tinf_inflate_uncompressed_block()

```
 tinf_align_src_to_byte_boundary()
 len=LittleEndian16bit[src+0]                          ;get len
 if LittleEndian16bit[src+2]<>(len XOR FFFFh) then ERROR  ;verify inverse len
 src=src+4                                             ;skip len values
 for i=0 to len-1, [dst]=[src], dst=dst+1, src=src+1, next i    ;copy block
 ret
```

tinf_inflate_compressed_block()
```
repeat
 sym1=tinf_decode_symbol(tinf_len_tree)
 if sym1<256
  [dst]=sym1, dst=dst+1
 if sym1>256
  len  = tinf_read_bits(length_bits[sym1-257])+length_base[sym1-257]
  sym2 = tinf_decode_symbol(tinf_dist_tree)
  dist = tinf_read_bits(dist_bits[sym2])+dist_base[sym2]
  for i=0 to len-1, [dst]=[dst-dist], dst=dst+1, next i
until sym1=256
ret
```

tinf_decode_symbol(tree)
```
sum=0, cur=0, len=0
repeat                        ;get more bits while code value is above sum
 cur=cur*2 + tinf_getbit()
 len=len+1
 sum=sum+tree.table[len]
 cur=cur-tree.table[len]
until cur<0
return tree.trans[sum+cur]
```

tinf_read_bits(num) ;get N bits from source stream
```
val=0
for i=0 to num-1, val=val+(tinf_getbit() shl i), next i
return val
```

tinf_getbit() ;get one bit from source stream
```
bit=tag AND 01h, tag=tag/2
if tag=00h then tag=[src], src=src+1, bit=tag AND 01h, tag=tag/2+80h
return bit
```

tinf_align_src_to_byte_boundary()
```
tag=01h   ;empty/end-bit (discard any bits, align src to byte-boundary)
ret
```

Inflate - Initialization & Tree Creation

tinf_init()
```
tinf_build_bits_base(length_bits, length_base, 4, 3)
length_bits[28]=0, length_base[28]=258
tinf_build_bits_base(dist_bits, dist_base, 2, 1)
ret
```

tinf_build_bits_base(bits,base,delta,base_val)
```
for i=0 to 29
 bits[i]=min(0,i-delta)/delta
 base[i]=base_val
 base_val=base_val+(1 shl bits[i])
ret
```

tinf_build_fixed_trees()
```
for i=0 to 6, tinf_len_tree.table[i]=0, next i       ;[0..6]=0   ;len tree...
tinf_len_tree.table[7,8,9]=24,152,112                ;[7..9]=24,152,112
for i=0 to 23,  tinf_len_tree.trans[i+0]  =i+256, next i  ;[0..23]   =256..279
for i=0 to 143, tinf_len_tree.trans[i+24] =i+0,   next i  ;[24..167] =0..143
for i=0 to 7,   tinf_len_tree.trans[i+168]=i+280, next i  ;[168..175]=280..287
for i=0 to 111, tinf_len_tree.trans[i+176]=i+144, next i  ;[176..287]=144..255
for i=0 to 4, tinf_dist_tree.table[i]=0, next i   ;[0..4]=0,0,0,0,0 ;\dist
tinf_dist_tree.table[5]=32                        ;[5]=32           ; tree
for i=0 to 31, tinf_dist_tree.trans[i]=i, next i  ;[0..31]=0..31    ;/
ret
```

tinf_decode_dynamic_trees()
```
hlit  = tinf_read_bits(5)+257          ;get 5 bits HLIT (257-286)
hdist = tinf_read_bits(5)+1            ;get 5 bits HDIST (1-32)
hclen = tinf_read_bits(4)+4            ;get 4 bits HCLEN (4-19)
for i=0 to 18, lengths[i]=0, next i
for i=0 to hclen-1                     ;read lengths for code length alphabet
 lengths[clcidx[i]]=tinf_read_bits(3)  ;get 3 bits code length (0-7)
tinf_build_tree(code_tree, lengths, 19) ;build code length tree
for num=0 to hlit+hdist-1              ;decode code lengths for dynamic trees
 sym = tinf_decode_symbol(code_tree)
 len=1, val=sym                        ;default (for sym=0..15)
 if sym=16 then len=tinf_read_bits(2)+3, val=lengths[num-1] ;3..6 previous
 if sym=17 then len=tinf_read_bits(3)+3, val=0              ;3..10 zeroes
 if sym=18 then len=tinf_read_bits(7)+11, val=0             ;11..138 zeroes
 for i=1 to len, lengths[num]=val, num=num+1, next i
tinf_build_tree(tinf_len_tree,  0,      hlit)    ;\build trees
tinf_build_tree(tinf_dist_tree, 0+hlit, hdist)   ;/
ret
```

tinf_build_tree(tree, first, num)
```
for i=0 to 15, tree.table[i]=0, next i     ;clear code length count table
;scan symbol lengths, and sum code length counts...
for i=0 to num-1, x=lengths[i+first], tree.table[x]=tree.table[x]+1, next i
tree.table[0]=0
sum=0          ;compute offset table for distribution sort
for i=0 to 15, offs[i]=sum, sum=sum+tree.table[i], next i
for i=0 to num-1  ;create code to symbol xlat table (symbols sorted by code)
 x=lengths[i+first], if x<>0 then tree.trans[offs[x]]=i, offs[x]=offs[x]+1
next i
ret
```

tinf_data

```
clclidx[0..18] = 16,17,18,0,8,7,9,6,10,5,11,4,12,3,13,2,14,1,15   ;constants

typedef struct TINF_TREE:
  unsigned short table[16]     ;table of code length counts
  unsigned short trans[288]    ;code to symbol translation table

TINF_TREE tinf_len_tree   ;length/symbol tree
TINF_TREE tinf_dist_tree  ;distance tree
TINF_TREE code_tree       ;temporary tree (for generating the dynamic trees)
unsigned char lengths[288+32]   ;temporary 288+32 x 8bit ;\for dynamic tree
unsigned short offs[16]         ;temporary 16 x 16bit    ;/creation

unsigned char  length_bits[30]
unsigned short length_base[30]
unsigned char  dist_bits[30]
unsigned short dist_base[30]
```

Inflate - Headers and Checksums

tinf_gzip_uncompress(void *dest, *destLen, *source, sourceLen)

```
src_start=src, dst_start=dst                ;memorize start addresses
if (src[0]<>1fh or src[1]<>8Bh) then ERROR  ;check id bytes
if (src[2]<>08h) then ERROR                 ;check method is deflate
flg=src[3]                                  ;get flag byte
if (flg AND 0E0h) then ERROR                ;verify reserved bits
src=src+10                                          ;skip base header
if (flg AND 04h) then src=src+2+LittleEndian16bit[src]     ;skip extra data
if (flg AND 08h) then repeat, src=src+1, until [src-1]=00h ;skip file name
if (flg AND 10h) then repeat, src=src+1, until [src-1]=00h ;skip file comment
hcrc=(tinf_crc32(src_start, src-src_start) & 0000ffffh))   ;calc header crc
if (flg AND 02h) then x=LittleEndian16bit[src], src=src+2  ;get header crc
if (flg AND 02h) then if x<>hcrc then ERROR                ;verify header
tinf_uncompress(dst, destLen, src, src_start+sourceLen-src-8)  ;----> inflate
crc32=LittleEndian32bit[src], src=src+4   ;get crc32 of decompressed data
dlen=LittleEndian32bit[src], src=src+4    ;get decompressed length
if (dlen<>destLen) then ERROR                              ;verify dest len
if (crc32<>tinf_crc32(dst_start,dlen)) then ERROR          ;verify crc32
ret
```

tinf_zlib_uncompress(dst, destLen, src, sourceLen)

```
src_start=src, dst_start=dst         ;memorize start addresses
hdr=BigEndian16bit[src], src=src+2   ;get header
if (hdr MOD 31)<>0 then ERROR        ;check header checksum (modulo)
if (hdr AND 20h)>0 then ERROR        ;check there is no preset dictionary
if (hdr AND 0F00h)<>0800h then ERROR ;check method is deflate
if (had AND 0F000h)>7000h then ERROR ;check window size is valid
tinf_uncompress(dst, destLen, src, sourceLen-6)      ;------> inflate
chk=BigEndian32bit[src], src=src+4                   ;get data checksum
if src-src_start<>sourceLen then ERROR               ;verify src len
if dst-dst_start<>destLen then ERROR                 ;verify dst len
if a32<>tinf_adler32(dst_start,destLen)) then ERROR  ;verify data checksum
ret
```

tinf_adler32(src, length)

```
s1=1, s2=0
while (length>0)
 k=max(length,5552)
 for i=0 to k-1, s1=s1+[src], s2=s2+s1, src=src+1, next i
 s1=s1 mod 65521, s2=s2 mod 65521, length=length-k
return (s2*10000h+s1)
```

Controllers and Memory Cards

Controllers/Memory Cards
Controller and Memory Card I/O Ports
Controller and Memory Card Misc
Controller and Memory Card Signals
Controller and Memory Card Multitap Adaptor

Controllers
Controllers - Communication Sequence
Controllers - Standard Digital/Analog Controllers
Controllers - Mouse
Controllers - Racing Controllers
Controllers - Lightguns
Controllers - Rumble Configuration
Controllers - Dance Mats
Controllers - Fishing Controllers
Controllers - I-Mode Adaptor (Mobile Internet)
Controllers - Additional Inputs
Controllers - Misc

Memory Cards
Memory Card Read/Write Commands
Memory Card Data Format
Memory Card Images
Memory Card Notes
Other Hardware that connects to Memory Card slot

Pocketstation (Memory Card with built-in LCD screen and buttons)
Pocketstation

Pinouts

Pinouts - Controller Ports and Memory-Card Ports

Controller and Memory Card I/O Ports

1F801040h JOY_TX_DATA (W)

```
0-7   Data to be sent
8-31  Not used
```

Writing to this register starts the transfer (if, or as soon as TXEN=1 and JOY_STAT.2=Ready), the written value is sent to the controller or memory card, and, simultaneously, a byte is received (and stored in RX FIFO if JOY_CTRL.1 or JOY_CTRL.2 is set).
The "TXEN=1" condition is a bit more complex: Writing to SIO_TX_DATA latches the current TXEN value, and the transfer DOES start if the current TXEN value OR the latched TXEN value is set (ie. if TXEN gets cleared after writing to SIO_TX_DATA, then the transfer may STILL start if the old latched TXEN value was set).

1F801040h JOY_RX_DATA (R)

```
0-7   Received Data      (1st RX FIFO entry) (oldest entry)
8-15  Preview            (2nd RX FIFO entry)
16-23 Preview            (3rd RX FIFO entry)
24-31 Preview            (4th RX FIFO entry) (5th..8th cannot be previewed)
```

A data byte can be read when JOY_STAT.1=1. Data should be read only via 8bit memory access (the 16bit/32bit "preview" feature is rather unusable, and usually there shouldn't be more than 1 byte in the FIFO anyways).

1F801044h JOY_STAT (R)

```
0     TX Ready Flag 1   (1=Ready/Started)
1     RX FIFO Not Empty (0=Empty, 1=Not Empty)
2     TX Ready Flag 2   (1=Ready/Finished)
3     RX Parity Error   (0=No, 1=Error; Wrong Parity, when enabled)  (sticky)
4     Unknown (zero)    (unlike SIO, this isn't RX FIFO Overrun flag)
5     Unknown (zero)    (for SIO this would be RX Bad Stop Bit)
6     Unknown (zero)    (for SIO this would be RX Input Level AFTER Stop bit)
7     /ACK Input Level  (0=High, 1=Low)
8     Unknown (zero)    (for SIO this would be CTS Input Level)
9     Interrupt Request (0=None, 1=IRQ7) (See JOY_CTRL.Bit4,10-12)   (sticky)
10    Unknown (always zero)
11-31 Baudrate Timer    (21bit timer, decrementing at 33MHz)
```

1F801048h JOY_MODE (R/W) (usually 000Dh, ie. 8bit, no parity, MUL1)

```
0-1   Baudrate Reload Factor (1=MUL1, 2=MUL16, 3=MUL64) (or 0=MUL1, too)
2-3   Character Length       (0=5bits, 1=6bits, 2=7bits, 3=8bits)
4     Parity Enable          (0=No, 1=Enable)
5     Parity Type            (0=Even, 1=Odd) (seems to be vice-versa...?)
6-7   Unknown (always zero)
8     Destroy Received Data  (0=Normal, 1=Force Data=FFh) (purpose unknown?)
9-15  Unknown (always zero)
```

1F80104Ah JOY_CTRL (R/W) (usually 1003h,3003h,0000h)

```
0     TX Enable (TXEN)  (0=Disable, 1=Enable)
1     /JOYn Output      (0=High, 1=Low/Select) (/JOYn as defined in Bit13)
2     RX Enable (RXEN)  (0=Normal, when /JOYn=Low, 1=Force Enable Once)
3     Unknown? (read/write-able) (for SIO, this would be TX Output Level)
4     Acknowledge       (0=No change, 1=Reset JOY_STAT.Bits 3,9)          (W)
5     Unknown? (read/write-able) (for SIO, this would be RTS Output Level)
6     Reset             (0=No change, 1=Reset most JOY_registers to zero) (W)
7     Not used             (always zero) (unlike SIO, no matter of FACTOR)
8-9   RX Interrupt Mode    (0..3 = IRQ when RX FIFO contains 1,2,4,8 bytes)
10    TX Interrupt Enable  (0=Disable, 1=Enable) ;when JOY_STAT.0-or-2 ;Ready
11    RX Interrupt Enable  (0=Disable, 1=Enable) ;when N bytes in RX FIFO
12    ACK Interrupt Enable (0=Disable, 1=Enable) ;when JOY_STAT.7  ;/ACK=LOW
13    Desired Slot Number  (0=/JOY1, 1=/JOY2) (set to LOW when Bit1=1)
14-15 Not used             (always zero)
```

Caution: After slot selection (via Bits 1,13), one should issue a short delay (XXX: unknown length) before sending the the first data byte; this is required for controllers like Mouse and Analog Joypads (whilst Digital Joypads may work without that delay).

1F80104Eh JOY_BAUD (R/W) (usually 0088h, ie. circa 250kHz, when Factor=MUL1)

```
0-15  Baudrate Reload value for decrementing Baudrate Timer
```

Timer reload occurs when writing to this register, and, automatically when the Baudrate Timer reaches zero. Upon reload, the 16bit Reload value is multiplied by the Baudrate Factor (see 1F801048h.Bit0-1), divided by 2, and then copied to the 21bit Baudrate Timer (1F801044h.Bit11-31). The 21bit timer decreases at 33MHz, and, it ellapses twice per bit (once for CLK=LOW and once for CLK=HIGH).

```
BitsPerSecond = (44100Hz*300h) / MIN(((Reload*Factor) AND NOT 1),1)
```

The default BAUD value is 0088h (equivalent to 44h cpu cycles), and default factor is MUL1, so CLK pulses are 44h cpu cycles LOW, and 44h cpu cycles HIGH, giving it a transfer rate of circa 250kHz per bit (33MHz divided by 88h cycles).
Note: The Baudrate Timer is always running; even if there's no transfer in progress.

/IRQ7 (/ACK) Controller and Memory Card - Byte Received Interrupt

Gets set after receiving a data byte - that only if an /ACK has been received from the peripheral (ie. there will be no IRQ if the peripheral fails to send an /ACK, or if there's no peripheral connected at all).

```
Actually, /IRQ7 means "more-data-request",
accordingly, it does NOT get triggered after receiving the LAST byte.
```

I_STAT.7 is edge triggered (that means it can be acknowledge before or after acknowledging JOY_STAT.9). However, JOY_STAT.9 is NOT edge triggered (that means it CANNOT be acknowledged while the external /IRQ input is still low; ie. one must first wait until JOY_STAT.7=0, and then set JOY_CTRL.4=1) (this is apparently a hardware glitch; note: the LOW duration is circa 100 clock cycles).

/IRQ10 (/IRQ) Controller - Lightpen Interrupt

Pin8 on Controller Port. Routed directly to the Interrupt Controller (at 1F80107xh). There are no status/enable bits in the JOY_registers (at 1F80104xh).

RX FIFO / TX FIFO Notes

The JOY registers can hold up to 8 bytes in RX direction, and almost 2 bytes in TX direction (just like the SIO registers, see there for details), however, normally only 1 byte should be in the RX/TX registers (one shouldn't send a 2nd byte until /ACK is sensed, and, since the transfer CLK is dictated by the CPU, the amount of incoming data cannot exceed 1 byte; provided that one reads received response byte after each transfer).
Unlike SIO, the JOY status register doesn't have a RX FIFO Overrun flag.

General Notes

RXEN should be usually zero (the hardware automatically enables receive when /JOYn is low). When RXEN is set, the next transfer causes data to be stored in RX FIFO even when /JOYn is high; the hardware automatically clears RXEN after the transfer.
For existing joypads and memory cards, data should be always transferred as 8bit no parity (although the JOY registers do support parity just like SIO registers).

Plugging and Unplugging Cautions
During plugging and unplugging, the Serial Data line may be dragged LOW for a moment; this may also affect other connected devices because the same Data line is shared for all controllers and memory cards (for example, connecting a joypad in slot 1 may corrupt memory card accesses in slot 2).
Moreover, the Sony Mouse does power-up with /ACK=LOW, and stays stuck in that state until it is accessed at least once (by at least sending one 01h byte to its controller port); this will also affect other devices (as a workaround one should always access BOTH controller ports; even if a game uses only one controller, and, code that waits for /ACK=HIGH should use timeouts).

Emulation Note
After sending a byte, the Kernel waits 100 cycles or so, and does THEN acknowledge any old IRQ7, and does then wait for the new IRQ7. Due to that bizarre coding, emulators can't trigger IRQ7 immediately within 0 cycles after sending the byte.

Controller and Memory Card Misc

BIOS Functions
Controllers can be probably accessed via InitPad and StartPad functions,
BIOS Joypad Functions
Memory cards can be accessed by the filesystem (with device names "bu00:" (slot1) and "bu10:" (slot2) or so). Before using that device names, it seems to be required to call InitCard, StartCard, and _bu_init (?).

Connectors
The PlayStation has four connectors (two controllers, two memory cards),

```
Memory Card 1         Memory Card 2
Controller 1          Controller 2
```

The controller ports have 9 pins, the memory cards only 8 pins. However, there are only 10 different pins in total.

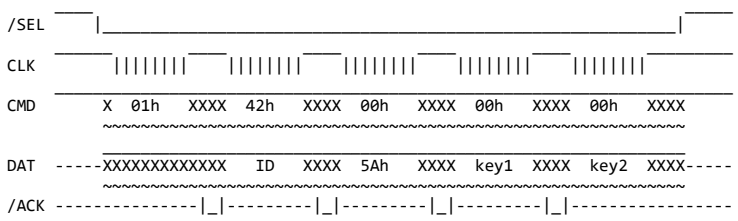
```
JOYDAT,JOYCMD,JOYCLK  Data in/out/clock
+7.5V,+3.5V,GND       Supply
/JOY1,/JOY2  Selects controller/memorycard 1, or controller/memorycard 2
/ACK         Indicates that the device is ready to send more data (IRQ7)
/IRQ10       Lightgun (controllers only, not memory card) (IRQ10)
```

Most of these pins are shared for all 4 connectors (eg. a CLK signal meant to be sent to one device will also arrive at the other 3 devices).
The /JOYn signals are selecting BOTH the corresponding controller, and the corresponding memory card (whether it is a controller access or memory card access depends on the first byte transferred via the CMD line; this byte should be 01h=Controller, or 81h=Memory Card).
The purpose of the Unknown pin is still... unknown? Reportedly, it is usually not connected at the controller-side, but it is probably connected to something at the console-side?

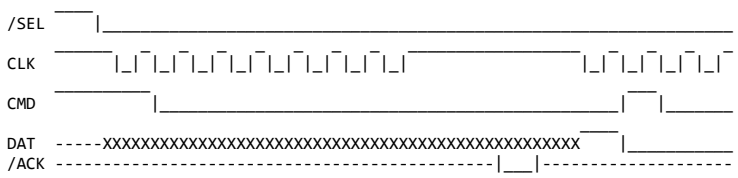
Data In/Out
The data is transferred in units of bytes, via separate input and output lines. So, when sending byte, the hardware does simultaneously receive a response byte.
One exception is the first command byte (which selects either the controller, or the memory card) until that byte has been sent, neither the controller nor memory card are selected (and so the first "response" byte should be ignored; probably containing more or less stable high-z levels).
The other exception is, when you have send all command bytes, and still want to receive further data, then you'll need to send dummy command bytes (should be usually 00h) to receive the response bytes.

Controller and Memory Card Signals

Overview



Top command. First comminucation(device check)



X = none, - = Hi-Z

* 0x81 is memory-card, 0x01 is standard-pad at top command.
* serial data transfer is LSB-First format.
* data is down edged output, PSX is read at up edge in shift clock.
* PSX expects No-connection if not returned Acknowledge less than 100 usec.
* clock pulse is 250KHz.
* no need Acknowledge at last data.
* Acknowledge signal width is more than 2 usec.
* time is 16msec between SEL from previous SEL.
* SEL- for memory card in PAD access.

Controller and Memory Card Multitap Adaptor

SCPH-1070 (Multitap)

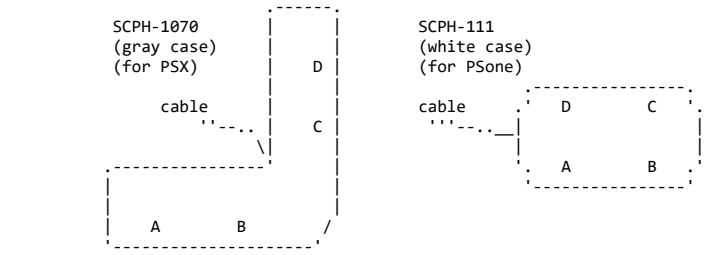
The Multitap is an external adaptor that allows to connect 4 controllers, and 4 memory cards to one controller port. When using two adaptors (one on each slot), up to 8 controllers and 8 memory cards can be used.

Multitap Controller Access
Normally joypad reading is done by sending this bytes to the pad:
```
01 42 00 00 ..   ;normal read
```
And with the multitap, there are even two different ways how to access extra pads:
```
01 42 01 00 ..   ;method 1: receive special ID and data from ALL four pads
0n 42 00 00 ..   ;method 2: receive data from pad number "n" (1..4)
```
The first method seems to be the more commonly used one (and its special ID is also good for detecting the multitap); see below for details.
The second method works more like "normal" reads, among other it's allowing to transfer more than 4 halfwords per slot, altough I don't know if any existing games are using it.
The IRQ10 signal (for Konami Lightguns) is simply wired to all four slots via small resistors (without special logic for activating/deactivating the IRQ on certain slots).

Multitap Controller Access, Method 1 Details
Below LONG response is activated by sending "01h" as third command byte; observe that sending that byte does NOT affect the current response. Instead, it does request that the NEXT command shall return special data, as so:
```
Halfword 0      --> Controller ID for MultiTap (5A80h=Multitap)
Halfword 1..4   --> Player A (Controller ID, Buttons, Analog Inputs, if any)
Halfword 5..8   --> Player B (Controller ID, Buttons, Analog Inputs, if any)
Halfword 9..12  --> Player C (Controller ID, Buttons, Analog Inputs, if any)
Halfword 13..16 --> Player D (Controller ID, Buttons, Analog Inputs, if any)
```
With this method, the Multitap is always sending 4 halfwords per slot (padded with FFFFh values for devices like Digital Joypads and Mice; which do use less than 4 halfwords); for empty slots it's padding all 4 halfwords with FFFFh.
Sending the request is possible ONLY if there is a controller in Slot A (if controller Slot A is empty then the Slot A access aborts after the FIRST byte, and it's thus impossible to send the request in the THIRD byte).
Sending the request works on access to Slot A, trying to send another request during the LONG response is glitchy (for whatever strange reason); one must thus REPEATEDLY do TWO accesses: one dummy Slot A access (with the request), followed by the long Slot A+B+C+D access.
```
Previous access had REQ=0 and returned Slot A data ---> returns Slot A data
Previous access had REQ=0 and returned Slot A-D data -> returns Slot A data
Previous access had REQ=1 and returned Slot A data ---> returns Slot A-D data
Previous access had REQ=1 and returned Slot A-D data -> returns garbage
Previous access had REQ=1 and returned garbage -------> returns Slot A-D data
```
In practice:
Toggling REQ on/off after each command: Returns responses toggling between normal Slot A data and long Slot A+B+C+D data.
Sending REQ=1 in ALL commands: Returns responses toggling between Garbage and long Slot A+B+C+D data.
Both of the above is working (one needs only the Slot A+B+C+D part, and it doesn't matter if the other part is Slot A, or Garbage; as long as the software is able/aware of ignoring the Garbage). Garbage response means that the multitap returns ONLY four bytes, like so: Hiz,80h,5Ah,LSB (ie. the leading HighZ byte, the 5A80h Multitap ID, and the LSB of the Slot A controller ID), and aborts transfer after that four bytes.

Multitap Memory Card Access
Normally memory card access is done by sending this bytes to the card:
```
80 xx .. ..      ;normal access
```
And with the multitap, memory cards can be accessed as so:
```
8n xx .. ..      ;access memory card in slot "n" (1..4)
```
That's the way how its done in Silent Hill. Although for the best of confusion, it doesn't actually work in that game (probably the developer has just linked in the multitap library, without actually supporting the multitap at higher program levels).

Multitap Games
```
Bomberman World
Breakout: Off the Wall Fun
Circuit Breakers
Crash Team Racing
FIFA series soccer games
Frogger
Gauntlet: Dark Legacy
Hot Shots Golf 2 & 3
NBA Live (any year) (up to 8 players with two multitaps)
Need For Speed 3
Need For Speed 5
Poy Poy (4 players hitting each other with rocks and trees)
Running Wild
```

Multitap Versions


The cable connects to one of the PSX controller ports (which also carriers the memory card signals). The PSX memory card port is left unused (and is blocked by a small edge on the Multitap's plug).

MultiTap Parsed Controller IDs
Halfword 0 is parsed (by the BIOS) as usually, ie. the LSB is moved to MSB, and LSB is replaced by status byte (so ID 5A80h becomes 8000h=Multitap/okay, or xxFFh=bad). Halfwords 1,5,9,13 are NOT parsed (neither by the BIOS nor by the Multitap hardware), however, some info in the internet is hinting that Sony's libraries might be parsing these IDs too (so for example 5A41h would become 4100h=DigitalPad/okay, or xxFFh=bad).

Power Supply
The Multitap is powered by the PSX controller port. Unknown if there are any power supply restrictions (up to eight controllers and eight cards may scratch some limits, especially when doing things like activating rumble on all joypads). However, the Multitap hardware itself doesn't do much on supply restrictions (+3.5V is passed through something; maybe some fuse, loop, or 1 ohm resistor or so) (and +7.5V is passed without any restrictions).

See also
Pinouts - Component List and Chipset Pin-Outs for Multitap

Controllers - Communication Sequence

Controller Communication Sequence

```
Send Reply Comment
01h  Hi-Z  Controller Access (unlike 81h=Memory Card access), dummy response
42h  idlo  Receive ID bit0..7 (variable) and Send Read Command (ASCII "B")
TAP  idhi  Receive ID bit8..15 (usually/always 5Ah)
MOT  swlo  Receive Digital Switches bit0..7
MOT  swhi  Receive Digital Switches bit8..15
--- transfer stops here for digital pad (or analog pad in digital mode) ---
00h  adc0  Receive Analog Input 0 (if any) (eg. analog joypad or mouse)
00h  adc1  Receive Analog Input 1 (if any) (eg. analog joypad or mouse)
--- transfer stops here for analog mouse ----------------------------------
00h  adc2  Receive Analog Input 2 (if any) (eg. analog joypad)
00h  adc3  Receive Analog Input 3 (if any) (eg. analog joypad)
--- transfer stops here for analog pad (in analog mode) -------------------
--- transfer stops here for nonstandard devices (steering/twist/paddle) ---
```

The TAP byte should be usually zero, unless one wants to activate Multitap (multi-player mode), for details, see
Controller and Memory Card Multitap Adaptor
The two MOT bytes are meant to control the rumble motors (for normal non-rumble controllers, that bytes should be 00h), however, the MOT bytes have no effect unless rumble is enabled via config commands, for details, see
Controllers - Rumble Configuration

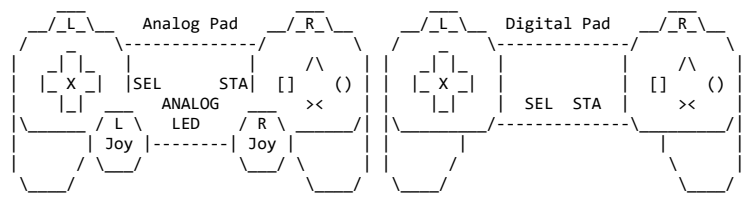
Controller ID (Halfword Number 0)

```
0-3  Number of following halfwords (01h..0Fh=1..15, or 00h=16 halfwords)
4-7  Controller Type (or currently selected Controller Mode)
8-15 Fixed (5Ah)
```

Known 16bit ID values are:

```
xx00h=N/A                (initial buffer value from InitPad BIOS function)
5A12h=Mouse              (two button mouse)
5A23h=NegCon             (steering twist/wheel/paddle)
5A31h=Konami Lightgun    (IRQ10-type)
5A41h=Digital Pad        (or analog pad/stick in digital mode; LED=Off)
5A53h=Analog Stick       (or analog pad in "flight mode"; LED=Green)
5A63h=Namco Lightgun     (Cinch-type)
5A73h=Analog Pad         (in normal analog mode; LED=Red)
5A80h=Multitap           (multiplayer adaptor) (when activated)
5AE3h=Jogcon             (steering dial)
5AF3h=Config Mode        (when in config mode; see rumble command 43h)
FFFFh=High-Z             (no controller connected, pins floating High-Z)
```

Controllers - Standard Digital/Analog Controllers



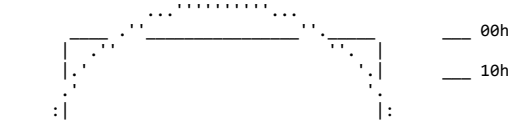
Standard Controllers

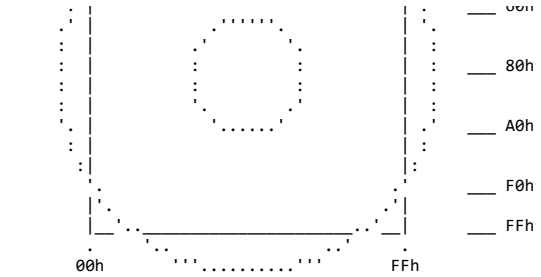
```
__Halfword 0 (Controller Info)_____________________________________________
0-15  Controller Info  (5A41h=digital, 5A73h=analog/pad, 5A53h=analog/stick)
__Halfword 1 (Digital Switches)____________________________________________
0   Select Button    (0=Pressed, 1=Released)
1   L3/Joy-button    (0=Pressed, 1=Released/None/Disabled) ;analog mode only
2   R3/Joy-button    (0=Pressed, 1=Released/None/Disabled) ;analog mode only
3   Start Button     (0=Pressed, 1=Released)
4   Joypad Up        (0=Pressed, 1=Released)
5   Joypad Right     (0=Pressed, 1=Released)
6   Joypad Down      (0=Pressed, 1=Released)
7   Joypad Left      (0=Pressed, 1=Released)
8   L2 Button        (0=Pressed, 1=Released) (Lower-left shoulder)
9   R2 Button        (0=Pressed, 1=Released) (Lower-right shoulder)
10  L1 Button        (0=Pressed, 1=Released) (Upper-left shoulder)
11  R1 Button        (0=Pressed, 1=Released) (Upper-right shoulder)
12  /\ Button        (0=Pressed, 1=Released) (Triangle, upper button)
13  () Button        (0=Pressed, 1=Released) (Circle, right button)
14  >< Button        (0=Pressed, 1=Released) (Cross, lower button)
15  [] Button        (0=Pressed, 1=Released) (Square, left button)
__Halfword 2 (Right joystick) (analog pad/stick in analog mode only)_________
0-7   adc0 RightJoyX (00h=Left, 80h=Center, FFh=Right)
8-15  adc1 RightJoyY (00h=Up,   80h=Center, FFh=Down)
__Halfword 3 (Left joystick) (analog pad/stick in analog mode only)__________
0-7   adc2 LeftJoyX  (00h=Left, 80h=Center, FFh=Right)
8-15  adc3 LeftJoyY  (00h=Up,   80h=Center, FFh=Down)
```

Analog Mode Note

On power-up, the controllers are in digital mode (with analog inputs disabled). Analog mode can be (de-)activated manually by pushing the Analog button. Alternately, analog mode can be (de-)activated by software via rumble configuration commands (though unknown if that's supported by older analog pads/sticks which did not include rumble motors).
The analog sticks are mechanically restricted to a "circular field of motion" (ie. the "min/max" values can be reached only in straight horizontal or vertical directions; not in diagonal directions).

Analog Joypad Range






```
Big Circle   --> Mechanically possible field of motion
Square Area  --> Digitally visible 8bit field of motion
Small Circle --> Resting position when releasing the joystick
```

Example min/center/max values for two different pads at 16'C room temperature:

```
gray analog joy1:  Min=(0E,0E), Mid: (6C..8A,75..79), Max=(ED,ED) at 16'C
white analog joy1: Min=(11,11), Mid: (8A..9F,70..96), Max=(FD,FD) at 16'C
```

Values may vary for other pads and/or different temperatures.

Dual Analog Pad in LED=Green Mode

Basically same as normal analog LED=Red mode, with following differences:

```
ID is 5A53h (identifying itself as analog stick) (rather than analog pad)
Left/right joy-buttons disabled (as for real analog stick, bits are always 1)
Some buttons are re-arranged: bit9=L1 bit10=[] bit11=/\ bit12=R1 bit15=R2
```

Concerning the button names, the real analog-stick does NOT have re-arranged buttons (eg. it's L1 button is in bit10), however, concerning the button locations, the analog stick's buttons are arranged completely differently as on analog pads (so it might be rather uncomfortable to play analog stick games on analog pads in LED=Red mode; the LED=Green mode is intended to solve that problem).

Might be useful for a few analog-stick games like MechWarrior 2, Ace Combat 2, Descent Maximum, and Colony Wars. In most other cases the feature is rather confusing (that's probably why the LED=Green mode wasn't implemented on the Dual Shock).

See also

Pinouts - Component List and Chipset Pin-Outs for Digital Joypad
Pinouts - Component List and Chipset Pin-Outs for Analog Joypad

Controllers - Mouse

Sony Mouse Controller

```
__Halfword 0 (Controller Info)_________________
0-15  Controller Info  (5A12h=Mouse)
__Halfword 1 (Mouse Buttons)___________________
0-7   Not used         (All bits always 1)
8-9   Unknown          (Seems to be always 0) (maybe SNES-style sensitivity?)
10    Right Button     (0=Pressed, 1=Released)
11    Left Button      (0=Pressed, 1=Released)
12-15 Not used         (All bits always 1)
__Halfword 2 (Mouse Motion Sensors)____________
0-7   Horizontal Motion (-80h..+7Fh = Left..Right) (00h=No motion)
8-15  Vertical Motion   (-80h..+7Fh = Up..Down)    (00h=No motion)
```

Sony Mouse Hardware Bug on Power-On

On Power-on (or when newly connecting it), the Sony mouse does draw /ACK to LOW on power-on, and does then hold /ACK stuck in the LOW position.

For reference: Normal controllers and memory cards set /ACK=LOW only for around 100 clk cycles, and only after having received a byte from the console.

The /ACK pin is shared for both controllers and both memory cards, so the stuck /ACK is also "blocking" all other connected controllers/cards. To release the stuck /ACK signal: Send a command (at least one 01h byte) to both controller slots.

Sony Mouse Compatible Games

```
3D Lemmings
Alien Resurrection
Area 51
Ark of Time
Atari Anniversary Edition
Atlantis: The Lost Tales
Breakout: Off the Wall Fun
Broken Sword: The Shadow of the Templars
Broken Sword II: The Smoking Mirror
Clock Tower: The First Fear
Clock Tower II: The Struggle Within
Command & Conquer: Red Alert
Command & Conquer: Red Alert - Retaliation
Constructor (Europe)
Die Hard Trilogy
Die Hard Trilogy 2: Viva Las Vegas
Discworld
Discworld II: Missing Presumed...!?
Discworld Noir
Dune 2000
Final Doom
Galaxian 3
Ghoul Panic
Klaymen Klaymen: Neverhood no Nazon (Japan)
Lemmings and Oh No! More Lemmings
Monopoly
Music 2000
Myst
Neorude (Japan)
Perfect Assassin
Policenauts (Japan)
Puchi Carat
Quake II
Railroad Tycoon II
Rescue Shot
Risk
Riven: The Sequel to Myst
```

```
RPG Maker
Sentinel Returns
SimCity 2000
Syndicate Wars
Tempest 2000 (Tempest X3)
Theme Aquarium (Japan)
Transport Tycoon
Warhammer: Dark Omen
Warzone 2100
X-COM: Enemy Unknown
X-COM: Terror from the Deep
Z
```

Note: There are probably many more mouse compatible games.
Plus: Dracula - The Resurrection

Sony Mouse Component List
PCB "TD-T41V/\, MITSUMI"
Component Side:

```
1x 3pin   4.00MHz "[M]4000A, 85 2"
2x 2pin   button (left/right)
1x 8pin   connector (to cable with shield and 7 wires)
1x 3pin   "811, T994I"
2x 3pin   photo transistor (black)  ;\or so, no idea which one is
2x 2pin   photo diode (transparent) ;/sender and which is sensor
1x 2pin   electrolyt capacitor 16V, 10uF
```

Solder/SMD Side:

```
1x 32pin  "(M), SC442116, FB G22K, JSAA815B"
1x 14pin  "BA10339F, 817 L67" (Quad Comparator)
2x 3pin   "LC" (amplifier for photo diodes)
1x 3pin   "24-" (looks like a dual-diode or so)
plus many SMD resistors/capacitors
```

Cable:

```
PSX.Controller.Pin1 JOYDAT ---- brown  -- Mouse.Pin4
PSX.Controller.Pin2 JOYCMD ---- red    -- Mouse.Pin3
PSX.Controller.Pin3 +7.5V  ---- N/A
PSX.Controller.Pin4 GND    ---- orange -- Mouse.Pin7 GND (G)
PSX.Controller.Pin5 +3.5V  ---- yellow -- Mouse.Pin1
PSX.Controller.Pin6 /JOYn  ---- green  -- Mouse.Pin5
PSX.Controller.Pin7 JOYCLK ---- blue   -- Mouse.Pin2
PSX.Controller.Pin8 /IRQ10 ---- N/A
PSX.Controller.Pin9 /ACK   ---- purple -- Mouse.Pin6
PSX.Controller.Shield --------- shield -- Mouse.Pin8 GND (SHIELD)
```

RS232 Mice
Below is some info on RS232 serial mice. That info isn't directly PSX related as the PSX normally doesn't support those mice.
With some efforts, one upgrade the PSX SIO port to support RS232 voltages, and with such a modded console one could use RS232 mice (in case one wants to do that).
The nocash PSX bios can map a RS232 mouse to a spare controller slot (thereby simulating a Sony mouse), that trick may work with various PSX games.

Standard Serial Mouse
A serial mouse should be read at 1200 bauds, 7 data bits, no parity, 1 stop bit (7N1) with DTR and RTS on. For best compatibility, the mouse should output 2 stop bits (so it could be alternately also read as 7N2 or 8N1). When the mouse gets moved, or when a button gets pressed/released, the mouse sends 3 or 4 characters:

```
__First Character_____________________
6    First Character Flag (1)
5    Left Button  (1=Pressed)
4    Right Button (1=Pressed)
2-3  Upper 2bit of Vertical Motion
0-1  Upper 2bit of Horizontal Motion
__Second Character____________________
6    Non-first Character Flag (0)
5-0  Lower 6bit of Horizontal Motion
__Third Character_____________________
6    Non-first Character Flag (0)
5-0  Lower 6bit of Vertical Motion
__Fourth Character (if any)___________
6    Non-first Character Flag (0)
5    Middle Button (1=Pressed)
4    Unused ???
3-0  Wheel  ???
```

Additionally, the mouse outputs a detection character (when switching RTS (or DTR?) off and on:

```
"M" = Two-Button Mouse (aka "Microsoft" mouse)
"3" = Three-Button Mouse (aka "Logitech" mouse)
"Z" = Mouse-Wheel
```

Normally, the detection response consist of a single character (usually "M"), though some mice have the "M" followed by 11 additional characters of garbage or version information (these extra characters have bit6=0, so after detection, one should ignore all characters until receiving the first data character with bit6=1).

Mouse Systems Serial Mouse
Accessed at 1200 bauds, just like standard serial mouse, but with 8N1 instead 7N1, and with different data bytes.

```
__First Byte_________________________
7-3  First Byte Code (10000b)
2    Left? Button   (0=Pressed)
1    Middle? Button (0=Pressed)
0    Right? Button  (0=Pressed)
__Second Byte________________________
7-0  Horizontal Motion (X1)
__Third Byte_________________________
7-0  Vertical Motion   (Y1)
__Fourth Byte________________________
7-0  Horizontal Motion (X2)
__Fifth Byte_________________________
7-0  Vertical Motion   (Y2)
```

The strange duplicated 8bit motion values are usually simply added together, ie. X=X1+X2 and Y=Y1+Y2, producing 9bit motion values.

Notes
The Sony Mouse connects directly to the PSX controller port. Alternately serial RS232 mice can be connected to the SIO port (with voltage conversion adaptor) (most or all commercial games don't support SIO mice, nor does the original BIOS do so, however, the nocash BIOS maps SIO mice to unused controller slots, so they can be used even with commercial games; if the game uses BIOS functions to read controller data).
Serial Mice (and maybe also the Sony mouse) do return raw mickeys, so effects like double speed threshold must (should) be implemented by software. Mice are

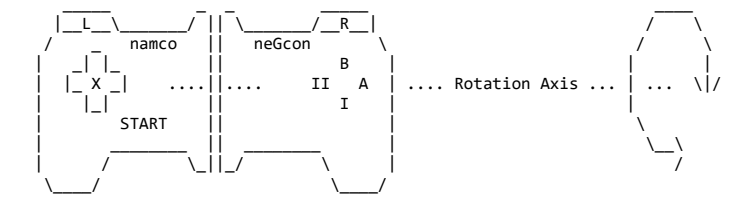
rather rarely used by PSX games. The game "Perfect Assassin" includes ultra-crude mouse support, apparently without threshold, and without properly matching the cursor range to the screen resolution.

Controllers - Racing Controllers

neGcon Racing Controller (Twist) (NPC-101/SLPH-00001/SLEH-0003)

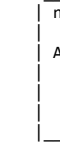
```
__Halfword 0 (Controller Info)_____________________________________________
0-15  Controller Info  (5A23h=neGcon)
__Halfword 1 (Digital Switches)____________________________________________
0-2   Not used       (always 1)       (would be Select, L3, R3 on other pads)
3     Start Button   (0=Pressed, 1=Released)
4     Joypad Up      (0=Pressed, 1=Released)
5     Joypad Right   (0=Pressed, 1=Released)
6     Joypad Down    (0=Pressed, 1=Released)
7     Joypad Left    (0=Pressed, 1=Released)
8-10  Not used       (always 1)       (would be L2, R2, L1 on other pads)
11    R Button       (0=Pressed, 1=Released) (would be R1 on other pads)
12    B Button       (0=Pressed, 1=Released) (would be /\ on other pads)
13    A Button       (0=Pressed, 1=Released) (would be () on other pads)
14-15 Not used       (always 1)             (would be ><, [] on other pads)
__Halfword 2 (Right joystick) (analog pad/stick in analog mode only)_________
0-7   Steering Axis    (00h=Left, 80h=Center, FFh=Right) (or vice-versa?)
8-15  Analog I button  (00h=Out ... FFh=In)  (Out=released, in=pressed?)
__Halfword 3 (Left joystick) (analog pad/stick in analog mode only)__________
0-7   Analog II button (00h=Out ... FFh=In)  (Out=released, in=pressed?)
8-15  Analog L button  (00h=Out ... FFh=In)  (Out=released, in=pressed?)
```

The Twist controller works like a paddle or steering wheel, but doesn't have a wheel or knob, instead, it can be twisted: To move into one direction (=maybe right?), turn its right end away from you (or its left end towards you). For the opposite direction (=maybe left?), do it vice-versa.



Namco Volume Controller (a paddle with two buttons) (SLPH-00015)

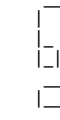
This is a cut-down variant of the neGcon, just a featureless small box. It does have the same ID value as neGcon (ID=5A23h), but, it excludes most digital, and all analog buttons.




```
Halfword 1 (digital buttons):
Bit3  Button A (0=Pressed) (aka neGcon Start button)
Bit13 Button B (0=Pressed) (aka neGcon A button aka () button)
Other bits     (not used, always 1)
Halfword 2 and 3 (analog inputs):
Steering Axis  (00h..FFh)  (as for neGcon)
Analog I,II,L button values (not used, always 00h)
```

SANKYO N.ASUKA aka Nasca Pachinco Handle (SLPH-00007)

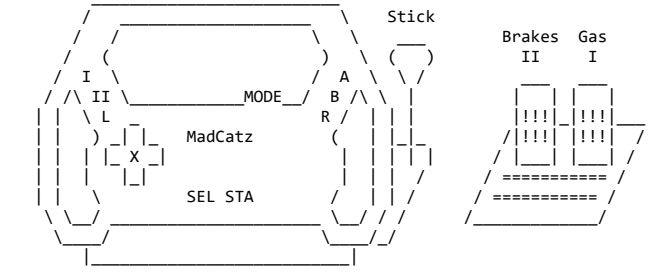
Another cut-down variant of the neGcon (with ID=5A23h, too). But, this one seems to have only one button. Unlike Namco's volume controller it doesn't look featureless. It looks pretty much as shown in the ascii-arts image below. Seems to be supported by several irem titles. No idea what exactly it is used for, it's probably not a sewing machine controller, nor an electronic amboss.



```
Halfword 1 (digital buttons):
Bit12 Button   (0=Pressed) (aka neGcon B button aka /\ button)
Other bits     (not used, always 1)
Halfword 2 and 3 (analog inputs):
Steering Axis  (00h..FFh)  (as for neGcon)
Analog I,II,L button values (not used, always 00h)
```

Mad Catz Steering Wheel (SLEH-0006)

A neGcon compatible controller. The Twist-feature has been replaced by a steering wheel (can be turned by 270 degrees), and the analog I and II buttons by foot pedals. The analog L button has been replaced by a digital button (ie. in neGcon mode, the last byte of the controller data can be only either 00h or FFh). When not using the pedals, the I/II buttons on the wheel can be used (like L button, they aren't analog though).



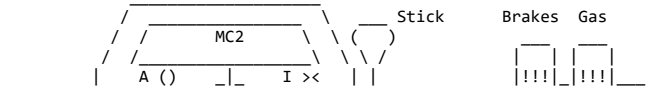
Unlike the neGon, the controller has Select, >< and [] buttons, and a second set of L/R buttons (at the rear-side of the wheel) (no idea if L1/R1 or L2/R2 are at front?). Aside from the neGcon mode, the controller can be also switched to Digital mode (see below for button chart).

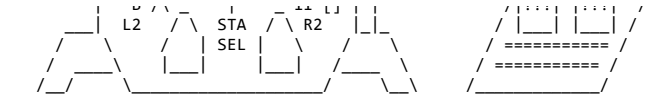
MadCatz Dual Force Racing Wheel

Same as above, but with a new Analog mode (additionally to Digital and neGcon modes). The new mode is for racing games that support only Analog Joypads (instead of neGcon). Additionally it supports vibration feedback.

MadCatz MC2 Vibration compatible Racing Wheel and Pedals

Same as above, but with a redesigned wheel with rearranged buttons, the digital pad moved to the center of the wheel, the L/R buttons at the rear-side of the wheel have been replaced by 2-way butterfly buttons ("pull towards user" acts as normal, the new "push away from user" function acts as L3/R3).





MadCatz Button Chart

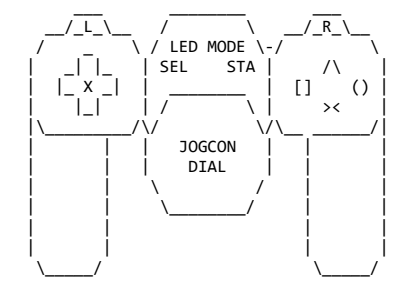
```
Mode     Buttons..................... Gas Brake Stick Wheel
Digital  >< [] () /\ L1 R1 L2 R2 L1 R1 ><  ()    L1/R1 lt/rt
Analog   >< [] () /\ L1 R1 L2 R2 L3 R3 UP  DN    L1/R1 LT/RT
Negcon   I  II A  B  L  R  L  R  L  R  I   II    up/dn Twist
```

Whereas, lt/rt/up/dn=Digital Pad, UP/DN=Left Analog Pad Up/Down, LT/RT=Right Analog Pad Left/Right. Analog mode is supported only by the Dual Force and MC2 versions, L3/R3 only by the MC2 version.

Namco Jogcon (NPC-105/SLEH-0020/SLPH-00126/SLUH-00059)

```
__Halfword 0 (Controller Info)_____________________
 0-15  Controller Info  (5AE3h=Jogcon in Jogcon mode) (ie. not Digital mode)
halfword1: buttons: same as digital pad
halfword2:
  0    unknown (uh, this isn't LSB of rotation?)
  1-15 dial rotation (signed offset since last read?) (or absolute position?)
halfword3:
  0    flag: dial was turned left  (0=no, 1=yes)
  1    flag: dial was turned right (0=no, 1=yes)
  2-15 unknown
```

Rotations of the dial are recognized by an optical sensor (so, unlike potentiometers, the dial can be freely rotated; by more than 360 degrees). The dial is also connected to a small motor, giving it a real force-feedback effect (unlike all other PSX controllers which merely have vibration feedback). Although that's great, the mechanics are reportedly rather cheap and using the controller doesn't feel too comfortable. The Jogcon is used only by Ridge Racer 4 for PS1 (and Ridge Racer 5 for PS2), and Breakout - Off the Wall Fun.
The Mode button probably allows to switch between Jogcon mode and Digital Pad mode (similar to the Analog button on other pads), not sure if the mode can be also changed by software via configuration commands...? Unknown how the motor is controlled; probably somewhat similar to vibration motors, ie. by the M1 and/or M2 bytes, but there must be also a way to select clockwise and anticlockwise direction)...? The controller does reportedly support config command 4Dh (same as analog rumble).



Controllers - Lightguns

There are two different types of PSX lightguns (which are incompatible with each other).

Namco Lightgun (GunCon)
Namco's Cinch-based lightguns are extracting Vsync/Hsync timings from the video signal (via a cinch adaptor) (so they are working completely independed of software timings).
Controllers - Lightguns - Namco (GunCon)

Konami Lightgun (IRQ10)
Konami's IRQ10-based lightguns are using the lightgun input on the controller slot (which requires IRQ10/timings being properly handled at software side).
Controllers - Lightguns - Konami Justifier/Hyperblaster (IRQ10)
The IRQ10-method is reportedly less accurate (although that may be just due to bugs at software side).

Third-Party Lightguns
There are also a lot of unlicensed lightguns which are either IRQ10-based, or Cinch-based, or do support both.
For example, the Blaze Scorpion supports both IRQ10 and Cinch, and it does additionally have a rumble/vibration function; though unknown how that rumble feature is accessed, and which games are supporting it).

Lightgun Games
Controllers - Lightguns - PSX Lightgun Games

Compatibilty Notes (IRQ10 vs Cinch, PAL vs NTSC, Calibration)
Some lightguns are reportedly working only with PAL or only with NTSC games (unknown which guns, and unknown what is causing problems; the IRQ10 method should be quite hardware independed, the GunCon variant, too, although theoretically, some GunCon guns might have problems to extract Vsync/Hsync from either PAL or NTSC composite signals).
Lightguns from different manufacturers are reportedly returning slightly different values, so it would be recommended to include a calibration function in the game, using at least one calibration point (that would also resolve different X/Y offsets caused by modifying GP1 display control registers).
Lightguns are needing to sense light from the cathode ray beam; as such they won't work on regions of the screen that contain too dark/black graphics.

Controllers - Lightguns - Namco (GunCon)

GunCon Cinch-based Lightguns (Namco)

```
__Halfword 0 (Controller Info)_____________________
0-15  Controller Info  (5A63h=Namco Lightgun; GunCon/Cinch Type)
__Halfword 1 (Buttons)_____________________________
0-2   Not used              (All bits always 1)
3     Button A (Left Side)  (0=Pressed, 1=Released) ;aka Joypad Start
4-12  Not used              (All bits always 1)
13    Trigger Button        (0=Pressed, 1=Released) ;aka Joypad O-Button
14    Button B (Right Side) (0=Pressed, 1=Released) ;aka Joypad X-Button
15    Not used              (All bits always 1)
__Halfword 2 (X)___________________________________
```

```
0-15  8MHz clks since HSYNC (01h=Error, or 04Dh..1CDh)
__Halfword 3 (Y)_____________________________
0-15  Scanlines since VSYNC (05h/0Ah=Error, PAL=20h..127h, NTSC=19h..F8h)
```

Caution: The gun should be read only shortly after begin of VBLANK.

Error/Busy Codes

Coordinates X=0001h, Y=0005h indicates "unexpected light":

```
ERROR: Sensed light during VSYNC (eg. from a Bulb or Sunlight).
```

Coordinates X=0001h, Y=000Ah indicates "no light", this can mean either:

```
ERROR: no light sensed at all (not aimed at screen, or screen too dark).
BUSY: no light sensed yet (when trying to read gun during rendering).
```

To avoid the BUSY error, one should read the gun shortly after begin of VBLANK (ie. AFTER rendering, but still BEFORE vsync). Doing that isn't as simple as one might think:

On a NTSC console, time between VBLANK and VSYNC is around 30000 cpu clks, reading the lightgun (or analog joypads) takes around 15000 cpu clks. So, reading two controllers within that timeframe may be problematic (and reading up to eight controllers via multitaps would be absolutely impossible). As a workaround, one may arrange the read-order to read lightguns at VBLANK (and joypads at later time). If more than one lightgun is connected, then one may need to restrict reading to only one (or maybe: max two) guns per frame.

Minimum Brightness

Below are some average minimum brightness values, the gun may be unable to return position data near/below that limits (especially coordinates close to left screen border are most fragile). The exact limits may vary from gun to gun, and will also depend on the TV Set's brightness setting.

```
666666h Minimum Gray
770000h Minimum Blue
007700h Minimum Green
000099h Minimum Red
```

The gun does also work with mixed colors (eg. white bold text on black background works without errors, but the returned coordinates are a bit "jumpy" in that case; returning the position of the closest white pixels).

BUG: On a plain RED screen, aiming at Y>=00F0h, the gun is randomly returning either Y, or Y-80h (that error occurs in about every 2nd frame, ie. at 50% chance). It's strange... no idea what is causing that effect.

Coordinates

The coordinates are updated in all frames (as opposed to some lightguns which do update them only when pulling the trigger).

The absolute min/max coordinates may vary from TV set to TV set (some may show a few more pixels than others). The relation of the gun's Screen Coodinates to VRAM Coordinates does (obviously) depend on where the VRAM is located on the screen; ie. on the game's GP1(06h) and GP1(07h) settings.

Vertical coordinates are counted in scanlines (ie. equal to pixels). Horizontal coordinates are counted in 8MHz units (which would equal a resolution of 385 pixels; which can be, for example, converted to 320 pixel resolution as X=X*320/385).

Misinformation (from bugged homebrew source code)

```
__Halfword 2 (X)_____________________________
0-7   X-Coordinate  (actual: see X-Offset)            ;\with unspecified
8-15  X-Offset      (00h: X=X-80, Nonzero: X=X-80+220) ;/dotclock?
__Halfword 3 (Y)_____________________________
0-7   Y-Coordinate  (actual: Y=Y-25) (but then, max is only 230, not 263 ?)
8-15  Pad ID        (uh, what id?) (reportedly too dark/bright error flag?)
```

Namco Lightgun Drawing

```
         _-_______________________--_
----->  |    namco            \\\\   \    Namco G-Con 45 (light gray) (cinch)
sensor   |............ .. .....\\\\...|_
        |_ :          :..   _____      _\
          |  O        :__../  )))|    (
           \__________/  |_\____/|     \
             :               :   |      |
             :               :   |      |  NPC-103
       A-Button (Left)   Trigger |      |  SLPH-00034/SLEH-0007/SLUH-00035
       B-Button (Right)          |______|
```

Konami Lightgun "NPC-103, (C) 1996 NAMCO LTD." Component List

PCB "DNP-0500A, NPC10300, namco, CMK-P3X"

```
U1  44pin NAMCO103P, 1611U1263, JAPAN 9847EAI, D0489AAF
U2   8pin 7071, 8C19
XTAL 2pin CSA 8.00WT
PS1  3pin Light sensor with metal shielding
J1   9pin Connector for 9pin cable to PSX controller and GunCon plugs
plus resistors and capacitors, and A1,A2,B1,B2,T1,T2 wires to buttons
```

PCB "DN-P-0501"

```
DIP Button (with black T1,T2 wires) (trigger)
```

PCB "DN-P-0502"

```
Button A (with red A1,A2 wires) (left side)
Button B (with white B1,B2 wires) (right side)
```

Other Components

```
Lens
```

Cable Pinouts

```
J1.Pin1 green  PSX.Controller.Pin5 +3.5V
J1.Pin2 brown  PSX.Controller.Pin4 GND
J1.Pin3 black  PSX.Controller.Pin9 /ACK/IRQ7
J1.Pin4 red    PSX.Controller.Pin6 /JOYn
J1.Pin5 yellow PSX.Controller.Pin1 JOYDAT
J1.Pin6 orange PSX.Controller.Pin2 JOYCMD
J1.Pin7 blue   PSX.Controller.Pin7 JOYCLK
J1.Pin8 gray   GunCon shield (GND)
J1.Pin9 white  GunCon composite video
N/A            PSX.Controller.Pin3 +7.5V
N/A            PSX.Controller.Pin8 /IRQ10
N/A            PSX.Controller Shield
```

Controllers - Lightguns - Konami Justifier/Hyperblaster (IRQ10)

Overall IRQ10-Based Lightgun Access

```
Send  01h 42h 00h x0h 00h
Reply HiZ 31h 5Ah buttons
```

The purpose of the "x0h" byte is probably to enable IRQ10 (00h=off, 10h=on), this would allow to access more than one lightgun (with only one per frame having the IRQ enabled).

Standard IRQ10-based Lightguns (Konami)
The Controller Data simply consists of the ID and buttons states:
```
  __Halfword 0 (Controller Info)_____________________
  0-15  Controller Info  (5A31h=Konami Lightgun; Timer/IRQ10 type)
  __Halfword 1 (Buttons)
  0-2   Not used                 (All bits always 1)
  3     Start Button (Left Side) (0=Pressed, 1=Released)  ;aka Joypad Start
  4-13  Not used                 (All bits always 1)
  14    Back Button  (Rear End)  (0=Pressed, 1=Released)  ;aka Joypad X-Button
  15    Trigger Button           (0=Pressed, 1=Released)  ;aka Joypad []-Button
```
The coordinates aren't part of the controller data, instead they must be read from Timer 0 and 1 upon receiving IRQ10 (see IRQ10 Notes below).

Konami Lightgun Drawing
```
   __                 ______ _
  _|__\_____________/   ___ \ \   Konami Justifier/Hyperblaster (light green)
 |  _____________ __ /   \ \ \
 |__|    _ _ _ _    |==|    O| \O\ .... Back Button (Rear End)
    |__:_:_:_:_:__  |___\__ /  ( (
                  |_|  ) \  :   \ \
     Trigger ......  \___/| :...|.|.... Start Button (Left Side)
                          |     | |
                          |     | | SLPH-00014/SLEH-0005/SLUH-00017
                         /     _|_|
                         \___--
```

Konami IRQ10 Notes
The PSX does have a lightgun input (Pin 8 of the controller), but, Sony did apparently "forget" to latch the current cathode ray beam coordinates by hardware when sensing the lightgun signal (quite strange, since that'd be a simple, inexpensive, and very obvious feature for a gaming console).
Instead, the lightgun signal triggers IRQ10, and the interrupt handler is intended to "latch" the coordinates by software (by reading Timer 0 and 1 values, which must configured to be synchronized with the GPU).
That method requires IRQ handling to be properly implemented in software (basically, IRQs should not be disabled for longer periods, and DMA transfers should not block the bus for longer periods). In practice, most programmers probably don't realize how to do that, to the worst, Sony seems to have delivered a slightly bugged library (libgun) to developers.
For details on Timers, see:
Timers
In some consoles, IRQ10 seems to be routed through a Secondary IRQ Controller, see:
EXP2 DTL-H2000 I/O Ports

IRQ10 Priority
For processing IRQ10 as soon as possible, it should be assigned higher priority than all other IRQs (ie. when using the SysEnqIntRP BIOS function, it should be the first/newest element in priority chain 0). The libgun stuff assigns an even higher priority by patching the BIOS exception handler, causing IRQ10 to be processed shortly before processing the priority chains (the resulting IRQ priority isn't actually higher as when using 1st element of chain 0; the main difference is that it skips some time consuming code which pushes registers R4..R30). For details on that patch, see:
BIOS Patches
Even if IRQ10 has highest priority, execution of (older) other IRQs may cause a new IRQ10 to be executed delayed (because IRQs are disabled during IRQ handling), to avoid that problem: Best don't enable any other IRQs except IRQ0 and IRQ10, or, if you need other IRQs, best have them enabled only during Vblank (there are no scanlines drawn during vblank, so IRQ10 should never trigger during that period). DMAs might also slow down IRQ execution, so best use them only during Vblank, too.

IRQ10 Timer Reading
To read the current timer values the IRQ10 handler would be required to be called <immediately> after receiving the IRQ10 signal, which is more or less impossible; if the main program is trying to read a mul/div/gte result while the mul/div/gte operation is still busy may stop the CPU for some dozens of clock cycles, and active DMA transfers or cache hits and misses in the IRQ handler may cause different timings, moreover, timings may become completely different if IRQs are disabled (eg. while another IRQ is processed).
However, IRQ10 does also get triggered in the next some scanlines, so the first IRQ10 is used only as a notification that the CPU should watch out for further IRQ10's. Ie. the IRQ10 handler should disable all DMAs, acknowledge IRQ10, and then enter a waitloop that waits for the IRQ10 bit in I_STAT to become set again (or abort if a timeout occurs) and then read the timers, reportedly like so:
```
  IF NTSC then X=(Timer0-140)*0.198166, Y=Timer1
  IF PAL  then X=(Timer0-140)*0.196358, Y=Timer1
```
No idea why PAL/NTSC should use different factors, that factors are looking quite silly/bugged, theoretically, the pixel-to-clock ratio should be the exactly same for PAL and NTSC...?
Mind that reading Timer values in Dotclock/Hblank mode is unstable, for Timer1 this can be fixed by the read-retry method, for Timer0 this could be done too, but one would need to subtract the retry-time to get a correct coordinate; alternately Timer0 can run at system clock (which doesn't require read-retry), but it must be then converted to video clock (mul 11, div 7), and then from video clock to dot clock (eg. div 8 for 320-pixel mode).
Above can be repeated for the next some scanlines (allowing to take the medium values as result, and/or to eliminate faulty values which are much bigger or smaller than the other values). Once when you have collected enough values, disable IRQ10, so it won't trigger on further scanlines within the current frame.

IRQ10 Bugs
BUG: The "libgun" library doesn't acknowledge the old IRQ10 <immediately> before waiting for a new IRQ10, so the timer values after sensing the new IRQ10 are somewhat random (especially for the first processed scanline) (the library allows to read further IRQ10's in further scanlines, which return more stable results).
No idea how many times IRQ10 gets typically repeated? Sporting Clays allocates a buffer for up to 20 scanlines (which would cause pretty much of a slowdown since the CPU is just waiting during that period) (nethertheless, the game uses only the first timer values, ie. the bugged libgun-random values).

Unknown if/how two-player games (with 2 lightguns) are working with the IRQ10 method... if IRQ10 is generated ONLY after pressing the trigger button, then it may work, unless both players have Trigger pressed at the same time... and, maybe one can enable/disable the lightguns by whatever commmand being sent to the controller (presumably that "x0h" byte, see above), so that gun 1 generates IRQ10 only in each second frame, and gun 2 only in each other frame...?

Controllers - Lightguns - PSX Lightgun Games

PSX Lightgun Games
Some games are working only with IRQ10 or only with Cinch, some games support both methods:
```
  Area 51 (Mesa Logic/Midway) (IRQ10)
  Crypt Killer (Konami) (IRQ10)
  Die Hard Trilogy 1: (Probe Entertainment) (IRQ10)
  Die Hard Trilogy 2: Viva Las Vegas (n-Space) (IRQ10/Cinch)
  Elemental Gearbolt (Working Designs) (IRQ10/Cinch)
  Extreme Ghostbusters: Ultimate Invasion (LSP) (Cinch)
  Galaxian 3 (Cinch)
  Ghoul Panic (Namco) (Cinch)
```

```
Judge Dredd (Gremlin) (Cinch)
Lethal Enforcers 1-2 (Konami) (IRQ10)
Maximum Force (Midway) (IRQ10/Cinch)
Mighty Hits Special (Altron) (EU/JPN) (Cinch)
Moorhuhn series (Phenomedia) (Cinch)
Point Blank 1-3 (Namco) (Cinch)
Project Horned Owl (Sony) (IRQ10)
Rescue Shot (Namco) (Cinch)
Resident Evil: Gun Survivor (Capcom) (JPN/PAL versions) (Cinch)
Silent Hill (IRQ10) ("used for an easter egg")
Simple 1500 Series Vol.024 - The Gun Shooting (unknown type)
Simple 1500 Series Vol.063 - The Gun Shooting 2 (unknown type)
Snatcher (IRQ10)
Sporting Clays (Charles Doty) (homebrew with buggy source code) (IRQ10/Cinch)
Star Wars: Rebel Assault II (IRQ10)
Time Crisis, and Time Crisis 2: Project Titan (Namco) (Cinch)
```

Note: The RPG game Dragon Quest Monsters does also contain IRQ10 lightgun code (though unknown if/when/where the game does use that code).

Controllers - Rumble Configuration

Rumble (aka "Vibration Function") is basically controlled by two previously unused bytes of the standard controller Read command. However, before that bytes can be used, one must unlock the rumble feature, for that purpose Sony has invented a "slightly" overcomplicated protocol with not less than 16 new commands (the rumble relevant commands are 43h and 4Dh, also, command 44h may be useful for activating analog inputs by software).

Normal Mode

```
42h "B" Read Buttons (and analog inputs when in analog mode)
43h "C" Enter/Exit Configuration Mode (stay normal, or enter)
```

Configuration Mode

```
40h "@" Unknown (response HiZ F3h 5Ah 6x00h)
41h "A" Unknown (response HiZ F3h 5Ah 6x00h)
42h "B" Read Buttons AND analog inputs (even when in digital mode)
43h "C" Enter/Exit Configuration Mode (stay config, or exit)
44h "D" Set LED State (analog mode on/off)
45h "E" Get LED State (and whatever other status/version values)
46h "F" Get Variable Response A (depending on incoming bit)
47h "G" Get whatever values (response HiZ F3h 5Ah 00h 00h 02h 00h 01h 00h)
48h "H" Unknown (response HiZ F3h 5Ah 00h 00h 00h 00h 01h 00h)
49h "I" Unknown (response HiZ F3h 5Ah 6x00h)
4Ah "J" Unknown (response HiZ F3h 5Ah 6x00h)
4Bh "K" Unknown (response HiZ F3h 5Ah 6x00h)
4Ch "L" Get Variable Response B (depending on incoming bit)
4Dh "M" Unlock Rumble (and select response length)
4Eh "N" Unknown (response HiZ F3h 5Ah 6x00h)
4Fh "O" Unknown (response HiZ F3h 5Ah 6x00h)
```

Normal Mode - Command 42h "B" - Read Buttons (and analog inputs when enabled)

```
Send  01h 42h 00h xx  yy  (00h 00h 00h 00h)
Reply HiZ id  5Ah buttons ( analog-inputs )
```

The normal read command, see Standard Controller chapter for details on buttons and analog inputs. The xx/yy bytes have effect only if rumble is unlocked; use Command 43h to enter config mode, and Command 4Dh to unlock rumble. Command 4Dh has billions of combinations, among others allowing to unlock only one of the two motors, and to exchange the xx/yy bytes, however, with the default values, xx/yy are assigned like so:

```
yy.bit0-7 ---> Left/Large Motor M1 (analog slow/fast) (00h=stop, FFh=fastest)
xx.bit0   ---> Right/small Motor M2 (digital on/off)  (0=off, 1=on)
```

The Left/Large motor starts spinning at circa min=50h..60h, and, once when started keeps spinning downto circa min=38h. The exact motor start boundary depends on the current position of the weight (if it's at the "falling" side, then gravity helps starting), and also depends on external movements (eg. it helps if the user or the other rumble motor is shaking the controller), and may also vary from controller to controller, and may also depend on the room temperature, dirty or worn-out mechanics, etc.

Normal Mode - Command 43h "C" - Enter/Exit Configuration Mode

```
Send  01h 43h 00h xx  00h (zero padded...)
Reply HiZ id  5Ah buttons (analog inputs...)
```

When issuing command 43h from inside normal mode, the response is same as for command 42h (button data) (and analog inputs when in analog mode) (but without M1 and M2 parameters). While in config mode, the ID bytes are always "F3h 5Ah" (instead of the normal analog/digital ID bytes).

```
xx=00h Stay in Normal mode
xx=01h Enter Configuration mode
```

Caution: Additionally to activating configuration commands, entering config mode does also activate a Watchdog Timer which does reset the controller if there's been no communication for about 1 second or so. The watchdog timer remains active even when returning to normal mode via Exit Config command. The reset does disable and lock rumble motors, and switches the controller to Digital Mode (with LED=off, and analog inputs disabled). To prevent this, be sure to keep issuing joypad reads even when not needing user input (eg. when loading data from CDROM).
Caution 2: A similar reset occurs when the user pushes the Analog button; this is causing rumble motors to be stopped and locked, and of course, the analog/digital state gets changed.
Caution 3: If config commands were used, and the user does then push the analog button, then the 5Ah-byte gets replaced by 00h (ie. responses change from "HiZ id 5Ah ..." to "HiZ id 00h ...").

Config Mode - Command 42h "B" - Read Buttons AND analog inputs

```
Send  01h 42h 00h M2  M1  00h 00h 00h 00h
Reply HiZ F3h 5Ah buttons  analog-inputs
```

Same as command 42h in normal mode, but with forced analog response (ie. analog inputs and L3/R3 buttons are returned even in Digital Mode).

Config Mode - Command 43h "C" - Enter/Exit Configuration Mode

```
Send  01h 43h 00h xx  00h 00h 00h 00h 00h
Reply HiZ F3h 5Ah 00h 00h 00h 00h 00h 00h
```

Equivalent to command 43h in normal mode, but returning 00h bytes rather than button data, can be used to return to normal mode.

```
xx=00h Enter Normal mode (Exit Configuration mode)
xx=01h Stay in Configuration mode
```

Back in normal mode, the rumble motors (if they were enabled) can be controlled with normal command 42h.

Config Mode - Command 44h "D" - Set LED State (analog mode on/off)

```
Send  01h 44h 00h val sel 00h 00h 00h 00h
Reply HiZ F3h 5Ah 00h 00h 00h 00h 00h 00h
```

If "sel=02h", then "val" is applied as new LED state (val=00h for LED off aka Digital mode, and val=01h for LED on/red aka analog mode).

Config Mode - Command 45h "E" - Get LED State (and whatever values)
```
Send  01h 45h 00h 00h 00h 00h 00h 00h 00h
Reply HiZ F3h 5Ah 01h 02h val 02h 01h 00h
```
Returns val=00h for LED off, and val=01h for LED on/red. The other values might indicate the number of rumble motors, analog inputs, or version information, or so.

Config Mode - Command 46h "F" - Get Variable Response A
```
Send  01h 46h 00h xx  00h 00h 00h 00h 00h
Reply Hiz F3h 5Ah 00h 00h yy  yy  yy  yy
```
Purpose unknown. Response varies: If xx=00h then yy=01h,02h,00h,0ah, else if xx=01h then yy=01h,01h,01h,14h.

Config Mode - Command 47h "G" - Get whatever values
```
Send  01h 47h 00h 00h 00h 00h 00h 00h 00h
Reply HiZ F3h 5Ah 00h 00h 02h 00h 01h 00h
```
Purpose unknown.

Config Mode - Command 4Ch "L" - Get Variable Response B
```
Send  01h 4Ch 00h xx  00h 00h 00h 00h 00h
Reply Hiz F3h 5Ah 00h 00h 00h yy  00h 00h
```
Purpose unknown. Response varies: If xx=00h then yy=04h, else if xx=01h then yy=07h.

Config Mode - Command 4Dh "M" - Unlock Rumble (and select response length)
```
Send  01h 4Dh 00h aa  bb  cc  dd  ee  ff
Reply Hiz F3h 5Ah <-----old values----->
```
This command does basically unlock the rumble motors (so they can be controlled with command 42h), this is usually done by setting aa..ff to 00h, 01h, FFh, FFh, FFh, FFh. There are billons of other combinations for aa..ff, some rules are:
```
when cc.bit1-7 are all zero --> transfer one extra halfword
when ee.bit1-7 are all zero --> transfer another extra halfword
when cc,dd,ee,ff are ALL nonzero --> unlock small motor
when aa=01h --> unlock large motor (and swap VAL1 and VAL2)
when bb=01h --> unlock large motor (default)
when cc.bit0=1 --> disable large motor (unless cc.bit7=1)
when ff.bit0=1 --> disable large motor (unless something)
```
The extra halfword(s) increase the transfer length by 1 or 2 halfwords (and accordingly, the digital mode ID changes from 41h to 42h or 43h), the controller returns 00h bytes for the extra halfwords, in analog mode, the ID and transfer length remains unchanged (since it always has extra halfwords for analog responses). So, the extra halfwords seem to be output to the controller (rather than response data), possibly intended to control additional rumble motors or to output data to other hardware features.

Config Mode - Command 48h "H" - Unknown (response HiZ F3h 5Ah 4x00h 01h 00h)
```
Send  01h 48h 00h 00h 00h 00h 00h 00h 00h
Reply HiZ F3h 5Ah 00h 00h 00h 00h 01h 00h
```
This command does return a bunch of 00h bytes, and one 01h byte. Purpose unknown. The command does not seem to be used by any games.

Config Mode - Command 40h "@" - Unknown (response HiZ F3h 5Ah 6x00h)
Config Mode - Command 41h "A" - Unknown (response HiZ F3h 5Ah 6x00h)
Config Mode - Command 49h "I" - Unknown (response HiZ F3h 5Ah 6x00h)
Config Mode - Command 4Ah "J" - Unknown (response HiZ F3h 5Ah 6x00h)
Config Mode - Command 4Bh "K" - Unknown (response HiZ F3h 5Ah 6x00h)
Config Mode - Command 4Eh "N" - Unknown (response HiZ F3h 5Ah 6x00h)
Config Mode - Command 4Fh "O" - Unknown (response HiZ F3h 5Ah 6x00h)
```
Send  01h 4xh 00h 00h 00h 00h 00h 00h 00h
Reply HiZ F3h 5Ah 00h 00h 00h 00h 00h 00h
```
These commands do return a bunch of 00h bytes. Purpose unknown. These commands do not seem to be used by any games.

Note
Above info is for a SCPH-110 controller (with 2 rumble motors; shipped with PSone consoles). The SCPH-1200 is probably accessed identically (since it has 2 motors too). However, the SCPH-1150 (with only 1 motor) is probably accessed somehow differently. The SCPH-1180 (without motors) obviously can't rumble, but it might support some config commands (eg. for activating analog mode).
Some Fishing Controllers do also somehow support rumble/vibration. And the "Blaze Scorpion" Lightgun does reportedly include rumble motor(s) too, but it's unknown how to access that function?
Rumble is a potentially annoying feature, so games that do support rumble should also include an option to disable it.

Controllers - Dance Mats

PSX Dance Mats are essentially normal joypads with uncommonly arranged buttons, the huge mats are meant to be put on the floor, so the user could step on them.

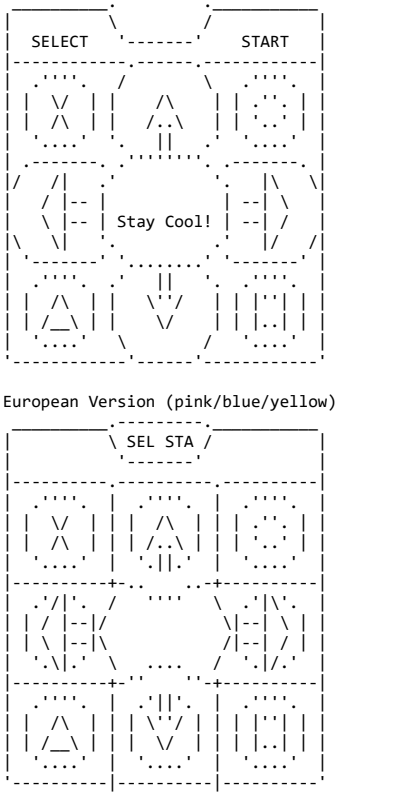
Dance Mat vs Joypad Compatibility
There are some differences to normal joypads: First of, the L1/L2/R1/R2 shoulder buttons are missing in most variants. And, the mats are allowing to push Left+Right and Up+Down at once, combinations that aren't mechanically possible on normal joypads (some dancing games do actually require those combinations, whilst some joypad games may get confused on them).

Dance Mat Unknown Things
Unknown if the mat was sold in japan, and if so, with which SLPH/SCPH number.
Unknown if the mat's middle field is also having a button assigned.
Unknown if the mat is having a special controller ID, or if there are other ways to detect mats (the mats are said to be compatible with skateboard games, so the mats are probably identifying themselves as normal digital joypad; assuming that those skateboard games haven't been specifically designed for mats).

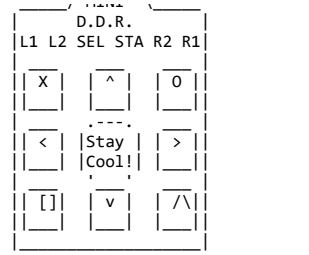
Dance Mat Games
```
D.D.R. Dance Dance Revolution 2nd Remix
(and maybe whatever further games)
```
The mats can be reportedly also used with whatever skateboard games.

Dance Mat Variants
There is the US version (DDR Dance Pad, SLUH-00071), and a slightly different European version (Official Dance Mat, SLEH-00023: shiny latex style with perverted colors, and Start/Select arranged differently). Unknown if there has been also a japanese version with SLPH/SCPH number (one should think so, but nobody in the internet seems to have ever seen one).
And there is a handheld version (with additional L1/L2/R2/R1 buttons; maybe unlicensed; produced as MINI DDR, and also as Venom Mini Dance Pad).

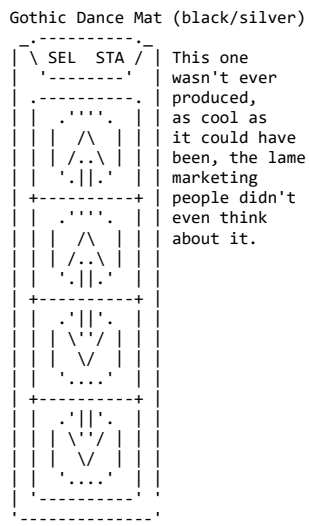
```
US Version (white/black/red/blue)        Handheld Version (blue/gray)
```



European Version (pink/blue/yellow)



Gothic Dance Mat (black/silver)



Stay Cool?
Despite of the "Stay Cool!" slogan, the mat wasn't very cool - not at all! It offered only two steps back-and-forth, and also allowed to do extremly uncool side-steps. Not to mention that it would melt when dropping a burning cigarette on it. Stay Away!

Controllers - Fishing Controllers

The fishing rods are (next to lightguns) some of the more openly martial playstation controllers - using the credo that "as long as you aren't using dynamite: it's okay to kill them cause they don't have any feelings."

PSX Fishing Controller Games

```
Action Bass (Syscom Entertainment) (1999) (SLPH-00100)
Bass Landing (ASCII/agetec) (1999) (SLPH-00100, SLUH-00063)
Bass Rise, Fishing Freaks (Bandai) (1999) (BANC-0001)
Bass Rise Plus, Fishing Freaks (Bandai) (2000) (BANC-0001, SLPH-00100)
Breath of Fire IV (Capcom) (SLUH-00063)
Championship Bass (EA Sports) (2000) (SLUH-00063)
Fish On! Bass (Pony Canyon) (1999) (BANC-0001, SLPH-00100)
Fisherman's Bait 2/Exiting Bass2 - Big Ol'Bass(Konami)(SLPH-00100,SLUH-00063)
Fishing Club: (series with 3 titles) (have "headset-logo" on back?)
Lake Masters II (1999) (Dazz/Nexus) (SLPH-00100)
Lake Masters Pro (1999) (Dazz/Nexus) (BANC-0001, SLPH-00100)
Let's Go Bassfishing!: Bass Tsuri ni Ikou! (Banpresto) (1999) (SLPH-00100)
Matsukata Hiroki no World Fishing (BPS The Choice) (1999) (SLPH-00100)
Murakoshi Seikai-Bakuchou Nihon Rettou (Victor) (SLPH-00100)
Murakoshi Masami-Bakuchou Nippon Rettou:TsuriConEdition (1999) (SLPH-00100)
Pakuchikou Seabass Fishing (JP, 03/25/99) (Victor) (SLPH-00100)
Perfect Fishing: Bass Fishing (2000) (Seta) (yellow/green logo)
Perfect Fishing: Rock Fishing (2000) (Seta) (yellow/green logo)
Oyaji no Jikan: Nechan, Tsuri Iku De! (2000) (Visit) (BANC-0001, SLPH-00100)
Reel Fishing II / Fish Eyes II (2000)(Natsume/Victor)(SLPH-00100, SLUH-00063)
Simple 1500 Series Vol. 29: The Tsuri (2000) (yellow/green logo)
Suizokukan Project: Fish Hunter e no Michi (1999)(Teichiku)(SLPH-00100)
Super Bass Fishing (1999) (King) (BANC-0001, SLPH-00100, yellow/green logo)
Super Black Bass X2 (2000) (Starfish) (SLPH-00100)
Tsuwadou Keiryuu Mizuumihen (Best Edition)(2000) (ASCII PS1+PS2 controllers?)
Tsuwadou Seabass Fishing (PlayStation the Best) (1999) (Oz Club) (SLPH-00100)
Uki Uki Tsuri Tengoku Nagami/Uokami Densetsu Oe (2000) (Teichiku)(SLPH-00100)
Umi no Nushi Tsuri-Takarajima ni Mukatte (1999)(Victor)(BANC-0001,SLPH-00100)
Winning Lure (Hori) (2000) (for Hori HPS-97 controller)  AKA HPS-98 ?
```

For more see: http://www.gamefaqs.com/ps/list-109 (sports->nature->fishing)

Logos on CD Covers
US Fishing games should have a "SLUH-00063" logo. European Fishing games don't have any fishing logos; apparently fishing controllers haven't been officially released/supported in Europe.
Japanese Fishing games can have a bunch of logos: Usually BANC-0001 or SLPH-00100 (or both).
Moreover, some japanese games have a yellow/green fishing logo with japanese text (found on Perfect Fishing: Bass Fishing, Perfect Fishing: Rock Fishing, Simple 1500 Series Vol. 29: The Tsuri, Super Bass Fishing) (unknown if that logo refer to other special hardware, or if it means the "normal" BANC-0001 or SLPH-00100 controllers.
And Moreover, some japanese games have some sort of "headset" logos with japanese text, these seem to have same meaning as SLPH-00100; as indicated by photos on CD cover of Tsuwadou Keiryuu Mizuumihen (Best Edition) (2000); that CD cover also has a "headset 2" logo, which seems to mean a newer PS2 variant of the SLPH-00100.

PSX Fishing Controllers

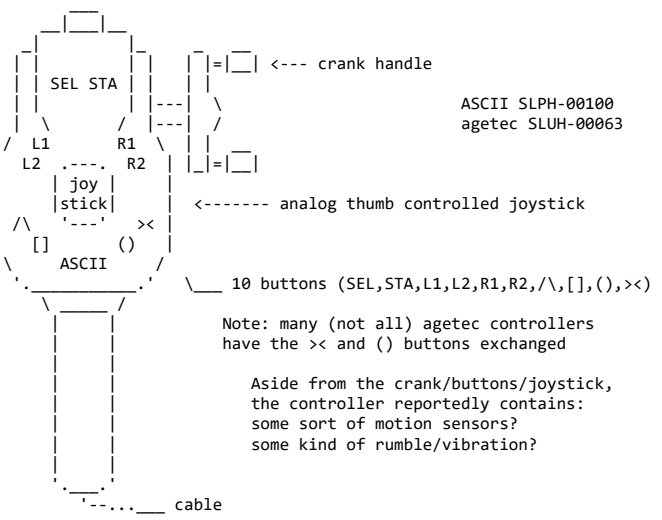
```
ASCII SLPH-00100 (silver)
ASCII PS2-version? (silver) (similar to SLPH-00100, with new mode switch?)
agetec SLUS-00063 (silver) (US version of ASCII's SLPH-00100 controller)
```

```
Bandai BANC-0001 (dark gray/blue) (has less buttons than ASCII/agetec)
Interact Fission (light gray/blue)(similar to ASCII/agetec, 2 extra buttons?)
Naki (transparent blue) (looks like a clone of the ASCII/agetec controllers)
Hori HPS-97/HPS-98 (black/gray) (a fishing rod attached to a plastic fish)
```

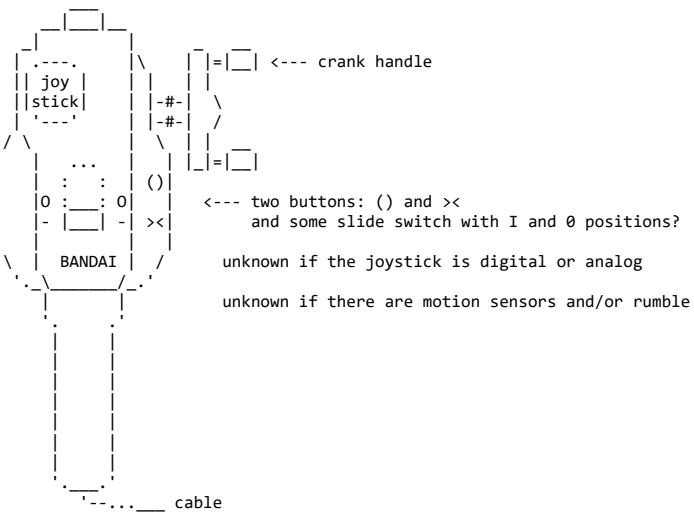
Of these, the ASCII/agetec controllers seem to be most popular (and most commonly supported). The Bandai contoller is also supported by a couple of games (though the Bandai controller itself seems to be quite rare). The Interact/Naki controllers are probably just clones of the ASCII/agetec ones. The Hori controller is quite rare (and with its string and plastic fish, it's apparently working completely different than the other fishing controllers).

Tech Info (all unknown)
Unknown how to detect fishing controllers.
Unknown how to read buttons, joystick, crank, motion sensors.
Unknown how to control rumble/vibration.
Unknown if/how Bandai differs from ASCII/agetec (aside from less buttons).
Unknown how the Hori thing works.

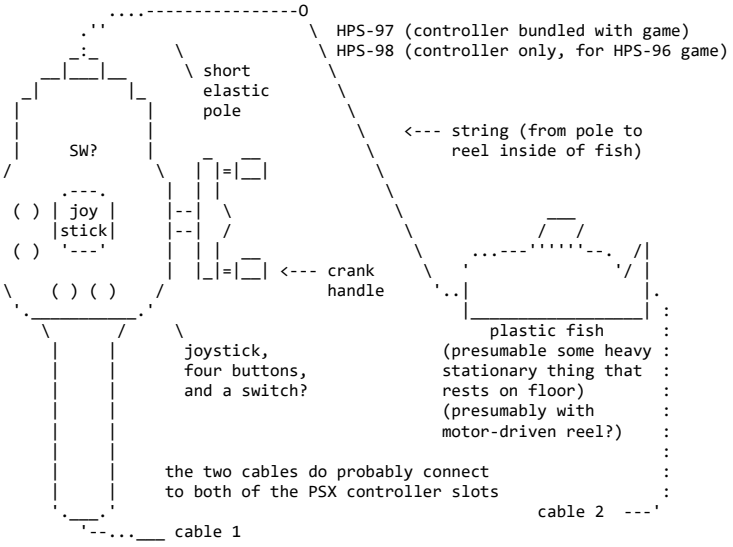
ASCII SLPH-00100 / agetec SLUH-00063 (silver)



Bandai BANC-0001 (dark gray/blue)



Hori HPS-97 / HPS-98 (black/gray)



Controllers - I-Mode Adaptor (Mobile Internet)

The I-Mode Adaptor cable (SCPH-10180) allows to connect an I-mode compatible mobile phone to the playstation's controller port; granting a mobile internet connection to japanese games.

PSX Games for I-Mode Adaptor (Japan only)
```
Doko Demo Issyo (PlayStation the Best release only) (Sony) 2000
Doko Demo Issyo Deluxe Pack (Bomber eXpress/Sony) 2001
Hamster Club-I (SLPS-03266) (Jorudan) 2002
iMode mo Issyo: Dokodemo Issho Tsuika Disc (Bomber/Sony) 2001
Keitai Eddy (iPC) 2000 (but, phone connects to SIO port on REAR side of PSX?)
Komocchi (Victor) 2001
Mobile Tomodachi (Hamster) 2002
Motto Trump Shiyouyo! i-Mode de Grand Prix (Pure Sound) 2002
One Piece Mansion (Capcom) 2001 (japanese version only)
```
The supported games should have a I-Mode adaptor logo on the CD cover (the logo depicts two plugs: the PSX controller plug, and the smaller I-Mode plug).
Note: "Dragon Quest Monsters 1 & 2" was announced/rumoured to support I-mode (however, its CD cover doesn't show any I-Mode adapter logo).

Tech Details (all unknown)
Unknown how to detect the thing, and how to do the actual data transfers.
The cable does contain a 64pin chip, an oscillator, and some smaller components (inside of the PSX controller port connector).

Hardware Variant
Keitai Eddy seems to have the phone connect to the SIO port (on rear side of the PSX, at least it's depicted like so on the CD cover). This is apparently something different than the SCPH-10180 controller-port cable. Unknown what it is exactly - probably some mobile internet connection too, maybe also using I-mode, or maybe some other protocol.

Controllers - Additional Inputs

Reset Button
PSX only (not PSone). Reboots the PSX via /RESET signal. Probably including for forcefully getting through the WHOLE BIOS Intro, making it rather useless/annoying? No idea if it clears ALL memory during reboot?

CDROM Shell Open
Status bit of the CDROM controller. Can be used to sense if the shell is opened (and also memorizes if the shell was opened since last check; allowing to sense possible disk changes).

PocketStation
Memory Card with built-in LCD screen and Buttons (which can be used as miniature handheld console). However, when it is connected to the PSX, the buttons are vanishing in the cartridge slot, so the buttons cannot be used as additional inputs for PSX games.

Serial Port PSX only (not PSone)
With an external adaptor (voltage conversion), the serial port can be used (among others) to connect a RS232 Serial Mouse. Although, most or all commercial games with mouse input are probably (?) supporting only Sony's Mouse (on the controller port) (rather than standard RS232 devices on the serial port).

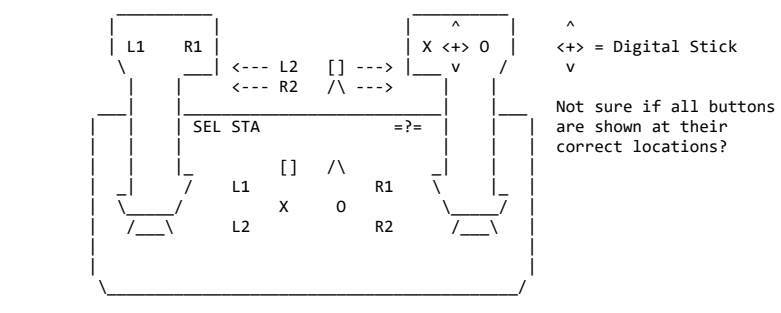
TTY Debug Terminal
If present, the external DUART can be used for external keyboard input, at the BIOS side, this is supported as "std_in".

Controllers - Misc

Standard Controllers
```
SCPH-1010  digital joypad (with short cable)
SCPH-1080  digital joypad (with longer cable)
SCPH-1030  mouse (with short cable)
SCPH-1090  mouse (with longer cable)
SCPH-1092  mouse (european?)
SCPH-1110  analog joystick
SCPH-1150  analog joypad (with one vibration motor, with red/green led)
SCPH-1180  analog joypad (without vibration motors, with red/green led)
SCPH-1200  analog joypad (with two vibration motors) (dualshock)
SCPH-110   analog joypad (with two vibration motors) (dualshock for psone)
SCPH-10010 dualshock2 (analog buttons, except L3/R3/Start/Select) (for ps2)
SCPH-1070  multitap
```

Special Controllers
```
SCPH-4010 VPick (guitar-pick controller) (for Quest for Fame, Stolen Song)
```
SLPH-0001 (nejicon)
BANDAI "BANC-0002" - 4 Buttons (Triangle, Circle, Cross, Square) (nothing more)

Joystick




```
The thumb buttons on the left act as L1 and R1,
   the trigger is L2, the pinky button is R2
The thumb buttons on the right act as X and O,
```

```
    the trigger is square and the pinky button is triangle.
 I find this odd as the triggers should've been L1 and R1,
    the pinkies L2 and R2.
 The buttons are redundantly placed on the base as large buttons like what
    you'd see on a fight/arcade stick. Also with Start and Select.
 There is also a physical analog mode switch,
    not a button like on dual shock.
```

Memory Card Read/Write Commands

Reading Data from Memory Card

```
  Send Reply Comment
  81h  N/A   Memory Card Access (unlike 01h=Controller access), dummy response
  52h  FLAG  Send Read Command (ASCII "R"), Receive FLAG Byte
  00h  5Ah   Receive Memory Card ID1
  00h  5Dh   Receive Memory Card ID2
  MSB  (00h) Send Address MSB  ;\sector number (0..3FFh)
  LSB  (pre) Send Address LSB  ;/
  00h  5Ch   Receive Command Acknowledge 1  ;<-- late /ACK after this byte-pair
  00h  5Dh   Receive Command Acknowledge 2
  00h  MSB   Receive Confirmed Address MSB
  00h  LSB   Receive Confirmed Address LSB
  00h  ...   Receive Data Sector (128 bytes)
  00h  CHK   Receive Checksum (MSB xor LSB xor Data bytes)
  00h  47h   Receive Memory End Byte (should be always 47h="G"=Good for Read)
```

Non-sony cards additionally send eight 5Ch bytes after the end flag.
When sending an invalid sector number, original Sony memory cards respond with FFFFh as Confirmed Address (and do then abort the transfer without sending any data, checksum, or end flag), third-party memory cards typically respond with the sector number ANDed with 3FFh (and transfer the data for that adjusted sector number).

Writing Data to Memory Card

```
  Send Reply Comment
  81h  N/A   Memory Card Access (unlike 01h=Controller access), dummy response
  57h  FLAG  Send Write Command (ASCII "W"), Receive FLAG Byte
  00h  5Ah   Receive Memory Card ID1
  00h  5Dh   Receive Memory Card ID2
  MSB  (00h) Send Address MSB  ;\sector number (0..3FFh)
  LSB  (pre) Send Address LSB  ;/
  ...  (pre) Send Data Sector (128 bytes)
  CHK  (pre) Send Checksum (MSB xor LSB xor Data bytes)
  00h  5Ch   Receive Command Acknowledge 1
  00h  5Dh   Receive Command Acknowledge 2
  00h  4xh   Receive Memory End Byte (47h=Good, 4Eh=BadChecksum, FFh=BadSector)
```

Get Memory Card ID Command

```
  Send Reply Comment
  81h  N/A   Memory Card Access (unlike 01h=Controller access), dummy response
  53h  FLAG  Send Get ID Command (ASCII "S"), Receive FLAG Byte
  00h  5Ah   Receive Memory Card ID1
  00h  5Dh   Receive Memory Card ID2
  00h  5Ch   Receive Command Acknowledge 1
  00h  5Dh   Receive Command Acknowledge 2
  00h  04h   Receive 04h
  00h  00h   Receive 00h
  00h  00h   Receive 00h
  00h  80h   Receive 80h
```

This command is supported only by original Sony memory cards. Not sure if all sony cards are responding with the same values, and what meaning they have, might be number of sectors (0400h) and sector size (0080h) or whatever.

Invalid Commands

```
  Send Reply Comment
  81h  N/A   Memory Card Access (unlike 01h=Controller access), dummy response
  xxh  FLAG  Send Invalid Command (anything else than "R", "W", or "S")
```

Transfer aborts immediately after the faulty command byte, or, occasionally after one more byte (with response FFh to that extra byte).

FLAG Byte

The initial value of the FLAG byte on power-up (and when re-inserting the memory card) is 08h.
Bit3=1 is indicating that the directory wasn't read yet (allowing to sense memory card changes). For some strange reason, bit3 is NOT reset when reading from the card, but rather when writing to it. To reset the flag, games are usually issuing a dummy write to sector number 003Fh, more or less unneccessarily stressing the lifetime of that sector.
Bit2=1 seems to be intended to indicate write errors, however, the write command seems to be always finishing without setting that bit, instead, the error flag may get set on the NEXT command.
Note: Some (not all) non-sony cards also have Bit5 of the FLAG byte set.

Timings

IRQ7 is usually triggered circa 1500 cycles after sending a byte (counted from the begin of the first bit), except, the last byte doesn't trigger IRQ7, and, after the 7th byte of the Read command, an additional delay of circa 31000 cycles occurs before IRQ7 gets triggered (that strange extra delay occurs only on original Sony cards, not on cards from other manufacturers).
There seems to be no extra delays in the Write command, as it seems, the data is written on the fly, and one doesn't need to do any write-busy handling... although, theoretically, the write shouldn't start until verifying the checksum... so it can't be done on the fly at all...?

Notes

Responses in brackets are don't care, (00h) means usually zero, (pre) means usually equal to the previous command byte (eg. the response to LSB is MSB).

Memory cards are reportedly "Flash RAM" which sounds like bullshit, might be battery backed SRAM, or FRAM, or slower EEPROM or FLASH ROM, or vary from card to card...?

Memory Card Data Format

Data Size

```
  Total Memory 128KB = 131072 bytes = 20000h bytes
  1 Block 8KB = 8192 bytes = 2000h bytes
```

```
  1 Frame 128 bytes = 80h bytes
```

The memory is split into 16 blocks (of 8 Kbytes each), and each block is split into 64 sectors (of 128 bytes each). The first block is used as Directory, the remaining 15 blocks are containing Files, each file can occupy one or more blocks.

Header Frame (Block 0, Frame 0)

```
  00h-01h Memory Card ID (ASCII "MC")
  02h-7Eh Unused (zero)
  7Fh     Checksum (all above bytes XORed with each other) (usually 0Eh)
```

Directory Frames (Block 0, Frame 1..15)

```
  00h-03h Block Allocation State
            00000051h - In use ;first-or-only block of a file
            00000052h - In use ;middle block of a file (if 3 or more blocks)
            00000053h - In use ;last block of a file   (if 2 or more blocks)
            000000A0h - Free   ;freshly formatted
            000000A1h - Free   ;deleted (first-or-only block of file)
            000000A2h - Free   ;deleted (middle block of file)
            000000A3h - Free   ;deleted (last block of file)
  04h-07h Filesize in bytes (2000h..1E000h; in multiples of 8Kbytes)
  08h-09h Pointer to the NEXT block number (minus 1) used by the file
            (ie. 0..14 for Block Number 1..15) (or FFFFh if last-or-only block)
  0Ah-1Eh Filename in ASCII, terminated by 00h (max 20 chars, plus ending 00h)
  1Fh     Zero (unused)
  20h-7Eh Garbage (usually 00h-filled)
  7Fh     Checksum (all above bytes XORed with each other)
```

Filesize [04h..07h] and Filename [0Ah..1Eh] are stored only in the first directory entry of a file (ie. with State=51h or A1h), other directory entries have that bytes zero-filled.

Filename Notes

The first some letters of the filename should indicate the game to which the file belongs, in case of commercial games this is conventionally done like so: Two character region code:

```
  "BI"=Japan, "BE"=Europe, "BA"=America
```

followed by 10 character game code,

```
  in "AAAA-NNNNN" form     ;for Pocketstation executables replace "-" by "P"
```

where the "AAAA" part does imply the region too; (SLPS/SCPS=Japan, SLUS/SCUS=America, SLES/SCES=Europe) (SCxS=Made by Sony, SLxS=Licensed by Sony), followed by up to 8 characters,

```
  "abcdefgh"
```

(which may identify the file if the game uses multiple files; this part often contains a random string which seems to be allowed to contain any chars in range of 20h..7Fh, of course it shouldn't contain "?" and "*" wildcards).

Broken Sector List (Block 0, Frame 16..35)

```
  00h-03h Broken Sector Number (Block*64+Frame) (FFFFFFFFh=None)
  04h-7Eh Garbage (usually 00h-filled) (some cards have [08h..09h]=FFFFh)
  7Fh     Checksum (all above bytes XORed with each other)
```

If Block0/Frame(16+N) indicates that a given sector is broken, then the data for that sector is stored in Block0/Frame(36+N).

Broken Sector Replacement Data (Block 0, Frame 36..55)

```
  00h-7Fh Data (usually FFh-filled, if there's no broken sector)
```

Unused Frames (Block 0, Frame 56..62)

```
  00h-7Fh Unused (usually FFh-filled)
```

Write Test Frame (Block 0, Frame 63)

Reportedly "write test". Usually same as Block 0 ("MC", 253 zero-bytes, plus checksum 0Eh).

Title Frame (Block 1..15, Frame 0) (in first block of file only)

```
  00h-01h  ID (ASCII "SC")
  02h      Icon Display Flag
             11h...Icon has 1 frame  (static) (same image shown forever)
             12h...Icon has 2 frames (animated) (changes every 16 PAL frames)
             13h...Icon has 3 frames (animated) (changes every 11 PAL frames)
            Values other than 11h..13 seem to be treated as corrupted file
            (causing the file not to be listed in the bootmenu)
  03h      Block Number (1-15)  "icon block count"  Uh?
                  (usually 01h or 02h... might be block number within
                  files that occupy 2 or more blocks)
                  (actually, that kind of files seem to HAVE title frames
                  in ALL of their blocks; not only in their FIRST block)
                  (at least SOME seem to have such duplicated title frame,
                  but not all?)
  04h-43h  Title in Shift-JIS format (64 bytes = max 32 characters)
  44h-4Fh  Reserved (00h)
  50h-5Fh  Reserved (00h)  ;<-- this region is used for the Pocketstation
  60h-7Fh  Icon 16 Color Palette Data (each entry is 16bit CLUT)
```

For more info on entries [50h..5Fh], see

Pocketstation File Header/Icons

Icon Frame(s) (Block 1..15, Frame 1..3) (in first block of file only)

```
  00h-7Fh  Icon Bitmap (16x16 pixels, 4bit color depth)
```

Note: The icons are shown in the BIOS bootmenu (which appears when starting the PlayStation without a CDROM inserted). The icons are drawn via GP0(2Ch) command, ie. as Textured four-point polygon, opaque, with texture-blending, whereas the 24bit blending color is 808080h (so it's quite the same as raw texture without blending). As semi-transparency is disabled, Palette/CLUT values can be 0000h=FullyTransparent, or 8000h=SolidBlack (the icons are usually shown on a black background, so it doesn't make much of a difference).

Data Frame(s) (Block 1..15, Frame N..63; N=excluding any Title/Icon Frames)

```
  00h-7Fh  Data
```

Note: Files that occupy more than one block are having only ONE Title area, and only one Icon area (in the first sector(s) of their first block), the additional blocks are using sectors 0..63 for plain data.

Shift-JIS Character Set (16bit) (used in Title Frames)

Can contain japanese or english text, english characters are encoded like so:

```
  81h,40h      --> SPC
  81h,43h..97h --> punctuation marks
  82h,4Fh..58h --> "0..9"
  82h,60h..79h --> "A..Z"
```

Titles shorter than 32 characters are padded with 00h-bytes.
Note: The titles are <usually> in 16bit format (even if they consist of raw english text), however, the BIOS memory card manager does also accept 8bit characters 20h..7Fh (so, in the 8bit form, the title could be theoretically up to 64 characters long, but, nethertheless, the BIOS displays only max 32 chars).
For displaying Titles, the BIOS includes a complete Shift-JIS character set,
BIOS Character Sets
Shift-JIS is focused on asian languages, and does NOT include european letters (eg. such with accent marks). Although the non-japanese PSX BIOSes DO include a european character set, the BIOS memory card manager DOESN'T seem to translate any title character codes to that character set region.

Memory Card Images

There are a lot of different ways to get a save from a memory card onto your PC's hard disk, and these ways sometimes involve sticking some additional information into a header at the beginning of the file.

Raw Memory Card Images (without header) (ie. usually 128K in size)

```
SmartLink .PSM,
WinPSM .PS,
DataDeck .DDF,
FPSX .MCR,
ePSXe .MCD...
```

don't stick any header on the data at all, so you can just read it in and treat it like a raw memory card.

All of these headers contain a signature at the top of the file. The three most common formats and their signatures are:

```
Connectix Virtual Game Station format (.MEM): "VgsM", 64 bytes
PlayStation Magazine format (.PSX): "PSV", 256 bytes
```

some programs will OMIT any blank or unallocated blocks from the end of the memory card -- if only three save blocks on the card are in use, for example, saving the other twelve is pointless.

Xploder and Action Replay Files (54 byte header)

```
00h..14h Filename in ASCII, terminated by 00h (max 20 chars, plus ending 00h)
15h..35h Title in ASCII, terminated by 00h (max 32 chars, plus ending 00h)
36h..    File Block(s) (starting with the Title sector)
```

This format contains only a single file (not a whole memory card). The filename should be the same as used in the Memory Card Directory. The title is more or less don't care; it may be the SHIFT-JIS title from the Title Sector converted to ASCII.

Other
"There exists another single-save format with a 128 byte header containing a raw index frame for the initial block, which must be updated to match the destination card, and the raw save data. I have seen this format once, but I don't remember what it was called or where it came from. You may want to account for this possibility in your format detection logic."

.GME Files (usually 20F40h bytes)
InterAct GME format, produced by the DexDrive.

```
000h 12   ASCII String "123-456-STD",00h
00Ch 4    Usually zerofilled (or meaningless garbage in some files)
010h 5    Always 00h,00h,01h,00h,01h
015h 16   Copy of Sector 0..15 byte[00h] ;"M", followed by allocation states
025h 16   Copy of Sector 0..15 byte[08h] ;00h, followed by next block values
035h 11   Usually zerofilled (or meaningless garbage in some files)
040h F00h Fifteen Description Strings (each one 100h bytes, padded with 00h)
F40h 128K Memory Card Image (128K) (unused sectors 00h or FFh filled)
```

This is a very strange file format, no idea where it comes from. It contains a F40h bytes header (mainly zerofilled), followed by the whole 128K of FLASH memory (mainly zerofilled, too, since it usually contains only a small single executable file).

Memory Card Notes

Sony Cards
Sony seems to have manufactured only 128KByte memory cards, no bigger ones?

Third Party Cards
Other manufacturers have reportedly sold PSX memory cards with more blocks... parts due to using larger memory chips, parts due to using data compression.

Bigger Cards
Not sure how more than 15 data blocks can be implemented... there are some unused directory entries which might be used... though their number is limited... eventually the directory can be made bigger than one block... but not sure if that'd be supported by normal games... and if it'd be supported by the bios bootmemnu...?

Passwords
Some older games are using passwords instead of memory cards to allow the user to continue at certain game positions. Nice for people without memory card, unfortunately many of that games are restricted to it - it'd be more user friendly to support both passwords, or optionally, memory cards.

Other Hardware that connects to Memory Card slot

Pocketstation
Pocketstation

Yaroze Access Cards
xxx...

Joytech Smart Card Adaptor
The smart card adaptor plugs into memory card slot, and allows to use special credit card-shaped memory cards. There don't seem to be any special features, ie. the hardware setup does just behave like normal PSX memory cards.

Pocketstation

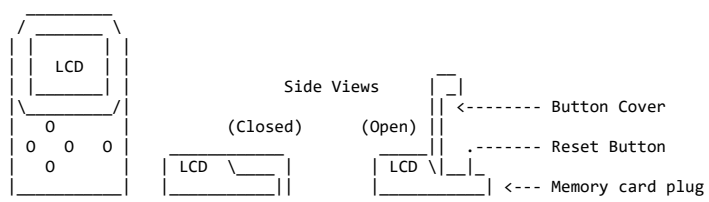
Pocketstation

Pocketstation Overview
Pocketstation I/O Map
Pocketstation Memory Map
Pocketstation IO Video and Audio
Pocketstation IO Interrupts and Buttons
Pocketstation IO Timers and Real-Time Clock
Pocketstation IO Infrared
Pocketstation IO Memory-Control
Pocketstation IO Communication Ports
Pocketstation IO Power Control
Pocketstation SWI Function Summary
Pocketstation SWI Misc Functions
Pocketstation SWI Communication Functions
Pocketstation SWI Execute Functions
Pocketstation SWI Date/Time/Alarm Functions
Pocketstation SWI Flash Functions
Pocketstation SWI Useless Functions
Pocketstation BU Command Summary
Pocketstation BU Standard Memory Card Commands
Pocketstation BU Basic Pocketstation Commands
Pocketstation BU Custom Pocketstation Commands
Pocketstation File Header/Icons
Pocketstation File Images
Pocketstation XBOO Cable

Pocketstation Overview

Sony's Pocketstation (SCPH-4000) (1998)
The Pocketstation is a memory card with built-in LCD screen and buttons; aside from using it as memory storage device, it can be also used as miniature handheld console.

```
CPU         ARM7TDMI (32bit RISC Processor) (var ? MHz)
Memory      2Kbytes SRAM (battery backed), 16Kbytes BIOS ROM, 128Kbytes FLASH
Display     32x32 pixel LCD (black and white) (without any grayscales)
Sound       Mini Speaker "(12bit PCM) x 1 unit" / "8bit PCM with 12bit range"
Controls    5 input buttons, plus 1 reset button
Infrared    Bi-directional (IrDA based)
Connector   Playstation memory card interface
RTC         Battery backed Real-Time Clock with time/date function
Supply      CR2032 Battery (3VDC) (used in handheld mode, and for SRAM/RTC)
```



The RTC Problem
The main problem of the Pocketstation seems to be that it tends to reset the RTC to 1st January 1999 with time 00:00:00 whenever possible.
The BIOS contains so many RTC-reset functions, RTC-reset buttons, RTC-reset flags, RTC-reset communication commands, RTC-reset parameters, RTC-reset exceptions, RTC-reset sounds, and RTC-reset animations that it seems as if Sony actually WANTED the Time/Date to be destroyed as often as possible.
The only possible reason for doing this is that the clock hardware is so inaccurate that Sony must have decided to "solve" the problem at software engineering side, by erasing the RTC values before the user could even notice time inaccuracies.

CPU Specs
For details on the ARM7TDMI CPUs opcodes and exceptions, check GBATEK at,
http://problemkaputt.de/gbatek.htm (or .txt)
The GBA uses an ARM7TDMI CPU, too.

Thanks to Exophase, Orion, Fezzik, Dr.Hell for Pocketstation info.

Pocketstation I/O Map

Memory and Memory-Control Registers

```
00000000h RAM        (2KB RAM) (first 512 bytes bytes reserved for kernel)
02000000h FLASH1     Flash ROM (virtual file-mapped addresses in this region)
04000000h BIOS_ROM      Kernel and GUI (16KB)
06000000h F_CTRL        Control of Flash ROM
06000004h F_STAT        Unknown?
06000008h F_BANK_FLG    FLASH virtual bank mapping enable flags(16 bits)(R/W)
0600000Ch F_WAIT1       waitstates...?
06000010h F_WAIT2       waitstates, and FLASH-Write-Control-and-Status...?
06000100h F_BANK_VAL    FLASH virtual bank mapping addresses (16 words) (R/W)
06000300h F_EXTRA       Extra FLASH (256 bytes, including below F_SN, F_CAL)
06000300h F_SN_LO       Extra FLASH Serial Number LSBs (nocash: 6BE7h)
06000302h F_SN_HI       Extra FLASH Serial Number MSBs (nocash: 426Ch)
06000304h F_?           Extra FLASH Unknown ?          (nocash: 05CAh)
06000306h F_UNUSED1     Extra FLASH Unused halfword    (nocash: FFFFh)
06000308h F_CAL         Extra FLASH LCD Calibration    (nocash: 001Ah)
0600030Ah F_UNUSED2     Extra FLASH Unused halfword    (nocash: FFFFh)
0600030Ch F_?           Extra FLASH Unknown ?          (nocash: 0010h)
0600030Eh F_UNUSED3     Extra FLASH Unused halfword    (nocash: FFFFh)
```

```
  0000510h F_UNUSED4    Extra FLASH Unused (510h..5FFh) (Nocash: FFFFh-filled)
  08000000h FLASH2      Flash ROM (128KB) (physical addresses in this region)
  08002A54h F_KEY1      Flash Unlock Address 1 (W)
  080055AAh F_KEY2      Flash Unlock Address 2 (W)
```

Interrupts and Timers

```
  0A000000h INT_LATCH     Interrupt hold (R)
  0A000004h INT_INPUT     Interrupt Status (R)
  0A000008h INT_MASK_READ Read Interrupt Mask (R)
  0A000008h INT_MASK_SET  Set Interrupt Mask (W)
  0A00000Ch INT_MASK_CLR  Clear Interrupt Mask (W)
  0A000010h INT_ACK       Clear Interrupt hold (W)
  0A800000h T0_RELOAD     Timer 0 Maximum value
  0A800004h T0_COUNT      Timer 0 Current value
  0A800008h T0_MODE       Timer 0 Mode
  0A800010h T1_RELOAD     Timer 1 Maximum value
  0A800014h T1_COUNT      Timer 1 Current value
  0A800018h T1_MODE       Timer 1 Mode
  0A800020h T2_RELOAD     Timer 2 Maximum value
  0A800024h T2_COUNT      Timer 2 Current value
  0A800028h T2_MODE       Timer 2 Mode
  0B000000h CLK_MODE      Clock control (CPU and Timer Speed) (R/W)
  0B000004h CLK_STOP      Clock stop (Sleep Mode)
  0B800000h RTC_MODE      RTC Mode
  0B800004h RTC_ADJUST    RTC Adjust
  0B800008h RTC_TIME      RTC Time (R)
  0B80000Ch RTC_DATE      RTC Date (R)
```

Communication Ports, Audio/Video

```
  0C000000h COM_MODE      Com Mode
  0C000004h COM_STAT1     Com Status Register 1 (Bit1=Error)
  0C000008h COM_DATA      Com RX Data (R) and TX Data (W)
  0C000010h COM_CTRL1     Com Control Register 1
  0C000014h COM_STAT2     Com Status Register 2 (Bit0=Ready)
  0C000018h COM_CTRL2     Com Control Register 2
  0C800000h IRDA_MODE     Infrared Control (R/W)
  0C800004h IRDA_DATA     Infrared TX Data
  0C80000Ch IRDA_MISC     Infrared Unknown/Reserved
  0D000000h LCD_MODE      Video Control (R/W)
  0D000004h LCD_CAL       Video Calibration (?)
  0D000100h LCD_VRAM      Video RAM (80h bytes; 32x32bit) (R/W)
  0D800000h IOP_CTRL      IOP control
  0D800004h IOP_STAT      Read Current Start/Stop bits? (R)
  0D800004h IOP_STOP      Stop bits? (W)
  0D800008h IOP_START     Start bits? (W)
  0D80000Ch IOP_DATA      IOP data?   (not used by bios)
  0D800010h DAC_CTRL      DAC Control (R/W)
  0D800014h DAC_DATA      DAC data
  0D800020h BATT_CTRL     Battery Monitor Control
```

BIOS and FLASH may be read only in 16bit and 32bit units, reading 8bit units doesn't seem to be supported...? strange... ARM CPUs should be theoretically able to extract 8bit values from 16bit data busses...?
Upon reset, BIOS ROM is mirrored to address 00000000h (instead of RAM).
For most I/O ports, it is unknown if they are (R), (W), or (R/W)...?
I/O ports are usually accessed at 32bit width, occassionally some ports are (alternately) accessed at 16bit width, though not sure if that works with ALL ports, and no idea if/how far ports can be accessed at 8bit width...? A special case are the F_SN registers which seem to be required to be accessed at 16bit (not 32bit).

Unknown/Unused Memory Locations

```
  00000800h-00FFFFFFh  Mirrors of 00000000h-000007FFh (2K RAM)
  01000000h-01FFFFFFh  Invalid (read causes data abort) (unused 16MB area)
  020xxxxxh-0201FFFFh  Invalid (read causes data abort) (disabled FLASH banks)
  02020000h-02FFFFFFh  Invalid (read causes data abort) (no Virt FLASH mirrors)
  03000000h-03FFFFFFh  Invalid (read causes data abort) (unused 16MB area)
  04004000h-04FFFFFFh  Mirrors of 04000000h-04003FFFh (16K BIOS)
  05000000h-05FFFFFFh  Invalid (read causes data abort)
  06000014h-060000FFh   Zerofilled (or maybe mirror of a ZERO port?) (F_xxx)
  06000140h-060002FFh   Zerofilled (or maybe mirror of a ZERO port?) (F_xxx)
  06000400h-06FFFFFFh   Zerofilled (or maybe mirror of a ZERO port?) (F_xxx)
  07000000h-07FFFFFFh  Invalid (read causes data abort) (unused 16MB area)
  08020000h-08FFFFFFh  Mirrors of 08000000h-0801FFFFh (128K Physical FLASH)
  09000000h-09FFFFFFh  Invalid (read causes data abort) (unused 16MB area)
  0A000014h-0A7FFFFFh  Mirrors of 0A000008h-0A00000Bh (INT_MASK_READ) (I_xxx)
  0A80000Ch            Mirror of 0A800000h-0A800003h (T0_RELOAD) (T0_xxx)
  0A80001Ch            Mirror of 0A800000h-0A800003h (T0_RELOAD) (T1_xxx)
  0A80002Ch            Mirror of 0A800000h-0A800003h (T0_RELOAD) (T2_xxx)
  0A800030h-0AFFFFFFh  Mirrors of 0A800000h-0A800003h (T0_RELOAD) (T_xxx)
  0B000008h-0B7FFFFFh  Mirrors of .... ? (CLK_xxx)
  0B800010h-0BFFFFFFh  Mirrors of 0B800008h-0B80000Bh (RTC_TIME)
  0C00000Ch-0C00000Fh  Zero (COM_xxx)
  0C00001Ch-0C7FFFFFh   Zerofilled (or maybe mirror of a ZERO port?) (COM_xxx)
  0C800008h-0CFFFFFFh   ? (IRDA_xxx)
  0D000008h-0D0000FFh   Zerofilled (or maybe mirror of a ZERO port?) (LCD_xxx)
  0D000180h-0D7FFFFFh   Zerofilled (or maybe mirror of a ZERO port?) (LCD_xxx)
  0D800018h             ? (DAC_xxx)
  0D80001Ch             ? (DAC_xxx)
  0D800024h-0DFFFFFFh   Zerofilled (or maybe mirror of a ZERO port?) (BATT_xxx)
  0E000000h-FFFFFFFFh  Invalid (read causes data abort) (unused 3872MB area)
```

Some of the unused memory locations are reportedly containing mirrors, others are reportedly causing Data Abort exceptions or so...?

Unsupported 8bit Reads

```
  02000000h-0201FFFFh VIRT_FLASH   ;\
  04000000h-04FFFFFFh BIOS_ROM     ; Mirror to ARM opcode [$+8+(addr and 3)]
  06000300h-060003FFh EXTRA_FLASH  ; (but unstable, ORed with garbage)
  08000000h-08FFFFFFh PHYS_FLASH   ;/
  0A800001h-0AFFFFFFh Timer area, odd addresses (with A0=1) mirror to 0A800001h
  0B800001h-0BFFFFFFh RTC area, odd addresses (with A0=1) mirror to ...?
```

Unsupported 16bit Reads

```
  0B800002h-0BFFFFFEh RTC area, odd addresses (with A1=1) mirror to 0B80000Ah
```

Pocketstation Memory Map

Overall Memory Map
```
00000000h RAM      RAM (2K)   (or mirror of BIOS ROM upon reset)
02000000h FLASH1   Flash ROM  (virtual file-mapped addresses in this region)
04000000h BIOS_ROM BIOS (16K) (Kernel and GUI)
06000300h F_SN...  Seems to contain a bunch of additional FLASH bytes?
08000000h FLASH2   Flash ROM  (128K) (physical addresses in this region)
0D000100h LCD_VRAM Video RAM  (128 bytes) (32x32 pixels, 1bit per pixel)
```

00000000h..000001FFh - Kernel RAM
The first 200h bytes of RAM are reserved for the kernel.
```
0000000h 20h  Exception handler opcodes (filled with LDR R15,[$+20h] opcodes)
0000020h 20h  Exception handler addresses (in ARM state, no THUMB bit here)
0000040h 80h  Sector buffer (and BU command parameter work space)
00000C0h 8    ComFlags (see GetPtrToComFlags(), SWI 06h for details)
00000C8h 2    BU Command FUNC3 Address (see GetPtrToFunc3addr() aka SWI 17h)
00000CAh 1    Value from BU Command_50h, reset by SWI 05h (sense_auto_com)
00000CBh 2    Not used
00000CDh 1    Old Year (BCD, 00h..99h) (for sensing wrapping to new century)
00000CEh 1    Alternate dir_index (when [0D0h]=0) (see SWI 15h and SWI 16h)
00000CFh 1    Current Century (BCD, 00h..99h) (see GetBcdDate() aka SWI 0Dh)
00000D0h 2    Current dir_index (for currently executed file, or 0=GUI)
00000D2h 2    New dir_index (PrepareExecute(flag,dir_index,param), SWI 08h)
00000D4h 4    New param     (PrepareExecute(flag,dir_index,param), SWI 08h)
00000D8h 8    Alarm Setting        (see GetPtrToAlarmSetting() aka SWI 13h)
00000E0h 4    Pointer to SWI table (see GetPtrToPtrToSwiTable() aka SWI 14h)
00000E4h 3x4  Memory Card BU Command variables
00000F0h 1    Memory Card FLAG byte (bit3=new_card, bit2=write_error)
00000F1h 1    Memory Card Error offhold (0=none, 1=once)
00000F2h 6    Not used
00000F8h 4x4  Callback Addresses (set via SetCallbacks(index,proc), SWI 01h)
0000108h 4    Snapshot ID (0xh,00h,"SE")
000010Ch 74h  IRQ and SWI stack (stacktop at 180h)
0000180h 80h  FIQ stack (stacktop at 200h)
```
Although one can modify that memory, one usually shouldn't do that, or at least one must backup and restore the old values before returning control to the GUI or to other executables. Otherwise, the only way to restore the original values would be to press the Reset button (which would erase the RTC time/date).

00000200h..000007FFh - User RAM and User stack (stacktop at 800h)
This region can be freely used by the game. The memory is zerofilled when the game starts.

02000000h - FLASH1 - Flash ROM (virtual file-mapped addresses in this region)
This region usually contains the currently selected file (including its title and icon sectors), used to execute the file in this region, mapped to continous addresses at 2000000h and up.

08000000h - FLASH2 - Flash ROM (128K) (physical addresses in this region)
This region is used by the BIOS when reading the memory card directory (and when writing data to the FLASH memory). The banking granularity is 2000h bytes (one memory card block), that means that the hardware cannot map Replacement Sectors which may be specified in the for Broken Sector List.

04000000h - BIOS ROM (16K) - Kernel and GUI
```
4000000h 1E00h Begin of Kernel (usually 1E00h bytes)
4000014h 4     BCD Date in YYYYMMDDh format (19981023h for ALL versions)
4001DFCh 4     Core Kernel Version  (usually "C061" or "C110")
4001E00h 2200h Begin of GUI (usually 2200h bytes)
4003FFCh 4     Japanese GUI Version (usually "J061" or "J110")
```
The "110" version does contain some patches, but does preserve same function addresses as the "061" version, still it'd be no good to expect the BIOS to contain any code/data at fixed locations (except maybe the GUI version string). Kernel functions can be accessed via SWI Opcodes, and, from the PSX-side, via BU Commands.

Bus-Width Restrictions
FLASH and BIOS ROM seem to be allowed to be read only in 16bit and 32bit units, not in 8bit units? Similar restrictions might apply for some I/O ports...? RAM can be freely read/written in 8bit, 16bit, and 32bit units.

Waitstates
Unknown if and how many waitstates are applied to the different memory regions. The F_WAIT1 and F_WAIT2 registers seem to be somehow waitstate related. FLASH memory does probably have a 16bit bus, so 32bit data/opcode fetches might be slower then 16bit reads...? Similar delays might happen for other memory and I/O regions...?

Pocketstation IO Video and Audio

0D000000h - LCD_MODE - LCD control word (R/W)
```
0-2  Draw mode; seems to turn off bits of the screen;
       0: All 32 rows on      ;\
       1: First 8 rows on     ;
       2: Second 8 rows on    ;
       3: Third 8 rows on     ; (these are not necessarily all correct?)
       4: Fourth 8 rows on    ;
       5: First 16 rows on    ;
       6: Middle 16 rows on   ;
       7: Bottom 16 rows on   ;/
3    CPEN      (0=Does some weird fade out of the screen, 1=Normal)
4-5  Refresh rate
       0: Makes a single blue (yes, blue, yes, on a black/white display)
          line appear at the top or middle of the screen - don't use!
       1: 64Hz? (might be 32Hz too, like 2)
       2: 32Hz
       3: 16Hz (results in less intensity on black pixels)
6    Display active               (0=Off, 1=On)
7    Rotate display by 180 degrees (0=For Handheld Mode, 1=For Docked Mode)
8-31 Unknown (should be zero)
```
Software should usually set LCD_MODE.7 equal to INT_INPUT.Bit11 (docking flag). In handheld mode, the button-side is facing towards the player, whilst in Docked mode (when the Pocketstation is inserted into the PSX controller port), the button-side is facing towards the PSX, so the screen coordinates become vice-versa, which can be "undone" by the Rotation flag.

0D000004h - LCD_CAL - LCD Calibration (maybe contrast or so?)
Upon the reset, the kernel sets LCD_CAL = F_CAL AND 0000003Fh. Aside from that, it doesn't use LCD_CAL.

0D000100h..D00017Fh - LCD_VRAM - 32x32 pixels, 1bit color depth (R/W)

This region consists of 32 words (32bit values),

```
[D000100h]=Top, through [D00017Ch]=Bottom-most scanline
```

The separate scanlines consist of 32bit each,

```
Bit0=Left, through Bit31=Right-most Pixel (0=White, 1=Black)
```

That [D000100h].Bit0=Upper-left arrangement applies if the Rotate bit in LCD_MODE.7 is set up in the conventional way, if it is set the opposite way, then it becomes [D00017Ch].Bit31=Upper-left.

The LCD_VRAM area is reportedly mirrored to whatever locations?

0D800010h - DAC_CTRL - Audio Control (R/W)

```
0     Audio Enable enable    (0=Off, 1=On)
1-31  Unknown, usually zero
```

Note: Aside from the bit in DAC_CTRL, audio must be also enabled/disabled via IOP_STOP/IOP_START bit5. Not sure if/which different purposes that bits have.

0D800014h - DAC_DATA - Audio D/A Converter

Not sure how many bits are passed to the D/A converter, probably bit8-15, ie. 8 bits...?

```
0-7   Probably unused, usually zero (or fractional part when lowered volume)
8-15  Signed Audio Outut Level (usually -7Fh..+7Fh) (probably -80h works too)
16-31 Probably unused, usually sign-expanded from bit15
```

The Pocketstation doesn't have any square wave or noise generator (nor a sound DMA channel). So the output levels must be written to DAC_DATA by software, this is usually done via Timer1/IRQ-8 (to reduce CPU load caused by high audio frequencies, it may be much more recommended to use Timer2/FIQ-13, because the FIQ handler doesn't need to push r8-r12).

For example, to produce a 1kHz square wave, the register must be toggled high/low at 2kHz rate. If desired, multiple channels can be mixed by software. High frequencies and multiple voices may require high CPU speed settings, and thus increase battery consumption (aside from that, battery consumption is probably increased anyways when the speaker is enabled).

Pocketstation IO Interrupts and Buttons

0A000004h - INT_INPUT - Raw Interrupt Signal Levels (R)

```
Bit   Type    Meaning
0     IRQ     Button Fire     (0=Released, 1=Pressed)
1     IRQ     Button Right    (0=Released, 1=Pressed)
2     IRQ     Button Left     (0=Released, 1=Pressed)
3     IRQ     Button Down     (0=Released, 1=Pressed)
4     IRQ     Button Up       (0=Released, 1=Pressed)
5     ?       Unknown?        (?)
6     FIQ (!) COM              ;for the COM_registers?      (via /SEL Pin?)
7     IRQ     Timer 0
8     IRQ     Timer 1
9     IRQ     RTC             (square wave or so...?)
10    IRQ     Battery Low     (0=Normal, 1=Battery Low)
11    IRQ     Docked ("IOP")  (0=Undocked, 1=Docked to PSX) (via VCC Pin?)
12    IRQ     Infrared Rx
13    FIQ (!) Timer 2
14-15 N/A     Not used
```

The buttons are usually read directly from this register (rather than being configured to trigger IRQs) (except in Sleep mode, where the Fire Button IRQ is usually used to wakeup). Also, bit9-11 are often read from this register.

The direction keys seem to be separate buttons, ie. unlike as on a joystick or DPAD, Left/Right (and Up/Down) can be simultaneously pressed...?

0A000008h - INT_MASK_SET - Set Interrupt Mask (W)
0A00000Ch - INT_MASK_CLR - Clear Interrupt Mask (W)
0A000008h - INT_MASK_READ - Read Interrupt Mask (R)

```
INT_MASK_SET  Enable Interrupt Flags         (0=No change, 1=Enable)   (W)
INT_MASK_CLR  Disable Interrupt Flags        (0=No change, 1=Disable)  (W)
INT_MASK_READ Current Interrupt Enable Flags (0=Disabled, 1=Enabled)   (R)
```

The locations of the separate bits are same as in INT_INPUT (see there).

0A000000h - INT_LATCH - Interrupt Request Flags (R)
0A000010h - INT_ACK - Acknowledge Interrupts (W)

```
INT_LATCH Latched Interrupt Requests   (0=None, 1=Interrupt Request)   (R)
INT_ACK   Clear Interrupt Requests     (0=No change, 1=Acknowledge)    (W)
```

The locations of the separate bits are same as in INT_INPUT (see there).

The interrupts seem to be edge-triggered (?), ie. when the corresponding bits in INT_INPUT change from 0-to-1. Not sure if the request bits get set when the corresponding interrupt is disabled in INT_MASK...?

ATTENTION: The GUI doesn't acknowledge Fire Button interrupts on wakeup... so, it seems as if button interrupts are NOT latched... ie. the button "INT_LATCH" bits seem to be just an unlatched mirror of the "INT_INPUT" bits... that might also apply for some other interrupt...?

However, after wakeup, the gui does DISABLE the Fire Button interrupt, MAYBE that does automatically acknowledge it... in that case it might be latched...?

Reading outside the readable region (that is where exactly?) seems to mirror to 0A000008h. Enabling IRQs for the buttons seems to make it impossible to poll them... is that really true? The RTC interrupt is a square wave with varying duty cycles that can be changed (how exactly?) in register 0B800000h.

Pocketstation IO Timers and Real-Time Clock

Timer and RTC interrupts

```
INT_INPUT.7     Timer 0 IRQ    ;used as 30Hz frame rate IRQ by GUI
INT_INPUT.8     Timer 1 IRQ    ;used as Audio square wave IRQ by GUI
INT_INPUT.13    Timer 2 FIQ (this one via FIQ vector, not IRQ vector)
INT_INPUT.9     RTC IRQ
```

0A800000h - T0_RELOAD - Timer 0 Reload Value
0A800010h - T1_RELOAD - Timer 1 Reload Value
0A800020h - T2_RELOAD - Timer 2 Reload Value

```
0-15  Reload Value (when timer becomes less than zero)
```

Writes to this register are ignored if the timer isn't stopped?

0A800004h - T0_COUNT - Timer 0 Current value
0A800014h - T1_COUNT - Timer 1 Current value
0A800024h - T2_COUNT - Timer 2 Current value

```
  0-15  Current value (decrementing)
```

Timer interrupts: The timers will automatically raise interrupts if they're enabled, there's no need to set a bit anywhere for IRQs (but you need to enable the respect interrupts in INT_MASK).

0A800008h - T0_MODE - Timer 0 Control
0A800018h - T1_MODE - Timer 1 Control
0A800028h - T2_MODE - Timer 2 Control

```
  0-1  Timer Divider (0=Div2, 1=Div32, 2=Div512, 3=Div2 too)
  2    Timer Enable  (0=Stop, 1=Decrement)
  3-15 Unknown (should be zero)
```

Timers are clocked by the System Clock (usually 4MHz, when CLK_MODE=7), divided by the above divider setting. Note that the System Clock changes when changing the CPU speed via CLK_MODE, so Timer Divider and/or Timer Reload must be adjusted accordingly.

0B800000h - RTC_MODE - RTC control word

```
  0    Pause RTC (0=Run, 1=Pause)
  1-3  Select value to be modified via RTC_ADJUST... and select Duty...?
  4-31 Not used?
```

The selection bits can be:

```
  00h = Second        ;\
  01h = Minute        ;
  02h = Hour          ; used in combination with RTC_ADJUST
  03h = Day of Week   ; while RTC is paused
  04h = Day           ;
  05h = Month         ;
  06h = Year          ;/
  07h = Unknown       ;-usually used when RTC isn't paused
```

The selection bits do reportedly also vary the duty cycle of the RTC interrupt signal... unknown which duty cycles exist... maybe per minute through per year... and/or maybe per less than 1 second... or maybe square waves and such with shorter "1" and longer "0" pulses...?
"There is a RTC-based counter capable of operating at 1/sec or at 4000/sec, but I don't know exactly how it works yet." Uh, does that refer to duty, or to something else?

0B800004h - RTC_ADJUST - Modify value (write only)
Writing a value here seems to increment the current selected parameter (by the RTC control). What is perhaps (?) clear is that you have to wait for the RTC interrupt signal to go low before writing to this.

0B800008h - RTC_TIME - Real-Time Clock Time (read only)

```
  0-7   Seconds     (00h..59h, BCD)
  8-15  Minutes     (00h..59h, BCD)
  16-23 Hours       (00h..23h, BCD)
  24-31 Day of week (1=Sunday, ..., 7=Saturday)
```

Reading RTC_TIME seems to be somewhat unstable: the BIOS uses a read/retry loop, until it has read twice the same value (although it does read the whole 32bit at once by a LDR opcode, the data is maybe passed through a 8bit or 16bit bus; so the LSBs might be a few clock cycles older than the MSBs...?).

0B80000Ch - RTC_DATE - Real-Time Clock Date (read only)

```
  0-7   Day     (01h..31h, BCD)
  8-11  Month   (01h..12h, BCD)
  16-23 Year    (00h..99h, BCD)
  24-31 Unknown? (this is NOT used as century)
```

Reading RTC_DATE seems to require the same read/retry method as RTC_TIME (see there). Note: The century is stored in battery-backed RAM (in the reserved kernel RAM region) rather than in the RTC_DATE register. The whole date, including century, can be read via SWI 0Dh, GetBcdDate().

Reading (which?) other locations seems to mirror to 0B800008h.

Pocketstation IO Infrared

The BIOS doesn't contain any IR functions (aside from doing some basic initialization and power-down stuff).
IR is used in Final Fantasy 8's Chocobo World (press Left/Right in the Map screen to go to the IR menu), and in Metal Gear Solid Integral (Press Up in the main screen), and in PDA Remote 1 & 2 (one-directional TV remote control).

0C800000h - IRDA_MODE - Controlling the protocol - send/recv, etc. (R/W)

```
  0    Transfer Direction  (0=Receive, 1=Transmit)
  1    Disable IRDA        (0=Enable, 1=Disable)
  2    Unknown (reportedly IR_SEND_READY, uh?)
  3    Unknown (reportedly IR_RECV_READY, uh?)
  4-31 Unknown (should be zero)
```

0C800004h - IRDA_DATA - Infrared TX Data

```
  0    Transmit Data in Send Direction (0=LED Off, 1=LED On)
  1-31 Unknown (should be zero)
```

Bits are usually encoded as long or short ON pulses, separated by short OFF pulses. Where long is usually twice as long as short.

0C80000Ch - IRDA_MISC
Unknown? Reportedly reserved.

INT_INPUT.12 - IRQ - Infrared RX Interrupt
Seems to get triggered on raising or falling (?) edges of incoming data. The interrupt handler seems to read the current counter value from one of the timers (usually Timer 2, with reload=FFFFh) to determine the length of the incoming IR pulse.

IR Notes
Mind that IR hardware usually adopts itself to the normal light conditions, so if it receives an IR signal for a longer period, then it may treat that as the normal light conditions (ie. as "OFF" state). To avoid that, one would usually send a group of ON-OFF-ON-OFF pulses, instead of sending a single long ON pulse:

```
  ___----------------___  One HIGH bit send as SINGLE-LONG-ON pulse   (BAD)
  ___-_-_-_-_-_-_-_-___  One HIGH bit send as MULTIPLE-ON-OFF pulses (OK)
```

that might be maybe done automatically by the hardware...?

Reportedly, Bit4 of Port 0D80000Ch (IOP_DATA) is also somewhat IR related...?

Pocketstation IO Memory-Control

06000000h - F_CTRL
```
0-31  Unknown
```
Written values are:
```
00000000h  Used when disabling all virtual flash banks
00000001h  Used before setting new virtual bank values
00000002h  Used after setting virtual bank enable bits
03h        Replace ROM at 00000000h by RAM (used after reset)
```
The GUI does additionally read from this register (and gets itself trapped in a bizarre endless loop if bit0 was zero). Unknown if it's possible to re-enable ROM at location 00000000h by writing any other values to this register?

06000004h F_STAT
```
0-31  Unknown
```
The kernel issues a dummy read from this address (before setting F_CTRL to 00000001h).

06000008h F_BANK_FLG ;FLASH virtual bank mapping enable flags (16 bits)(R/W)
```
0-15   Enable physical banks 0..15 in virtual region (0=Disable, 1=Enable)
16-31  Unknown (should be zero)
```

06000100h F_BANK_VAL ;FLASH virtual bank mapping addresses (16 words)(R/W)
This region contains 16 words, the first word at 06000100h for physical bank 0, the last word at 0600013Ch for physical bank 15. Each word is:
```
0-3    Virtual bank number
4-31   Should be 0
```
Unused physical banks are usually mapped to 0Fh (and are additionally disabled in the F_BANK_FLG register).

0600000Ch F_WAIT1 ;waitstates...?
Unknown, seems to control some kind of memory waitstates for FLASH (or maybe RAM or BIOS ROM). Normally it is set to the following values:
```
F_WAIT1=00000000h when CPU Speed = 00h..07h
F_WAIT1=00000010h when CPU Speed = 08h..0Fh
```
Note: The kernels Docking/Undocking IRQ-11 handler does additionally do this: "F_WAIT1=max(08h,(CLK_MODE AND 0Fh))" (that is a bug, what it actually wants to do is to READ the current F_WAIT.Bit4 setting).

06000010h F_WAIT2 ;waitstates, and FLASH-Write-Control-and-Status...?
Unknown, seems to control some kind of memory waitstates, maybe for another memory region than F_WAIT1, or maybe F_WAIT2 is for writing, and F_WAIT1 for reading or so. Normally it is set to the following values:
```
F_WAIT2=00000000h when CPU Speed = 00h..07h  ;\same as F_WAIT1
F_WAIT2=00000010h when CPU Speed = 08h..0Fh  ;/
```
In SWI 0Fh and SWI 10h it is also set to:
```
F_WAIT2=00000021h  ;SWI 10h, FlashWritePhysical(sector,src)
F_WAIT2=00000041h  ;SWI 0Fh, FlashWriteSerial(serial_number)
```
Before completion, those SWIs do additionally,
```
wait until reading returns F_WAIT2.Bit2 = 1
and then set F_WAIT2=00000000h
```

08002A54h - F_KEY1 - Flash Unlock Address 1 (W)
080055AAh - F_KEY2 - Flash Unlock Address 2 (W)
Unlocks FLASH memory for writing. The complete flowchart for writing sector data (or header values) is:
```
if write_sector                          ;\
  F_WAIT2=00000021h                      ; write enable or so
if write_header                          ;
  F_WAIT2=00000041h                      ;/
[80055AAh]=FFAAh                         ;\
[8002A54h]=FF55h                         ; unlock flash
[80055AAh]=FFA0h                         ;/
if write_sector                          ;\
  for i=0 to 3Fh                         ;
    [8000000h+sector*80h+i*2]=src[i*2]   ; write data
if write_header                          ;
  [8000000h]=new F_SN_LO value           ;
  [8000002h]=new F_SN_HI value           ;
  [8000008h]=new F_CAL value             ;/
first, wait 4000 clock cycles            ;\wait
then, wait until F_WAIT2.Bit2=1          ;/
F_WAIT2=00000000h                        ;-write disable or so
```
During the write operation one can (probably?) not read data (nor opcodes) from FLASH memory, so the above code must be executed either in RAM, or in BIOS ROM (see SWI 03h, SWI 0Fh, SWI 10h).

06000300h - F_SN_LO - Serial Number LSBs
06000302h - F_SN_HI - Serial Number MSBs
06000308h - F_CAL - Calibration value for LCD
```
0-15  Data
```
This seems to be an additional "header" region of the FLASH memory (additionally to the 128K of data). The F_SN registers contain a serial number or so (purpose unknown, maybe intended as some kind of an "IP" address for more complex infrared network applications), the two LO/HI registers must be read by separate 16bit LDRH opcodes (not by a single 32bit LDR opcode). The F_CAL register contains a 6bit calibration value for LCD_CAL (contrast or so?).
Although only the above 3 halfwords are used by the BIOS, the "header" is unlike to be 6 bytes in size, probably there are whatever number of additional "header" locations at 06000300h and up...?
Note: Metal Gear Solid Integral uses F_SN as some kind of copy protection (the game refuses to run and displays "No copy" if F_SN is different as when the pocketstation file was initially created).

F_BANK_VAL and F_BANK_FLG Notes
Observe that the physical_bank number (p) is used as array index, and that the virtual bank number (v) is stored in that location, ie. table[p]=v, which is unlike as one may have expected it (eg. on a 80386 CPU it'd be vice-versa: table[v]=p).
Due to the table[p]=v assignment, a physical block cannot be mirrored to multiple virtual blocks, instead, multiple physical blocks can be mapped to the same virtual block (not sure what happens in that case, maybe the data becomes ANDed together).

Pocketstation IO Communication Ports

0C000000h - COM_MODE - Com Mode
```
0     Data Output Enable  (0=None/HighZ, 1=Output Data Bits)
1     /ACK Output Level   (0=None/HighZ, 1=Output LOW)
2     Unknown (should be set when expecting a NEW command...?)
3-31  Unknown (should be zero)
```

0C000000h - COM_DATA - Com RX/TX Data
```
0-7   Data (Write: to be transmitted to PSX, Read: been received from PSX)
8-31  Unknown
```

0C000004h - COM_STAT1 - Com Status Register 1 (Bit1=Error)
```
0     Unknown
1     Error flag or so (0=Okay, 1=Error)
2-31  Unknown
```
Seems to indicate whatever error (maybe /SEL disabled during transfer, or timeout, or parity error or something else?) in bit1. Meaning of the other bits is unknown.
Aside from checking the error flag, the kernel does issue a dummy read at the end of each transfer, maybe to acknowledge something, maybe the hardware simply resets the error bit after reading (although the kernel doesn't handle the bit like so when receiving the 1st command byte).
Aside from the above error flag, one should check if INT_INPUT.11 becomes zero during transfer (which indicates undocking).

0C000014h - COM_STAT2 - Com Status Register 2 (Bit0=Ready)
```
0     Ready flag (0=Busy, 1=Ready) (when 8bits have been transferred)
1-31  Unknown
```

0C000010h - COM_CTRL1 - Com Control Register 1
```
0     Unknown (should be set AT BEGIN OF A NEW command...?)
1     Unknown (0=Disable something, 1=Enable something)
2-31  Unknown (should be zero)
```
Used values are:
```
00000000h = unknown?  disable
00000002h = unknown?  enable
00000003h = unknown?  at BEGIN of a new command
```
When doing the enable thing, Bit1 should be set to 0-then-1...? Bit0 might enable the data shift register... and bit1 might be a master enable and master acknowledge for the COM interrupt... or something else?

0C000018h - COM_CTRL2 - Com Control Register 2
```
0     Unknown (should be set, probably starts or acknowledges something)
1     Unknown (should be set when expecting a NEW command...?)
2-31  Unknown (should be zero)
```
Used values are:
```
00000001h = unknown?  used before AND after each byte-transfer
00000003h = unknown?  used after LAST byte of command (and when init/reset)
```
Maybe that two bits acknowledge the ready/error bits?

INT_INPUT.6 FIQ (!) COM for the COM_registers? (via /SEL Pin?)
```
(via FIQ vector, not IRQ vector)
```

INT_INPUT.11 IRQ Docked ("IOP") (0=Undocked, 1=Docked to PSX)
Probably senses the voltage on the cartridge slots VCC Pin. Becomes zero when Undocked (and probably also when the PSX is switched off).
The Kernel uses IRQ-11 for BOTH sensing docking and undocking, ie. as if the IRQ would be triggered on both 0-to-1 and 1-to-0 transistions... though maybe that feature just relies on switch-bounce. For the same reason (switch bounce), the IRQ-11 handler performs a delay before it checks the new INT_INPUT.11 setting (ie. the delay skips the unstable switch bound period, and allows the signal to stabilize).

IOP_START/IOP_STOP.Bit1
The BIOS adjusts this bit somehow in relation to communication. No idea when/why/how it must be used. For details on IOP_START/IOP_STOP see Power Control chapter.

Opcode E6000010h (The Undefined Instruction) - Write chr(r0) to TTY
This opcode is used by the SN Systems emulator to write chr(r0) to a TTY style text window. r0 can be ASCII characters 20h and up, or 0Ah for CRLF. Using that opcode is a not too good idea because the default BIOS undef instruction handler simply runs into an endless loop, so games that are using it (eg. Break-Thru by Jason) won't work on real hardware. That, unless the game would change the undef instruction vector at [04h] in Kernel RAM, either replacing it by a MOVS R15,R14 opcode (ignore exception and return to next opcode), or by adding exception handling that outputs the character via IR or via whatever cable connection. Observe that an uninitialized FUNC3 accidently destroys [04h], so first init FUNC3 handler via SWI 17h, before trying to change [04h], moreover, mind that SWI 05h may reset FUNC3, causing the problem to reappear.
Altogether, it'd be MUCH more stable to write TTY characters to an unused I/O port... only problem is that it's still unknown which I/O ports are unused... ie. which do neither trap data aborts, nor do mirror to existing ports...?

Pocketstation IO Power Control

0B000000h - CLK_MODE - Clock control (CPU and Timer Speed) (R/W)
```
0-3  Clock Ratio (1-8, 31232Hz SHL N) (usually 7 = 4MHz)    (R/W)
4    Clock Change State (0=Busy, 1=Ready)                   (Read-only)
5-15 ?
```
Allows to change the CPU clock (and Timer clock, which is usually one half of the CPU clock, or less, depending on the Timer Divider). Unknown what happens on value 00h and 09h..0Fh? Not sure if it's really exactly 31232Hz? (32768Hz oscillators are more commonly manufactured). Before changing CLK_MODE, F_WAIT1 and F_WAIT2 should be adjusted accordingly (see there for details).
Maximum CPU operating frequency of about 7.99MHz, but linked to the system clock can be reduced to about 62.5Khz.

0B000004h - CLK_STOP - Clock stop (Sleep Mode)
Stops the CPU until an interrupt occurs. The pocketstation doesn't have a power-switch nor standby button, the closest thing to switch "power off" is to enter sleep mode. Software should do that when the user hasn't pressed buttons for 1-2 seconds (that, only in handheld mode, not when docked to the PSX... apparently it uses the PSX power supply instead of the battery in that case...?).
```
0    Stop Clock (1=Stop)
1-15 ?
```
Wakeup is usually done by IRQ-0 (Fire Button) and IRQ-11 (Docking). If alarm is enabled, then the GUI also enables IRQ-9 (RTC), and compares RTC_TIME against the alarm setting each time when it wakes up.
Before writing to CLK_STOP, one should do:
```
DAC_CTRL=0                          ;\disable sound
IOP_STOP=20h                        ;/
LCD_MODE=0                          ;-disable video
IRDA=whatever                       ;-disable infrared (if it was used)
BATT_CTRL=BATT_CTRL AND FFFFFFFCh   ;-do whatever
INT_MASK_SET=801h                   ;-enable Docking/Fire wakeup interrupts
```
The GUI uses CLK_STOP only for Standby purposes (not for waiting for its 30Hz "frame rate" timer 0 interrupt; maybe that isn't possible, ie. probably CLK_STOP does completely disable the system clock, and thus does stop Timer0-2...?)

0D800000h - IOP_CTRL - Configures whatever...? (R/W)

```
  4-31 Unknown/Unused (seems to be always zero)
```

Unknown. Set to 0000000Fh by BIOS upon reset. Aside from that, the BIOS does never use that register.

0D800004h - IOP_STAT (R) - Read Current bits? -- No, seems to be always 0
0D800004h - IOP_STOP (W) - Set IOP_DATA Bits
0D800008h - IOP_START (W) - Clear IOP_DATA Bits

These two ports are probably accessing a single register, writing "1" bits to IOP_STOP sets bits in that register, and writing "1" bits to IOP_START clears bits... or vice-versa...? Writing "0" bits to either port seems to leave that bits unchanged. The meaning of most bits is still unknown:

```
  0    Unknown, STARTED by Kernel upon reset
  1    Red LED, Communication related (START=Whatever, STOP=Whatelse) (?)
  2    Unknown, STARTED by Kernel upon reset
  3    Unknown, STARTED by Kernel upon reset
  4    Never STARTED nor STOPPED by BIOS (maybe an INPUT, read via IOP_DATA)
  5    Sound Enable (START=On, STOP=Off)
  6    Unknown, STOPPED by Kernel upon reset
  7-31 Unknown, never STARTED nor STOPPED by BIOS
```

Aside from Bit1, it's probably not neccessary to change the unknown bits...?
Sound is usually disabled by setting IOP_STOP=00000020h. IOP_STAT is rarely used. Although, one piece of code in the BIOS disables sound by setting IOP_STOP=IOP_STAT OR 00000020h, that is probably nonsense, probably intended to keep bits stopped if they are already stopped (which would happen anyways), however, the strange code implies that reading from 0D800004h returns the current status of the register, and that the bits in that register seem to be 0=Started, and 1=Stopped...?

0D80000Ch - IOP_DATA (R)

```
  0    ?
  1    Red LED (0=On, 1=Off)
  2    ?
  3    ?
  4    Seems to be always 1 (maybe Infrared input?)
  5-31 Unknown/Unused (seems to be always zero)
```

Unknown. Not used by the BIOS. Reportedly this register is 0010h if IR Connection...? This register is read by Rewrite ID, and by Harvest Moon. Maybe bit4 doesn't mean <if> IR connection exist, but rather <contains> the received IR data level...?

0D800020h - BATT_CTRL - Battery Monitor Control?

Unknown. Somehow battery saving related. Upon reset, and upon leaving sleep mode, the BIOS does set BATT_CTRL=00000000h. Before entering sleep mode, it does set BATT_CTRL=BATT_CTRL AND FFFFFFFCh, whereas, assuming that BATT_CTRL was 00000000h, ANDing it with FFFFFFFCh would simply leave it unchanged... unless the hardware (or maybe a game) sets some bits in BATT_CTRL to nonzero values...?

Battery Low Interrupt

```
  INT_INPUT.10    IRQ     Battery Low     (0=Normal, 1=Battery Low)
```

Can be used to sense if the battery is low, if so, one may disable sound output and/or reduce the CPU speed to increase the remaining battery lifetime. Unknown how long the battery lasts, and how much the lifetime is affected by audio, video, infrared, cpu speed, and sleep mode...? No idea what happens when it becomes empty... ie. if the pocketstation can be still used as memory card when/if it gets powered via the VCC pin on the cartridge slot...?

Pocketstation SWI Function Summary

SWI Function Summary

BIOS functions can be called via SWI opcodes (from both ARM and THUMB mode) (in ARM mode, the SWI function number is in the lower 8bit of the 24bit field; unlike as for example on the GBA, where it'd be in the upper 8bit). Parameters (if any) are passed in r0,r1,r2. Return value is stored in r0 (all other registers are left unchanged).

```
  SWI 00h - Reset()   ;don't use            out: everything destroyed
  SWI 01h - SetCallbacks(index,proc)        out: old proc
  SWI 02h - CustomSwi2(r0..r6,r8..r10)      out: r0
  SWI 03h - FlashWriteVirtual(sector,src)   out: 0=okay, 1=failed
  SWI 04h - SetCpuSpeed(speed)              out: old_speed
  SWI 05h - SenseAutoCom()                  out: garbage
  SWI 06h - GetPtrToComFlags()              out: ptr (usually 0C0h)
  SWI 07h - ChangeAutoDocking(flags.16-18)  out: incoming flags AND 70000h
  SWI 08h - PrepareExecute(flag,dir_index,param)  out: dir_index (new or old)
  SWI 09h - DoExecute(snapshot_saving_flag) out: r0=r0 (failed) or r0=param
  SWI 0Ah - FlashReadSerial()               out: F_SN
  SWI 0Bh - ClearComFlagsBit10()            out: new [ComFlags] (with bit10=0)
  SWI 0Ch - SetBcdDateTime(date,time)       out: garbage (RTC_DATE/10000h)
  SWI 0Dh - GetBcdDate()                    out: date (with century in MSBs)
  SWI 0Eh - GetBcdTime()                    out: time and day-of-week
  SWI 0Fh - FlashWriteSerial(serial_number) out: garbage (r0=0) ;old BIOS only!
  SWI 10h - FlashWritePhysical(sector,src)  out: 0=okay, 1=failed
  SWI 11h - SetComOnOff(flag)               out: garbage retadr to swi handler
  SWI 12h - TestSnapshot(dir_index)         out: 0=normal, 1=MCX1 with 1,0,"SE"
  SWI 13h - GetPtrToAlarmSetting()          out: ptr to alarm_setting
  SWI 14h - GetPtrToPtrToSwiTable()         out: ptr-to-ptr to swi_table
  SWI 15h - MakeAlternateDirIndex(flag,dir_index)  out: alt_dir_index (new/old)
  SWI 16h - GetDirIndex()                   out: dir_index (or alternate)
  SWI 17h - GetPtrToFunc3addr()             out: ptr to func3 address
  SWI 18h - FlashReadWhateverByte(sector)   out: [8000000h+sector*80h+7Eh]
  SWI 19h..FFh - garbage
  SWI 100h..FFFFFFh - mirrors of SWI 00h..FFh
```

The BIOS uses the same memory region for SWI and IRQ stacks, so both may not occur simultaneously, otherwise one stack would be destroyed by the other (normally that is no problem; IRQs are automatically disabled by the CPU during SWI execution, SWIs aren't used from inside of default IRQ handlers, and SWIs shouldn't be used from inside of hooked IRQ handlers).

Pocketstation SWI Misc Functions

SWI 01h - SetCallbacks(index,proc)

```
  r0=0  Set SWI 02h callback              (r1=proc, or r1=0=reset/default)
  r0=1  Set IRQ callback                  (r1=proc, or r1=0=none/default)
  r0=2  Set FIQ callback                  (r1=proc, or r1=0=none/default)
  r0=3  Set Download Notification callback (r1=proc, or r1=0=bugged/default)
```

All callbacks are called via BX opcodes (ie. proc.bit0 can be set for THUMB code). SetCallbacks returns the old proc value (usually zero). The callbacks are automatically reset to zero when (re-)starting an executable, or when returning control to the GUI, so there's no need to restore the values by software.

IRQ and FIQ Callbacks

Registers r0,r1,r12 are pushed by the kernels FIQ/IRQ handlers (so the callbacks can use that registers without needing to push them). The FIQ handler can additionally use r8..r11 without pushing them (the CPU uses a separate set of r8..r12 registers in FIQ mode, nethertheless, the kernel DOES push r12 in FIQ mode, without reason). Available stack is 70h bytes for the FIQ callback, and 64h bytes for the IRQ callback.
The callbacks don't receive any incoming parameters, and don't need to respond with a return value. The callback should return to the FIQ/IRQ handler (via normal BX r14) (ie. it should not try to return to User mode).
The kernel IRQ handler does (after the IRQ callback) process IRQ-11 (IOP) (which does mainly handle docking/undocking), and IRQ-9 (RTC) (which increments the century if the year wrapped from 99h to 00h).
And the kernel FIQ handler does (before the FIQ callback) process IRQ-6 (COM) (which does, if ComFlags.Bit9 is set, handle bu_cmd's) (both IRQs and FIQs are disabled, and the main program is stopped until the bu_cmd finishes, or until a joypad command is identified irrelevant, among others that means that sound/timer IRQs aren't processed during that time, so audio output may become distorted when docked).
When docked, the FIQ callback should consist of only a handful of opcodes, eg. it may contain a simple noise, square wave generator, or software based sound "DMA" function, but it should not contain more time-consuming code like sound envelope processing; otherwise IRQ-6 (COM) cannot be executed fast enough to handle incoming commands.

SWI 02h - CustomSwi2(r0..r6,r8..r10) out: r0
Calls the SWI2 callback function (which can be set via SWI 01h). The default callback address is 00000000h (so, by default, it behaves identically as SWI 00h). Any parameters can be passed in r0..r6 and r8..r10 (the other registers aren't passed to the callback function). Return value can be stored in r0 (all other registers are pushed/popped by the swi handler, as usually). Available space on the swi stack is 38h bytes.
SWI2 can be useful to execute code in privileged mode (eg. to initialize FIQ registers r8..r12 for a FIQ based sound engine) (which usually isn't possible because the main program runs in non-privileged user mode).

SWI 04h - SetCpuSpeed(speed) out: old_speed
Changes the CPU speed. The BIOS uses it with values in range 01h..07h. Not sure if value 00h can be also used? The function also handles values bigger than 07h, of which, some pieces of BIOS code look as if 08h would be the maximum value...?
Before setting the new speed, the function sets F_WAIT1 and F_WAIT2 to 00000000h (or to 00000010h if speed.bit3=1). After changing the speed (by writing the parameter to CLK_MODE) it does wait until the new speed is applied (by waiting for CLK_MODE.bit4 to become zero). The function returns the old value of CLK_MODE, anded with 0Fh.

Pocketstation SWI Communication Functions

Communication (aka BU Commands, received from the PSX via the memory card slot) can be handled by the pocketstations kernel even while a game is running. However, communications are initially disabled when starting a game, so the game should enable them via SWI 11h, and/or via calling SWI 05h once per frame.

SWI 11h - SetComOnOff(flag)
Can be used to enable/disable communication. When starting an executable, communication is initially disabled, so it'd be a good idea to enable them (otherwise the PSX cannot communicate with the Pocketstation while the game is running).
When flag=0, disables communication: Intializes the COM_registers, disables IRQ-6 (COM), and clears ComFlags.9. When flag=1, enables communication: Intializes the COM_registers, enables IRQ-6 (COM), sets ComFlags.9 (when docked), or clears Sys.Flags.9 (when undocked), and sets FAST cpu_speed=7 (only when docked). The function returns garbage (r0=retadr to swi_handler).

SWI 06h - GetPtrToComFlags()
Returns a pointer to the ComFlags word in RAM, which contains several communication related flags (which are either modified upon docking/undocking, or upon receiving certain bu_cmd's). The ComFlags word consists of the following bits:
```
0-3   Whatever (set/cleared when docked/undocked, and modified by bu_cmd's)
4-7   Not used (should be zero)
8     IRQ-11 (IOP) occurred  (set by irq handler, checked/cleared by SWI 05h)
9     Communication Enabled And Docked (0=No, 1=Yes; prevents DoExecute)
10    Reject writes to Broken Sector Region (sector 16..55)     (0=No, 1=Yes)
11    Start file request (set by bu_cmd_59h, processed by GUI, not by Kernel)
12-15 Not used (should be zero)
16    Automatically power-down DAC audio on insert/removal      (0=No, 1=Yes)
17    Automatically power-down IRDA infrared on insert/removal  (0=No, 1=Yes)
18    Automatically adjust LCD screen rotate on insert/removal  (0=No, 1=Yes)
19-27 Not used (should be zero)
28    Indicates if a standard bu_cmd (52h/53h/57h) was received (0=No, 1=Yes)
29    Set date/time request  (set by bu_cmd FUNC0, processed by BIOS)
30    Destroy RTC and Start GUI request  (set by bu_cmd_59h, dir_index=FFFEh)
31    Not used (should be zero)
```
Bit16-18 can be changed via SWI 07h, ChangeAutoDocking(flags). Bit10 can be cleared by SWI 0Bh, ClearComFlagsBit10().

SWI 07h - ChangeAutoDocking(flags.16-18)
```
0-15  Not used (should be zero)
16    Automatically power-down DAC audio on insert/removal     (0=No, 1=Yes)
17    Automatically power-down IRDA infrared on insert/removal (0=No, 1=Yes)
18    Automatically adjust LCD screen rotate on insert/removal (0=No, 1=Yes)
19-31 Not used (should be zero)
```
Copies bit16-18 of the incoming parameter to ComFlags.16-18, specifying how the kernel IRQ-11 (IOP) handler shall process docking/undocking from the PSX cartridge slot. The function returns the incoming flags value ANDed with 70000h.

SWI 0Bh - ClearComFlagsBit10()
Resets ComFlags.Bit10, ie. enables bu_cmd_57h (write_sector) to write to the Broken Sector region in FLASH memory (sector 16..55). SWI 0Bh returns the current ComFlags value (the new value, with bit10=0).
Aside from calling SWI 0Bh, ComFlags.10 is also automatically cleared upon IRQ-10 (IOP) (docking/undocking). ComFlags.10 can get set/cleared by the Download Notification callback.

SWI 05h - SenseAutoCom()
Checks if docking/undocking has occurred (by examining ComFlags.8, which gets set by the kernel IRQ-11 (IOP) handler). If that flag was set, then the function does reset it, and does then reset FUNC3=0000h and [0CAh]=00h (both only if docked, not when undocked), and, no matter if docked or undocked, it enables communication; equivalent to SetComOnOff(1); which sets/clears ComFlags.9. The function returns garbage (r0=whatever).
The GUI is calling SWI 05h once per frame. The overall purpose is unknown. It's a good idea to reset FUNC3 and to Enable Communication (although that'd be required only when docked, not when undocked), but SWI 05h is doing that only on (un-)docking transitions (not when it was already docked). In general, it'd make more sense to do proper initializations via SWI 11h and SWI 17h as than trusting SWI 05h to do the job. The only possibly useful effect is that SWI 05h does set/clear ComFlags.9 when docked/undocked.

SWI 17h - GetPtrToFunc3addr()
Returns a pointer to a halfword in RAM which contains the FUNC3 address (for bu_cmd_5bh and bu_cmd_5ch). The address is only 16bit, originated at 02000000h in FLASH (ie. it can be only in the first 64K of the file), bit0 can be set for THUMB code. The default address is zero, which behaves bugged: It accidently sets [00000004h]=00000000h, ie. replaces the Undefined Instruction exception vector by a "andeq r0,r0,r0" opcode, due to that NOP-like opcode, any Undefined

Instruction exceptions will run into the SWI vector at [00000008h], and randomly execute an SWI function, with some bad luck that may execute one of the FlashWrite functions and destroy the saved files.
Although setting 0000h acts bugged, one should restore that setting before returning control to GUI or other executables; otherwise the address would still point to the FUNC3 address of the old unloaded executable, which is worse than the bugged effect.
The FUNC3 address is automatically reset to 0000h when (if) SWI 05h (SenseAutoCom) senses new docking.

Download Notification callback
Can be used to mute sound during communication, see SWI 01h, SetCallbacks(index,proc), and BU Command 5Dh for details.

Pocketstation SWI Execute Functions

SWI 08h - PrepareExecute(flag,dir_index,param)
dir_index should 0=GUI, or 1..15=First block of game. When calling DoExecute, param is passed to the entrypoint of the game or GUI in r0 register (see notes on GUI <param> values belows). For games, param may be interpreted in whatever way.
When flag=0, the function simply returns the old dir_index value. When flag=1, the new dir_index and param values are stored in Kernel RAM (for being used by DoExecute); the values are stored only if dir_index=0 (GUI), or if dir_index belongs to a file with "SC" and "MCX0" or "MCX1" IDs in it's title sector. If dir_index was accepted, then the new dir_index value is returned, otherwise the old dir_index is returned.

GUI <param> values - for PrepareExecute(1,0,param)
PrepareExecute(1,0,param) prepares to execute the GUI (rather than a file). When executing the GUI, <param> consists of the following destructive bits:
```
0-7  Command number (see below, MSBs=Primary command, LSBs=another dir_index)
8    Do not store Alarm setting in Kernel RAM (0=Normal, 1=Don't store)
9-31 Not used (should be zero)
```
The command numbers can be:
```
Command 0xh --> Erase RTC time/date
Command 1xh --> Enter GUI Time Screen with speaker symbol
Command 20h --> Enter GUI Time Screen with alarm symbol
Command 2xh --> Prompt for new Date/Time, then start dir_index (x)
Command 3xh --> Enter GUI File Selection Screen, with dir_index (x) selected
Command xxh --> Erase RTC time/date (same as Command 0xh)
```
For Command 2xh and 3xh, the lower 4bit of the command (x) must be a valid dir_index of the 1st block of a pocketstation executable, otherwise the BIOS erases the RTC time/date. Bit8 is just a "funny" nag feature, allowing the user to change the alarm setting, but with the changes being ignored (bit8 can be actually useful in BU Command 59h, after FUNC2 was used for changing alarm).

SWI 09h - DoExecute(), or DoExecute(snapshot_saving_flag) for MCX1
Allows to return control to the GUI (when dir_index=0), or to start an executable (when dir_index=1..15). Prior to calling DoExecute, parameters should be set via PrepareExecute(1,dir_index,param), when not doing that, DoExecute would simply restart the current executable (which may be a desired effect in some cases).
The "snapshot_saving_flag" can be ommited for normal (MCX0) files, that parameter is used only for special (MCX1) files (see Snapshot Notes for details).
Caution: DoExecute fails (and returns r0=unchanged) when ComFlags.9=1 (which indicates that communications are enabled, and that the Pocketstation is believed to be docked to the PSX). ComFlags.9 can be forcefully cleared by calling SetComOnOff(0), or it can be updated according to the current docking-state by calling SetComOnOff(1) or SenseAutoCom().

SWI 16h - GetDirIndex()
Returns the dir_index for the currently executed file. If that value is zero, ie. if there is no file executed, ie. if the function is called by the GUI, then it does instead return the "alternate" dir_index (as set via SWI 15h).

SWI 15h - MakeAlternateDirIndex(flag,dir_index) out: alt_dir_index (new/old)
Applies the specified dir_index as "alternate" dir_index (for being retrieved via SWI 16h for whatever purpose). The dir_index is applied only when flag=1, and only if dir_index is 0=none, or if it is equal to the dir_index of the currently executed file (ie. attempts to make other files being the "alternate" one are rejected). If successful, the new dir_index is returned, otherwise the old dir_index is returned (eg. if flag=0, or if the index was rejected).

SWI 12h - TestSnapshot(dir_index)
Tests if the specified file contains a load-able snapshot, ie. if it does have the "SC" and "MCX1" IDs in the title sector, and the 01h,00h,"SE" ID in the snapshot header. If so, it returns r0=1, and otherwise returns r0=0.

Snapshot Notes (MCX1 Files)
Snapshots are somewhat automatically loaded/saved when calling DoExecute:
If the old file (the currently executed file) contains "SC" AND "MCX1" IDs in the title sector, then the User Mode CPU registers and User RAM at 200h..7FFh are automatically saved in the files snapshot region in FLASH memory, with the snapshot_saving_flag being applied as bit0 of the 0xh,00h,"SE" ID of the snapshot header).
If the new file (specified in dir_index) contains load-able snapshot data (ie. if it has "SC" and "MCX1" IDs in title sector, and 01h,00h,"SE" ID in the snapshot region), then the BIOS starts the saved snapshot data (instead of restarting the executable at its entrypoint). Not too sure if that feature is really working... the snapshot loader seems to load User RAM from the wrong sectors... and it seems to jump directly to User Mode return address... without removing registers that are still stored on SWI stack... causing the SWI stack to underflow after loading one or two snapshots...?

Pocketstation SWI Date/Time/Alarm Functions

SWI 0Ch - SetBcdDateTime(date,time)
Sets the time and date, the parameters are having the same format as SWI 0Dh and SWI 0Eh return values (see there). The SWI 0Ch return value contains only garbage (r0=RTC_DATE/10000h).

SWI 0Dh - GetBcdDate()
```
0-7   Day     (01h..31h, BCD)
8-11  Month   (01h..12h, BCD)
16-31 Year    (0000h..9999h, BCD)
```
Returns the current date, the lower 24bit are read from RTC_DATE, the century in upper 8bit is read from Kernel RAM.

SWI 0Eh - GetBcdTime()
```
0-7   Seconds     (00h..59h, BCD)
8-15  Minutes     (00h..59h, BCD)
16-23 Hours       (00h..23h, BCD)
24-31 Day of week (1=Sunday, ..., 7=Saturday)
```
Returns the current time and day of week, read from RTC_TIME.

SWI 13h - GetPtrToAlarmSetting()
Returns a pointer to a 64bit value in Kernel RAM, the upper word (Bit32-63) isn't actually used by the BIOS, except that, the bu_cmd FUNC3 does transfer the

whole 64bits. The meaning of the separate bits is:

```
  0-7   Alarm Minute    (00h..59h, BCD)
  8-15  Alarm Hour      (00h..23h, BCD)
  16    Alarm Enable    (0=Off, 1=On)
  17    Button Lock     (0=Normal, 1=Lock) (pressing all 5 buttons in GUI)
  18-19 Volume Shift    (0=Normal/Loud, 1=Medium/Div4, 2=Mute/Off)
  20-22 Not used        (should be zero)
  23    RTC Initialized (0=Not yet, 1=Yes, was initialized from within GUI)
  24-31 Not used        (should be zero)
  32-63 Pointer to 8x8 BIOS Charset (characters "0"..."9" plus strange symbols)
```

The RTC hardware doesn't have a hardware-based alarm feature, instead, the alarm values must be compared with the current time by software. Alarm is handled only by the GUI portion of the BIOS. The Kernel doesn't do any alarm handling, so alarm won't occur while a game is executed (unless the game contains code that handles alarm).

Games are usually using only the lower 16bit of the charset address, ORed with 04000000h (although the full 32bit is stored in RAM).

```
  CHR(00h..09h) = Digits "0..9"
  CHR(0Ah) = Space " "
  CHR(0Bh) = Colon ":"
  CHR(0Ch) = Button Lock (used by Final Fantasy 8's Chocobo World)
  CHR(0Dh) = Speaker Medium; or loud if followed by chr(0Eh)
  CHR(0Eh) = Speaker Loud; to be appended to chr(0Dh)
  CHR(0Fh) = Speaker Off
  CHR(10h) = Battery Low (used by PocketMuuMuu's Cars)
  CHR(11h) = Alarm Off
  CHR(12h) = Alarm On
  CHR(13h) = Memory Card symbol
```

Pocketstation SWI Flash Functions

SWI 10h - FlashWritePhysical(sector,src)
Writes 80h-bytes at src to the physical sector number (0..3FFh, originated at 08000000h), and does then compare the written data with the source data. Returns 0=okay, or 1=failed.

SWI 03h - FlashWriteVirtual(sector,src)
The sector number (0..3FFh) is a virtual sector number (originated at 02000000h), the function uses the F_BANK_VAL settings to translate it to a physical sector number, and does then write the 80h-bytes at src to that location (via the FlashWritePhysical function). Returns 0=okay, or 1=failed (if the write failed, or if the sector number exceeded the filesize aka the virtually mapped memory region).

SWI 0Ah - FlashReadSerial()
Returns the 32bit value from the two 16bit F_SN registers (see F_SN for details).

SWI 0Fh - FlashWriteSerial(serial_number) ;old BIOS only!
Changes the 32bit F_SN value in the "header" region of the FLASH memory. The function also rewrites the F_CAL value (but it simply rewrites the old value, so it's left unchanged). The function isn't used by the BIOS, no idea if it is used by any games. No return value (always returns r0=0).
This function is supported by the old "061" version BIOS only (the function is padded with jump opcodes which hang the CPU in endless loops on newer "110" version).

SWI 18h - FlashReadWhateverByte(sector)
Returns [8000000h+sector*80h+7Eh] AND 00FFh. Purpose is totally unknown... the actual FLASH memory doesn't contain any relevant information at that locations (eg. the in the directory sectors, that byte is unused, usually zero)... and, reading some kind of status or manufacturer information would first require to command the hardware to output that info...?

Pocketstation SWI Useless Functions

SWI 00h - Reset() ;don't use, destroys RTC settings
Reboots the pocketstation, similar as when pressing the Reset button. Don't use! The BIOS bootcode does (without any good reason) reset the RTC registers and alarm/century settings in RAM to Time 00:00:00, Date 01 Jan 1999, and Alarm 00:00 disabled, so, after reset, the user would need to re-enter that values.
Aside from the annoying destroyed RTC settings, the function is rather unstable: it does jump to address 00000000h in RAM, which should usually redirect to 04000000h in ROM, however, most pocketstation games are programmed in C language, where "pointer" is usually pronounced "pointer?" without much understanding of whether/why/how to initialize that "strange things", so there's a good probability that one of the recently executed games has accidently destroyed the reset vector at [00000000h] in battery-backed RAM.

SWI 14h - GetPtrToPtrToSwiTable()
Returns a pointer to a word in RAM, which contains another pointer which usually points to SWI table in ROM. Changing that word could be (not very) useful for setting up a custom SWI table in FLASH or in RAM. When doing that, one must restore the original setting before returning control to the GUI or to another executable (the setting isn't automatically restored).

Pocketstation BU Command Summary

The Pocketstation supports the standard Memory Card commands (Read Sector, Write Sector, Get Info), plus a couple of special commands.

BU Command Summary

```
  50h  Change a FUNC 03h related value or so
  51h  N/A
  52h  Standard Read Sector command
  53h  Standard Get ID command
  54h  N/A
  55h  N/A
  56h  N/A
  57h  Standard Write Sector command
  58h  Get an ID or Version value or so
  59h  Prepare File Execution with Dir_index, and Parameter
  5Ah  Get Dir_index, ComFlags, F_SN, Date, and Time
  5Bh  Execute Function and transfer data from Pocketstation to PSX
  5Ch  Execute Function and transfer data from PSX to Pocketstation
  5Dh  Execute Custom Download Notification Function   ;via SWI 01h with r0=3
  5Eh  Get-and-Send ComFlags.bit1,3,2
  5Fh  Get-and-Send ComFlags.bit0
```

Commands 5Bh and 5Ch can use the following functions:

```
FUNC 01h - Get or Set Memory Block
FUNC 02h - Get or Set Alarm/Flags
FUNC 03h - Custom Function 3              ;via SWI 17h, GetPtrToFunc3addr()
FUNC 80h..FFh - Custom Functions 80h..FFh ;via Function Table in File Header
```

Pocketstation BU Standard Memory Card Commands

For general info on the three standard memory card commands (52h, 53h, 57h), and for info on the FLAG response value, see:
Memory Card Read/Write Commands

BU Command 52h (Read Sector)
Works much as on normal memory cards, except that, on the Pocketstation, the Read Sector command return 00h as dummy values; instead of the "(pre)" dummies that occur on normal memory cards.
The Read Sector command does reproduce the strange delay (that occurs between 5Ch and 5Dh bytes), similar as on normal original Sony memory cards, maybe original cards did (maybe) actually DO something during that delay period, the pocketstation BIOS simply blows up time in a wait loop (maybe for compatibility with original cards).

BU Command 53h (Get ID)
The Get ID command (53h) returns exactly the same values as normal original Sony memory cards.

BU Command 57h (Write Sector)
The Write Sector command has two new error codes (additonally to the normal 47h="G"=Good, 4Eh="N"=BadChecksum, FFh=BadSector responses). The new error codes are (see below for details):

```
FDh Reject write to Directory Entries of currently executed file
FEh Reject write to write-protected Broken Sector region (sector 16..55)
```

And, like Read Sector, it returns 00h instead of "(pre)" as dummy values.

Write Error Code FDh (Directory Entries of currently executed file)
The FDh error code is intended to prevent the PSX bootmenu (or other PSX games) to delete the currently executed file (which would crash the pocketstation - once when the deleted region gets overwritten by a new file), because the PSX bootmenu and normal PSX games do not recognize the new FDh error code the pocketstation does additionally set FLAG.3 (new card), which should be understood by all PSX programs.
The FDh error code occurs only on directory sectors of the file (not on its data blocks). However, other PSX games should never modify files that belong to other games (so there should be no compatibility problem with other PSX programs that aren't aware of the file being containing currently executed code).
However, the game that has created the executable pocketstation file must be aware of that situation. If the file is broken into a Pocketstation Executable region and a PSX Gameposition region, then it may modify the Gameposition stuff even while the Executable is running. If the PSX want to overwrite the executable then it must first ensure that it isn't executed (eg. by retrieving the dir_index of the currently executed file via BU Command 5Ah, and comparing it against the first block number in the files FCB at the PSX side; for file handle "fd", the first block is found at "[104h]+fd*2Ch+24h" in PSX memory).

Write Error Code FEh (write-protected Broken Sector region, sector 16..55)
The write-protection is enabled by ComFlags.bit10 (which can be set/cleared via BU Command 5Dh). That bit should be set before writing Pocketstation excecutables (the Virtual Memory banking granularity is 2000h bytes, which allows to map whole blocks only, but cannot map single sectors, which would be required for files with broken sector replacements).
Unlike Error FDh, this error code doesn't set FLAG.3 for notifying normal PSX programs about the error (which is no problem since normally Error FEh should never occur since ComFlags.10 is usually zero). For more info on ComFlags.10, see SWI 0Bh aka ClearComFlagsBit10(), and BU Command 5Dh.

Pocketstation BU Basic Pocketstation Commands

BU Command 50h (Change a FUNC 03h related value or so)

```
Send Reply Comment
81h  N/A   Memory Card Access
50h  FLAG  Send Command 50h
VAL  00h   Send new [0CAh], receive length of following data (00h)
```

Might be somehow related to FUNC 03h...?

BU Command 58h (Get an ID or Version value or so)

```
Send Reply Comment
81h  N/A   Memory Card Access
58h  FLAG  Send Command 58h
(0)  02h   Send dummy/zero, receive length of following data (02h)
(0)  01h   Send dummy/zero, receive whatever value           (01h)
(0)  01h   Send dummy/zero, receive another value            (01h)
```

BU Command 59h (Prepare File Execution with Dir_index, and Parameter)

```
Send Reply Comment
81h  N/A   Memory Card Access
59h  FLAG  Send Command 59h
(0)  06h   Send dummy/zero, receive length of following data (06h)
NEW  OLD   Send new dir_index.8-15, receive old dir_index.8-15
NEW  OLD   Send new dir_index.0-7, receive old dir_index.0-7
PAR  (0)   Send exec_parameter.0-7, receive dummy/zero
PAR  (0)   Send exec_parameter.8-15, receive dummy/zero
PAR  (0)   Send exec_parameter.16-23, receive dummy/zero
PAR  (0)   Send exec_parameter.24-31, receive dummy/zero
```

The new dir_index can be the following:

```
0000h..000Fh --> Request to Start GUI or File (with above parameter bits)
0010h..FFFDh --> Not used, acts same as FFFFh (see below)
FFFEh --> Request to Destroy RTC and Start GUI (with parameter 00000000h)
FFFFh --> Do nothing (transfer all bytes, but don't store the new values)
```

Upon dir_index=0000h (Start GUI) or 0001..000Fh (start file), a request flag in ComFlags.11 is set, the GUI does handle that request, but the Kernel doesn't handle it (so it must be handled in the game; ie. check ComFlags.11 in your mainloop, and call DoExecute when that bit is set, there's no need to call PrepareExecute, since that was already done by the BU Command).
Caution: When dir_index=0000h, then <param> should be a value that does NOT erase the RTC time/date (eg. 10h or 20h) (most other values do erase the RTC, see SWI 08h for details).
Upon dir_index=FFFEh, a similar request flag is set in ComFlags.30, and, the Kernel (not the GUI) does handle that request in its FIQ handler (however, the request is: To reset the RTC time/date and to start the GUI with uninitialized irq/svc stack pointers, so this unpleasant and bugged feature shouldn't ever be used). Finally, dir_index=FFFFh allows to read the current dir_index value (which could be also read via BU Command 5Ah).

BU Command 5Ah (Get Dir_index, ComFlags, F_SN, Date, and Time)

```
  Send Reply Comment
  81h  N/A   Memory Card Access
  5Ah  FLAG  Send Command 5Ah
  (0)  12h   Send dummy/zero, receive length of following data (12h)
  (0)  INDX  Send dummy/zero, receive curr_dir_index.bit8-15   (00h)
  (0)  INDX  Send dummy/zero, receive curr_dir_index.bit0-7    (00h..0Fh)
  (0)  FLG   Send dummy/zero, receive ComFlags.bit0            (00h or 01h)
  (0)  FLG   Send dummy/zero, receive ComFlags.bit1            (00h or 01h)
  (0)  FLG   Send dummy/zero, receive ComFlags.bit3            (00h or 01h)
  (0)  FLG   Send dummy/zero, receive ComFlags.bit2            (00h or 01h)
  (0)  SN    Send dummy/zero, receive F_SN.bit0-7              (whatever)
  (0)  SN    Send dummy/zero, receive F_SN.bit8-15             (whatever)
  (0)  SN    Send dummy/zero, receive F_SN.bit16-23            (whatever)
  (0)  SN    Send dummy/zero, receive F_SN.bit24-31            (whatever)
  (0)  DATE  Send dummy/zero, receive BCD Day                  (01h..31h)
  (0)  DATE  Send dummy/zero, receive BCD Month                (01h..12h)
  (0)  DATE  Send dummy/zero, receive BCD Year                 (00h..99h)
  (0)  DATE  Send dummy/zero, receive BCD Century              (00h..99h)
  (0)  TIME  Send dummy/zero, receive BCD Second               (00h..59h)
  (0)  TIME  Send dummy/zero, receive BCD Minute               (00h..59h)
  (0)  TIME  Send dummy/zero, receive BCD Hour                 (00h..23h)
  (0)  TIME  Send dummy/zero, receive BCD Day of Week          (01h..07h)
```

At midnight, the function may accidently return the date for the old day, and the time for the new day.

BU Command 5Eh (Get-and-Send ComFlags.bit1,3,2)

```
  Send Reply Comment
  81h  N/A   Memory Card Access
  5Eh  FLAG  Send Command 5Eh
  (0)  03h   Send dummy/zero, receive length of following data (03h)
  NEW  OLD   Send new ComFlags.bit1, receive old ComFlags.bit1 (00h or 01h)
  NEW  OLD   Send new ComFlags.bit3, receive old ComFlags.bit3 (00h or 01h)
  NEW  OLD   Send new ComFlags.bit2, receive old ComFlags.bit2 (00h or 01h)
```

BU Command 5Fh (Get-and-Send ComFlags.bit0)

```
  Send Reply Comment
  81h  N/A   Memory Card Access
  5Fh  FLAG  Send Command 5Fh
  (0)  01h   Send dummy/zero, receive length of following data (01h)
  NEW  OLD   Send new ComFlags.bit0, receive old ComFlags.bit0 (00h or 01h)
```

Pocketstation BU Custom Pocketstation Commands

BU Command 5Bh (Execute Function and transfer data from Pocketstation to PSX)

```
  Send Reply Comment
  81h  N/A   Memory Card Access
  5Bh  FLAG  Send Command 5Bh
  FUNC FFh   Send Function Number, receive FFh (indicating variable length)
  (0)  LEN1  Send dummy/zero, receive length of parameters (depending on FUNC)
  ...  (0)   Send parameters (LEN1 bytes), and receive dummy/zero
   <-------- at this point, the function is executed for the first time
  (0)  LEN2  Send dummy/zero, receive length of data (depending on FUNC)
  (0)  ...   Send dummy/zero, receive data (LEN2 bytes) from pocketstation
  (0)  FFh   Send dummy/zero, receive FFh
   <-------- at this point, the function is executed for the second time
```

See below for more info on the FUNC value and the corresponding functions.

BU Command 5Ch (Execute Function and transfer data from PSX to Pocketstation)

```
  Send Reply Comment
  81h  N/A   Memory Card Access
  5Ch  FLAG  Send Command 5Ch
  FUNC FFh   Send Function Number, receive FFh (indicating variable length)
  (0)  LEN1  Send dummy/zero, receive length of parameters (depending on FUNC)
  ...  (0)   Send parameters (LEN1 bytes), and receive dummy/zero
   <-------- at this point, the function is executed for the first time
  (0)  LEN2  Send dummy/zero, receive length of data (depending on FUNC)
  ...  (0)   Send data (LEN2 bytes) to pocketstation, receive dummy/zero
  (0)  FFh   Send dummy/zero, receive FFh
   <-------- at this point, the function is executed for the second time
```

See below for more info on the FUNC value and the corresponding functions.

BU Command 5Dh (Execute Custom Download Notification Function)

Can be used to notify the GUI (or games that do support this function) about following "download" operations (or uploads or other BU commands).
BU commands are handled inside of the kernels FIQ handler, that means both IRQs and FIQs are disabled during a BU command transmission, so any IRQ or FIQ based audio frequency generators will freeze during BU commands. To avoid distorted noise, it's best to disable sound for the duration specified in bit0-7. If the PSX finishes before the originally specified duration has expired, then it can resend this command with bit8=1 to notify the pocketstation that the "download" has completed.

```
  Send Reply Comment
  81h  N/A   Memory Card Access
  5Dh  FLAG  Send Command 5Dh
  (0)  03h   Send dummy/zero, receive length of following data (03h)
  VAL  (0)   Send receive value.16-23 (whatever), receive dummy/zero
  VAL  (0)   Send receive value.8-15 (download flags), receive dummy/zero
  VAL  (0)   Send receive value.0-7 (download duration), receive dummy/zero
```

The Download Notification callback address can be set via SWI 01h, SetCallbacks(3,proc), see there for details. At kernel side, the function execution is like so:

```
  If value.8-15 = 00h, then ComFlags.bit10=1, else ComFlags.bit10=0.
  If download_callback<>0 then call download_callback with r0=value.0-23.
```

In the GUI, the bu_cmd_5dh_hook/callback handles parameter bits as so (and games should probably handle that bits in the same fashion, too):

```
  bit0-7  download duration   (in whatever units... 30Hz, RTC, seconds...?)
  bit8    download finished   (0=no, 1=yes, cancel any old/busy duration)
  bit9-23 not used by gui
```

If PSX games send any of the standard commands (52h,53h,57h) to access the memory card without using command 5Dh, then GUI automatically sets the duration to 01h (and pauses sound only for that short duration).

FUNC 00h - Get or Set Date/Time (FUNC0)

LEN1 is 00h (no parameters), and LEN2 is 08h (eight data bytes):

```
  DATE  Get or Send BCD Day         (01h..31h)
  DATE  Get or Send BCD Month       (01h..12h)
```

```
  DATE  Get or Send BCD Century     (00h..99h)
  TIME  Get or Send BCD Second      (00h..59h)
  TIME  Get or Send BCD Minute      (00h..59h)
  TIME  Get or Send BCD Hour        (00h..23h)
  TIME  Get or Send BCD Day of Week (01h..07h)
```

At midnight, the function may accidently return the date for the old day, and the time for the new day.

FUNC 01h - Get or Set Memory Block (FUNC1)

LEN1 is 05h (five parameters bytes):

```
  ADDR  Send Pocketstation Memory Address.bit0-7
  ADDR  Send Pocketstation Memory Address.bit8-15
  ADDR  Send Pocketstation Memory Address.bit16-23
  ADDR  Send Pocketstation Memory Address.bit24-31
  LEN2  Send Desired Data Length (00h..80h, automatically clipped to max=80h)
```

LEN2 is variable (using the 5th byte of the above parameters):

```
  ...   Get or Send LEN2 Data byte(s), max 80h bytes
```

Can be used to write to RAM (and eventually also to I/O ports; when you know what you are doing). In the read direction it can read almost anything: RAM, BIOS ROM, I/O Ports, Physical and Virtual FLASH memory. Of which, trying to read unmapped Virtual FLASH does probably (?) cause a Data Abort exception (and crash the Pocketstation), so that region may be read only if a file is loaded (check that dir_index isn't zero, via BU Command 5Ah, and, take care not to exceed the filesize of that file).

BUG: When sending more than 2 data bytes in the PSX-to-Pocketstation direction, then ADDR must be word-aligned (the BIOS tries to handle odd destination addresses, but when doing that, it messes up the alignment of another internal pointer).

FUNC 02h - Get or Set Alarm/Flags (FUNC2)

LEN1 is 00h (no parameters), and LEN2 is 08h (eight data bytes):

```
  DATA  Get or Send Alarm.bit0-7,   Alarm Minute    (00h..59h, BCD)
  DATA  Get or Send Alarm.bit8-15,  Alarm Hour      (00h..23h, BCD)
  DATA  Get or Send Alarm.bit16-23, Flags, see SWI 13h, GetPtrToAlarmSetting()
  DATA  Get or Send Alarm.bit24-31, Not used (usually 00h)
  DATA  Get or Send Alarm.bit32-39, BIOS Charset Address.0-7
  DATA  Get or Send Alarm.bit40-47, BIOS Charset Address.8-15
  DATA  Get or Send Alarm.bit48-55, BIOS Charset Address.16-23
  DATA  Get or Send Alarm.bit56-63, BIOS Charset Address.24-31
```

Changing the alarm value while the GUI is running works only with some trickery: For a sinister reason, the GUI copies the alarm setting to User RAM when it gets started, that copy isn't affected by FUNC2, so the GUI believes that the old alarm setting does still apply (and writes that old values back to Kernel RAM when leaving the GUI). The only workaround is:

Test if the GUI is running, if so, restart it via Command 59h (with dir_index=0, and param=0120h or similar, ie. with param.bit8 set), then execute FUNC2, then restart the GUI again (this time with param.bit8 zero).

FUNC 03h - Custom Function 3 (aka FUNC3)

LEN1 is 04h (fixed) (four parameters bytes):

```
  VAL   Send Parameter Value.bit0-7
  VAL   Send Parameter Value.bit8-15
  VAL   Send Parameter Value.bit16-23
  VAL   Send Parameter Value.bit24-31
```

LEN2 is variable (depends on the return value of the 1st function call):

```
  ...   Get or Send LEN2 Data byte(s)
```

The function address can be set via SWI 17h, GetPtrToFunc3addr(), see there for details.

Before using FUNC 03h one must somehow ensure that the desired file is loaded (and that it does have initialized the function address via SWI 17h, otherwise the pocketstation would crash).

The FUNC3 address is automatically reset to 0000h when (if) SWI 05h (SenseAutoCom) senses new docking.

Note: The POC-XBOO circuit uses FUNC3 to transfer TTY debug messages.

FUNC 80h..FFh - Custom Function 80h..FFh

LEN1 is variable (depends on the LEN1 value in Function Table in File Header):

```
  ...   Send LEN1 Parameter Value(s), max 80h bytes (destroys Kernel when >80h)
```

LEN2 is variable (depends on the return value of the 1st function call):

```
  ...   Get or Send LEN2 Data byte(s), max 80h bytes (clipped to max=80h)
```

The function addresses (and LEN1 values) are stored in the Function Table FLASH memory (see Pocketstation File Header for details).

```
     ;above LEN1 should be 00h..80h (the parameters are stored
     ;in a 80h-byte buffer in kernel RAM, so len LEN1=81h..FFh would
     ;destroy the kernel RAM that is located after that buffer)
```

Before using FUNC 80h..FFh one must somehow ensure that the desired file is loaded (ie. that the function table with the desired functions is mapped to flash memory; otherwise the pocketstation would crash).

First Function Call (Pre-Data)

Incoming parameters on 1st Function Call:

```
  r0=flags (09h=Pre-Data to PSX, or 0Ah=Pre-Data from PSX)
  r1=pointer to parameter buffer (which contains LEN1 bytes) (in Kernel RAM)
```

Return Value on 1st Function Call:

```
  r0 = Pointer to 64bit memory location (or r0=00000000h=Failed)
```

That 64bits are:

```
  0-31    BUF2 address of data buffer (src/dst)
  32-63   LEN2 (00000000h..00000080h) (clipped to max 00000080h if bigger)
```

dst is written in 8bit units

src is read in 16bit units (and then split to 8bit units)

Second Function Call (Post-Data)

Incoming parameters on 2nd Function Call:

```
  r0=flags (11h=Post-Data to PSX, 12h=Post-Data from PSX; plus 04h if Bad-Data)
  r1=pointer to data buffer (which contains LEN2 bytes) (BUF2 address)
```

Return Value on 2nd Function Call:

```
  There's no return value required on 2nd call (although the kernel
  functions seem to return the same stuff as on 1st call).
```

Function flags (r0)

For each function, there is only one single function vector which is called for both To- and From-PSX, and both Pre- and Post-Data, and also on errors. The function must decipher the flags in r0 to figure out which of that operations it should handle:

```
  0    To-PSX   (when used by Command 5Bh)
  1    From-PSX (when used by Command 5Ch)
  2    Error occurred during Data transfer
  3    Pre-Data
  4    Post-Data
```

There are only six possible flags combinations:

```
09h Pre-Data to PSX
0Ah Pre-Data from PSX
11h Post-Data to PSX
12h Post-Data from PSX
15h Post-Bad-Data to PSX
16h Post-Bad-Data from PSX
```

The kernel doesn't call FUNC 03h if the Error bit is set (ie. Post-Bad-Data needs to be handled only by FUNC 80h..FFh, not by FUNC 03h.)

Pocketstation File Header/Icons

Pocketstation File Content

Pocketstation files consists of the following elements (in that order):

```
PSX Title Sector              ;80h bytes
PSX Colored Icon(s)           ;(hdr[02h] AND 0Fh)*80h bytes
Pocketstation Saved Snapshot  ;800h bytes if hdr[52h]="MCX1", else 0 bytes
Pocketstation Function Table  ;(hdr[57h]*8+7Fh) AND NOT 7Fh bytes
Pocketstation File Viewer Mono Icon     ;hdr[50h]*80h bytes
Pocketstation Executable Mono Icon List ;hdr[56h]*8 bytes
Body (Pocketstation Executable Code/Data, PSX Game Position, Exec-Icons)
```

The Title sector contains some information about the size of the above regions, but not about their addresses (ie. to find a given region, one must compute the size of the preceeding regions).

Special "P" Filename in Directory Sector

For pocketstation executables, the 7th byte of the filename must be a "P" (for other files that location does usually contain a "-", assuming the file uses a standard filename, eg. "BESLES-12345abcdefgh" for a Sony licensed european title).

Special Pocketstation Entries in the Title Sector at [50h..5Fh]

```
50h 2  Number of File Viewer Mono Icon Frames (or 0000h=Use Exec-Icons)
52h 4  Pocketstation Identifier ("MCX0"=Normal, "MCX1"=With Snapshot)
56h 1  Number of entries in Executable Mono Icon List (01h..FFh)
57h 1  Number of BU Command 5Bh/5Ch Get/Set Functions (00h..7Fh, usually 00h)
58h 4  Reserved (zero)
5Ch 4  Entrypoint in FLASH1 (ie. Fileoffset plus 02000000h) (bit0=THUMB)
```

In normal PSX files, the region at 50h..5Fh is usually zerofilled. For more info on the standard entries in the Title Sector (and for info on Directory Entries), see:
Memory Card Data Format

Snapshot Region (in "MCX1" Files only)

For a load-able snapshot the Snapshot ID must be 01h,00h,"SE", the Kernel uses snapshots only once (after loading a snapshot, it forcefully changes the ID to 00h,00h,"SE" in FLASH memory).

```
000h  r1..r12 (ie. without r0)
030h  r13_usr (sp_usr)
034h  r14_usr (lr_usr)
038h  r15     (pc)
03Ch  psr_fc
040h  Snapshot ID (0xh,00h,"SE")
044h  unused (3Ch bytes)
200h  Copy of user RAM at 200h..7FFh
```

For MCX1 files, snapshots can be automatically loaded and saved via the SWI 09h, DoExecute function (the snapshot handling seems to be bugged though; see SWI 09h for details).

Function Table (FUNC 80h..FFh)

The table can contain 00h..7Fh entries, for FUNC 80h..FFh. Each entry occupies 8 bytes:

```
00h 4   LEN1 (00000000h..00000080h) (destroys Kernel RAM if bigger)
04h 4   Function Address (bit0 can be set for THUMB code)
```

If the number of table entries isn't a multiple of 10h, then the table should be zero-padded to a multiple of 80h bytes (the following File Viewer Mono Icon data is located on the next higher 80h-byte boundary after the Function Table).
For details see BU Commands 5Bh and 5Ch.

File Viewer Mono Icon

Animation Length (0001h..any number) (icon frames) is stored in hdr[50h], for the File Viewer Icon, the Animation Delay is fixed (six 30Hz units per frame).
The File Viewer Icon is shown in the Directory Viewer (which is activated when holding the Down-button pressed for some seconds in the GUI screen with the speaker and memory card symbols, and which shows icons for all files, including regular PSX game positions, whose colored icons are converted without any contrast optimizations to unidentify-able dithered monochrome icons). If the animation length of the File Viewer Icon is 0000h, then the Directory Viewer does instead display the first Executable Mono Icon.
Each icon frame is 32x32 pixels with 1bit color depth (32 words, =128 bytes),

```
1st word = top-most scanline, 31st word = bottom-most scanline
bit0 = left-most pixel, bit31 = right-most pixel (0=white, 1=black)
```

A normal icon occupies 80h bytes, animated icons have more than one frame and do occupy N*80h bytes.

Executable Mono Icon List

The number of entries in the Executable Mono Icon List is specified in hdr[56h] (usually 01h). Each entry in the Icon List occupies 8 bytes:

```
00h 2  Animation Length (0001h..any number) (icon frames)
02h 2  Animation Delay (N 30Hz units per icon frame)
04h 4  Address of icon frame(s) in Virtual FLASH (at 02000000h and up)
```

The icon frame(s) can be anywhere on a word-aligned location in the file Body (as specified in the above Address entry), the format of the frame(s) is the same as for File Viewer Mono Icons (see there).
The Executable Icons are shown in the Executable File Selection Menu (which occurs when pressing Left/Right buttons in the GUI). Pressing Fire button in that menu starts the selected executable. If the Icon List has more than 1 entry, then pressing Up/Down buttons moves to the previous/next entry (this just allows to view the corresponding icons, but doesn't have any other purpose, namely, the current list index is NOT passed to the game when starting it).
The Executable Mono Icon List is usually zero-padded to 80h-bytes size (although that isn't actually required, the following file Body could start at any location).

Entrypoint

The whole file (including the header and icons) gets mapped to 02000000h and up. The entryoint can be anywhere in the file Body, and it gets called with a parameter value in r0 (when started by the GUI, that parameter is always zero, but it may be nonzero when the executable was started by a game, ie. the <param> from SWI 08h, PrepareExecute, or the <param> from BU Command 59h).
Caution: Games (and GUI) are started with the ARM CPU running in Non-privileged User Mode (however, there are several ways to hook IRQ/FIQ handlers, and from there one can switch to Privileged System Mode).

Returning to the GUI

Games should always include a way to return to the GUI (eg. an option in the game over screen, a key combination, a watchdog timer, and/or the docking signal) (conventionally, games should prompt Exit/Continue when holding Fire pressed for 5 seconds), otherwise it wouldn't be possible to start other games - except by pushing the Reset button (which is no good idea since the bizarre BIOS reset handler does reset the RTC time for whatever reason).
The kernel doesn't pass any return address to the entryoint (neither in R14, nor on stack). To return control to the GUI, use SWI functions PrepareExecute(1,0,GetDirIndex()+30h), and then DoExecute(0).

Pocketstation File Images

Pocketstation files are normally stored in standard Memory Card images,
Memory Card Images

Pocketstation specific files
Aside from that standard formats, there are two Pocketstation specific formats, the "SC" and "SN" variants. Both contain only the raw file, without any Directory sectors, and thus not including a "BESLESP12345"-style filename string. The absence of the filename means that a PSX game couldn't (re-)open these files via filenames, so they are suitable only for "standalone" pocketstation games.

Pocketstation .BIN Files ("SC" variant)
Contains the raw Pocketstation Executable (ie. starting with the "SC" bytes in the title sector, followed by icons, etc.), the filesize should be padded to a 2000h-byte block boundary.

Pocketstation .BIN Files ("SN" variant)
This is a strange incomplete .BIN file variant which starts with a 4-byte ID ("SN",00h,00h), which is directly followed by executable code, without any title sector, and without any icons.

```
It seems as if the file (including the 4-byte ID) is intended to be
mapped to address 02000000h, and that the entrypoint is fixed at
02000004h (in ARM state).
Since the File doesn't have a valid file header with "SC" and "MCXn" IDs,
it won't be recognized by real hardware, the PSX BIOS would treat it as
a corrupted/deleted file, the Pocketstation BIOS would treat it as a
non-executable file.
So, that fileformat is apparently working only on whatever emulators,
apparently on the one developed by SN Systems.
If one should want to use that files on real hardware, one could add
a 2000h byte stub at the begin of the file; with valid headers, and
with a small executable that remaps the "SN" stuff to 02000000h via
the F_BANK_VAL registers.
Ah, and the "SN" files seem to access RAM at 01000000h and up, unknown
if RAM is mirrored to that location on real hardware, reportedly that
region is unused... and doesn't contain RAM...?
  Some games use The Undefined Instruction for TTY Output.
  Most games do strange 8bit writes to LCD_MODE+0 and LCD_MODE+1
  The games usually don't allow to return to the GUI (except by Reset).
```

The filesize is don't care (no padding to block, sector, word, or halfword boundaries required).

Pocketstation XBOO Cable

This circuit allows to connect a pocketstation to PC parallel port, allowing to upload executables to real hardware, and also allowing to download TTY debug messages (particulary useful as the 32x32 pixel LCD screen is ways too small to display any detailed status info).

POC-XBOO Circuit
Use a standard parallel port cable (with 36pin centronics connector or 25pin DB connector) and then build a small adaptor like this:

```
Pin CNTR  DB25          Pocketstation          _____________________
ACK 10    10  --------- 1 JOYDTA              |       |       |     |
D0  2     2   --------- 2 JOYCMD              | 9 7 6 | 5 4 3 |  2 1  | CARD
GND 19-30 18-25 ------- 4 GND                |_______|_______|_______|
D1  3     3   --------- 6 /JOYSEL             _____________________
D2  4     4   --------- 7 JOYCLK             |       |       |     |
PE  12    12  --------- 9 /JOYACK (/IRQ7)     | 9 8 7 | 6 5 4 | 3 2 1 | PAD
NC ---------------------- 8 /JOYGUN (/IRQ10)    \_____|_____|_____/
NC ---------------------- 3 7.5V (rumble.supply)
SUPPLY.5V --|>|---|>|-- 5 3.5V (VCC) (eg. PC's +5V via two 1N4001 diodes)
SUPPLY.0V ------------- 4 GND (not needed when same as GND on CNTR/DB25)
```

The circuit is same as for "Direct Pad Pro" (but using a memory card connector instead of joypad connector, and needing +5V from PC power supply instead of using parallel port D3..D7 as supply). Note: IRQ7 is optional (for faster/early timeout).

POC-XBOO Upload
The upload function is found in no$gba "Utility" menu. It does upload the executable and autostart it via standard memory card/pocketstation commands (ie. it doesn't require any special transmission software installed on the pocketstation side).
Notes: Upload is overwriting ALL files on the memory card, and does then autostart the first file. Upload is done as "read and write only if different", this provides faster transfers and higher lifetime.

POC-XBOO TTY Debug Messages
TTY output is conventionally done by executing the ARM CPU's Undefined Opcode with an ASCII character in R0 register (for that purpose, the undef opcode handler should simply point to a MOVS PC,LR opcode).
That kind of TTY output works in no$gba's pocketstation emulation. It can be also used via no$gba's POC-XBOO cable, but requires some small customization in the executable:
First of, the executable needs "TTY+" ID in some reserved bytes of the title sector (telling the xboo uploader to stay in transmission mode and to keep checking for TTY messages after the actual upload):

```
TitleSector[58h] = "TTY+"
```

With that ID, and with the XBOO-hardware being used, the game will be started with with "TTY+" in R0 (notifying it that the XBOO hardware is present, and that it needs to install special transmission handlers):

```
;------------------
.data?
org  200h
...
tty_bufsiz equ 128  ;max=128=fastest (can be smaller if you are short of RAM)
func3_info:                                  ;\ ;\
 func3_buf_base     dd 0   ;fixed="func3_buf"     ;  ;  func3_info+00h
 func3_buf_len      dd 0   ;range=0..128          ;/ ;  func3_info+04h
 func3_stack        dd 0                             ;  func3_info+08h
```

```
ptr_to_comflags     dd 0
...
.code
...
;------------------
tty_wrchr:   ;in: r0=char
 dd      0e6000010h ;=undef opcode            ;-Write chr(r0) to TTY
 bx      lr
;------------------
init_tty:  ;in: r0=param (from entrypoint)
 ldr     r1,=2B595454h ;"TTY+"                ;\check if xboo-cable present
 cmp     r1,r0                                ; (r0=incoming param from
 beq     @@tty_by_xboo_cable                  ;/executable's entrypoint)
;- - -
 mov     r1,0                                 ;\dummy und_handler
 ldr     r2,=0e1b0f00eh  ;=movs r15,r14       ; (just return from exception,
 str     r2,[r1,04h]  ;und_handler            ;/for normal cable-less mode)
 b       @@finish
;---
@@tty_by_xboo_cable:
 swi     17h  ;GetPtrToFunc3addr()            ;\
 ldr     r1,=(tty_func3_handler AND 0ffffh)   ; init FUNC3 aka TTY handler
 strh    r1,[r0]                              ;/
 ldr     r1,=func3_info                       ;\
 mov     r0,0                              ;\ ; mark TTY as len=empty
 str     r0,[r1,4]        ;func3_buf_len  ;/ ; and
 add     r0,r1,0ch ;=func3_buffer         ;\ ; init func3 base
 str     r0,[r1,0]        ;func3_buf_base ;/ ;/
 mov     r1,0                                 ;\
 ldr     r2,=0e59ff018h  ;=ldr r15,[pc,NN]    ;
 str     r2,[r1,04h]  ;und_handler            ; special xboo und_handler
 add     r2,=tty_xboo_und_handler             ;
 str     r2,[r1,24h]  ;und_vector             ;/
@@finish:
 swi     06h ;GetPtrToComFlags()              ;\
 ldr     r1,=ptr_to_comflags                  ; get ptr to ComFlags
 str     r0,[r1]                              ;/
 bx      lr
;------------------
tty_xboo_und_handler:   ;in: r0=char
 ldr     r13,=func3_info ;aka sp_und          ;-base address (in sp_und)
 str     r12,[r13,8] ;func3_stack             ;-push r12
@@wait_if_buffer_full:                        ;\
 ldr     r12,=ptr_to_comflags                 ; ;\exit if execute file request
 ldr     r12,[r12]  ;ptr to ComFlags          ; ; ComFlg.Bit11 ("bu_cmd_59h"),
 ldr     r12,[r12]  ;read ComFlags            ; ; ie. allow that flag to be
 tst     r12,1 shl 11  ;test bit11            ; ; processed by main program,
 bne     @@exit                               ; ;/without hanging here
 ldrb    r12,[r13,4] ;func3_buf_len           ; wait if buffer full
 cmp     r12,tty_bufsiz                       ; (until drained by FIQ)
 beq     @@wait_if_buffer_full                ;/
 mov     r12,1bh+0c0h  ;mode=und, FIQ/IRQ=off ;\disable FIQ (no COMMUNICATION
 mov     cpsr_ctl,r12                         ;/interrupt during buffer write)
 ldrb    r12,[r13,4]  ;func3_buf_len          ;\
 add     r12,1        ;raise len              ; write char to buffer
 strb    r12,[r13,4]  ;func3_buf_len          ; and raise buffer length
 add     r12,0ch-1    ;=func3_buffer+INDEX    ;
 strb    r0,[r13,r12] ;append char to buf     ;/
@@exit:
 ldr     r12,[r13,8]  ;func3_stack            ;-pop r12
 movs    r15,r14        ;return from exception (and restore old IRQ/FIQ state)
;------------------
tty_func3_handler:   ;in: r0=flags, r1=ptr
 tst     r0,10h  ;test if PRE/POST data (pre: Z, post: NZ)
;ldreq   r1,[r1]   ;read 32bit param (aka the four LEN1 bytes of FUNC3)
 ldr     r0,=func3_info   ;ptr to two 32bit values (FUNC3 return value)
 movne   r1,0                          ;\for POST data: mark buffer empty
 strne   r1,[r0,4] ;func3_buf_len=0    ;/
 bx      lr                            ;-for PRE data: return r0=func3_info
```

Usage: Call "init_tty" at the executable's entrypoint (with incoming R0 passed on). Call "tty_wrchr" to output ASCII characters.
Note: The TTY messages are supported only in no$gba debug version (not no$gba gaming version).

Serial Port (SIO)

1F801050h SIO_TX_DATA (W)

```
0-7   Data to be sent
8-31  Not used
```

Writing to this register starts transmit (if, or as soon as, TXEN=1 and CTS=on and SIO_STAT.2=Ready). Writing to this register while SIO_STAT.0=Busy causes the old value to be overwritten.
The "TXEN=1" condition is a bit more complex: Writing to SIO_TX_DATA latches the current TXEN value, and the transfer DOES start if the current TXEN value OR the latched TXEN value is set (ie. if TXEN gets cleared after writing to SIO_TX_DATA, then the transfer may STILL start if the old latched TXEN value was set; this appears for SIO transfers in Wipeout 2097).

1F801050h SIO_RX_DATA (R)

```
0-7   Received Data      (1st RX FIFO entry) (oldest entry)
8-15  Preview            (2nd RX FIFO entry)
16-23 Preview            (3rd RX FIFO entry)
24-31 Preview            (4th RX FIFO entry) (5th..8th cannot be previewed)
```

A data byte can be read when SIO_STAT.1=1. Data should be read only via 8bit memory access (the 16bit/32bit "preview" feature is rather unusable).

1F801054h SIO_STAT (R)

```
0     TX Ready Flag 1   (1=Ready/Started)  (depends on CTS) (TX requires CTS)
1     RX FIFO Not Empty (0=Empty, 1=Not Empty)
2     TX Ready Flag 2   (1=Ready/Finished) (depends on TXEN and on CTS)
3     RX Parity Error   (0=No, 1=Error; Wrong Parity, when enabled) (sticky)
4     RX FIFO Overrun   (0=No, 1=Error; Received more than 8 bytes) (sticky)
5     RX Bad Stop Bit   (0=No, 1=Error; Bad Stop Bit) (when RXEN)   (sticky)
6     RX Input Level    (0=Normal, 1=Inverted) ;only AFTER receiving Stop Bit
```

```
  8     CTS Input Level   (0=Off, 1=On) (remote RTS) ;CTS required for TX
  9     Interrupt Request (0=None, 1=IRQ)                          (sticky)
  10    Unknown           (always zero)
  11-25 Baudrate Timer    (15bit timer, decrementing at 33MHz)
  26-31 Unknown (usually zero, sometimes all bits set)
```

Note: Bit0 gets cleared after sending the Startbit, Bit2 gets cleared after sending all bits up to including the Stopbit.

1F801058h SIO_MODE (R/W) (eg. 004Eh --> 8N1 with Factor=MUL16)

```
  0-1   Baudrate Reload Factor (1=MUL1, 2=MUL16, 3=MUL64) (or 0=STOP)
  2-3   Character Length       (0=5bits, 1=6bits, 2=7bits, 3=8bits)
  4     Parity Enable          (0=No, 1=Enable)
  5     Parity Type            (0=Even, 1=Odd) (seems to be vice-versa...?)
  6-7   Stop bit length        (0=Reserved/1bit, 1=1bit, 2=1.5bits, 3=2bits)
  8-15  Not used (always zero)
```

1F80105Ah SIO_CTRL (R/W)

```
  0     TX Enable (TXEN)  (0=Disable, 1=Enable, when CTS=On)
  1     DTR Output Level  (0=Off, 1=On)
  2     RX Enable (RXEN)  (0=Disable, 1=Enable)  ;Disable also clears RXFIFO
  3     TX Output Level   (0=Normal, 1=Inverted, during Inactivity & Stop bits)
  4     Acknowledge       (0=No change, 1=Reset SIO_STAT.Bits 3,4,5,9)      (W)
  5     RTS Output Level  (0=Off, 1=On)
  6     Reset             (0=No change, 1=Reset most SIO_registers to zero) (W)
  7     Unknown? (read/write-able when FACTOR non-zero) (otherwise always zero)
  8-9   RX Interrupt Mode    (0..3 = IRQ when RX FIFO contains 1,2,4,8 bytes)
  10    TX Interrupt Enable  (0=Disable, 1=Enable) ;when SIO_STAT.0-or-2 ;Ready
  11    RX Interrupt Enable  (0=Disable, 1=Enable) ;when N bytes in RX FIFO
  12    DSR Interrupt Enable (0=Disable, 1=Enable) ;when SIO_STAT.7  ;DSR=On
  13-15 Not used (always zero)
```

1F80105Ch SIO_MISC (R/W)

This is an internal register, which usually shouldn't be accessed by software. Messing with it has rather strange effects: After writing a value "X" to this register, reading returns "X ROR 8" eventually "ANDed with 1F1Fh and ORed with C0C0h or 8080h" (depending on the character length in SIO_MODE).

1F80105Eh SIO_BAUD (R/W) (eg. 00DCh --> 9600 bauds; when Factor=MUL16)

```
  0-15  Baudrate Reload value for decrementing Baudrate Timer
```

The Baudrate is calculated (based on SIO_BAUD, and on Factor in SIO_MODE):

```
  BitsPerSecond = (44100Hz*300h) / MIN(((Reload*Factor) AND NOT 1),Factor)
```

SIO_TX_DATA Notes

The hardware can hold (almost) 2 bytes in the TX direction (one being currently transferred, and, once when the start bit was sent, another byte can be stored in SIO_TX_DATA). When writing to SIO_TX_DATA, both SIO_STAT.0 and SIO_STAT.2 become zero. As soon as the transfer starts, SIO_STAT.0 becomes set (indicating that one can write a new byte to SIO_TX_DATA; although the transmission is still busy). As soon as the transfer of the most recently written byte ends, SIO_STAT.2 becomes set.

SIO_RX_DATA Notes

The hardware can hold 8 bytes in the RX direction (when receiving further byte(s) while the RX FIFO is full, then the last FIFO entry will by overwritten by the new byte, and SIO_STAT.4 gets set; the hardware does NOT automatically disable RTS when the FIFO becomes full).
Data can be read from SIO_RX_DATA when SIO_STAT.1 is set, that flag gets automatically cleared after reading from SIO_RX_DATA (unless there are still further bytes in the RX FIFO). Note: The hardware does always store incoming data in RX FIFO (even when Parity or Stop bits are invalid).
Note: A 16bit read allows to read two FIFO entries at once; nethertheless, it removes only ONE entry from the FIFO. On the contrary, a 32bit read DOES remove FOUR entries (although, there's nothing that'd indicate if the FIFO did actually contain four entries).
Reading from Empty RX FIFO returns either the most recently received byte or zero (the hardware stores incoming data in ALL unused FIFO entries; eg. if five entries are used, then the data gets stored thrice, after reading 6 bytes, the FIFO empty flag gets set, but nethertheless, the last byte can be read two more times, but doing further reads returns 00h).

Interrupt Acknowledge Notes

First reset I_STAT.8, then set SIO.CTRL.4 (when doing it vice-versa, the hardware may miss a new IRQ which may occur immediately after setting SIO.CTRL.4) (and I_STAT.8 is edge triggered, so that bit can be reset even while SIO_STAT.9 is still set).
When acknowledging via SIO_CTRL.4 with the enabled condition(s) in SIO_CTRL.10-12 still being true (eg. the RX FIFO is still not empty): the IRQ does trigger again (almost) immediately (it goes off only for a very short moment; barely enough to allow I_STAT.8 to sense a edge).

SIO_BAUD Notes

Timer reload occurs when writing to SIO_BAUD, and, automatically when the Baudrate Timer reaches zero. There should be two 16bit SIO timers (for TX and RX), the upper 15bit of one of that timers can be read from SIO_STAT (not sure which one, and no idea if there's a way to read the other timer, too).
Or... maybe there is only ONE timer, and RX/TX are separated only by separate "timer ellapsed" counters, in that case the MUL1 factor won't work properly, but, with the MUL16 or MUL64 factors, RX could start anytime (eg. when TX has already ellapsed a bunch of times)...?
The maximum baud rate may vary depending on the length and quality of the cable, whether and how many inverters and anti-inverters are used (on the mainboard and in external adaptor, and on whether signals are externally converted to +/-12V levels)... anyways, rates up to 9600 baud should be working in all cases.
However, running in no$psx, Wipeout 2097 seems to use about 2 million bauds... although, in older no$psx versions, I believe I did see it using some kind of baudrate detection, where it did try different rates in steps of 200 bauds or so...?

SIO Ports vs JOY Ports

SIO uses I/O Addresses 1F801050h..1F80105Fh, which seem to be organized similar to the Controller/Memory Card registers at 1F801040h..1F80104Fh, though not identical, and with an additional register at 1F80105Ch, which has no corresponding port at 1F80104Ch.
SIO_BAUD is <effectively> same as for JOY_BAUD, but, <internally> they are a bit different:

```
  JOY_BAUD is multiplied by Factor, and does then ellapse "2" times per bit.
  SIO_BAUD is NOT multiplied, and, instead, ellapses "2*Factor" times per bit.
```

Unlike for the Controller/Memory Card ports, the data is transferred without CLK signal, instead, it's using RS232 format, ie. the transfer starts with a start bit, and is then transferred at a specific baudrate (which must be configured identically at the receiver side). For RS232, data is usually 8bit, and may optionally end with a parity bit, and one or two stop bits.

Note

For SIO Pinouts, PSone SIO upgrading, and for building RS232 adaptors, see:
Pinouts - SIO Pinouts
Aside from the internal SIO port, the PSX BIOS supports two additional external serial ports, connected to the expansion port,
EXP2 Dual Serial Port (for TTY Debug Terminal)

SIO Games

The serial port is used (for 2-player link) by Wipeout 2097 (that game accidently assumes BAUDs based on 64*1024*1025 Hz rather than on 600h*44100 Hz).

Ridge Racer Revolution is also said to support 2P link.
Keitai Eddy seems to allow to connect a mobile phone to the SIO port (the games CD cover suggests so; this seems to be something different than the "normal" I-Mode adaptor, which would connect to controller port, not to SIO port).

8251A Note
The Playstation Serial Port is apparently based/inspired on the Intel 8251A USART chip; which has very similar 8bit Mode/Command/Status registers.

Expansion Port (PIO)

Expansion Port can contain ROM, RAM, I/O Ports, etc. For ROM, the first 256 bytes would contain the expansion ROM header.

For region 1, the CPU outputs a chip select signal (CPU Pin 98, /EXP).
For region 2, the CPU doesn't produce a chip select signal (the region is intended to contain multiple I/O ports, which do require an address decoder anyways, that address decoder could treat any /RD or /WR with A13=Hi and A23=Hi and A22=Lo as access to expansion region 2 (for /WR, A22 may be ignored; assuming that the BIOS is read-only).

Size/Bus-Width
The BIOS initalizes Expansion Region 1 to 512Kbyte with 8bit bus, and Region 2 to 128 bytes with 8bit bus. However, the size and data bus-width of these regions can be changed, see:
Memory Control
For Region 1, 32bit reads are supported even in 8bit mode (eg. 32bit opcode fetches are automatically processed as four 8bit reads).
For Region 2, only 8bit access seems to be supported (except that probably 16bit mode allows 16bit access), anyways, larger accesses seem to cause exceptions... not sure if that can be disabled...?

Expansion 1 - EXP1 - Intended to contain ROM
EXP1 Expansion ROM Header

Expansion 2 - EXP2 - Intended to contain I/O Ports
EXP2 Dual Serial Port (for TTY Debug Terminal)
EXP2 DTL-H2000 I/O Ports
EXP2 Post Registers
EXP2 Nocash Emulation Expansion

Expansion 3 - EXP3 - Intended to contain RAM
Not used by BIOS nor by any games. Seems to contain 1Mbyte RAM with 16bit databus (ie. 512Kx16) in DTL-H2000.

Other Expansions
Aside from the above, the Expansion regions can be used for whatever purpose, however, mind that the BIOS is reading from the ROM header region, and is writing to the POST register (so 1F000000h-1F0000FFh and 1F802041h should be used only if the hardware isn't disturbed by those accesses).

Missing Expansion Port
The expansion port is installed only on older PSX boards, newer PSX boards and all PSone boards don't have that port. However, the CPU should still output all expansion signals, and there should be big soldering points on the board, so it'd be easy to upgrade the console.
Only problem would be that most of the pinouts of the original expansion port are still unknown, so it's presently impossible to built a port that has 1:1 the same pinouts as original older PSX consoles?

Latched Address Bus
Note that A0..A23 are latched outputs, so they can be used as general purpuse 24bit outputs, provided that the system bus isn't used for other purposes (such like /BIOS, /SPU, /CD accesses) (A0..A23 are not affected by Main RAM and GPU addressing, nor by internal I/O ports like Timer and IRQ registers).

EXP1 Expansion ROM Header

Expansion 1 - ROM Header (accessed with 8bit databus setting)
```
  Address  Size Content
  1F000000h 4   Post-Boot Entrypoint (eg. 1F000100h and up)
  1F000004h 2Ch Post-Boot ID ("Licensed by Sony Computer Entertainment Inc.")
  1F000030h 50h Not used     (should be zero, but may contain code/data/io)
  1F000080h 4   Pre-Boot Entrypoint  (eg. 1F000100h and up)
  1F000084h 2Ch Pre-Boot ID  ("Licensed by Sony Computer Entertainment Inc.")
  1F0000B0h 50h Not used     (should be zero, but may contain code/data/io)
  1F000100h ..  Code, Data, I/O Ports, etc.
```
The entrypoints are called if their corresonding ID strings are present, return address to BIOS is passed in R31, so the expansion ROM may return control to BIOS, if that should be desired.

Pre-Boot Function
The Pre-Boot function is called almost immediately after Reset, with only some Memory Control registers initialized, the BIOS function vectors at A0h, B0h, and C0h are NOT yet initialized, so the Pre-Boot function can't use them.

Post-Boot Function
The Post-Boot function gets called while showing the "PS" logo, but before loading the .EXE file. The BIOS function vectors at A0h, B0h, and C0h are already installed and can be used by the Post-Boot Function.
Note that the Post-Boot Function is called ONLY when the "PS" logo is shown (ie. not if the CDROM drive is empty, or if it contains an Audio CD).

Mid-Boot Hook
One common trick to hook the Kernel after BIOS initialization, but before CDROM loading is to use the Pre-Boot handler to set a COP0 opcode fetch hardware breakpoint at 80030000h (after returning from the Pre-Boot handler, the Kernel will initialize important things like A0h/B0h/C0h tables, and will then break again when starting the GUI code at 80030000h) (this trick is used by Action Replay v2.0 and up).

EXP2 Dual Serial Port (for TTY Debug Terminal)

SCN2681 Dual Asynchronous Receiver/Transmitter (DUART)
The PSX/PSone retail BIOS contains some TTY Debug Terminal code; using an external SCN2681 chip which can be connected to the expansion port.
Whilst supported by all PSX/PSone retail BIOSes on software side, there aren't any known PSX consoles/devboards/expansions actually containing DUARTs on

hardware side.

1F802023h/Read - RHRA - DUART Rx Holding Register A (FIFO) (R)
1F80202Bh/Read - RHRB - DUART Rx Holding Register B (FIFO) (R)
1F802023h/Write - THRA - DUART Tx Holding Register A (W)
1F80202Bh/Write - THRB - DUART Tx Holding Register B (W)

```
7-0  Data (aka Character)
```

The hardware can hold max 2 Tx characters per channel (1 in the THR register, and one currently processed in the Tx Shift Register), and max 4 Rx characters (3 in the RHR FIFO, plus one in the Rx Shift Register) (when receiving a 5th character, the "old newest" value in the Rx Shift Register is lost, and the overrun flag is set).

1F802020h/FirstAccess - MR1A - DUART Mode Register 1.A (R/W)
1F802028h/FirstAccess - MR1B - DUART Mode Register 1.B (R/W)

```
7    RxRTS Control       (0=No, 1=Yes)
6    RxINT Select        (0=RxRDY, 1=FFULL)
5    Error Mode          (0=Char, 1=Block)
4-3  Parity Mode   (0=With Parity, 1=Force Parity, 2=No Parity, 3=Multidrop)
2    Parity Type         (0=Even, 1=Odd)
1-0  Bits per Character  (0=5bit, 1=6bit, 2=7bit, 3=8bit)
```

Note: In block error mode, block error conditions must be cleared by using the error reset command (command 4) or a receiver reset (command 2).

1F802020h/SecondAccess - MR2A - DUART Mode Register 2.A (R/W)
1F802028h/SecondAccess - MR2B - DUART Mode Register 2.B (R/W)

```
7-6  Channel Mode       (0=Normal, 1=Auto-Echo, 2=Local loop, 3=Remote loop)
5    TxRTS Control      (0=No, 1=Yes) (when 1 --> OP0=RTSA / OP1=RTSB)
4    CTS Enable         (0=No, 1=Yes) (when 1 --> IP0=CTSA / IP1=CTSB)
3-0  Tx Stop Bit Length (00h..0Fh = see below)
```

Stop Bit Lengths:

```
0=0.563  1=0.625  2=0.688  3=0.750  4=0.813  5=0.875  6=0.938  7=1.000
8=1.563  9=1.625  A=1.688  B=1.750  C=1.813  D=1.875  E=1.938  F=2.000
```

Add 0.5 to values shown for 0..7 if channel is programmed for 5 bits/char.

1F802021h/Write - CSRA - DUART Clock Select Register A (W)
1F802029h/Write - CSRB - DUART Clock Select Register B (W)

```
7-4  Rx Clock Select  (0..0Ch=See Table, 0Dh=Timer, 0Eh=16xIP, 0Fh=1xIP)
3-0  Tx Clock Select  (0..0Ch=See Table, 0Dh=Timer, 0Eh=16xIP, 0Fh=1xIP)
```

The 2681 has some sets of predefined baud rates (set1/set2 selected via ACR.7), additionally, in BRG Test Mode, set3/set4 are used instead of set1/set2), the baud rates for selections 00h..0Dh are:

```
Rate 00h  01h 02h   03h   04h   05h   06h    07h  08h   09h  0Ah   0Bh  0Ch
Set1 50   110 134.5 200   300   600   1200   1050 2400  4800 7200  9600 38400
Set2 75   110 134.5 150   300   600   1200   2000 2400  4800 1800  9600 19200
Set3 4800 880 1076  19200 28800 57600 115200 1050 57600 4800 57600 9600 38400
Set4 7200 880 1076  14400 28800 57600 115200 2000 57600 4800 14400 9600 19200
```

Selection 0Eh/0Fh are using an external clock source (derived from IP3,IP4,IP5,IP6 pins; for TxA,RxA,TxB,RxB respectively).

1F802022h/Write - CRA - DUART Command Register A (W)
1F80202Ah/Write - CRB - DUART Command Register B (W)

```
7    Not used    (should be 0)
6-4  Miscellaneous Commands (0..7 = see below)
3    Disable Tx  (0=No change, 1=Disable)
2    Enable Tx   (0=No change, 1=Enable) ;Don't use with Command 3 (Reset Rx)
1    Disable Rx  (0=No change, 1=Disable)
0    Enable Rx   (0=No change, 1=Enable) ;Don't use with Command 2 (Reset Tx)
```

The command values for CRA (or CRB) are:

```
0 No command          ;no effect
1 Reset MR pointer    ;force "FirstAccess" state for MR1A (or MR1B) access
2 Reset receiver      ;reset RxA (or RxB) registers, disable Rx, flush Fifo
3 Reset transmitter   ;reset TxA (or TxB) registers
4 Reset Error Flags           ;reset SRA.7-4 (or SRB.7-4) to zero
5 Reset Break-Change IRQ Flag ;reset ISR.2 (or ISR.6) to zero
6 Start break  ;after current char, pause Tx with TxDA=Low (or TxDB=Low)
7 Stop break   ;output one High bit, then continue Tx (ie. undo pause)
```

Access to the upper four bits of the command register should be separated by 3 edges of the X1 clock. A disabled transmitter cannot be loaded.

1F802025h/Read - ISR - DUART Interrupt Status Register (R)
1F802025h/Write - IMR - DUART Interrupt Mask Register (W)

```
7    Input Port Change   (0=No, 1=Yes) (Ack via reading IPCR) ;see ACR.3-0
6    Break Change B      (0=No, 1=Yes) (Ack via CRB/Command5)
5    RxRDYB/FFULLB       (0=No, 1=Yes) (Ack via reading data) ;see MR1B.6
4    THRB Empty (TxRDYB) (0=No, 1=Yes) (Ack via writing data) ;same as SRB.2
3    Counter Ready       (0=No, 1=Yes) (Ack via CT_STOP)
2    Break Change A      (0=No, 1=Yes) (Ack via CRA/Command5)
1    RxRDYA/FFULLA       (0=No, 1=Yes) (Ack via reading data) ;see MR1A.6
0    THRA Empty (TxRDYA) (0=No, 1=Yes) (Ack via writing data) ;same as SRA.2
```

1F802021h/Read - SRA - DUART Status Register A (R)
1F802029h/Read - SRB - DUART Status Register B (R)

```
7    Rx Received Break*        (0=No, 1=Yes) ;received 00h without stop bit
6    Rx Framing Error*         (0=No, 1=Yes) ;received data without stop bit
5    Rx Parity Error*          (0=No, 1=Yes) ;received data with bad parity
4    Rx Overrun Error          (0=No, 1=Yes) ;Rx FIFO full + RxShiftReg full
3    Tx Underrun  (TxEMT)      (0=No, 1=Yes) ;both TxShiftReg and THR empty
2    Tx THR Empty (TxRDY)      (0=No, 1=Yes) ;same as ISR.0 / ISR.4
1    Rx FIFO Full (FFULL)      (0=No, 1=Yes) ;set upon 3 or more characters
0    Rx FIFO Not Empty (RxRDY) (0=No, 1=Yes) ;set upon 1 or more characters
```

Bit7-5 are appended to the corresponding data character in the receive FIFO. A read of the status provides these bits (7:5) from the top of the FIFO together with bits (4:0). These bits are cleared by a "reset error status" command. In character mode they are discarded when the corresponding data character is read from the FIFO. In block error mode, block error conditions must be cleared by using the error reset command (command 4x) or a receiver reset.

1F802024h/Write - ACR - DUART Aux. Control Register (W)

```
7    Select Baud Rate Generator (BRG) Set   (0=Set1/Set3, 1=Set2/Set4)
6-4  Counter/Timer Mode and Source          (see below)
3-0  IP3..IP0 Change Interrupt Enable Flags (0=Off, 1=On)
```

Counter/Timer Mode and Clock Source Settings:

```
Num  Mode      Clock Source
0h   Counter   External (IP2)
1h   Counter   TxCA - 1x clock of Channel A transmitter
```

```
  3h   Counter   Crystal or external clock (x1/CLK) divided by 16
  4h   Timer     External (IP2)
  5h   Timer     External (IP2) divided by 16
  6h   Timer     Crystal or external clock (x1/CLK)
  7h   Timer     Crystal or external clock (x1/CLK) divided by 16
```

In Counter Mode, the Counter Ready flag is set on any underflow, and the counter wraps to FFFFh and keeps running (but may get stopped by software).
In Timer Mode, automatic reload occurs on any underflow, the counter flag (which can be output to OP3) is toggled on any underflow, but the Counter Ready flag is set only on each 2nd underflow (unlike as in Counter mode).

1F802024h/Read - IPCR - DUART Input Port Change Register (R)

```
  7-4  IP3..IP0 Change Occured Flags (0=No, 1=Yes)    ;auto reset after read
  3-0  Current IP3-IP0 Input states  (0=Low, 1=High)  ;Same as IP.3-0
```

Reading from this register automatically resets IPCR.7-4 and ISR.7.

1F80202Dh/Read - IP - DUART Input Port (R)

```
  7    Not used (always 1)
  6-0  Current IP6-IP0 Input states (0=Low, 1=High)  ;LSBs = Same as IPCR.3-0
```

IP0-6 can be used as general purpose inputs, or for following special purposes:

```
  IP6 External RxB Clock     ;see CSRB.7-4
  IP5 External TxB Clock     ;see CSRB.3-0
  IP4 External RxA Clock     ;see CSRA.7-4
  IP3 External TxA Clock     ;see CSRA.3-0
  IP2 External Timer Input   ;see AUX.6-4
  IP1 Clear to Send B (CTSB) ;see MR2B.5
  IP0 Clear to Send A (CTSA) ;see MR2A.5
```

Note: The 24pin chip doesn't have any inputs, the 28pin chip has only one input (IP2), the 40pin/44pin chips have seven inputs (IP0-IP6).

1F80202Eh/Write - DUART Set Output Port Bits Command (Set means Out=LOW)
1F80202Fh/Write - DUART Reset Output Port Bits Command (Reset means Out=HIGH)

```
  7-0  Change "OPR" OP7-OP0 Output states (0=No change, 1=Set/Reset)
```

Note: The 24pin chip doesn't have any outputs, the 28pin chip has only two outputs (OP0,OP1), the 40pin/44pin chips have eight outputs (OP0-OP7).

1F80202Dh/Write - OPCR - DUART Output Port Configuration Register (W)

```
  7    OP7      (0=OPR.7, 1=TxRDYB)
  6    OP6      (0=OPR.6, 1=TxRDYA)
  5    OP5      (0=OPR.5, 1=RxRDY/FFULLB)
  4    OP4      (0=OPR.4, 1=RxRDY/FFULLA)
  3-2  OP3      (0=OPR.3, 1=Clock/Timer Output, 2=TxCB(1x), 3=RxCB(1x))
  1-0  OP2      (0=OPR.2, 1=TxCA(16x), 2=TxCA(1x), 3=RxCA(1x))
```

Additionally, the OP0 and OP1 outputs are controlled via MR2A.5 and MR2B.5.

1F802022h/Read - - DUART Toggle Baud Rate Generator Test Mode (Read=Strobe)
1F80202Ah/Read - - DUART Toggle 1X/16X Test Mode (Read=Strobe)

```
  7-0  Not used (just issue a dummy-read to toggle the test mode on/off)
```

BGR Test switches between Baud Rate Set1/Set2 and Set3/Set4.
1X/16X Test switches between whatever...?

1F80202Eh/Read - CT_START - DUART Start Counter Command (Read=Strobe)
1F80202Fh/Read - CT_STOP - DUART Stop Counter Command (Read=Strobe)

```
  7-0  Not used (just issue a dummy-read to strobe start/stop command)
```

Start: Forces reload (copies CTLR/CTUR to CTL/CTU), and starts the timer.
Stop-in-Counter-Mode: Resets ISR.3, and stops the timer.
Stop-in-Timer-Mode: Resets ISR.3, but doesn't stop the timer.

1F802026h/Read - CTU - DUART Counter/Timer Current Value, Upper/Bit15-8 (R)
1F802027h/Read - CTL - DUART Counter/Timer Current Value, Lower/Bit7-0 (R)
1F802026h/Write - CTUR - DUART Counter/Timer Reload Value, Upper/Bit15-8 (W)
1F802027h/Write - CTLR - DUART Counter/Timer Reload Value, Lower/Bit7-0 (W)
The CTLR/CTUR reload value is copied to CTL/CTU upon Start Counter Command. In Timer mode (not in Counter mode), it is additionally copied automatically when the timer undeflows.

1F80202Ch - N/A - DUART Reserved Register (neither R nor W)
Reserved.

Chip versions
The SCN2681 is manufactured with 24..44 pins, the differences are:

```
  24pin basic cut-down version          ;without IP0-1/OP0-1 = without CTS/RTS
  28pin additional IP2,OP0,OP1,X2       ;without IP0-1 = without CTS
  40pin additional IP0-IP6,OP0-OP7,X2   ;full version
  44pin same as 40pin with four NC pins ;full version (SMD)
```

Unknown which of them is supposed to be used with the PSX?
Note: The Motorola 68681 should be the same as the Philips/Signetics 2681.

Notes
Unknown if the Interrupt signal is connected to the PSX... there seems to be no spare IRQ for it, though it <might> share an IRQ with whatever other hardware...?
The BIOS seems to use only one of the two channels; for the std_io functions:
BIOS TTY Console (std_io)
Aside from the external DUART, the PSX additionally contains an internal UART,
Serial Port (SIO)
The DTL-H2000 devboard uses a non-serial "ATCONS" channel for TTY stuff,
EXP2 DTL-H2000 I/O Ports

EXP2 DTL-H2000 I/O Ports

The DTL-H2000 contains extended 8Mbyte Main RAM (instead of normal 2Mbyte), plus additional 1MByte RAM in Expansion Area at 1FA00000h, plus some I/O ports at 1F8020xxh:

1F802000h - DTL-H2000: EXP2: - ATCONS STAT (R)

```
  0    Unknown, used for something
```

```
2    Unknown, used for something
3    TTY/Atcons TX Ready     (0=Busy, 1=Ready)
4    TTY/Atcons RX Available (0=None, 1=Yes)
5-7  Unknown/unused
```

1F802002h - DTL-H2000: EXP2: - ATCONS DATA (R and W)

```
0-7  TTY/Atcons RX/TX Data
```

TTY channel for message output (TX) and debug console keyboard input (RX). The DTL-H2000 is using this "ATCONS" stuff instead of the DUART stuff used in retail console BIOSes ("CONS" seems to refer to "Console", and "AT" might refer to PC/AT or whatever).

1F802004h - DTL-H2000: EXP2: - 16bit - ?

```
0-15 Data...?
```

1F802030h - DTL-H2000: Secondary IRQ10 Controller (IRQ Flags)

This register does expand IRQ10 (Lightgun) to more than one IRQ source. The register contains only Secondary IRQ Flags (there seem to be no Secondary IRQ Enable bits; at least not for Lightguns).

```
0     ... used for something
1    Lightgun IRQ (write: 0=No change, 1=Acknowledge) (read: 0=None, 1=IRQ)
2-3  Unknown/unused (write: 0=Normal)
4     ... acknowledged at 1FA00B04h, otherwise unused
5     ... TTY RX ?
6-7  Unknown/unused (write: 0=Normal)
8-31 Not used by DTL-H2000 BIOS (but Lightgun games write 0 to these bits)
```

Retail games that support IRQ10-based "Konami" Lightguns are containing code for detecting and accessing port 1F802030h. The detection works by examining a value in the BIOS ROM like so:

```
IF [BFC00104h]=00002000h then Port 1F802030h does exist     (DTL-H2000)
IF [BFC00104h]=00002500h then Port 1F802030h does NOT exist
IF [BFC00104h]=00000003h then Port 1F802030h does NOT exist (default)
IF [BFC00104h]= <other>  then Port 1F802030h does NOT exist
```

Normal consoles don't include Port 1F802030h, and IRQ10 is wired directly to the controller port, and the value at [BFC00104h] is always 00000003h. Accordingly, one cannot upgrade the console just by plugging a Secondary IRQ10 controller to the expansion port (instead, one would need to insert the controller between the CPU and controller plug, and to install a BIOS with [BFC00104h]=00002000h).

The DTL-H2000 BIOS accesses 1F802030h with 8bit load/store opcodes, however, the Lightgun games use 32bit load/store - which is theoretically overlapping port 1F802032h, though maybe the memory system does ignore the upper bits.

1F802032h - DTL-H2000: EXP2: - maybe IRQ enable?

```
0    Used for something (CLEARED on some occassions)
1-3  Unknown/unused
4    Used for something (SET on some occassions)
5-7  Unknown/unused
```

1F802040h - DTL-H2000: EXP2: 1-byte - DIP Switch?

```
0-7  DIP Value (00h..FFh, but should be usually 00h..02h)
```

This register selects the DTL-H2000 boot mode, for whatever reason it's called "DIP Switch" register, although the DTL-H2000 boards don't seem to contain any such DIP Switches (instead, it's probably configured via some I/O ports on PC side). Possible values are:

```
DIP=0 --> .. long delay before TTY?   with "PSX>" prompt, throws CDROM cmds
DIP=1 --> .. long delay before TTY?   no "PSX>" prompt           PSY-Q?
DIP=2 --> .. instant TTY?             with "PSX>" prompt
DIP=3 --> Lockup
DIP=04h..FFh --> Lockup with POST=04h..FFh
```

1F802042h - DTL-H2000: EXP2: POST/LED (R/W)

EXP2 Post Registers

EXP2 Post Registers

1F802041h - POST - External 7-segment Display (W)

```
0-3  Current Boot Status (00h..0Fh)
4-7  Not used by BIOS    (always set to 0)
```

During boot, the BIOS writes incrementing values to this register, allowing to display the current boot status on an external 7 segment display (much the same as Port 80h used in PC BIOSes).

1F802042h - DTL-H2000: EXP2: POST/LED (R/W)

```
0-7 Post/LED value
```

8bit wide, otherwise same as POST 1F802041h on retail consoles.

1F802070h - POST2 - Unknown? (W) - PS2

Might be a configuration port, or it's another POST register (which is used prior to writing the normal POST bytes to 1FA00000h).
The first write to 1F802070h is 32bit, all further writes seem to be 8bit.

1FA00000h - POST3 - External 7-segment Display (W) - PS2

Similar to POST, but PS2 BIOS uses this address.

EXP2 Nocash Emulation Expansion

1F802060h Emu-Expansion ID1 "E" (R)
1F802061h Emu-Expansion ID2 "X" (R)
1F802062h Emu-Expansion ID3 "P" (R)
1F802063h Emu-Expansion Version (01h) (R)

Contains ID and Version.

1F802064h Emu-Expansion Enable1 "O" (R/W)
1F802065h Emu-Expansion Enable2 "N" (R/W)

Activates the Halt and Turbo Registers (when set to "ON").

1F802066h Emu-Expansion Halt (R)

When enabled (see above), doing an 8bit read from this address stops the CPU emulation unless/until an Interrupt occurs (when "CAUSE AND SR AND FF00h" becomes nonzero). Can be used to reduce power consumption, and to make the emulation faster.

1F802067h Emu-Expansion Turbo Mode Flags (R/W)
When enabled (see above), writing to this register activates/deactivates "turbo" mode, which is causing new data to arrive immediately after acknowledging the previous interrupt.

```
0   CDROM Turbo       (0=Normal, 1=Turbo)
1   Memory Card Turbo (0=Normal, 1=Turbo)
2   Controller Turbo  (0=Normal, 1=Turbo)
3-7 Reserved (must be zero)
```

Memory Control

The Memory Control registers are initialized by the BIOS, and, normally software doesn't need to change that settings. Some registers are useful for expansion hardware (allowing to increase the memory size and bus width).

1F801000h - Expansion 1 Base Address (usually 1F000000h)
1F801004h - Expansion 2 Base Address (usually 1F802000h)

```
0-23   Base Address   (Read/Write)
24-31  Fixed          (Read only, always 1Fh)
```

For Exansion 1, the address is forcefully aligned to the selected expansion size (see below), ie. if the size is bigger than 1 byte, then the lower bit(s) of the base address are ignored.
For Expansion 2, trying to use ANY other value than 1F802000h seems to disable the Expansion 2 region, rather than mapping it to the specified address (ie. Port 1F801004h doesn't seem to work).
For Expansion 3, the address seems to be fixed (1FA00000h).

1F801008h - Expansion 1 Delay/Size (usually 0013243Fh) (512Kbytes, 8bit bus)
1F80100Ch - Expansion 3 Delay/Size (usually 00003022h) (1 byte)
1F801010h - BIOS ROM Delay/Size (usually 0013243Fh) (512Kbytes, 8bit bus)
1F801014h - SPU Delay/Size (200931E1h) (use 220931E1h for SPU-RAM reads)
1F801018h - CDROM Delay/Size (00020843h or 00020943h)
1F80101Ch - Expansion 2 Delay/Size (usually 00070777h) (128 bytes, 8bit bus)

```
0-3   Unknown (R/W)
4-7   Access Time        (00h..0Fh=00h..0Fh Cycles)
8     Use COM0 Time      (0=No, 1=Yes, add to Access Time)
9     Use COM1 Time      (0=No, 1=Probably Yes, but has no effect?)
10    Use COM2 Time      (0=No, 1=Yes, add to Access Time)
11    Use COM3 Time      (0=No, 1=Yes, clip to MIN=(COM3+6) or so?)
12    Data Bus-width     (0=8bit, 1=16bit)
13-15 Unknown (R/W)
16-20 Memory Window Size (1 SHL N bytes) (0..1Fh = 1 byte ... 2 gigabytes)
21-23 Unknown (always zero)
24-27 Unknown (R/W) ;must be non-zero for SPU-RAM reads
28    Unknown (always zero)
29    Unknown (R/W)
30    Unknown (always zero)
31    Unknown (R/W) (Port 1F801008h only; always zero for other ports)
```

Trying to access addresses that exceed the selected size causes an exception. Maximum size would be Expansion 1 = 17h (8MB), BIOS = 16h (4MB), Expansion 2 = 0Dh (8KB), Expansion 3 = 15h (2MB). Trying to select larger sizes would overlap the internal I/O ports, and crash the PSX. The Size bits seem to be ignored for SPU/CDROM. The SPU timings seem to be applied for both the 200h-byte SPU region at 1F801C00h and for the 200h-byte unknown region at 1F801E00h.

1F801020h - COM_DELAY / COMMON_DELAY (00031125h or 0000132Ch or 00001325h)

```
0-3   COM0 - Offset A   ;used for SPU/EXP2 (and for adjusted CDROM timings)
4-7   COM1 - No effect? ;used for EXP2
8-11  COM2 - Offset B   ;used for BIOS/EXP1/EXP2
12-15 COM3 - Min Value  ;used for CDROM
16-17 COM? - Unknown    ;used for whatever
18-31 Unknown/unused (read: always 0000h)
```

This register contains clock cycle offsets that can be added to the Access Time values in Port 1F801008h..1Ch. Works (somehow) like so:

```
1ST=0, SEQ=0, MIN=0
IF Use_COM0 THEN 1ST=1ST+COM0-1, SEQ=SEQ+COM0-1
IF Use_COM2 THEN 1ST=1ST+COM2,   SEQ=SEQ+COM2
IF Use_COM3 THEN MIN=COM3
IF 1ST<6 THEN 1ST=1ST+1   ;(somewhat like so)
1ST=1ST+AccessTime+2, SEQ=SEQ+AccessTime+2
IF 1ST<(MIN+6) THEN 1ST=(MIN+6)
IF SEQ<(MIN+2) THEN SEQ=(MIN+2)
```

The total access time is the sum of First Access, plus any Sequential Access(es), eg. for a 32bit access with 8bit bus: Total=1ST+SEQ+SEQ+SEQ.
If the access is done from code in (uncached) RAM, then 0..4 cycles are added to the Total value (the exact number seems to vary depending on the used COMx values or so).
And the purpose... probably allows to define the length of the chipselect signals, and of gaps between that signals...?

1F801060h - RAM_SIZE (R/W) (usually 00000B88h) (or 00000888h)

```
0-2   Unknown (no effect)
3     Crashes when zero (except older consoles whose BIOS <did> set bit3=0)
4-6   Unknown (no effect)
7     Delay on simultaneous CODE+DATA fetch from RAM (0=None, 1=One Cycle)
8     Unknown (no effect) (should be set for 8MB, cleared for 2MB)
9-11  Define 8MB Memory Window (first 8MB of KUSEG,KSEG0,KSEG1)
12-15 Unknown (no effect)
16-31 Unknown (Garbage)
```

Possible values for Bit9-11 are:

```
0 = 1MB Memory + 7MB Locked
1 = 4MB Memory + 4MB Locked
2 = 1MB Memory + 1MB HighZ + 6MB Locked
3 = 4MB Memory + 4MB HighZ
4 = 2MB Memory + 6MB Locked              ;<--- would be correct for PSX
5 = 8MB Memory                           ;<--- default by BIOS init
6 = 2MB Memory + 2MB HighZ + 4MB Locked
7 = 8MB Memory
```

The BIOS initializes this to setting 5 (8MB) (ie. the 2MB RAM repated 4 times), although the "correct" would be setting 4 (2MB, plus other 6MB Locked). The remaining memory, after the first 8MB, and up to the Expansion/IO/BIOS region seems to be always Locked.
The HighZ regions are FFh-filled, that even when grounding data lines on the system bus (ie. it is NOT a mirror of the PIO expansion region).
Locked means that the CPU generates an exception when accessing that area.
Note: Wipeout uses a BIOS function that changes RAM_SIZE to 00000888h (ie. with corrected size of 2MB, and with the unknown Bit8 cleared). Gundam Battle Assault 2 does actually use the "8MB" space (with stacktop in mirrored RAM at 807FFFxxh).

FFFE0130h Cache Control (R/W)

```
  0-2   Unknown (Read/Write)                                        (R/W)
  3     Scratchpad Enable 1 (0=Disable, 1=Enable when Bit7 is set, too) (R/W)
  4-5   Unknown (Read/Write)                                        (R/W)
  6     Unknown (read=always zero)                 (R) or (W) or unused..?
  7     Scratchpad Enable 2 (0=Disable, 1=Enable when Bit3 is set, too) (R/W)
  8     Unknown                                                     (R/W)
  9     Crash (0=Normal, 1=Crash if code-cache enabled)             (R/W)
  10    Unknown (read=always zero)                 (R) or (W) or unused..?
  11    Code-Cache Enable (0=Disable, 1=Enable)                     (R/W)
  12-31 Unknown                                                     (R/W)
```

Used by BIOS to initialize cache (in combination with COP0), like so:

```
 Init Cache Step 1:
  [FFFE0130h]=00000804h, then set cop0_sr=00010000h, then
  zerofill each FOURTH word at [0000..0FFFh], then set cop0_sr=zero.
 Init Cache Step 2:
  [FFFE0130h]=00000800h, then set cop0_sr=00010000h, then
  zerofill ALL words at [0000h..0FFFh], then set cop0_sr=zero.
 Finish Initialization:
  read 8 times 32bit from [A0000000h], then set [FFFE0130h]=0001E988h
```

Note: FFFE0130h is described in LSI's "L64360" datasheet, chapter 14 (and probably also in their LR33300/LR33310 datasheet, if it were available in internet).

Unpredictable Things

Normally, I/O ports should be accessed only at their corresponding size (ie. 16bit read/write for 16bit ports), and of course, only existing memory and I/O addresses should be used. When not recursing that rules, some more or less (un-)predictable things may happen...

I/O Write Datasize

Address	Content	W.8bit	W.16bit	W.32bit	
00000000h-00xFFFFFh	Main RAM	OK	OK	OK	
1F800000h-1F8003FFh	Scratchpad	OK	OK	OK	
1F801000h-1F801023h	MEMCTRL	(w32)	(w32)	OK	
1F80104xh	JOY_xxx	(w16)	OK	CROP	
1F80105xh	SIO_xxx	(w16)	OK	CROP	
1F801060h-1F801063h	RAM_SIZE	(w32)	(w32)	OK	(with crash)
1F801070h-1F801077h	IRQCTRL	(w32)	(w32)	OK	
1F8010x0h-1F8010x3h	DMAx.ADDR	(w32)	(w32)	OK	
1F8010x4h-1F8010x7h	DMAx.LEN	OK	OK	OK	
1F8010x8h-1F8010xFh	DMAx.CTRL/MIRR	(w32)	(w32)	OK	
1F8010F0h-1F8010F7h	DMA.DPCR/DICR	(w32)	(w32)	OK	
1F8010F8h-1F8010FFh	DMA.unknown	IGNORE	IGNORE	IGNORE	
1F801100h-1F80110Bh	Timer 0	(w32)	(w32)	OK	
1F801110h-1F80110Bh	Timer 1	(w32)	(w32)	OK	
1F801120h-1F80110Bh	Timer 2	(w32)	(w32)	OK	
1F801800h-1F801803h	CDROM	OK	?	?	
1F801810h-1F801813h	GPU.GP0	?	?	OK	
1F801814h-1F801817h	GPU.GP1	?	?	OK	
1F801820h-1F801823h	MDEC.CMD/DTA	?	?	OK	
1F801824h-1F801827h	MDEC.CTRL	?	?	OK	
1F801C00h-1F801E7Fh	SPU	(i16)	OK	OK	
1F801E80h-1F801FFFh	SPU.UNUSED	IGNORE	IGNORE	IGNORE	
1F802020h-1F80202Fh	DUART	OK	?	?	
1F802041h	POST	OK	?	?	
FFFE0130h-FFFE0133h	CACHE.CTRL	(i32)	(i32)	OK	

Whereas,

```
  OK    works
  (w32) write full 32bits (left-shifted if address isn't word-aligned)
  (w16) write full 16bits (left-shifted if address isn't halfword-aligned)
  (i32) write full 32bits (ignored if address isn't word-aligned)
  (i16) write full 16bits (ignored if address isn't halfword-aligned)
  CROP  write only lower 16bit (and leave upper 16bit unchanged)
```

It's somewhat "legit" to use 16bit writes on 16bit registers like RAM_SIZE, I_STAT, I_MASK, and Timer 0-2.
Non-4-byte aligned 8bit/16bit writes to RAM_SIZE do crash (probably because the "(w32)" effect is left-shifting the value, so lower 8bit become zero).
Results on unaligned I/O port writes (via SWL/SWR opcodes) are unknown.

I/O Read Datasize

In most cases, I/O ports can be read in 8bit, 16bit, or 32bit units, regardless of their size, among others allowing to read two 16bit ports at once with a single 32bit read. If there's only one 16bit port within a 32bit region, then 32bit reads often return garbage in the unused 16bits. Also, 8bit or 16bit VRAM data reads via GPUREAD probably won't work? Expansion 2 Region can be accessed only via 8bit reads, and 16bit/32bit reads seem to cause exceptions (or rather: no such exception!) (except, probably 16bit reads are allowed when the region is configured to 16bit databus width).
There are at least some special cases:

```
  FFFE0130h-FFFE0133h  8bit (+16bit?) read works ONLY from word-aligned address
```

Cache Problems

The functionality of the Cache is still widely unknown. Not sure if DMA transfers are updating or invalidating cache. Cached Data within KSEG0 should be automatically also cached at the corresponding mirrored address in KUSEG and vice versa. Mirrors within KSEG1 (or within KUSEG) may be a different thing, eg. when using addresses spead across the first 8MB region to access the 2MB RAM. Same problems may occor for Expansion and BIOS mirrors, although, not sure if that regions are cached.

Writebuffer Problems

The writebuffer seems to be disabled for the normal I/O area at 1F801000h, however, it appears to be enabled for the Expansion I/O region at 1F802000h (after writing to 1F802041h, the BIOS issues 4 dummy writes to RAM, apparently (?) in order to flush the writebuffer). The same might apply for Expansion Memory region at 1F000000h, although usually that region would contain ROM, so it'd be don't care whether it is write-buffered or not.

CPU Load/Store Problems

XXcpuREG ---> applies ONLY to LOAD (not to store)
Memory read/write opcodes take a 1-cycle delay until the data arrives at the destination, ie. the next opcode should not use the destination register (or more unlikely, the destination memory location) as source operand. Usually, when trying to do so, the second opcode would receive the OLD value - however, if an exception occurs between the two opcodes, then the read/write operation may finish, and the second opcode would probably receive the NEW value.

CPU Register Problems - R1 (AT), R26 (K0), R29 (SP)

Exception handlers cannot preserve all registers, before returning, they must load the return address into a general purpose register (conventionally R26 aka K0), so be careful not to use that register, unless you are 100% sure that no interrupts and no other exceptions can occur. Some exception handlers might also destroy R27

aka AT (although exception handler in the PSX Kernel leaves that register unchanged).
Some assemblers (not a22i in nocash syntax mode) are internally using R1 aka AT as scratch register for some pseudo opcodes, including for a "sw rx,imm32" pseudo opcode (which is nearly impossible to separate from the normal "sw rx,imm16" opcode), be careful not to use R1, unless you can trust your assembler not to destroy that register behind your back.
The PSX Kernel uses "Full-Decrementing-Wasted-Stack", where "Wasted" means that when calling a sub-function with N parameters, then the caller must pre-allocate N works on stack, and the sub-function may freely use and destroy these words; at [SP+0..N*4-1].

Locked Locations in Memory and I/O Area

```
00800000h           ;-when Main RAM configured to end at 7FFFFFh
1F080000h 780000h   ;-when Expansion 1 configured to end at 7FFFFh
1F800400h C00h      ;-region after Scratchpad
1F801024h 1Ch       ;\
1F801064h 0Ch       ;
1F801078h 08h       ;
1F801140h 6C0h      ; gaps in I/O region
1F801804h 0Ch       ;
1F801818h 08h       ;
1F801828h 3D8h      ;/
1F802080h 3FDF80h   ;-when Expansion 2 configured to end at 7Fh
1FC80000h 60380000h ;-when BIOS ROM configured to end at 7FFFFh
C0000000h 1FFE0000h ;\
FFFE0020h E0h       ; gaps in KSEG2 (cache control region)
FFFE0140h 1FEC0h    ;/
```

Trying to access these locations generates an exception. For KSEG0 and KSEG1, locked regions are same as for first 512MB of KUSEG.

Mirrors in I/O Area

```
1F80108Ch+N*10h - D#_CHCR Mirrors - (N=0..6, for DMA channel 0..6)
```

Read/writeable mirrors of DMA Control registers at 1F801088h+N*10h.

Garbage Locations in I/O Area

```
1F801062h (2 bytes)  ;\
1F801072h (2 bytes)  ; unused addresses in Memory and Interrupt Control area
1F801076h (2 bytes)  ;/
1F801102h (2 bytes)  ;\
1F801106h (2 bytes)  ; unused addresses in Timer 0 area
1F80110Ah (6 bytes)  ;/
1F801112h (2 bytes)  ;\
1F801116h (2 bytes)  ; unused addresses in Timer 1 area
1F80111Ah (6 bytes)  ;/
1F801122h (2 bytes)  ;\
1F801126h (2 bytes)  ; unused addresses in Timer 2 area and next some bytes
1F80112Ah (22 bytes) ;/
1F801820h (4 bytes)  ;-read MDEC Data-Out port (if there is no data)
FFFE0000h (32 bytes) ;\
FFFE0100h (48 bytes) ; unused addresses in Cache control area
FFFE0132h (2 bytes)  ; (including write-only upper 16bit of Port FFFE0130h)
FFFE0134h (12 bytes) ;/
```

Unlike all other unused I/O addresses, these addresses are unlocked (ie. they do not trigger exceptions on access), however they do not seem to contain anything useful. The BIOS never seems to use them. Writing any values to them seems to have no effect. And reading acts somewhat unstable:
Usually returns zeros in most cases. Except that, the first byte on a 10h-byte boundary often returns the lower 8bit of the memory address (eg. [FFFE0010h]=10h). And, when [FFFE0130h].Bit11=0, then reading from these registers does return the 32bit opcode that is to be executed next (or at some locations, the opcode thereafter).

PSX as Abbreviation for Playstation 1
In gaming and programming scene, "PSX" is most commonly used as abbreviation for the original Playstation series (occasionally including PSone). Sony has never officially used that abbreviation, however, the Playstation BIOS contains the ASCII strings "PSX" and "PS-X" here and there. The letters "PS" are widely believed to stand for PlayStation, and the meaning of the "X" is totally unknown (although, actually it may stand for POSIX.1, see below).

PSX as Abbreviation for POSIX.1
According to JMI Software Systems, "PSX" is a trademark of themselves, and stands for "single-user, single-group, subset of POSIX.1" (POSIX stands for something commonly used by HLL programmers under UNIX or so). That "PSX" kernel from JMI is available for various processors, including MIPS processors, and like the playstation, it does include functions like "atoi", and does support TTY access via Signetics 2681 DUART chips. The DTL-H2000 does also have POSIX-style "PSX>" prompt. So, altogether, it's quite possible that Sony has licensed the kernel from JMI.

PSX as Abbreviation for an Extended Playstation 2
As everybody agrees, PSX should be used only as abbreviation for Playstation 1, and nobody should never ever use it for the Playstation 2. Well, nobody, except Sony... despite of the common use as abbreviation for Playstation 1 (and despite of the JMI trademark)... in 2003, Sony has have released a "Playstation 2 with built-in HDD/DVD Videorecorder" and called that thing "PSX" for the best of confusion.

CPU Specifications

CPU
CPU Registers
CPU Opcode Encoding
CPU Load/Store Opcodes
CPU ALU Opcodes
CPU Jump Opcodes
CPU Coprocessor Opcodes
CPU Pseudo Opcodes

System Control Coprocessor (COP0)
COP0 - Register Summary
COP0 - Exception Handling
COP0 - Misc
COP0 - Debug Registers

CPU Registers

All registers are 32bit wide.

```
(R0)       zero     Constant (always 0) (this one isn't a real register)
R1         at       Assembler temporary (destroyed by some pseudo opcodes!)
R2-R3      v0-v1    Subroutine return values, may be changed by subroutines
R4-R7      a0-a3    Subroutine arguments, may be changed by subroutines
R8-R15     t0-t7    Temporaries, may be changed by subroutines
R16-R23    s0-s7    Static variables, must be saved by subs
R24-R25    t8-t9    Temporaries, may be changed by subroutines
R26-R27    k0-k1    Reserved for kernel (destroyed by some IRQ handlers!)
R28        gp       Global pointer (rarely used)
R29        sp       Stack pointer
R30        fp(s8)   Frame Pointer, or 9th Static variable, must be saved
R31        ra       Return address (used so by JAL,BLTZAL,BGEZAL opcodes)
-          pc       Program counter
-          hi,lo    Multiply/divide results, may be changed by subroutines
```

R0 is always zero.
R31 can be used as general purpose register, however, some opcodes are using it to store the return address: JAL, BLTZAL, BGEZAL. (Note: JALR can optionally store the return address in R31, or in R1..R30. Exceptions store the return address in cop0r14 - EPC).

R29 (SP) - Full Decrementing Wasted Stack Pointer
The CPU doesn't explicitly have stack-related registers or opcodes, however, conventionally, R29 is used as stack pointer (SP). The stack can be accessed with normal load/store opcodes, which do not automatically increase/decrease SP, so the SP register must be manually modified to (de-)allocate data.
The PSX BIOS is using "Full Decrementing Wasted Stack".
Decrementing means that SP gets decremented when allocating data (that's common for most CPUs) - Full means that SP points to the first ALLOCATED word on the stack, so the allocated memory is at SP+0 and above, free memory at SP-1 and below, Wasted means that when calling a sub-function with N parameters, then the caller must pre-allocate N works on stack, and the sub-function may freely use and destroy these words; at [SP+0..N*4-1].

For example, "push ra,r16,r17" would be implemented as:

```
sub  sp,20h
mov  [sp+14h],ra
mov  [sp+18h],r16
mov  [sp+1Ch],r17
```

where the allocated 20h bytes have the following purpose:

```
[sp+00h..0Fh]  wasted stack (may, or may not, be used by sub-functions)
[sp+10h..13h]  8-byte alignment padding (not used)
[sp+14h..1Fh]  pushed registers
```

CPU Opcode Encoding

Primary opcode field (Bit 26..31)

```
00h=SPECIAL 08h=ADDI  10h=COP0 18h=N/A   20h=LB   28h=SB   30h=LWC0 38h=SWC0
01h=BcondZ  09h=ADDIU 11h=COP1 19h=N/A   21h=LH   29h=SH   31h=LWC1 39h=SWC1
02h=J       0Ah=SLTI  12h=COP2 1Ah=N/A   22h=LWL  2Ah=SWL  32h=LWC2 3Ah=SWC2
03h=JAL     0Bh=SLTIU 13h=COP3 1Bh=N/A   23h=LW   2Bh=SW   33h=LWC3 3Bh=SWC3
04h=BEQ     0Ch=ANDI  14h=N/A  1Ch=N/A   24h=LBU  2Ch=N/A  34h=N/A  3Ch=N/A
05h=BNE     0Dh=ORI   15h=N/A  1Dh=N/A   25h=LHU  2Dh=N/A  35h=N/A  3Dh=N/A
06h=BLEZ    0Eh=XORI  16h=N/A  1Eh=N/A   26h=LWR  2Eh=SWR  36h=N/A  3Eh=N/A
07h=BGTZ    0Fh=LUI   17h=N/A  1Fh=N/A   27h=N/A  2Fh=N/A  37h=N/A  3Fh=N/A
```

Secondary opcode field (Bit 0..5) (when Primary opcode = 00h)

```
00h=SLL   08h=JR      10h=MFHI 18h=MULT  20h=ADD  28h=N/A  30h=N/A  38h=N/A
01h=N/A   09h=JALR    11h=MTHI 19h=MULTU 21h=ADDU 29h=N/A  31h=N/A  39h=N/A
02h=SRL   0Ah=N/A     12h=MFLO 1Ah=DIV   22h=SUB  2Ah=SLT  32h=N/A  3Ah=N/A
03h=SRA   0Bh=N/A     13h=MTLO 1Bh=DIVU  23h=SUBU 2Bh=SLTU 33h=N/A  3Bh=N/A
04h=SLLV  0Ch=SYSCALL 14h=N/A  1Ch=N/A   24h=AND  2Ch=N/A  34h=N/A  3Ch=N/A
05h=N/A   0Dh=BREAK   15h=N/A  1Dh=N/A   25h=OR   2Dh=N/A  35h=N/A  3Dh=N/A
06h=SRLV  0Eh=N/A     16h=N/A  1Eh=N/A   26h=XOR  2Eh=N/A  36h=N/A  3Eh=N/A
07h=SRAV  0Fh=N/A     17h=N/A  1Fh=N/A   27h=NOR  2Fh=N/A  37h=N/A  3Fh=N/A
```

Opcode/Parameter Encoding

```
31..26 |25..21|20..16|15..11|10..6 |  5..0  |
 6bit  | 5bit | 5bit | 5bit | 5bit |  6bit  |
-------+------+------+------+------+--------+------------
000000 | N/A  | rt   | rd   | imm5 | 0000xx | shift-imm
000000 | rs   | rt   | rd   | N/A  | 0001xx | shift-reg
000000 | rs   | N/A  | N/A  | N/A  | 001000 | jr
000000 | rs   | N/A  | rd   | N/A  | 001001 | jalr
000000 | <-----comment20bit------> | 00110x | sys/brk
000000 | N/A  | N/A  | rd   | N/A  | 0100x0 | mfhi/mflo
000000 | rs   | N/A  | N/A  | N/A  | 0100x1 | mthi/mtlo
000000 | rs   | rt   | N/A  | N/A  | 0110xx | mul/div
000000 | rs   | rt   | rd   | N/A  | 10xxxx | alu-reg
000001 | rs   | 00000| <--immediate16bit--> | bltz
000001 | rs   | 00001| <--immediate16bit--> | bgez
000001 | rs   | 10000| <--immediate16bit--> | bltzal
000001 | rs   | 10001| <--immediate16bit--> | bgezal
00001x | <---------immediate26bit---------> | j/jal
00010x | rs   | rt   | <--immediate16bit--> | beq/bne
00011x | rs   | N/A  | <--immediate16bit--> | blez/bgtz
001xxx | rs   | rt   | <--immediate16bit--> | alu-imm
001111 | N/A  | rt   | <--immediate16bit--> | lui-imm
100xxx | rs   | rt   | <--immediate16bit--> | load rt,[rs+imm]
101xxx | rs   | rt   | <--immediate16bit--> | store rt,[rs+imm]
x1xxxx | <------coprocessor specific------> | coprocessor (see below)
```

Coprocessor Opcode/Parameter Encoding

```
31..26 |25..21|20..16|15..11|10..6 |  5..0  |
 6bit  | 5bit | 5bit | 5bit | 5bit |  6bit  |
-------+------+------+------+------+--------+------------
0100nn |0|0000| rt   | rd   | N/A  | 000000 | MFCn rt,rd_dat  ;rt = dat
0100nn |0|0010| rt   | rd   | N/A  | 000000 | CFCn rt,rd_cnt  ;rt = cnt
0100nn |0|0100| rt   | rd   | N/A  | 000000 | MTCn rt,rd_dat  ;dat = rt
0100nn |0|0110| rt   | rd   | N/A  | 000000 | CTCn rt,rd_cnt  ;cnt = rt
0100nn |0|1000|00000 | <--immediate16bit--> | BCnF target ;jump if false
0100nn |0|1000|00001 | <--immediate16bit--> | BCnT target ;jump if true
0100nn |1| <--------immediate25bit--------> | COPn imm25
```

```
010000 |1|0000| N/A  | N/A  | N/A  | 000010 | COP0 02h  ;=TLBWI
010000 |1|0000| N/A  | N/A  | N/A  | 000110 | COP0 06h  ;=TLBWR
010000 |1|0000| N/A  | N/A  | N/A  | 001000 | COP0 08h  ;=TLBP
010000 |1|0000| N/A  | N/A  | N/A  | 010000 | COP0 10h  ;=RFE
1100nn | rs   | rt   | <--immediate16bit--> | LWCn rt_dat,[rs+imm]
1110nn | rs   | rt   | <--immediate16bit--> | SWCn rt_dat,[rs+imm]
```

Illegal Opcodes
All opcodes that are marked as "N/A" in the Primary and Secondary opcode tables are causing a Reserved Instruction Exception (excode=0Ah).
The unused operand bits (eg. Bit21-25 for LUI opcode) should be usually zero, but do not necessarily trigger exceptions if set to nonzero values.

CPU Load/Store Opcodes

Load instructions
```
movbs rt,[imm+rs]  lb  rt,imm(rs)    rt=[imm+rs]  ;byte sign-extended
movb  rt,[imm+rs]  lbu rt,imm(rs)    rt=[imm+rs]  ;byte zero-extended
movhs rt,[imm+rs]  lh  rt,imm(rs)    rt=[imm+rs]  ;halfword sign-extended
movh  rt,[imm+rs]  lhu rt,imm(rs)    rt=[imm+rs]  ;halfword zero-extended
mov   rt,[imm+rs]  lw  rt,imm(rs)    rt=[imm+rs]  ;word
```
Load instructions can read from the data cache (if the data is not in the cache, or if the memory region is uncached, then the CPU gets halted until it has read the data) (however, the PSX doesn't have a data cache).

Caution - Load Delay
The loaded data is NOT available to the next opcode, ie. the target register isn't updated until the next opcode has completed. So, if the next opcode tries to read from the load destination register, then it would (usually) receive the OLD value of that register (unless an IRQ occurs between the load and next opcode, in that case the load would complete during IRQ handling, and so, the next opcode would receive the NEW value).

Store instructions
```
movb  [imm+rs],rt  sb  rt,imm(rs)    [imm+rs]=(rt AND FFh)   ;store 8bit
movh  [imm+rs],rt  sh  rt,imm(rs)    [imm+rs]=(rt AND FFFFh) ;store 16bit
mov   [imm+rs],rt  sw  rt,imm(rs)    [imm+rs]=rt             ;store 32bit
```
Store operations are passed to the write-buffer, so they can execute within a single clock cycle (unless the write-buffer was full, in that case the CPU gets halted until there's room in the buffer). But, the PSX doesn't have a writebuffer...?

Load/Store Alignment
Halfword addresses must be aligned by 2, word addresses must be aligned by 4, trying to access mis-aligned addresses will cause an exception. There's no alignment restriction for bytes.

Unaligned Load/Store
```
lwr   rt,imm(rs)     load right bits of rt from memory (usually imm+0)
lwl   rt,imm(rs)     load left  bits of rt from memory (usually imm+3)
swr   rt,imm(rs)     store right bits of rt to memory (usually imm+0)
swl   rt,imm(rs)     store left  bits of rt to memory (usually imm+3)
```
There's no delay required between lwl and lwr, so you can use them directly following eachother, eg. to load a word anywhere in memory without regard to alignment:
```
lwl   r2,$0003(t0)   ;\no delay required between these
lwr   r2,$0000(t0)   ;/(although both access r2)
nop                  ;-requires load delay HERE (before reading from r2)
and   r2,r2,0ffffh   ;-access r2 (eg. reducing it to unaligned 16bit data)
```

Unaligned Load/Store (Details)
LWR/SWR transfers the right (=lower) bits of Rt, up-to 32bit memory boundary:
```
lwr/swr [N*4+0]     transfer whole 32bit of Rt to/from [N*4+0..3]
lwr/swr [N*4+1]     transfer lower 24bit of Rt to/from [N*4+1..3]
lwr/swr [N*4+2]     transfer lower 16bit of Rt to/from [N*4+2..3]
lwr/swr [N*4+3]     transfer lower  8bit of Rt to/from [N*4+3]
```
LWL/SWL transfers the left (=upper) bits of Rt, down-to 32bit memory boundary:
```
lwl/swl [N*4+0]     transfer upper  8bit of Rt to/from [N*4+0]
lwl/swl [N*4+1]     transfer upper 16bit of Rt to/from [N*4+0..1]
lwl/swl [N*4+2]     transfer upper 24bit of Rt to/from [N*4+0..2]
lwl/swl [N*4+3]     transfer whole 32bit of Rt to/from [N*4+0..3]
```
The CPU has four separate byte-access signals, so, within a 32bit location, it can transfer all fragments of Rt at once (including for odd 24bit amounts). The transferred data is not zero- or sign-expanded, eg. when transferring 8bit data, the other 24bit of Rt and [mem] will remain intact.

Note: The aligned variant can also misused for blocking memory access on aligned addresses (in that case, if the address is known to be aligned, only one of the opcodes are needed, either LWL or LWR).... Uhhhhhhhm, OR is that NOT allowed... more PROBABLY that doesn't work?

CPU ALU Opcodes

arithmetic instructions
```
addt rd,rs,rt    add   rd,rs,rt         rd=rs+rt (with overflow trap)
add  rd,rs,rt    addu  rd,rs,rt         rd=rs+rt
subt rd,rs,rt    sub   rd,rs,rt         rd=rs-rt (with overflow trap)
sub  rd,rs,rt    subu  rd,rs,rt         rd=rs-rt
addt rt,rs,imm   addi  rt,rs,imm        rt=rs+(-8000h..+7FFFh) (with ov.trap)
add  rt,rs,imm   addiu rt,rs,imm        rt=rs+(-8000h..+7FFFh)
```
The opcodes "with overflow trap" do trigger an exception (and leave rd unchanged) in case of overflows.

comparison instructions
```
setlt slt   rd,rs,rt  if rs<rt then rd=1 else rd=0 (signed)
setb  sltu  rd,rs,rt  if rs<rt then rd=1 else rd=0 (unsigned)
setlt slti  rt,rs,imm if rs<(-8000h..+7FFFh)  then rt=1 else rt=0 (signed)
setb  sltiu rt,rs,imm if rs<(FFFF8000h..7FFFh) then rt=1 else rt=0(unsigned)
```

logical instructions
```
and  rd,rs,rt    and  rd,rs,rt         rd = rs AND rt
or   rd,rs,rt    or   rd,rs,rt         rd = rs OR  rt
xor  rd,rs,rt    xor  rd,rs,rt         rd = rs XOR rt
nor  rd,rs,rt    nor  rd,rs,rt         rd = FFFFFFFFh XOR (rs OR rt)
and  rt,rs,imm   andi rt,rs,imm        rt = rs AND (0000h..FFFFh)
```

```
  xor  rt,rs,imm   xori rt,rs,imm       rt = rs XOR (0000h..FFFFh)
```

shifting instructions

```
  shl  rd,rt,rs    sllv rd,rt,rs        rd = rt SHL (rs AND 1Fh)
  shr  rd,rt,rs    srlv rd,rt,rs        rd = rt SHR (rs AND 1Fh)
  sar  rd,rt,rs    srav rd,rt,rs        rd = rt SAR (rs AND 1Fh)
  shl  rd,rt,imm   sll  rd,rt,imm       rd = rt SHL (00h..1Fh)
  shr  rd,rt,imm   srl  rd,rt,imm       rd = rt SHR (00h..1Fh)
  sar  rd,rt,imm   sra  rd,rt,imm       rd = rt SAR (00h..1Fh)
  mov  rt,i*10000h lui  rt,imm          rt = (0000h..FFFFh) SHL 16
```

Unlike many other opcodes, shifts use 'rt' as second (not third) operand.
The hardware does NOT generate exceptions on SHL overflows.

Multiply/divide

```
  smul rs,rt       mult   rs,rt         hi:lo = rs*rt (signed)
  umul rs,rt       multu  rs,rt         hi:lo = rs*rt (unsigned)
  sdiv rs,rt       div    rs,rt         lo = rs/rt, hi=rs mod rt (signed)
  udiv rs,rt       divu   rs,rt         lo = rs/rt, hi=rs mod rt (unsigned)
  mov  rd,hi       mfhi   rd            rd=hi  ;move from hi
  mov  rd,lo       mflo   rd            rd=lo  ;move from lo
  mov  hi,rs       mthi   rs            hi=rs  ;move to hi
  mov  lo,rs       mtlo   rs            lo=rs  ;move to lo
```

The mul/div opcodes are starting the multiply/divide operation, starting takes only a single clock cycle, however, trying to read the result from the hi/lo registers while the mul/div operation is busy will halt the CPU until the mul/div has completed. For multiply, the execution time depends on rs (ie. "small*large" can be much faster than "large*small").

```
  __umul_execution_time_____________________________________________
  Fast  (6 cycles)   rs = 00000000h..000007FFh
  Med   (9 cycles)   rs = 00000800h..000FFFFFh
  Slow  (13 cycles)  rs = 00100000h..FFFFFFFFh
  __smul_execution_time_____________________________________________
  Fast  (6 cycles)   rs = 00000000h..000007FFh, or rs = FFFFF800h..FFFFFFFFh
  Med   (9 cycles)   rs = 00000800h..000FFFFFh, or rs = FFF00000h..FFFFF801h
  Slow  (13 cycles)  rs = 00100000h..7FFFFFFFh, or rs = 80000000h..FFF00001h
  __udiv/sdiv_execution_time________________________________________
  Fixed (36 cycles)  no matter of rs and rt values
```

For example, when executing "umul 123h,12345678h" and "mov r1,lo", one can insert up to six (cached) ALU opcodes, or read one value from PSX Main RAM (which has 6 cycle access time) between the "umul" and "mov" opcodes without additional slowdown.
The hardware does NOT generate exceptions on divide overflows, instead, divide errors are returning the following values:

```
  Opcode  Rs              Rd       Hi/Remainder  Lo/Result
  udiv    0..FFFFFFFFh    0   -->  Rs            FFFFFFFFh
  sdiv    0..+7FFFFFFFh   0   -->  Rs            -1
  sdiv    -80000000h..-1  0   -->  Rs            +1
  sdiv    -80000000h      -1  -->  0             -80000000h
```

For udiv, the result is more or less correct (as close to infinite as possible). For sdiv, the results are total garbage (about farthest away from the desired result as possible).
Note: After accessing the lo/hi registers, there seems to be a strange rule that one should not touch the lo/hi registers in the next 2 cycles or so... not yet understood if/when/how that rule applies...?

CPU Jump Opcodes

jumps and branches

Note that the instruction following the branch will always be executed.

```
  jmp  dest        j      dest        pc=(pc and F0000000h)+(imm26bit*4)
  call dest        jal    dest        pc=(pc and F0000000h)+(imm26bit*4),ra=$+8
  jmp  rs          jr     rs          pc=rs
  call rs,ret=rd   jalr (rd,)rs(,rd)  pc=rs, rd=$+8 ;see caution
  je   rs,rt,dest  beq    rs,rt,dest  if rs=rt  then pc=$+4+(-8000h..+7FFFh)*4
  jne  rs,rt,dest  bne    rs,rt,dest  if rs<>rt then pc=$+4+(-8000h..+7FFFh)*4
  js   rs,dest     bltz   rs,dest     if rs<0   then pc=$+4+(-8000h..+7FFFh)*4
  jns  rs,dest     bgez   rs,dest     if rs>=0  then pc=$+4+(-8000h..+7FFFh)*4
  jgtz rs,dest     bgtz   rs,dest     if rs>0   then pc=$+4+(-8000h..+7FFFh)*4
  jlez rs,dest     blez   rs,dest     if rs<=0  then pc=$+4+(-8000h..+7FFFh)*4
  calls  rs,dest   bltzal rs,dest     if rs<0   then pc=$+4+(..)*4, ra=$+8
  callns rs,dest   bgezal rs,dest     if rs>=0  then pc=$+4+(..)*4, ra=$+8
```

JALR cautions

Caution: The JALR source code syntax varies (IDT79R3041 specs say "jalr rs,rd", but MIPS32 specs say "jalr rd,rs"). Moreover, JALR may not use the same register for both operands (eg. "jalr r31,r31") (doing so would destroy the target address; which is normally no problem, but it can be a problem if an IRQ occurs between the JALR opcode and the following branch delay opcode; in that case BD gets set, and EPC points "back" to the JALR opcode, so JALR is executed twice, with destroyed target address in second execution).

exception opcodes

Unlike for jump/branch opcodes, exception opcodes are immediately executed (ie. without executing the following opcode).

```
  syscall  imm20        generates a system call exception
  break    imm20        generates a breakpoint exception
```

The 20bit immediate doesn't affect the CPU (however, the exception handler may interprete it by software; by examing the opcode bits at [epc-4]).

CPU Coprocessor Opcodes

Coprocessor Instructions (COP0..COP3)

```
  mov  rt,cop#Rd(0-31)       mfc# rt,rd       ;rt = cop#datRd ;data regs
  mov  rt,cop#Rd(32-63)      cfc# rt,rd       ;rt = cop#cntRd ;control regs
  mov  cop#Rd(0-31),rt       mtc# rt,rd       ;cop#datRd = rt ;data regs
  mov  cop#Rd(32-63),rt      ctc# rt,rd       ;cop#cntRd = rt ;control regs
  mov  cop#cmd,imm25         cop# imm25       ;exec cop# command 0..1FFFFFFh
  mov  cop#Rt(0-31),[rs+imm] lwc# rt,imm(rs)  ;cop#dat_rt = [rs+imm]  ;word
  mov  [rs+imm],cop#Rt(0-31) swc# rt,imm(rs)  ;[rs+imm] = cop#dat_rt  ;word
  jf   cop#flg,dest          bc#f dest        ;if cop#flg=false then pc=$+disp
  jt   cop#flg,dest          bc#t dest        ;if cop#flg=true  then pc=$+disp
  rfe                        rfe              ;return from exception (COP0)
  tlb<xx>                    tlb<xx>          ;virtual memory related (COP0)
```

Unknown if any tlb-opcodes (tlbr,tlbwi,tlbwr,tlbp) are implemented in the psx?

Caution - Load Delay
When reading from a coprocessor register, the next opcode cannot use the destination register as operand (much the same as the Load Delays that occur when reading from memory; see there for details).
Reportedly, the Load Delay applies for the next TWO opcodes after coprocessor reads, but, that seems to be nonsense (the PSX does finish both COP0 and COP2 reads after ONE opcode).

Caution - Store Delay
In some cases, a similar delay occurs when writing to a coprocessor register. COP0 is more or less free of store delays (eg. one can read from a cop0 register immediately after writing to it), the only known exception is the cop2 enable bit in cop0r12.bit30 (setting that cop0 bit acts delayed, and cop2 isn't actually enabled until after 2 clock cycles or so).
Writing to cop2 registers has a delay of 2..3 clock cycles. In most cases, that is probably (?) only 2 cycles, but special cases like writing to IRGB (which does additionally affect IR1,IR2,IR3) take 3 cycles until the result arrives in all registers).
Note that Store Delays are counted in numbers of clock cycles (not in numbers of opcodes). For 3 cycle delay, one must usually insert 3 cached opcodes (or one uncached opcode).

CPU Pseudo Opcodes

Pseudo instructions (native/spasm)
```
nop                 ;alias for sll r0,r0,0
move rd,rs          ;alias for addu rd,rs,r0
la   rx,imm32       ;load address   (alias for lui rx / addiu rx)
li   rx,imm32       ;load immediate (alias for lui rx / ori   rx)
li   rx,imm16       ;load immediate (alias for ori, range 0..FFFFh)
li   rx,-imm15      ;load immediate (alias for addiu, range -1..-8000h)
li   rx,imm16*10000h ;load immediate (alias for lui)
lw   rx,imm32       ;load from address (lui rx / lw rx,rx)
sw   rx,imm32       ;store to address  (lui r1 / sw rx,r1) (destroys r1!)
lb,lh,lwl,lwr,lbu,lhu;as above pseudo lw
sb,sh,swl,swr       ;as above pseudo sw (ie. also destroys r1!)
alu  rx,op          ;alias for alu  rx,rx,op
alu(u) rx,rx,imm    ;alias for alui(u) rx,rx,imm
jalr  rx            ;alias for jalr (RA,)rx(,RA)
subi(u) rt,rs,imm   ;alias for addi(u) rt,rs,-imm
beqz rx,dest        ;alias for beq rx,r0,dest
bnez rx,dest        ;alias for bne rx,r0,dest
b   dest            ;alias for beq r0,r0,dest (jump relative/spasm)
bra dest            ;alias for ...? (jump relative/gnu)
bal dest            ;alias for ...? (call relative/spasm)
```

Pseudo instructions (nocash/a22i)
```
mov  rx,NNNN0000h   ;alias for lui  rx,NNNNh
mov  rx,0000NNNNh   ;alias for or   rx,r0,NNNNh  ;max +FFFFh
mov  rx,-imm15      ;alias for add  rx,r0,-NNNNh ;min -8000h
mov  rx,ry          ;alias for or   rx,ry,0  (or "addiu")
nop                 ;alias for shl  r0,r0,0
jrel dest           ;alias for blez R0,dest   ;relative jump
crel dest           ;alias for callns R0,dest ;relative call
jz   rx,dest        ;alias for je   rx,R0,dest
jnz  rx,dest        ;alias for jne  rx,R0,dest
call rx             ;alias for call rx,ret=RA
ret                 ;alias for jmp  ra
subt rt,rs,imm      ;alias for addt rt,rs,-imm
sub  rt,rs,imm      ;alias for add  rt,rs,-imm
alu  rx,op          ;alias for alu  rx,rx,op
neg(t) rx,ry        ;alias for sub(t) rx,R0,ry
not    rx,ry        ;alias for nor    rx,R0,ry
neg(t)/not rx       ;alias for neg(t)/not rx,rx
setz rx,ry          ;alias for setb rx,ry,1   (set if zero)
setnz rx,ry         ;alias for setb rx,R0,ry  (set if nonzero)
syscall/break       ;alias for syscall/break 000000h
```
Below are pseudo instructions combined of two 32bit opcodes...
```
movp rx,imm32       ;alias for lui  rx,imm16 -plus- or rx,rx,imm16)
mov(bhs)p rx,[imm32] ;load from address (lui rx,imm16 / mov rx,[rx+imm16])
movu [rs+imm]       ;alias for lwr/swr [rs+imm] plus lwl/swl [rs+imm+3]
reti                ;alias for jmp k0 plus rfe
```
Below are pseudo instructions combined of two or more 32bit opcodes...
```
push rlist          ;alias for sub sp,n*4 -- mov [sp+(1..n)*4],r1..rn
pop  rlist          ;alias for mov r1..rn,[sp+(1..n)*4] -- add sp,n*4
pop  pc,rlist       ;alias for pop ra,rlist -- jmp ra
```

Possible more Pseudos...
```
call x0000000h ;call y0000000h (could be half-working for mem mirrors?)
setae,setge    ;--> setb,setlt with swapped operands
```

Directives (nocash)
```
.mips          ;select MIPS instruction set (alternately .hc05 for MC68HC05)
.bios          ;create a .ROM file (instead of .EXE)
.auto_nop      ;append NOPs to jumps ;unless next opcode starts with a +
org imm        ;assume following code to be originated at address "imm"
db n(,n(..)))  ;define 8bit data values(s) or quoted ASCII strings
dw n(,n(..)))  ;define 16bit data values(s) (not 32bit data!)
dd n(,n(..)))  ;define 32bit data values(s)
.align imm
0              ;alias for immediate 0 and register R0 (whichever fits)
```

Directives (native)
```
org imm        ;self-explaining (but, default=$80010000 for spasm!)
align imm      ;self-explaining (probably zeropadded?)
db n(,n(..)))  ;define 8bit data values(s) or quoted ASCII strings
dh n(,n(..)))  ;define 16bit data values(s)
dw n(,n(..)))  ;define 32bit data values(s) (not 16bit data!)
dcb len,value  ;fill <len> bytes by <value> (different as DCB on ARM CPUs)
xyz            ;define label "xyz" at current address (without colon)
xyz equ n      ;assign value n to xyz
xyz = n        ;probably same/sililar as "equ"
```

```
incbin file.bin       ;import binary file
include file.asm      ;import asm file
zero          ;alias for r0
>imm32        ;alias for (i-(i AND 8000h))/10000h,  and/or i/10000h ?
<imm32        ;alias for (i AND 0FFFFh), used for SW(+/-) and ORI(+)?
end       ;N/A ;no "end" or ".end" directive needed/used by spasm
r1 aka at ;N/A ;some assemblers may (optionally) reject to use r1/at
```

Syntax for unknown assembler (for pad.s)
It uses "0x" for HEX values (but doesn't use "$" for registers).
It uses "#" instead of ";" for comments.
It uses ":" for labels (fortunately).
The assembler has at least one directive: ".byte" (equivalent to "db" on other assemblers).
I've no clue which assembler is used for that syntax... could that be the Psy-Q assembler?

COP0 - Register Summary

COP0 Register Summary

```
cop0r0-r2   - N/A
cop0r3      - BPC - Breakpoint on execute (R/W)
cop0r4      - N/A
cop0r5      - BDA - Breakpoint on data access (R/W)
cop0r6      - JUMPDEST - Randomly memorized jump address (R)
cop0r7      - DCIC - Breakpoint control (R/W)
cop0r8      - BadVaddr - Bad Virtual Address (R)
cop0r9      - BDAM - Data Access breakpoint mask (R/W)
cop0r10     - N/A
cop0r11     - BPCM - Execute breakpoint mask (R/W)
cop0r12     - SR - System status register (R/W)
cop0r13     - CAUSE - (R)  Describes the most recently recognised exception
cop0r14     - EPC - Return Address from Trap (R)
cop0r15     - PRID - Processor ID (R)
cop0r16-r31 - Garbage
cop0r32-r63 - N/A - None such (Control regs)
```

COP0 - Exception Handling

cop0r13 - CAUSE - (Read-only, except, Bit8-9 are R/W)
Describes the most recently recognised exception

```
0-1   -      Not used (zero)
2-6   Excode Describes what kind of exception occured:
               00h INT     Interrupt
               01h MOD     Tlb modification (none such in PSX)
               02h TLBL    Tlb load         (none such in PSX)
               03h TLBS    Tlb store        (none such in PSX)
               04h AdEL    Address error, Data load or Instruction fetch
               05h AdES    Address error, Data store
                           The address errors occur when attempting to read
                           outside of KUseg in user mode and when the address
                           is misaligned. (See also: BadVaddr register)
               06h IBE     Bus error on Instruction fetch
               07h DBE     Bus error on Data load/store
               08h Syscall Generated unconditionally by syscall instruction
               09h BP      Breakpoint - break instruction
               0Ah RI      Reserved instruction
               0Bh CpU     Coprocessor unusable
               0Ch Ov      Arithmetic overflow
               0Dh-1Fh     Not used
7     -      Not used (zero)
8-15  Ip     Interrupt pending field. Bit 8 and 9 are R/W, and
             contain the last value written to them. As long
             as any of the bits are set they will cause an
             interrupt if the corresponding bit is set in IM.
16-27 -      Not used (zero)
28-29 CE     Contains the coprocessor number if the exception
             occurred because of a coprocessor instuction for
             a coprocessor which wasn't enabled in SR.
30    -      Not used (zero)
31    BD     Is set when last exception points to the
             branch instuction instead of the instruction
             in the branch delay slot, where the exception
             occurred.
```

cop0r12 - SR - System status register (R/W)

```
0     IEc Current Interrupt Enable  (0=Disable, 1=Enable) ;rfe pops IUp here
1     KUc Current Kernal/User Mode  (0=Kernel, 1=User)    ;rfe pops KUp here
2     IEp Previous Interrupt Disable                      ;rfe pops IUo here
3     KUp Previous Kernal/User Mode                       ;rfe pops KUo here
4     IEo Old Interrupt Disable                       ;left unchanged by rfe
5     KUo Old Kernal/User Mode                        ;left unchanged by rfe
6-7   -   Not used (zero)
8-15  Im  8 bit interrupt mask fields. When set the corresponding
          interrupts are allowed to cause an exception.
16    Isc Isolate Cache (0=No, 1=Isolate)
            When isolated, all load and store operations are targetted
            to the Data cache, and never the main memory.
            (Used by PSX Kernel, in combination with Port FFFE0130h)
17    Swc Swapped cache mode (0=Normal, 1=Swapped)
            Instruction cache will act as Data cache and vice versa.
            Use only with Isc to access & invalidate Instr. cache entries.
            (Not used by PSX Kernel)
18    PZ  When set cache parity bits are written as 0.
19    CM  Shows the result of the last load operation with the D-cache
          isolated. It gets set if the cache really contained data
          for the addressed memory location.
20    PE  Cache parity error (Does not cause exception)
```

```
          matches 2 TLB entries.
          (initial value on reset allows to detect extended CPU version?)
  22    BEV Boot exception vectors in RAM/ROM (0=RAM/KSEG0, 1=ROM/KSEG1)
  23-24 -   Not used (zero)
  25    RE  Reverse endianness   (0=Normal endianness, 1=Reverse endianness)
            Reverses the byte order in which data is stored in
            memory. (lo-hi -> hi-lo)
            (Has affect only to User mode, not to Kernel mode) (?)
            (The bit doesn't exist in PSX ?)
  26-27 -   Not used (zero)
  28    CU0 COP0 Enable (0=Enable only in Kernal Mode, 1=Kernal and User Mode)
  29    CU1 COP1 Enable (0=Disable, 1=Enable) (none such in PSX)
  30    CU2 COP2 Enable (0=Disable, 1=Enable) (GTE in PSX)
  31    CU3 COP3 Enable (0=Disable, 1=Enable) (none such in PSX)
```

cop0r14 - EPC - Return Address from Trap (R)

```
  0-31  Return Address from Exception
```

This address is the instruction at which the exception took place, unless BD is set in CAUSE, then the instruction was at EPC+4.
Interrupts should always return to EPC+0, no matter of the BD flag. That way, if BD=1, the branch gets executed again, that's required because EPC stores only the current program counter, but not additionally the branch destination address.
Other exceptions may require to handle BD. In simple cases, when BD=0, the exception handler may return to EPC+0 (retry execution of the opcode), or to EPC+4 (skip the opcode that caused the exception). Note that jumps to faulty memory locations are executed without exception, but will trigger address errors and bus errors at the target location, ie. EPC (and BadVAddr, in case of address errors) point to the faulty address, not to the opcode that has jumped to that address).

Interrupts vs GTE Commands

If an interrupt occurs "on" a GTE command (cop2cmd), then the GTE command is executed, but nethertheless, the return address in EPC points to the GTE command. So, if the exeception handler would return to EPC as usually, then the GTE command would be executed twice. In best case, this would be a waste of clock cycles, in worst case it may lead to faulty result (if the results from the 1st execution are re-used as incoming parameters in the 2nd execution). To fix the problem, the exception handler must do:

```
  if (cause AND 7Ch)=00h              ;if excode=interrupt
   if ([epc] AND FE000000h)=4A000000h ;and opcode=cop2cmd
    epc=epc+4                         ;then skip that opcode
```

Note: The above exception handling is working only in newer PSX BIOSes, but in very old PSX BIOSes, it is only incompletely implemented (see "BIOS Patches" chapter for common workarounds; or write your own exception handler without using the BIOS).
Of course, the above exeption handling won't work in branch delays (where BD gets set to indicate that EPC was modified) (best workaround is not to use GTE commands in branch delays).

cop0cmd=10h - RFE opcode - Prepare Return from Exception

The RFE opcode moves some bits in cop0r12 (SR): bit2-3 are copied to bit0-1, and bit4-5 are copied to bit2-3, all other bits (including bit4-5) are left unchanged.
The RFE opcode does NOT automatically jump to EPC. Instead, the exception handler must copy EPC into a register (usually R26 aka K0), and then jump to that address. Because of branch delays, that would look like so:

```
  mov  k0,epc  ;get return address
  push k0      ;save epc in memory (if you expect nested exceptions)
  ...          ;whatever (ie. process CAUSE)
  pop  k0      ;restore from memory (if you expect nested exceptions)
  jmp  k0      ;jump to K0 (after executing the next opcode)
  +rfe         ;move SR bit4/5 --> bit2/3 --> bit0/1
```

If you expect exceptions to be nested deeply, also push/pop SR. Note that there's no way to leave all registers intact (ie. above code destroys K0).

cop0r8 - BadVaddr - Bad Virtual Address (R)

Contains the address whose reference caused an exception. Set on any MMU type of exceptions, on references outside of kuseg (in User mode) and on any misaligned reference. BadVaddr is updated ONLY by Address errors (Excode 04h and 05h), all other exceptions (including bus errors) leave BadVaddr unchanged.

Exception Vectors (depending on BEV bit in SR register)

```
  Exception     BEV=0         BEV=1
  Reset         BFC00000h     BFC00000h
  UTLB Miss     80000000h     BFC00100h (Virtual memory, none such in PSX)
  COP0 Break    80000040h     BFC00140h
  General       80000080h     BFC00180h
```

Note: Changing vectors at 800000xxh (kseg0) seems to be automatically reflected to the instruction cache without needing to flush cache (at least it worked SOMETIMES in my test proggy... but NOT always? ...anyways, it'd be highly recommended to flush cache when changing any opcodes), whilst changing mirrors at 000000xxh (kuseg) seems to require to flush cache.
The PSX uses only the BEV=0 vectors (aside from the reset vector, the PSX BIOS ROM doesn't contain any of the BEV=1 vectors).

Exception Priority

```
  Reset At any time (highest)            ;-reset
  AdEL Memory (Load instruction)         ;\
  AdES Memory (Store instruction)        ; memory (data load/store)
  DBE  Memory (Load or store)            ;/
  MOD  ALU (Data TLB)                    ;\
  TLBL ALU (DTLB Miss)                   ; none such
  TLBS ALU (DTLB Miss)                   ;/
  Ovf  ALU                               ;-overflow
  Int  ALU                               ;-interrupt
  Sys  RD (Instruction Decode)           ;\
  Bp   RD (Instruction Decode)           ;
  RI   RD (Instruction Decode)           ;
  CpU  RD (Instruction Decode)           ;/
  TLBL I-Fetch (ITLB Miss)               ;-none such
  AdEL IVA (Instruction Virtual Address) ;\memory (opcode fetch)
  IBE  RD (end of I-Fetch, lowest)       ;/
```

COP0 - Misc

cop0r15 - PRID - Processor ID (R)

```
  0-7   Revision
  8-15  Implementation
  16-31 Not used
```

For a Playstation with CXD8606CQ CPU, the PRID value is 00000002h.
Unknown if/which other Playstation CPU versions have other values...?

cop0r6 - JUMPDEST - Randomly memorized jump address (R)

This is a rather strange totally useless register. After certain exceptions, the CPU does memorize a jump destination address in this register. Once when it has memorized an address, the register becomes locked, and the memorized value won't change until it becomes unlocked by a new exception. Exceptions that do unlock the register are Reset and Interrupts (cause.bit10). Exceptions that do NOT unlock the register are syscall/break opcodes, and software generated interrupts (eg. cause.bit8).
In the unlocked state, the CPU does more or less randomly memorize one of the next some jump destinations - eg. the destination from the second jump after reset, or from a jump that occured immediately before executing the IRQ handler, or from a jump inside of the IRQ handler, or even from a later jump that occurs shortly after returning from the IRQ handler.
The register seems to be read-only (although the Kernel initialization code writes 0 to it for whatever reason).

cop0r0..r2, cop0r4, cop0r10, cop0r32..r63 - N/A
Registers 0,1,2,4,10 control virtual memory on some MIPS processors (but there's none such in the PSX), and Registers 32..63 (aka "control registers") aren't used in any MIPS processors. Trying to read any of these registers causes a Reserved Instruction Exception (excode=0Ah).

cop0cmd=01h,02h,06h,08h - TLBR,TLBWI,TLBWR,TLBP
The PSX supports only one cop0cmd (cop0cmd=10h aka RFE). Trying to execute the TLBxx opcodes causes a Reserved Instruction Exception (excode=0Ah).

jf/jt cop0flg,dest - conditional cop0 jumps
mov [mem],cop0reg / mov cop0reg,[mem] - coprocessor cop0 load/store
Not supported by the CPU. Trying to execute these opcodes causes a Coprocessor Unusable Exception (excode=0Bh, ie. unlike above, not 0Ah).

cop0r16-r31 - Garbage
Trying to read these registers returns garbage (but does not trigger an exception). When reading one of the garbage registers shortly after reading a valid cop0 register, the garbage value is usually the same as that of the valid register. When doing the read later on, the return value is usually 00000020h, or when reading much later it returns 00000040h, or even 00000100h. No idea what is causing that effect...?
Note: The garbage registers can be accessed (without causing an exception) even in "User mode with cop0 disabled" (SR.Bit1=1 and SR.Bit28=0); accessing any other existing cop0 registers (or executing the rfe opcode) in that state is causing Coprocessor Unusable Exceptions (excode=0Bh).

COP0 - Debug Registers

The COP0 debug registers seem to be PSX specific, normal R30xx CPUs like IDT's R3041 and R3051 don't have anything similar.

cop0r7 - DCIC - Breakpoint control (R/W)
```
0      Automatically set by hardware upon Any break            (R/W)
1      Automatically set by hardware upon BPC Code break       (R/W)
2      Automatically set by hardware upon BDA Data break       (R/W)
3      Automatically set by hardware upon BDA Data-Read break  (R/W)
4      Automatically set by hardware upon BDA Data-Write break (R/W)
5      Automatically set by hardware upon any-jump break       (R/W)
6-11   Not used (always zero)
12-13  Jump Redirection (0=Disable, 1..3=Enable) (see note)    (R/W)
14-15  Unknown? (R/W)
16-22  Not used (always zero)
23     Super-Master Enable 1 for bit24-29
24     Execution breakpoint     (0=Disabled, 1=Enabled) (see BPC, BPCM)
25     Data access breakpoint   (0=Disabled, 1=Enabled) (see BDA, BDAM)
26     Break on Data-Read       (0=No, 1=Break/when Bit25=1)
27     Break on Data-Write      (0=No, 1=Break/when Bit25=1)
28     Break on any-jump        (0=No, 1=Break on branch/jump/call/etc.)
29     Master Enable for bit28
30     Master Enable for bit24-27
31     Super-Master Enable 2 for bit24-29
```
When a breakpoint address match occurs the PSX jumps to 80000040h (ie. unlike normal exceptions, not to 80000080h). The Excode value in the CAUSE register is set to 09h (same as BREAK opcode), and EPC contains the return address, as usually. One of the first things to be done in the exception handler is to disable breakpoints (eg. if the any-jump break is enabled, then it must be disabled BEFORE jumping from 40h to the actual exception handler).

cop0r7.bit12-13 - Jump Redirection Note
If one or both of these bits are nonzero, then the PSX seems to check for the following opcode sequence,
```
mov rx,[mem]   ;load rx from memory
...            ;one or more opcodes that do not change rx
jmp/call rx    ;jump or call to rx
```
if it does sense that sequence, then it sets PC=[00000000h], but does not store any useful information in any cop0 registers, namely it does not store the return address in EPC, so it's impossible to determine which opcode has caused the exception. For the jump target address, there are 31 registers, so one could only guess which of them contains the target value; for "POP PC" code it'd be usually R31, but for "JMP [vector]" code it may be any register. So far the feature seems to be more or less unusable...?

cop0r5 - BDA - Breakpoint on Data Access Address (R/W)
cop0r9 - BDAM - Breakpoint on Data Access Mask (R/W)
Break condition is "((addr XOR BDA) AND BDAM)=0".

cop0r3 - BPC - Breakpoint on Execute Address (R/W)
cop0r11 - BPCM - Breakpoint on Execute Mask (R/W)
Break condition is "((PC XOR BPC) AND BPCM)=0".

Note (BREAK Opcode)
Additionally, the BREAK opcode can be used to create further breakpoints by patching the executable code. The BREAK opcode uses the same Excode value (09h) in CAUSE register. However, the BREAK opcode jumps to the normal exception handler at 80000080h (not 80000040h).

Note (LibCrypt)
The debug registers are mis-used by "Legacy of Kain: Soul Reaver" (and maybe also other games) for storing libcrypt copy-protection related values (ie. just as a "hidden" location for storing data, not for actual debugging purposes).
CDROM Protection - LibCrypt

Note (Cheat Devices/Expansion ROMs)
The Expansion ROM header supports only Pre-Boot and Post-Boot vectors, but no Mid-Boot vector. Cheat Devices are often using COP0 breaks for Mid-Boot Hooks, either with BPC=BFC06xxxh (break address in ROM, used in older cheat firmwares), or with BPC=80030000h (break address in RAM aka relocated GUI entrypoint, used in later cheat firmwares). Moreover, aside from the Mid-Boot Hook, the Xplorer cheat device is also supporting a special cheat code that uses the COP0 break feature.

Note (Datasheet)

Note: COP0 debug registers are described in LSI's "L64360" datasheet, chapter 14. And in their LR33900/LR33910 datasheet, chapter 4.

Kernel (BIOS)

BIOS Overview
BIOS Memory Map
BIOS Function Summary
BIOS File Functions
BIOS File Execute and Flush Cache
BIOS CDROM Functions
BIOS Memory Card Functions
BIOS Interrupt/Exception Handling
BIOS Event Functions
BIOS Event Summary
BIOS Thread Functions
BIOS Timer Functions
BIOS Joypad Functions
BIOS GPU Functions
BIOS Memory Allocation
BIOS Memory Fill/Copy/Compare (SLOW)
BIOS String Functions
BIOS Number/String/Character Conversion
BIOS Misc Functions
BIOS Internal Boot Functions
BIOS More Internal Functions
BIOS PC File Server
BIOS TTY Console (std_io)
BIOS Character Sets
BIOS Control Blocks
BIOS Versions
BIOS Patches

BIOS Overview

BIOS CDROM Boot
The main purpose of the BIOS is to boot games from CDROM, unfortunately, before doing that, it displays the Sony intro. It's also doing some copy protection and region checks, and refuses to boot unlicensed games, or illegal copies, or games for other regions.

BIOS Bootmenu
The bootmenu shows up when starting the Playstation without CDROM inserted. The menu allows to play Audio CDs, and to erase or copy game positions on Memory Cards.

BIOS Functions
The BIOS contains a number of more or less useful, and probably more or less inefficient functions that can be used by software.
No idea if it's easy to take full control of the CPU, ie. to do all hardware access and interrupt handling by software, without using the BIOS at all?
Eventually the BIOS functions for accessing the CDROM drive are important, not sure how complicated/compatible it'd be to access the CDROM drive directly via I/O ports... among others, there might be different drives used in different versions of the Playstation, which aren't fully compatible with each other?

BIOS Memory
The BIOS occupies 512Kbyte ROM with 8bit address bus (so the BIOS ROM is rather slow, for faster execution, portions of it are relocated to the first 64K of RAM). For some very strange reason, the original PSX BIOS executes all ROM functions in uncached ROM, which is incredible slow (nocash BIOS uses cached ROM, which does work without problems).
The first 64Kbyte of the 2Mbyte Main RAM are reserved for the BIOS (containing exception handlers, jump tables, other data, and relocated code). That reserved region does unfortunately include the "valuable" first 32Kbytes (valuable because that memory could be accessed directly via [R0+immediate], without needing to use R1..R31 as base register).

BIOS Memory Map

BIOS ROM Map (512Kbytes)

```
BFC00000h Kernel Part 1  (code/data executed in uncached ROM)
BFC10000h Kernel Part 2  (code/data relocated to cached RAM)
BFC18000h Intro/Bootmenu (code/data decompressed and relocated to RAM)
BFC64000h Character Sets
```

BIOS ROM Header/Footer

```
BFC00100h Kernel BCD date  (YYYYMMDDh)
BFC00104h Console Type     (see Port 1F802030h, Secondary IRQ10 Controller)
BFC00108h Kernel Maker/Version Strings (separated by one or more 00h bytes)
BFC7FF32h GUI Version/Copyright Strings (if any) (separated by one 00h byte)
```

BIOS RAM Map (1st 64Kbytes of RAM) (fixed addresses mainly in 1st 500h bytes)

```
00000000h 10h    Garbage Area (see notes below)
00000010h 30h    Unused/reserved
00000040h 20h    COP0 debug-break vector (not used by Kernel) (in KSEG0)
00000060h 4      RAM Size (in megabytes) (2 or 8)
00000064h 4      Unknown (set to 00000000h)
00000068h 4      Unknown (set to 000000FFh)
0000006Ch 14h    Unused/reserved
00000080h 10h    Exception vector (actually in KSEG0, ie. at 80000080h)
00000090h 10h    Unused/reserved
000000A0h 10h    A(nnh) Function Vector
000000B0h 10h    B(nnh) Function Vector
000000C0h 10h    C(nnh) Function Vector
000000D0h 30h    Unused/reserved
00000100h 58h    Table of Tables (BIOS Control Blocks) (see below)
```

```
00000180h 80h    Command line argument from SYSTEM.CNF; BOOT = fname argument
00000200h 300h   A(nnh) Jump Table
00000500h ...    Kernel Code/Data (relocated from ROM)
0000Cxxxh ...    Unused/reserved
0000DF80h 80h    Used for BIOS Patches (ie. used by games, not used by BIOS)
0000DFFCh 4      Response value from Intro/Bootmenu
0000E000h 2000h  Kernel Memory; ExCBs, EvCBs, and TCBs allocated via B(00h)
```

User Memory (not used by Kernel)

```
00010000h ...    Begin of User RAM (Exefile, Data, Heap, Stack, etc.)
001FFF00h ...    Default Stacktop (usually in KSEG0)
1F800000h 400h   Scratchpad (Data-Cache mis-used as Fast RAM)
```

Table of Tables (see BIOS Control Blocks for details)

Each table entry consists of two 32bit values; containing the base address, and total size (in bytes) of the corresponding control blocks.

```
00000100h  ExCB Exception Chain Entrypoints (addr=var, size=4*08h)
00000108h  PCB  Process Control Block       (addr=var, size=1*04h)
00000110h  TCB  Thread Control Blocks       (addr=var, size=N*C0h)
00000118h  -    Unused/reserved
00000120h  EvCB Event Control Blocks        (addr=var, size=N*1Ch)
00000128h  -    Unused/reserved
00000130h  -    Unused/reserved
00000138h  -    Unused/reserved
00000140h  FCB  File Control Blocks         (addr=fixed, size=10h*2Ch)
00000148h  -    Unused/reserved
00000150h  DCB  Device Control Blocks       (addr=fixed, size=0Ah*50h)
```

File handles (fd=00h..0Fh) can be simply converted as fcb=[140h]+fd*2Ch.
Event handles (event=F10000xxh) as evcb=[120h]+(event AND FFFFh)*1Ch.

Garbage Area at Address 00000000h

The first some bytes of memory address 00000000h aren't actually used by the Kernel, except for storing some garbage at that locations. However, this garbage is actually important for bugged games like R-Types and Fade to Black (ie. games that do read from address 00000000h due to using uninitialized pointers).
Initially, the garbage area is containing a copy of the 16-byte exception handler at address 80h, but the first 4-bytes are typically smashed (set to 00000003h from some useless dummy writes in some useless CDROM delays). Ie. the 16-bytes should have these values:

```
[00000000h]=3C1A0000h  ;<-- but overwritten by 00000003h after soon
[00000004h]=275A0C80h  ;<-- or 275A0C50h (in older BIOS)
[00000008h]=03400008h
[0000000Ch]=00000000h
```

For R-Types, the halfword at [0] must non-zero (else the game will do a DMA to address 0, and thereby destroy kernel memory). Fade to Black does several garbage reads from [0..9], a wrong byte value at [5] can cause the game to crash with an invalid memory access exception upon memory card access.

BIOS Function Summary

Parameters, Registers, Stack

Argument(s) are passed in R4,R5,R6,R7,[SP+10h],[SP+14h],etc.
Caution: When calling a sub-function with N parameters, the caller MUST always allocate N words on the stack, and, although the first four parameters are passed in registers rather than on stack, the sub-function is allowed to use/destroy these words at [SP+0..N*4-1].
BIOS Functions (and custom callback functions) are allowed to destroy registers R1-R15, R24-R25, R31 (RA), and HI/LO. Registers R16-R23, R29 (SP), and R30 (FP) must be left unchanged (if the function uses that registers, then it must push/pop them). R26 (K0) is reserved for exception handler and should be usually not used by other functions. R27 (K1) and R28 (GP) are left more or less unused by the BIOS, so one can more or less freely use them for whatever purpose.
The return value (if any) is stored in R2 register.

A-Functions (Call 00A0h with function number in R9 Register)

```
A(00h) or B(32h) FileOpen(filename,accessmode)
A(01h) or B(33h) FileSeek(fd,offset,seektype)
A(02h) or B(34h) FileRead(fd,dst,length)
A(03h) or B(35h) FileWrite(fd,src,length)
A(04h) or B(36h) FileClose(fd)
A(05h) or B(37h) FileIoctl(fd,cmd,arg)
A(06h) or B(38h) exit(exitcode)
A(07h) or B(39h) FileGetDeviceFlag(fd)
A(08h) or B(3Ah) FileGetc(fd)
A(09h) or B(3Bh) FilePutc(char,fd)
A(0Ah) todigit(char)
A(0Bh) atof(src)     ;Does NOT work - uses (ABSENT) cop1 !!!
A(0Ch) strtoul(src,src_end,base)
A(0Dh) strtol(src,src_end,base)
A(0Eh) abs(val)
A(0Fh) labs(val)
A(10h) atoi(src)
A(11h) atol(src)
A(12h) atob(src,num_dst)
A(13h) SaveState(buf)
A(14h) RestoreState(buf,param)
A(15h) strcat(dst,src)
A(16h) strncat(dst,src,maxlen)
A(17h) strcmp(str1,str2)
A(18h) strncmp(str1,str2,maxlen)
A(19h) strcpy(dst,src)
A(1Ah) strncpy(dst,src,maxlen)
A(1Bh) strlen(src)
A(1Ch) index(src,char)
A(1Dh) rindex(src,char)
A(1Eh) strchr(src,char)  ;exactly the same as "index"
A(1Fh) strrchr(src,char) ;exactly the same as "rindex"
A(20h) strpbrk(src,list)
A(21h) strspn(src,list)
A(22h) strcspn(src,list)
A(23h) strtok(src,list)  ;use strtok(0,list) in further calls
A(24h) strstr(str,substr) - buggy
A(25h) toupper(char)
A(26h) tolower(char)
A(27h) bcopy(src,dst,len)
A(28h) bzero(dst,len)
A(29h) bcmp(ptr1,ptr2,len)      ;Bugged
A(2Ah) memcpy(dst,src,len)
```

```
A(2Ch) memmove(dst,src,len)     ;Bugged
A(2Dh) memcmp(src1,src2,len)    ;Bugged
A(2Eh) memchr(src,scanbyte,len)
A(2Fh) rand()
A(30h) srand(seed)
A(31h) qsort(base,nel,width,callback)
A(32h) strtod(src,src_end) ;Does NOT work - uses (ABSENT) cop1 !!!
A(33h) malloc(size)
A(34h) free(buf)
A(35h) lsearch(key,base,nel,width,callback)
A(36h) bsearch(key,base,nel,width,callback)
A(37h) calloc(sizx,sizy)            ;SLOW!
A(38h) realloc(old_buf,new_siz)     ;SLOW!
A(39h) InitHeap(addr,size)
A(3Ah) SystemErrorExit(exitcode)
A(3Bh) or B(3Ch) std_in_getchar()
A(3Ch) or B(3Dh) std_out_putchar(char)
A(3Dh) or B(3Eh) std_in_gets(dst)
A(3Eh) or B(3Fh) std_out_puts(src)
A(3Fh) printf(txt,param1,param2,etc.)
A(40h) SystemErrorUnresolvedException()
A(41h) LoadExeHeader(filename,headerbuf)
A(42h) LoadExeFile(filename,headerbuf)
A(43h) DoExecute(headerbuf,param1,param2)
A(44h) FlushCache()
A(45h) init_a0_b0_c0_vectors
A(46h) GPU_dw(Xdst,Ydst,Xsiz,Ysiz,src)
A(47h) gpu_send_dma(Xdst,Ydst,Xsiz,Ysiz,src)
A(48h) SendGP1Command(gp1cmd)
A(49h) GPU_cw(gp0cmd)   ;send GP0 command word
A(4Ah) GPU_cwp(src,num) ;send GP0 command word and parameter words
A(4Bh) send_gpu_linked_list(src)
A(4Ch) gpu_abort_dma()
A(4Dh) GetGPUStatus()
A(4Eh) gpu_sync()
A(4Fh) SystemError
A(50h) SystemError
A(51h) LoadAndExecute(filename,stackbase,stackoffset)
A(52h) SystemError ----OR---- "GetSysSp()" ?
A(53h) SystemError          ;PS2: set_ioabort_handler(src)
A(54h) or A(71h) CdInit()
A(55h) or A(70h) _bu_init()
A(56h) or A(72h) CdRemove()  ;does NOT work due to SysDeqIntRP bug
A(57h) return 0
A(58h) return 0
A(59h) return 0
A(5Ah) return 0
A(5Bh) dev_tty_init()                                   ;PS2: SystemError
A(5Ch) dev_tty_open(fcb,and unused:"path\name",accessmode) ;PS2: SystemError
A(5Dh) dev_tty_in_out(fcb,cmd)                          ;PS2: SystemError
A(5Eh) dev_tty_ioctl(fcb,cmd,arg)                       ;PS2: SystemError
A(5Fh) dev_cd_open(fcb,"path\name",accessmode)
A(60h) dev_cd_read(fcb,dst,len)
A(61h) dev_cd_close(fcb)
A(62h) dev_cd_firstfile(fcb,"path\name",direntry)
A(63h) dev_cd_nextfile(fcb,direntry)
A(64h) dev_cd_chdir(fcb,"path")
A(65h) dev_card_open(fcb,"path\name",accessmode)
A(66h) dev_card_read(fcb,dst,len)
A(67h) dev_card_write(fcb,src,len)
A(68h) dev_card_close(fcb)
A(69h) dev_card_firstfile(fcb,"path\name",direntry)
A(6Ah) dev_card_nextfile(fcb,direntry)
A(6Bh) dev_card_erase(fcb,"path\name")
A(6Ch) dev_card_undelete(fcb,"path\name")
A(6Dh) dev_card_format(fcb)
A(6Eh) dev_card_rename(fcb1,"path\name1",fcb2,"path\name2")
A(6Fh) ?   ;card ;[r4+18h]=00000000h  ;card_clear_error(fcb) or so
A(70h) or A(55h) _bu_init()
A(71h) or A(54h) CdInit()
A(72h) or A(56h) CdRemove()   ;does NOT work due to SysDeqIntRP bug
A(73h) return 0
A(74h) return 0
A(75h) return 0
A(76h) return 0
A(77h) return 0
A(78h) CdAsyncSeekL(src)
A(79h) return 0               ;DTL-H: Unknown?
A(7Ah) return 0               ;DTL-H: Unknown?
A(7Bh) return 0               ;DTL-H: Unknown?
A(7Ch) CdAsyncGetStatus(dst)
A(7Dh) return 0               ;DTL-H: Unknown?
A(7Eh) CdAsyncReadSector(count,dst,mode)
A(7Fh) return 0               ;DTL-H: Unknown?
A(80h) return 0               ;DTL-H: Unknown?
A(81h) CdAsyncSetMode(mode)
A(82h) return 0               ;DTL-H: Unknown?
A(83h) return 0               ;DTL-H: Unknown?
A(84h) return 0               ;DTL-H: Unknown?
A(85h) return 0               ;DTL-H: Unknown?, or reportedly, CdStop (?)
A(86h) return 0               ;DTL-H: Unknown?
A(87h) return 0               ;DTL-H: Unknown?
A(88h) return 0               ;DTL-H: Unknown?
A(89h) return 0               ;DTL-H: Unknown?
A(8Ah) return 0               ;DTL-H: Unknown?
A(8Bh) return 0               ;DTL-H: Unknown?
A(8Ch) return 0               ;DTL-H: Unknown?
A(8Dh) return 0               ;DTL-H: Unknown?
A(8Eh) return 0               ;DTL-H: Unknown?
A(8Fh) return 0               ;DTL-H: Unknown?
A(90h) CdromIoIrqFunc1()
A(91h) CdromDmaIrqFunc1()
A(92h) CdromIoIrqFunc2()
```

```
A(94h) CdromGetInt5errCode(dst1,dst2)
A(95h) CdInitSubFunc()
A(96h) AddCDROMDevice()
A(97h) AddMemCardDevice()     ;DTL-H: SystemError
A(98h) AddDuartTtyDevice()    ;DTL-H: AddAdconsTtyDevice ;PS2: SystemError
A(99h) AddDummyTtyDevice()
A(9Ah) SystemError            ;DTL-H: AddMessageWindowDevice
A(9Bh) SystemError            ;DTL-H: AddCdromSimDevice
A(9Ch) SetConf(num_EvCB,num_TCB,stacktop)
A(9Dh) GetConf(num_EvCB_dst,num_TCB_dst,stacktop_dst)
A(9Eh) SetCdromIrqAutoAbort(type,flag)
A(9Fh) SetMemSize(megabytes)
```

Below functions A(A0h..B4h) not supported on pre-retail DTL-H2000 devboard:

```
A(A0h) WarmBoot()
A(A1h) SystemErrorBootOrDiskFailure(type,errorcode)
A(A2h) EnqueueCdIntr()  ;with prio=0 (fixed)
A(A3h) DequeueCdIntr()  ;does NOT work due to SysDeqIntRP bug
A(A4h) CdGetLbn(filename) ;get 1st sector number (or garbage when not found)
A(A5h) CdReadSector(count,sector,buffer)
A(A6h) CdGetStatus()
A(A7h) bu_callback_okay()
A(A8h) bu_callback_err_write()
A(A9h) bu_callback_err_busy()
A(AAh) bu_callback_err_eject()
A(ABh) _card_info(port)
A(ACh) _card_async_load_directory(port)
A(ADh) set_card_auto_format(flag)
A(AEh) bu_callback_err_prev_write()
A(AFh) card_write_test(port)  ;CEX-1000: jump_to_00000000h
A(B0h) return 0               ;CEX-1000: jump_to_00000000h
A(B1h) return 0               ;CEX-1000: jump_to_00000000h
A(B2h) ioabort_raw(param)     ;CEX-1000: jump_to_00000000h
A(B3h) return 0               ;CEX-1000: jump_to_00000000h
A(B4h) GetSystemInfo(index)   ;CEX-1000: jump_to_00000000h
A(B5h..BFh) N/A ;jump_to_00000000h
```

B-Functions (Call 00B0h with function number in R9 Register)

```
B(00h) alloc_kernel_memory(size)
B(01h) free_kernel_memory(buf)
B(02h) init_timer(t,reload,flags)
B(03h) get_timer(t)
B(04h) enable_timer_irq(t)
B(05h) disable_timer_irq(t)
B(06h) restart_timer(t)
B(07h) DeliverEvent(class, spec)
B(08h) OpenEvent(class,spec,mode,func)
B(09h) CloseEvent(event)
B(0Ah) WaitEvent(event)
B(0Bh) TestEvent(event)
B(0Ch) EnableEvent(event)
B(0Dh) DisableEvent(event)
B(0Eh) OpenThread(reg_PC,reg_SP_FP,reg_GP)
B(0Fh) CloseThread(handle)
B(10h) ChangeThread(handle)
B(11h) jump_to_00000000h
B(12h) InitPad(buf1,siz1,buf2,siz2)
B(13h) StartPad()
B(14h) StopPad()
B(15h) OutdatedPadInitAndStart(type,button_dest,unused,unused)
B(16h) OutdatedPadGetButtons()
B(17h) ReturnFromException()
B(18h) SetDefaultExitFromException()
B(19h) SetCustomExitFromException(addr)
B(1Ah) SystemError  ;PS2: return 0
B(1Bh) SystemError  ;PS2: return 0
B(1Ch) SystemError  ;PS2: return 0
B(1Dh) SystemError  ;PS2: return 0
B(1Eh) SystemError  ;PS2: return 0
B(1Fh) SystemError  ;PS2: return 0
B(20h) UnDeliverEvent(class,spec)
B(21h) SystemError  ;PS2: return 0
B(22h) SystemError  ;PS2: return 0
B(23h) SystemError  ;PS2: return 0
B(24h) jump_to_00000000h
B(25h) jump_to_00000000h
B(26h) jump_to_00000000h
B(27h) jump_to_00000000h
B(28h) jump_to_00000000h
B(29h) jump_to_00000000h
B(2Ah) SystemError  ;PS2: return 0
B(2Bh) SystemError  ;PS2: return 0
B(2Ch) jump_to_00000000h
B(2Dh) jump_to_00000000h
B(2Eh) jump_to_00000000h
B(2Fh) jump_to_00000000h
B(30h) jump_to_00000000h
B(31h) jump_to_00000000h
B(32h) or A(00h) FileOpen(filename,accessmode)
B(33h) or A(01h) FileSeek(fd,offset,seektype)
B(34h) or A(02h) FileRead(fd,dst,length)
B(35h) or A(03h) FileWrite(fd,src,length)
B(36h) or A(04h) FileClose(fd)
B(37h) or A(05h) FileIoctl(fd,cmd,arg)
B(38h) or A(06h) exit(exitcode)
B(39h) or A(07h) FileGetDeviceFlag(fd)
B(3Ah) or A(08h) FileGetc(fd)
B(3Bh) or A(09h) FilePutc(char,fd)
B(3Ch) or A(3Bh) std_in_getchar()
B(3Dh) or A(3Ch) std_out_putchar(char)
B(3Eh) or A(3Dh) std_in_gets(dst)
B(3Fh) or A(3Eh) std_out_puts(src)
B(40h) chdir(name)
```

```
B(42h) firstfile(filename,direntry)
B(43h) nextfile(direntry)
B(44h) FileRename(old_filename,new_filename)
B(45h) FileDelete(filename)
B(46h) FileUndelete(filename)
B(47h) AddDevice(device_info)  ;subfunction for AddXxxDevice functions
B(48h) RemoveDevice(device_name_lowercase)
B(49h) PrintInstalledDevices()
```

Below functions B(4Ah..5Dh) not supported on pre-retail DTL-H2000 devboard:

```
B(4Ah) InitCard(pad_enable)  ;uses/destroys k0/k1 !!!
B(4Bh) StartCard()
B(4Ch) StopCard()
B(4Dh) _card_info_subfunc(port)  ;subfunction for "_card_info"
B(4Eh) write_card_sector(port,sector,src)
B(4Fh) read_card_sector(port,sector,dst)
B(50h) allow_new_card()
B(51h) Krom2RawAdd(shiftjis_code)
B(52h) SystemError  ;PS2: return 0
B(53h) Krom2Offset(shiftjis_code)
B(54h) GetLastError()
B(55h) GetLastFileError(fd)
B(56h) GetC0Table
B(57h) GetB0Table
B(58h) get_bu_callback_port()
B(59h) testdevice(devicename)
B(5Ah) SystemError  ;PS2: return 0
B(5Bh) ChangeClearPad(int)
B(5Ch) get_card_status(slot)
B(5Dh) wait_card_status(slot)
B(5Eh..FFh) N/A ;jump_to_00000000h    ;CEX-1000: B(5Eh..F6h) only
B(100h....) N/A ;garbage              ;CEX-1000: B(F7h.....) and up
```

C-Functions (Call 00C0h with function number in R9 Register)

```
C(00h) EnqueueTimerAndVblankIrqs(priority) ;used with prio=1
C(01h) EnqueueSyscallHandler(priority)     ;used with prio=0
C(02h) SysEnqIntRP(priority,struc)  ;bugged, use with care
C(03h) SysDeqIntRP(priority,struc)  ;bugged, use with care
C(04h) get_free_EvCB_slot()
C(05h) get_free_TCB_slot()
C(06h) ExceptionHandler()
C(07h) InstallExceptionHandlers()  ;destroys/uses k0/k1
C(08h) SysInitMemory(addr,size)
C(09h) SysInitKernelVariables()
C(0Ah) ChangeClearRCnt(t,flag)
C(0Bh) SystemError  ;PS2: return 0
C(0Ch) InitDefInt(priority) ;used with prio=3
C(0Dh) SetIrqAutoAck(irq,flag)
C(0Eh) return 0               ;DTL-H2000: dev_sio_init
C(0Fh) return 0               ;DTL-H2000: dev_sio_open
C(10h) return 0               ;DTL-H2000: dev_sio_in_out
C(11h) return 0               ;DTL-H2000: dev_sio_ioctl
C(12h) InstallDevices(ttyflag)
C(13h) FlushStdInOutPut()
C(14h) return 0               ;DTL-H2000: SystemError
C(15h) tty_cdevinput(circ,char)
C(16h) tty_cdevscan()
C(17h) tty_circgetc(circ)    ;uses r5 as garbage txt for ioabort
C(18h) tty_circputc(char,circ)
C(19h) ioabort(txt1,txt2)
C(1Ah) set_card_find_mode(mode)  ;0=normal, 1=find deleted files
C(1Bh) KernelRedirect(ttyflag)   ;PS2: ttyflag=1 causes SystemError
C(1Ch) AdjustA0Table()
C(1Dh) get_card_find_mode()
C(1Eh..7Fh) N/A ;jump_to_00000000h
C(80h.....) N/A ;mirrors to B(00h.....)
```

SYS-Functions (Syscall opcode with function number in R4 aka A0 Register)

```
SYS(00h) NoFunction()
SYS(01h) EnterCriticalSection()
SYS(02h) ExitCriticalSection()
SYS(03h) ChangeThreadSubFunction(addr) ;syscall with r4=03h, r5=addr
SYS(04h..FFFFFFFFh) calls DeliverEvent(F0000010h,4000h)
```

The 20bit immediate in the "syscall imm" opcode is unused (should be zero).

BREAK-Functions (Break opcode with function number in opcode's immediate)

BRK opcodes may be used within devkits, however, the standard BIOS simply calls DeliverEvent(F0000010h,1000h) and SystemError_A_40h upon any BRK opcodes (as well as on any other unresolved exceptions).

```
BRK(1C00h) Division by zero (commonly checked/invoked by software)
BRK(1800h) Division overflow (-80000000h/-1, sometimes checked by software)
```

Below breaks are used in DTL-H2000 BIOS:

```
BRK(1h) Whatever lockup or so?
BRK(101h) PCInit()   Inits the fileserver.
BRK(102h) PCCreat(filename, fileattributes)
BRK(103h) PCOpen(filename, accessmode)
BRK(104h) PCClose(filehandle)
BRK(105h) PCRead(filehandle, length, memory_destination_address)
BRK(106h) PCWrite(filehandle, length, memory_source_address)
BRK(107h) PClSeek(filehandle, file_offset, seekmode)
BRK(3C400h) User has typed "break" command in debug console
```

The break functions have argument(s) in A1,A2,A3 (ie. unlike normal BIOS functions not in A0,A1,A2), and TWO return values (in R2, and R3). These functions require a commercial/homebrew devkit... consisting of a Data Cable (for accessing the PC's harddisk)... and an Expansion ROM (for handling the BREAK opcodes)... or so?

BIOS File Functions

A(00h) or B(32h) - FileOpen(filename, accessmode) - Opens a file for IO

```
out: V0  File handle (00h..0Fh), or -1 if error.
```

```
bit0     1=Read  ;\These bits aren't actually used by the BIOS, however, at
bit1     1=Write ;/least 1 should be set; won't work when all 32bits are zero
bit2     1=Exit without waiting for incoming data (when TTY buffer empty)
bit9     0=Open Existing File, 1=Create New file (memory card only)
bit15    1=Asynchronous mode (memory card only; don't wait for completion)
bit16-31 Number of memory card blocks for a new file on the memory card
```

The PSX can have a maximum of 16 files open at any time, of which, 2 handles are always reserved for std_io, so only 14 handles are available for actual files. Some functions (chdir, testdevice, FileDelete, FileUndelete, FormatDevice, firstfile, FileRename) are temporarily allocating 1 filehandle (FileRename tries to use 2 filehandles, but, it does accidently use only 1 handle, too). So, for example, FileDelete would fail if more than 13 file handles are opened by the game.

A(01h) or B(33h) - FileSeek(fd, offset, seektype) - Move the file pointer

```
seektype 0 = from start of file        (with positive offset)
         1 = from current file pointer (with positive/negative offset)
         2 = Bugs. Should be from end of file.
```

Moves the file pointer the number of bytes in A1, relative to the location specified by A2. Movement from the eof is incorrect. Also, movement beyond the end of the file is not checked.

A(02h) or B(34h) - FileRead(fd, dst, length) - Read data from an open file

```
out: V0  Number of bytes actually read, -1 if failed.
```

Reads the number of bytes from the specified open file. If length is not specified an error is returned. Read per $0080 bytes from memory card (bu:) and per $0800 from cdrom (cdrom:).

A(03h) or B(35h) - FileWrite(fd, src, length) - Write data to an open file

```
out: V0  Number of bytes written.
```

Writes the number of bytes to the specified open file. Write to the memory card per $0080 bytes. Writing to the cdrom returns 0.

A(04h) or B(36h) - FileClose(fd) - Close an open file

Returns r2=fd (or r2=-1 if failed).

A(08h) or B(3Ah) - FileGetc(fd) - read one byte from file

```
out: R2=character (sign-expanded) or FFFFFFFFh=error
```

Internally redirects to "FileRead(fd,tempbuf,1)". For some strange reason, the returned character is sign-expanded; so, a return value of FFFFFFFFh could mean either character FFh, or error.

A(09h) or B(3Bh) - FilePutc(char,fd) - write one byte to file

Observe that "fd" is the 2nd paramter (not the 1st paramter as usually).

```
out:  R2=Number of bytes actually written, -1 if failed
```

Internally redirects to "FileWrite(fd,tempbuf,1)".

B(40h) - chdir(name) - Change the current directory on target device

Changes the current directory on the specified device, which should be "cdrom:" (memory cards don't support directories). The PSX supports only a current directory, but NOT a current device (ie. after chdir, the directory name may be ommited from filenames, but the device name must be still included in all filenames).

```
in:  A0  Pointer to new directory path (eg. "cdrom:\path")
```

Returns 1=okay, or 0=failed.
The function doesn't verify if the directory exists. Caution: For cdrom, the function does always load the path table from the disk (even if it was already stored in RAM, so chdir is causing useless SLOW read/seek delays).

B(42h) - firstfile(filename,direntry) - Find first file to match the name

Returns r2=direntry (or r2=0 if no matching files).
Searches for the first file to match the specified filename; the filename may contain "?" and "*" wildcards. "*" means to ignore ALL following characters; accordingly one cannot specify any further characters after the "*" (eg. "DATA*" would work, but "*.DAT" won't work). "?" is meant to ignore a single character cell. Note: The "?" wildcards (but not "*") can be used also in all other file functions; causing the function to use the first matching name (eg. FileDelete "????" would erase the first matching file, not all matching files).
Start the name with the device you want to address. (ie. pcdrv:) Different drives can be accessed as normally by their drive names (a:, c:, huh?) if path is omitted after the device, the current directory will be used.
A direntry structure looks like this:

```
00h 14h Filename, terminated with 00h
14h 4   File attribute (always 0 for cdrom) (50h=norm or A0h=del for card)
18h 4   File size
1Ch 4   Pointer to next direntry? (not used?)
20h 4   First Sector Number
24h 4   Reserved (not used)
```

BUG: If "?" matches the ending 00h byte of a name, then any further characters in the search expression are ignored (eg. "FILE?.DAT" would match to "FILE2.DAT", but accidently also to "FILE").
BUG: For CDROM, the BIOS includes some code that is intended to realize disk changes during firstfile/nextfile operations, however, that code is so bugged that it does rather ensure that the BIOS does NOT realize new disks being inserted during firstfile/nextfile.
BUG: firstfile/nextfile is internally using a FCB. On the first call to firstfile, the BIOS is searching a free FCB, and does apply that as "search fcb", but it doesn't mark that FCB as allocated, so other file functions may accidently use the same FCB. Moreover, the BIOS does memorize that "search fcb", and, even when starting a new search via another call to firstfile, it keeps using that FCB for search (without checking if the FCB is still free). A possible workaround is not to have any files opened during firstfile/nextfile operations.

B(43h) - nextfile(direntry) - Searches for the next file to match the name

Returns r2=direntry (or r2=0 if no more matching files).
Uses the settings of a previous firstfile/nextfile command.

B(44h) - FileRename(old_filename, new_filename)

Returns 1=okay, or 0=failed.

B(45h) - FileDelete(filename) - Delete a file on target device

Returns 1=okay, or 0=failed.

B(46h) - FileUndelete(filename)

Returns 1=okay, or 0=failed.

B(41h) - FormatDevice(devicename)

Erases all files on the device (ie. for formatting memory cards).
Returns 1=okay, or 0=failed.

B(54h) - GetLastError()

FileDelete, FileUndelete, FormatDevice, FileRename). Use GetLastError() ONLY if an error has occured (the error code isn't reset to zero by functions that are passing okay). firstfile/nextfile do NOT affect GetLastError(). See below list of File Error Numbers for more info.

B(55h) - GetLastFileError(fd)
Basically same as B(54h), but allowing to specify a file handle for which error information is to be received; accordingly it doesn't work for functions that do use 'hidden' internal file handles (eg. FileDelete, or unsuccessful FileOpen). Returns FCB[18h], or FFFFFFFFh if the handle is invalid/unused.

A(05h) or B(37h) FileIoctl(fd,cmd,arg)
Used only for TTY.

A(07h) or B(39h) FileGetDeviceFlag(fd)
Returns bit1 of the file's DCB flags. That bit is set only for Duart/TTY, and is cleared for Dummy/TTY, Memory Card, and CDROM.

B(59h) - testdevice(devicename)
Whatever. Checks the devicename, and if it's accepted, calls a device specific function. For the existing devices (cdrom,bu,tty) that specific function simply returns without doing anything. Maybe other devices (like printers or modems) would do something more interesting.

File Error Numbers for B(54h) and B(55h)
```
00h okay (though many successful functions leave old error code unchanged)
02h file not found
06h bad device port number (tty2 and up)
09h invalid or unused file handle
10h general error (physical I/O error, unformatted, disk changed for old fcb)
11h file already exists error (create/undelete/rename)
12h tried to rename a file from one device to another device
13h unknown device name
16h sector alignment error, or fpos>=filesize, unknown seektype or ioctl cmd
18h not enough free file handles
1Ch not enough free memory card blocks
FFFFFFFFh invalid or unused file handle passed to B(55h) function
```

BIOS File Execute and Flush Cache

A(41h) - LoadExeHeader(filename, headerbuf)
Loads the 800h-byte exe file header to an internal sector buffer, and does then copy bytes [10h..4Bh] of that header to headerbuf[00h..3Bh].

A(42h) - LoadExeFile(filename, headerbuf)
Same as LoadExeHeader (see there for details), but additionally loads the body of the executable (using the size and destination address in the file header), and does call FlushCache. The exe can be then started via DoExecute (this isn't done automatically by LoadExeFile). Unlike "LoadAndExecute", the "LoadExeFile/DoExecute" combination allows to return the new exe file to return to the old exe file (instead of restarting the boot executable).
BUG: Uses the unstable FlushCache function (see there for details).

A(43h) - DoExecute(headerbuf, param1, param2)
Can be used to start a previously loaded executable. The function saves R16,R28,R30,SP,RA in the reserved region of headerbuf (rather than on stack), more or less slowly zerofills the memfill region specified in headerbuf, reads the stack base and offset values and sets SP and FP to base+offset (or leaves them unchanged if base=0), reads the GP value from headerbuf and sets GP to that value. Then calls the excecutables entrypoint, with param1 and param2 passed in r4,r5.
If the executable (should) return, then R16,R28,R30,SP,RA are restored from headerbuf, and the function returns with r2=1.

A(51h) - LoadAndExecute(filename, stackbase, stackoffset)
This is a rather bizarre function. In short, it does load and execute the specified file, and thereafter, it (tries to) reload and restart to boot executable.
Part1: Takes a copy of the filename, with some adjustments: Everything up to the first ":" or 00h byte is copied as is (ie. the device name, if it does exist, or otherwise the whole path\filename.ext;ver), the remaining characters are copied and converted to uppercase (ie. the path\filename.ext;ver part, or none if the device name didn't exist), finally, checks if a ";" exists (ie. the version suffix), if there's none, then it appends ";1" as default version. CAUTION: The BIOS allocates ONLY 28 bytes on stack for the copy of the filename, that region is followed by 4 unused bytes, so the maximum length would be 32 bytes (31 characters plus EOL) (eg. "device:\pathname\filename.ext;1",00h).
Part2: Enables IRQs via ExitCriticalSection, memorizes the stack base/offset values from the previously loaded executable (which should have been the boot executable, unless LoadAndExecute should have been used in nested fashion), does then use LoadExeFile to load the desired file, replaces the stack base/offset values in its headerbuf by the LoadAndExecute parameter values, and does then execute it via DoExecute(headerbuf,1,0).
Part3: If the exefile returns, or if it couldn't be loaded, then the boot file is (unsuccessfully) attempted to be reloaded: Enables IRQs via ExitCriticalSection, loads the boot file via LoadExeFile, replaces the stack base/offset values in its headerbuf by the values memorized in Part2 (which <should> be the boot executable's values from SYSTEM.CNF, unless the nesting stuff occurred), and does then execute the boot file via DoExecute(headerbuf,1,0).
Part4: If the boot file returns, or if it couldn't be loaded, then the function looks up in a "JMP $" endless loop (normally, returning from the boot exe causes SystemErrorBootOrDiskFailure("B",38Ch), however, after using LoadAndExecute, this functionality is replaced by the "JMP $" lockup.
BUG: Uses the unstable FlushCache function (see there for details).
BUG: Part3 accidently treats the first 4 characters of the exename as memory address (causing an invalid memory address exception on address 6F726463h, for "cdrom:filename.exe").

A(9Ch) - SetConf(num_EvCB, num_TCB, stacktop)
Changes the number of EvCBs and TCBs, and the stacktop. That values are usually initialized from the settings in the SYSTEM.CNF file, so using this function usually shouldn't ever be required.
The function deallocates all old ExCBs, EvCBs, TCBs (so all Exception handlers, Events, and Threads (except the current one) are lost, and all other memory that may have been allocated via alloc_kernel_memory(size) is deallocated, too. It does then allocate the new control blocks, and enqueue the default handlers. Despite of the changed stacktop, the current stack pointer is kept intact, and the function returns to the caller.

A(9Dh) - GetConf(num_EvCB_dst, num_TCB_dst, stacktop_dst)
Returns the number of EvCBs, TCBs, and the initial stacktop. There's no return value in the R2 register, instead, the three 32bit return values are stored at the specified "dst" addresses.

A(44h) - FlushCache()
Flushes the Code Cache, so opcodes are ensured to be loaded from RAM. This is required when loading program code via DMA (ie. from CDROM) (the cache controller apparently doesn't realize changes to RAM that are caused by DMA). The LoadExeFile and LoadAndExecute functions are automatically calling FlushCache (so FlushCache is required only when loading program code via "FileRead" or via "CdReadSector").
FlushCache may be also required when relocating or modifying program code by software (the cache controller doesn't seem to realize modifications to memory mirrors, eg. patching the exception handler at 80000080h seems to be work without FlushCache, but patching the same bytes at 00000080h doesn't).
Note: The PSX doesn't have a Data Cache (or actually, it has, but it's misused as Fast RAM, mapped to a fixed memory region, and which isn't accessable by DMA), so FlushCache isn't required for regions that contain data.

opcodes, then the k0 value gets destroyed by the exception handler, causing FlushCache to get trapped in an endless loop.
One workaround would be to disable all IRQs before calling FlushCache, however, the BIOS does internally call the function without IRQs disabled, that applies to:

```
load_file  A(42h)
load_exec  A(51h)
add_device B(47h) (and all "add_xxx_device" functions)
init_card  B(4Ah)
and by intro/boot code
```

for load_file/load_exec, IRQ2 (cdrom) and IRQ3 (dma) need to be enabled, so the "disable all IRQs" workaround cannot be used for that functions, however, one can/should disable as many IRQs as possible, ie. everything except IRQ2/IRQ3, and all DMA interrupts except DMA3 (cdrom).

Executable Memory Allocation
LoadExeFile and LoadAndExecute are simply loading the file to the address specified in the exe file header. There's absolutely no verification whether that memory is (or isn't) allocated via malloc, or if it is used by the boot executable, or by the kernel, or if it does contain RAM at all.
When using the "malloc" function combined with loading exe files, it may be recommended not to pass all memory to InitHeap (ie. to keep a memory region being reserved for loading executables).

Note
For more info about EXE files and their headers, see
CDROM File Formats

BIOS CDROM Functions

General File Functions
CDROMs are basically accessed via normal file functions, with device name "cdrom:" (which is an abbreviation for "cdrom0:", anyways, the port number is ignored).
BIOS File Functions
BIOS File Execute and Flush Cache
Before starting the boot executable, the BIOS automatically calls CdInit(), so the game doesn't need to do any initializations before using CDROM file functions.

Absent CD-Audio Support
The Kernel doesn't include any functions for playing Audio tracks. Also, there's no BIOS function for setting the XA-ADPCM file/channel filter values. So CD Audio can be used only by directly programming the CDROM I/O ports.

Asynchronous CDROM Access
The normal File functions are always using synchroneous access for CDROM (ie. the functions do wait until all data is transferred) (unlike as for memory cards, accessmode.bit15 cannot be used to activate asynchronous cdrom access).
However, one can read files in asynchrouneous fashion via CdGetLbn, CdAsyncSeekL, and CdAsyncReadSector. CDROM files are non-fragmented, so they can be read simply from incrementing sector numbers.

A(A4h) - CdGetLbn(filename)
Returns the first sector number used by the file, or -1 in case of error.
BUG: The function accidently returns -1 for the first file in the directory (the first file should be a dummy entry for the current or parent directory or so, so that bug isn't much of a problem), if the file is not found, then the function accidently returns garbage (rather than -1).

A(A5h) - CdReadSector(count,sector,buffer)
Reads <count> sectors, starting at <sector>, and writes data to <buffer>. The read is done in mode=80h (double speed, 800h-bytes per sector). The function waits until all sectors are transferred, and does then return the number of sectors (ie. count), or -1 in case of error.

A(A6h) - CdGetStatus()
Retrieves the cdrom status via CdAsyncGetStatus(dst) (see there for details; especially for cautions on door-open flag). The function waits until the event indicates completion, and does then return the status byte (or -1 in case of error).

A(78h) - CdAsyncSeekL(src)
Issues Setloc and SeekL commands. The parameter (src) is a pointer to a 3-byte sector number (MM,SS,FF) (in BCD format).
The function returns 0=failed, or 1=okay. Completion is indicated by events (class=F0000003h, and spec=20h, or 8000h).

A(7Ch) - CdAsyncGetStatus(dst)
Issues a GetStat command. The parameter (dst) is a pointer to a 1-byte location that receives the status response byte.
The function returns 0=failed, or 1=okay. Completion is indicated by events (class=F0000003h, and spec=20h, or 8000h).
Caution: The command acknowledges the door-open flag, but doesn't automatically reload the path table (which is required if a new disk is inserted); if the door-open flag was set, one should call a function that does forcefully load the path table (like chdir).

A(7Eh) - CdAsyncReadSector(count,dst,mode)
Issues SetMode and ReadN (when mode.bit8=0), or ReadS (when mode.bit8=1) commands. count is the number of sectors to be read, dst is the destination address in RAM, mode.bit0-7 are passed as parameter to the SetMode command, mode.bit8 is the ReadN/ReadS flag (as described above). The sector size (for DMA) depends on the mode value: 918h-bytes (bit4=1, bit5=X), 924h-bytes (bit4=0, bit5=1), or 800h-bytes (bit4,5=0).
Before CdAsyncReadSector, the sector number should be set via CdAsyncSeekL(src).
The function returns 0=failed, or 1=okay. Completion is indicated by events (class=F0000003h, and spec=20h, 80h, or 8000h).

A(81h) - CdAsyncSetMode(mode)
Similar to CdAsyncReadSector (see there for details), but issues only the SetMode command, without any following ReadN/ReadS command.

A(94h) - CdromGetInt5errCode(dst1,dst2)
Returns the first two response bytes from the most recent INT5 error: [dst1]=status, [dst2]=errorcode. The BIOS doesn't reset these values in case of successful completion, so the values are quite useless.

A(54h) or A(71h) - CdInit()
A(56h) or A(72h) - CdRemove() ;does NOT work due to SysDeqIntRP bug
A(90h) - CdromIoIrqFunc1()
A(91h) - CdromDmaIrqFunc1()
A(92h) - CdromIoIrqFunc2()
A(93h) - CdromDmaIrqFunc2()
A(95h) - CdInitSubFunc() ;subfunction for CdInit()
A(9Eh) - SetCdromIrqAutoAbort(type,flag)
A(A2h) - EnqueueCdIntr() ;with prio=0 (fixed)

Internally used CDROM functions for initialization and IRQ handling.

BIOS Memory Card Functions

General File Functions
Memory Cards aka Backup Units (bu) are basically accessed via normal file functions, with device names "bu00:" (Slot 1) and "bu10:" (Slot 2),
BIOS File Functions
Before using the file functions for memory cards, first call InitCard(pad_enable), then StartCard(), and then _bu_init().

File Header, Filesize, and Sector Alignment
The first 100h..200h bytes (2..4 sectors) of the file must contain the title and icon bitmap(s). For details, see:
Memory Card Data Format
The filesize must be a multiple of 2000h bytes (one block), the maximum size would be 1E000h bytes (when using all 15 blocks on the memory card). The filesize must be specified when creating the file (ie. accessmode bit9=1, and bit16-31=number of blocks). Once when the file is created, the BIOS does NOT allow to change the filesize (unless by deleting and re-creating the file).
When reading/writing files, the amount of data must be a multiple of 80h bytes (one sector), and the file position must be aligned to a 80h-byte boundary, too.
There's no restriction on fragmented files (ie. one may cross 2000h-byte block boundaries within the file).

Poor Memcard Performance
PSX memory card accesses are typically super-slow. That, not so much because the hardware would be slow, but rather because of improper inefficent code at the BIOS side. The original BIOS tries to synchronize memory card accesses with joypad accesses simply by accessing only one sector per frame (although it could access circa two sectors). To the worst, the BIOS accesses Slot 1 only on each second frame, and Slot 2 only each other frame (although in 99% of all cases only one slot is accessed at once, so the access time drops to 0.5 sectors per frame).
Moreover, the memory card id, directory, and broken sector list do occupy 26 sectors (although the whole information would fit into 4 or 5 sectors) (a workaround would be to read only the first some bytes, and to skip the additional unused bytes - though that'd also mean to skip the checksums which are unfortunately stored at the end of the sector).
And, anytime when opening a file (in synchronous mode), the BIOS does additionally read sector 0 (which is totally useless, and gets especially slow when opening a bunch of files; eg. when extracting the title/icon from all available files on the card).

Asynchronous Access
The BIOS supports synchronous and asynchronous memory card access. Synchronous means that the BIOS function doesn't return until the access has completed (which means, due to the poor performance, that the function spends about 75% of the time on inactivity) (except in nocash PSX bios, which has better performance), whilst asynchronous access means that the BIOS function returns immediately after invoking the access (which does then continue on interrupt level, and does return an event when finished).
The file "FileRead" and "FileWrite" functions act asynchronous when accessmode bit15 is set when opening the file. Additionally, the A(ACh) _card_async_load_directory(port) function can be used to tell the BIOS to load the directory entries and broken sector list to its internal RAM buffers (eg. during the games title screen, so the BIOS doesn't need to load that data once when the game enters its memory card menu). All other functions like FileDelete or FormatDevice always act synchronous. The FileOpen/findfirst/findnext functions do normally complete immediately without accessing the card at all (unless the directory wasn't yet read; in that case the directory is loading in synchronous fashion).
Unfortunately, the asynchronous response doesn't rely on a single callback event, but rather on a bunch of different events which must be all allocated and tested by the game (and of which, one event is delivered on completion) (which one depends on whether function completed okay, or if an error occurred).

Multitap Support (and Multitap Problems)
The BIOS does have some partial support for accessing more than two memory cards (via Multitap adaptors). Device/port names "bu01:", "bu02:", "bu03:" allow to access extra memory carts in slot1 (and "bu11:", "bu12:", "bu13:" in slot2). Namely, those names will send values 82h, 83h, 84h to the memory card slot (instead of the normal 81h value).
However, the BIOS directory_buffer and broken_sector_list do support only two memory cards (one in slot1 and one in slot2). So, trying to access more memory cards may cause great data corruption (though there might be a way to get the BIOS to reload those buffers before accessing a different memory card).
Aside from that problem, the BIOS functions are very-very-very slow even when accessing only two memory cards. Trying to use the BIOS to access up to eight memory cards would be very-extremly-very slow, which would be more annoying than useful.

B(4Ah) - InitCard(pad_enable) ;uses/destroys k0/k1 !!!
B(4Bh) - StartCard()
B(4Ch) - StopCard()
A(55h) or A(70h) - _bu_init()

```
  --- Below are some lower level memory card functions ---
```

A(ABh) - _card_info(port)
B(4Dh) - _card_info_subfunc(port) ;subfunction for "_card_info"
Can be used to check if the most recent call to write_card_sector has completed okay. Issues an incomplete dummy read command (similar to B(4Fh) - read_card_sector). The read command is aborted once when receiving the status byte from the memory card (the actual data transfer is skipped).

A(AFh) - card_write_test(port) ;not supported by old CEX-1000 version
Resets the card changed flag. For some strange reason, this flag isn't automatically reset after reading the flag, instead, the flag is reset upon sector writes. To do that, this function issues a dummy write to sector 3Fh.

B(50h) - allow_new_card()
Normally any memory card read/write functions fail if the BIOS senses the card change flag to be set. Calling this function tells the BIOS to ignore the card change flag on the next read/write operation (the function is internally used when loading the "MC" ID from sector 0, and when calling the card_write_test function to acknowledge the card change flag).

B(4Eh) - write_card_sector(port,sector,src)
B(4Fh) - read_card_sector(port,sector,dst)
Invokes asynchronous reading/writing of a single sector. The function returns 1=okay, or 0=failed (on invalid sector numbers). The actual I/O is done on IRQ level, completion of the I/O command transmission can be checked, among others, via get/wait_card_status(slot) functions (with slot=port/10h).
In case of the write function, completion of the <transmission> does NOT mean that the actual <writing> has completed, instead, write errors are indicated upon completion of the <next sector> read/write transmission (or, if there are no further sectors to be accessed; one can use _card_info to verify completion of the last written sector).
The sector number should be in range of 0..3FFh, for some strange reason, probably a BUG, the function also accepts sector 400h. The specified sector number is directly accessed (it is NOT parsed through the broken sector replacement list).

B(5Ch) - get_card_status(slot)
B(5Dh) - wait_card_status(slot)

Returns the status of the most recent I/O command, possible values are:

```
01h=ready
02h=busy/read
04h=busy/write
08h=busy/info
11h=failed/timeout (eg. when no cartridge inserted)
21h=failed/general error
```

get_card_status returns immediately, wait_card_status waits until a non-busy state occurs.

A(A7h) - bu_callback_okay()
A(A8h) - bu_callback_err_write()
A(A9h) - bu_callback_err_busy()
A(AAh) - bu_callback_err_eject()
A(AEh) - bu_callback_err_prev_write()
These five callback functions are internally used by the BIOS, notifying other BIOS functions about (un-)successful completion of memory card I/O commands.

B(58h) - get_bu_callback_port()
This is a subfunction for the five bu_callback_xxx functions (indicating whether the callback occured for a slot1 or slot2 access).

A(ACh) - _card_async_load_directory(port)
Invokes asynchronous reading of the memory card directory. The function isn't too useful because the BIOS tends to read the directory automatically in various places in synchronous mode, so there isn't too much chance to replace the automatic synchronous reading by asynchronous reading.

A(ADh) - set_card_auto_format(flag)
Can be used to enable/disable auto format (0=off, 1=on). The _bu_init function initializes auto format as disabled. If auto format is enabled, then the BIOS does automatically format memory cards if it has failed to read the "MC" ID bytes on sector 0. Although potentially useful, activating this feature is rather destructive (for example, read errors on sector 0 might occur accidently due to improperly inserted cards with dirty contacts, so it'd be better to prompt the user whether or not to format the card, rather than doing that automatically).

C(1Ah) - set_card_find_mode(mode)
C(1Dh) - get_card_find_mode()
Allows to get/set the card find mode (0=normal, 1=find deleted files), the mode setting affects only the firstfile/nextfile functions. All other file functions (FileOpen, FileRename, FileDelete, FileUndelete) are forcefully setting mode=0 (or mode=1 in case of FileUndelete).

BIOS Interrupt/Exception Handling

The Playstation's Kernel uses an uncredible inefficient and unstable exception handler; which may have been believed to be very powerful and flexible.

Inefficiency
For a maximum of slowness, it does always save and restore all CPU registers (including such that aren't used in the exception handler). It does then go through a list of installed interrupt handlers - and executes ALL of them. For example, a Timer0 IRQ is first passed to the Cdrom and Vblank handlers (which must reject it, no thanks), before it does eventually reach the Timer0 handler.

Unstable IRQ Handling
A fundamental mistake in the exception handler is that it doesn't memorize the incoming IRQ flags. So the various interrupt handlers must check Port 1F801070h one after each other. That means, if a high priority handler has rejected IRQ processing (because the desired IRQ flag wasn't set at that time), then a lower priority handler may process it (assuming that the IRQ flag got set in the meantime), and, in worst case it may even acknowledge it (so the high priority handler does never receive it).
To avoid such problems, there should be only ONE handler installed for each IRQ source. However, that isn't always possible, because the Kernel automatically installs some predefined handlers. Most noteworthy, the totally bugged DefaultInterruptHandlers is always installed (and cannot be removed), so it does randomly trigger Events. Fortunately, it does not acknowledge the IRQs (unless SetIrqAutoAck was used to enable that fatal behaviour).

B(18h) - SetDefaultExitFromException()
Applies the default "Exit" structure (which consists of a pointer to ReturnFromException, and the Kernel's exception stacktop (minus 4, for whatever reason), and zeroes for the R16..R23,R28,R30 registers. Returns the address of that structure.
See SetCustomExitFromException for details.

B(19h) - SetCustomExitFromException(addr)
addr points to a structure (with same format as for the SaveState function):

```
00h 4    r31/ra,pc ;usually ptr to ReturnFromException function
04h 4    r28/sp    ;usually exception stacktop, minus 4, for whatever reason
08h 4    r30/fp    ;usually 0
0Ch 4x8  r16..r23  ;usually 0
2Ch 4    r28/gp    ;usually 0
```

The hook function is executed only if the ExceptionHandler has been fully executed (after processing an IRQ, many interrupt handlers are calling ReturnFromException to abort further exception handling, and thus do skip the hook function). Once when the hook function has finished, it should execute ReturnFromException. The hook function is called with r2=1 (that is important if the hook address was recorded with SaveState, where it "returns" to the SaveState caller, with r2 as "return value").

Priority Chains
The Kernel's exception handler has four priority chains, each may contain one or more Interrupt or Exception handlers. The default handlers are:

```
Prio Chain Content
0    CdromDmaIrq, CdromIoIrq, SyscallException
1    CardSpecificIrq, VblankIrq, Timer2Irq, Timer1Irq, Timer0Irq
2    PadCardIrq
3    DefInt
```

The exception handler calls all handlers, starting with the first element in the priority 0 chain (ie. usually CdromDmaIrq). The separate handlers must check if they want to process the IRQ (eg. CdromDmaIrq would process only CDROM DMA IRQs, but not joypad IRQs or so). If it has processed and acknowledged the IRQ, then the handler may execute ReturnFromException, which causes the handlers of lower priority to be skipped (if there are still other unacknowledge IRQs pending, then the hardware will re-enter the exception handler as soon as the RFE opcode in ReturnFromException does re-enable interrupts).

C(02h) - SysEnqIntRP(priority,struc) ;bugged, use with care
Inserts a new element in the specified priority chain. The new element is inserted at the begin of the chain, so (within that priority chain) the new element has highest priority.

```
00h 4  pointer to next element    (0=none)  ;this pointer is inserted by BIOS
04h 4  pointer to SECOND function (0=none)  ;executed if func1 returns r2<>0
08h 4  pointer to FIRST  function (0=none)  ;executed first
0Ch 4  Not used (usually zero)
```

BUG: SysDeqIntRP can remove only the first element in the chain (see there for details), so adding new chain elements may cause OTHER functions to be unable to remove their chain elements. The BIOS seems to be occassionally adding/removing the "CardSpecificIrq" and "PadCardIrq" (with priority 1 and 2), so using that priorities may cause the BIOS to be unable to remove that IRQ handlers. Using priority 0 and 3 should work (as long as the software takes care to remove only the newest elements) (but there should be no conflicts with the BIOS which does never remove priority 0 and 3 elements) (leaving apart that DequeueCdIntr and CdRemove try to remove priority 0 elements, but that functions won't work anyways; due to the same bug).

C(03h) - SysDeqIntRP(priority,struc) ;bugged, use with care
Removes the specified element from the specified priority chain.
BUG: The function tries to search the whole chain (and to remove the element if it finds it). However, it does only check the first element properly, and, thereafter it reads a garbage value from an uninitialized stack location, and acts more or less unpredictable. So, it can remove only the first element in the chain, ie. it should be called only if you are SURE that there's no newer element in the chain, and only if you are SURE that the element IS in the chain.

SYS(01h) - EnterCriticalSection() ;syscall with r4=01h
Disables interrupts by clearing SR (cop0r12) Bit 2 and 10 (of which, Bit2 gets copied to Bit0 once when returning from the syscall exception). Returns 1 if both bits were set, returns 0 if one or both of the bits were already zero.

SYS(02h) - ExitCriticalSection() ;syscall with r4=02h
Enables interrupts by set SR (cop0r12) Bit 2 and 10 (of which, Bit2 gets copied to Bit0 once when returning from the syscall exception). There's no return value (all registers except SR and K0 are unchanged).

C(0Dh) - SetIrqAutoAck(irq,flag)
Specifies if the DefaultInterruptHandler shall automatically acknowledge IRQs. The "irq" paramter is the number of the interrupt, ie. 00h..0Ah = IRQ0..IRQ10. The "flag" value should be 0 to disable AutoAck, or non-zero to enable AutoAck. By default, AutoAck is disabled for all IRQs.
Mind that the DefaultInterruptHandler is totally bugged. Especially the AutoAck feature doesn't work very well: It may cause higher priority handlers to miss their IRQ, and it may even cause the DefaultInterruptHandler to miss its own IRQs.

C(06h) - ExceptionHandler()
The C(06h) vector points to the exception handler, ie. to the function that is invoked from the 4 opcodes at address 80000080h, that opcodes jump directly to the exception handler, so patching the C(06h) vector has no effect.
Reading the C(06h) entry can be used to let a custom 80000080h handler pass control back to the default handler (that, by a "direct" jump, not by the usual "MOV R9,06h / CALL 0C0h" method, which would destroy main programs R9 register).
Also, reading C(06h) may be useful for patching the exception handler (which contains a bunch of NOP opcodes, which seem to be intended to insert additional opcodes, such like debugger exception handling) (Note: some of that NOPs are reserved for Memory Card IRQ handling).
BUG: Early BIOS versions did try to examine a copy of cop0r13 in r2 register, but did forgot cop0r13 to r2 (so they examined garbage), this was fixed in newer BIOS versions, additionally, most commercial games still include patches for compatibility with the old BIOS.

B(17h) - ReturnFromException()
Restores the CPU registers (R1-R31,HI,LO,SR,PC) (except R26/K0) from the current TCB. This function is usually executed automatically at the end of the ExceptionHandler, however, functions in the exception chain may call ReturnFromException to to return immediately, without processing chain elements of lower priority.

C(00h) - EnqueueTimerAndVblankIrqs(priority) ;used with prio=1
C(01h) - EnqueueSyscallHandler(priority) ;used with prio=0
C(0Ch) - InitDefInt(priority) ;used with prio=3
Internally used to add some default IRQ and Exception handlers.

No Nested Exceptions
The Kernel doesn't support nested exceptions, that would require a decreasing exception stack, however, the kernel saves the incoming CPU registers in the current TCB, and an exception stack with fixed start address for internal push/pop during exception handling. So, nesting would overwrite these values. Do not enable IRQs, and don't trap other exceptions (like break or syscall opcodes, or memory or overlow errors) during exception handling.
Note: The execption stack has a fixed size of 1000h bytes (and is located somewhere in the first 64Kbytes of memory).

BIOS Event Functions

B(08h) - OpenEvent(class, spec, mode, func)
Adds an event structure to the event table.

```
    class,spec - triggers if BOTH values match
    mode - (1000h=execute function/stay busy, 2000h=no func/mark ready)
    func - Address of callback function (0=None) (used only when mode=1000h)
 out: R2 = Event descriptor (F1000000h and up), or FFFFFFFFh if failed
```

Opens an event, should be called within a critical section. The return value is used to identify the event to the other event functions. A list of event classes, specs and modes is at the end of this section. Initially, the event is disabled.
Note: The desired max number of events can be specified in the SYSTEM.CNF boot file (the default is "EVENT = 10" (which is a HEX number, ie. 16 decimal; of which 5 events are internally used by the BIOS for CDROM functions, so, of the 16 events, only 11 events are available to the game). A bigger amount of events will slowdown the DeliverEvent function (which always scans all EvCBs, even if all events are disabled).

B(09h) - CloseEvent(event) - releases event from the event table
Always returns 1 (even if the event handle is unused or invalid).

B(0Ch) - EnableEvent(event) - Turns on event handling for specified event
Always returns 1 (even if the event handle is unused or invalid).

B(0Dh) - DisableEvent(event) - Turns off event handling for specified event
Always returns 1 (even if the event handle is unused or invalid).

B(0Ah) WaitEvent(event)
Returns 0 if the event is disabled. Otherwise hangs in a loop until the event becomes ready, and returns 1 once when it is ready (and automatically switches the event back to busy status). Callback events (mode=1000h) do never set the ready flag (and thus WaitEvent would hang forever).
The main program simply hangs during the wait, so when using multiple threads, it may be more recommended to create an own waitloop that checks TestEvent, and to call ChangeThread when the event is busy.
BUG: The return value is unstable (sometimes accidently returns 0=disabled if the event status changes from not-ready to ready shortly after the function call).

B(0Bh) TestEvent(event)
Returns 0 if the event is busy or disabled. Otherwise, when it is ready, returns 1 (and automatically switches the event back to busy status). Callback events (mode=1000h) do never set the ready flag.

B(07h) DeliverEvent(class, spec)
This function is usually called by the kernel, it triggers all events that are enabled/busy, and that have the specified class and spec values. Depending on the mode, either the callback function is called (mode=1000h), or the event is marked as enabled/ready (mode=2000h).

B(20h) UnDeliverEvent(class, spec)
This function is usually called by the kernel, undelivers all events that are enabled/ready, and that have mode=2000h, and that have the specified class and spec values. Undeliver means that the events are marked as enabled/busy.

C(04h) get_free_EvCB_slot()
A subfunction for OpenEvent.

Event Classes
File Events:
```
0000000xh memory card (for file handle fd=x)
```
Hardware Events:
```
F0000001h IRQ0  VBLANK
F0000002h IRQ1  GPU
F0000003h IRQ2  CDROM Decoder
F0000004h IRQ3  DMA controller
F0000005h IRQ4  RTC0 (timer0)
F0000006h IRQ5/IRQ6 RTC1 (timer1 or timer2)
F0000007h N/A   Not used (this should be timer2)
F0000008h IRQ7  Controller (joypad/memcard)
F0000009h IRQ9  SPU
F000000Ah IRQ10 PIO ;uh, does the PIO have an IRQ signal? (IRQ10 is joypad)
F000000Bh IRQ8  SIO
F0000010h Exception ;CPU crashed (BRK,BadSyscall,Overflow,MemoryError, etc.)
F0000011h memory card (lower level BIOS functions)
F0000012h memory card (not used by BIOS; maybe used by Sony's devkit?)
F0000013h memory card (not used by BIOS; maybe used by Sony's devkit?)
```
Event Events:
```
F1xxxxxxh event (not used by BIOS; maybe used by Sony's devkit?)
```
Root Counter Events (Timers and Vblank):
```
F2000000h Root counter 0 (Dotclock)                   (hardware timer)
F2000001h Root counter 1 (horizontal retrace?)        (hardware timer)
F2000002h Root counter 2 (one-eighth of system clock) (hardware timer)
F2000003h Root counter 3 (vertical retrace?) (this is a software timer)
```
User Events:
```
F3xxxxxxh user (not used by BIOS; maybe used by games and/or Sony's devkit?)
```
BIOS Events (including such that have nothing to do with BIOS):
```
F4000001h memory card (higher level BIOS functions)
F4000002h libmath (not used by BIOS; maybe used by Sony's devkit?)
```
Thread Events:
```
FFxxxxxxh thread (not used by BIOS; maybe used by Sony's devkit?)
```

Event Specs
```
0001h counter becomes zero
0002h interrupted
0004h end of i/o
0008h file was closed
0010h command acknowledged
0020h command completed
0040h data ready
0080h data end
0100h time out
0200h unknown command
0400h end of read buffer
0800h end of write buffer
1000h general interrupt
2000h new device
4000h system call instruction ;SYS(04h..FFFFFFFFh)
8000h error happened
8001h previous write error happened
0301h domain error in libmath
0302h range error in libmath
```

Event modes
```
1000h Execute callback function, and stay busy (do NOT mark event as ready)
2000h Do NOT execute callback function, and mark event as ready
```

BIOS Event Summary

Below is a list of all events (class,spec values) that are delivered and/or undelivered by the BIOS in one way or another. The BIOS does internally open five events for cdrom (class=F0000003h with spec=10h,20h,40h,80h,8000h). The various other class/spec's are only delivered by the BIOS (but not received by the BIOS) (ie. a game may open/enable memory card events to receive notifications from the BIOS).

CDROM Events
```
F0000003h,10h  cdrom DMA finished (all sectors finished)
F0000003h,20h  cdrom ?
F0000003h,40h  cdrom dead feature (delivered only by unused functions)
F0000003h,80h  cdrom INT4 (reached end of disk)
F0000003h,100h         n/a ?  ;undelivered, but not opened, nor delivered
F0000003h,200h                ;undelivered, but not opened
F0000003h,8000h
```

Memory Card - Higher Level File/Device Events
```
0000000xh,4     card file handle (x=fd) done okay
F4000001h,4     card done okay (len=0)
F4000001h,100h  card err busy  ;A(A9h)
F4000001h,2000h card err eject ;A(AAh) or unformatted (bad "MC" id)
F4000001h,8000h card err write ;A(A8h) or A(AEh) or general error
```

Memory Card - Lower Level Hardware I/O Events
```
F0000011h,4      finished okay
F0000011h,100h   err busy
```

```
F0000011h,2000h  err
F0000011h,8000h  err
F0000011h,8001h  err (this one is NOT undelivered!)
```

Timer/Vblank Events

```
F2000000h,2   Timer0 (IRQ4)
F2000001h,2   Timer1 (IRQ5)
F2000002h,2   Timer2 (IRQ6)
F2000003h,2   Vblank (IRQ0) (unstable since IRQ0 is also used for joypad)
```

Default IRQ Handler Events (very unstable, don't use)

```
F0000001h,1000h ;IRQ0  (VBLANK)
F0000002h,1000h ;IRQ1  (GPU)
F0000003h,1000h ;IRQ2  (CDROM)
F0000004h,1000h ;IRQ3  (DMA)
F0000005h,1000h ;IRQ4  (TMR0)
F0000006h,1000h ;IRQ5  (TMR1)
F0000006h,1000h ;IRQ6  (TMR2) (accidently uses same event as TMR1)
F0000008h,1000h ;IRQ7  (joypad/memcard)
F0000009h,1000h ;IRQ9  (SPU)
F000000Ah,1000h ;IRQ10 (Joypad and PIO)
F000000Bh,1000h ;IRQ8  (SIO)
```

Unresolved Exception Events

```
F0000010h,1000h unknown exception ;neither IRQ nor SYSCALL
F0000010h,4000h unknown syscall   ;syscall(04h..FFFFFFFFh)
```

BIOS Thread Functions

B(0Eh) OpenThread(reg_PC,reg_SP_FP,reg_GP)
Searches a free TCB, marks it as used, and stores the inital program counter (PC), global pointer (GP aka R28), stack pointer (SP aka R29), and frame pointer (FP aka R30) (using the same value for SP and FP). All other registers are left uninitialized (eg. may contain values from an older closed thread, that includes the SR register, see note).
The return value is the new thread handle (in range FF000000h..FF000003h, assuming that 4 TCBs are allocated) or FFFFFFFFh if there's no free TCB. The function returns to the old current thread, use "ChangeThread" to switch to the new thread.
Note: The desired max number of TCBs can be specified in the SYSTEM.CNF boot file (the default is "TCB = 4", one initially used for the boot executable, plus 3 free threads).

BUG - Unitialized SR Register
OpenThread does NOT initialize the SR register (cop0r12) of the new thread. Upon powerup, the bootcode zerofills the TCB memory (so, the SR of new threads will be initially zero; ie. Kernel Mode, IRQ's disabled, and COP2 disabled). However, when closing/reopening threads, the SR register will have the value of the old closed thread (so it may get started with IRQs enabled, and, in worst case, if the old thread should have switched to User Mode, even without access to KSEG0, KSEG1 memory).
Or, ACTUALLY, the memory is NOT zerofilled on powerup... so SR is total random?

B(0Fh) CloseThread(handle)
Marks the TCB for the specified thread as unused. The function can be used for any threads, including for the current thread.
Closing the current thread doesn't terminate the current thread, so it may cause problems once when opening a new thread, however, it should be stable to execute the sequence "DisableInterrupts, CloseCurrentThread, ChangeOtherThread".
The return value is always 1 (even if the handle was already closed).

B(10h) ChangeThread(handle)
Pauses the current thread, and activates the selected new thread (or crashes if the specified handle was unused or invalid).
The return value is always 1 (stored in the R2 entry of the TCB of the old thread, so the return value will be received once when changing back to the old thread).
Note: The BIOS doesn't automatically switch from one thread to another. So, all other threads remain paused until the current thread uses ChangeThread to pass control to another thread.
Each thread is having it's own CPU registers (R1..R31,HI,LO,SR,PC), the registers are stored in the TCB of the old thread, and restored when switching back to that thread. Mind that other registers (I/O Ports or GTE registers aren't stored automatically, so, when needed, they need to be pushed/popped by software before/after ChangeThread).

C(05h) get_free_TCB_slot()
Subfunction for OpenThread, returns the number of the first free TCB (usually in range 0..3) or FFFFFFFFh if there's no free TCB.

SYS(03h) ChangeThreadSubFunction(addr) ;syscall with r4=03h, r5=addr
Subfunction for ChangeThread, R5 contains the address of the new TCB, just like all exceptions, the syscall exceptions saves the CPU registers in the current TCB, but does then apply the new TCB as current TCB, so, it does enter the new thread when returning from the exception.

BIOS Timer Functions

Timers (aka Root Counters)
The three hardware timers aren't internally used by any BIOS functions, so they can be freely used by the game, either via below functions, or via direct I/O access.

Vblank
Some of the below functions are allowing to use Vblank IRQs as a fourth "timer". However, Vblank IRQs are internally used by the BIOS for handling joypad and memory card accesses. One could theoretically use two separate Vblank IRQ handlers, one for joypad, and one as "timer", but the BIOS is much too unstable for such "shared" IRQ handling (it may occassionally miss one of the two handlers).
So, although Vblank IRQs are most important for games, the PSX BIOS doesn't actually allow to use them for purposes other than joypad access. A possible workaround is to examine the status byte in one of the joypad buffers (ie. the InitPad(buf1,22h,buf2,22h) buffers). Eg. a wait_for_vblank function could look like so: set buf1[0]=55h, then wait until buf1[0]=00h or buf1[0]=FFh.

B(02h) init_timer(t,reload,flags)
When t=0..2, resets the old timer mode by setting [1F801104h+t*16]=0000h, applies the reload value by [1F801108h+t*16]=reload, computes the new mode:

```
if flags.bit4=0 then mode=0048h else mode=0049h
if flags.bit0=0 then mode=mode OR 100h
if flags.bit12=1 then mode=mode OR 10h
```

and applies it by setting [1F801104h+t*16]=mode, and returns 1. Does nothing and returns zero for t>2.

Reads the current timer value: Returns halfword[1F801100h+t*16] for t=0..2. Does nothing and returns zero for t>2.

B(04h) enable_timer_irq(t)
B(05h) disable_timer_irq(t)
Enables/disables timer or vblank interrupt enable bits in [1F801074h], bit4,5,6 for t=0,1,2, or bit0 for t=3, or random/garbage bits for t>3. The enable function returns 1 for t=0..2, and 0 for t=3. The disable function returns always 1.

B(06h) restart_timer(t)
Sets the current timer value to zero: Sets [1F801100h+t*16]=0000h and returns 1 for t=0..2. Does nothing and returns zero for t>2.

C(0Ah) - ChangeClearRCnt(t,flag) ;root counter (aka timer)
Selects what the kernel's timer/vblank IRQ handlers shall do after they have processed an IRQ (t=0..2: timer 0..2, or t=3: vblank) (flag=0: do nothing; or flag=1: automatically acknowledge the IRQ and immediately return from exception). The function returns the old (previous) flag value.

BIOS Joypad Functions

Pad Input
Joypads should be initializes via InitPad(buf1,22h,buf2,22h), and StartPad(). The main program can read the pad data from the buf1/buf2 addresses (including Status, ID1, button states, and any kind of analogue inputs). For more info on ID1, Buttons and analogue inputs, see
Controllers and Memory Cards
Note: The BIOS doesn't include any functions for sending custom data to the pads (such like for controlling rumble motors).

B(12h) - InitPad(buf1, siz1, buf2, siz2)
Memorizes the desired buf1/buf2 addresses, zerofills the buffers by using the siz1/siz2 buffer size values (which should be 22h bytes each). And does some initialization on the PadCardIrq element (but doesn't enqueue it, that must be done by a following call to StartPad), and does set the "pad_enable_flag", that flag can be also set/cleared via InitCard(pad_enable), where it selects if the Pads are kept handled together with Memory Cards. buf1/buf2 are having the following format:

```
00h      Status (00h=okay, FFh=timeout/wrong ID2)
01h      ID1    (eg. 41h=digital_pad, 73h=analogue_pad, 12h=mouse, etc.)
02h..21h Data   (max 16 halfwords, depending on lower 4bit of ID1)
```

Note: InitPad does initially zerofill the buffers, so, until the first IRQ is processed, the initial status is 00h=okay, with buttons=0000h (all buttons pressed), to fix that situation, change the two status bytes to FFh after calling InitPad (or alternately, reject ID1=00h).
Once when the PadCardIrq is enqueued via StartPad, and while "pad_enable_flag" is set, the data for (both) Pad1 and Pad2 is read on Vblank interrupts, and stored in the buffers, the IRQ handler stores up to 22h bytes in the buffer (regardless of the siz1/siz2 values) (eg. a Multitap adaptor uses all 22h bytes).

B(13h) - StartPad()
Should be used after InitPad. Enqueues the PadCardIrq handler, and does additionally initialize some flags.

B(14h) - StopPad()
Dequeues the PadCardIrq handler. Note that this handler is also used for memory cards, so it'll "stop" cards, too.

B(15h) - OutdatedPadInitAndStart(type, button_dest, unused, unused)
This is an extremely bizarre and restrictive function - don't use! The function fails unless type is 20000000h or 20000001h (the type value has no other function).
The function uses "buf1/buf2" addresses that are located somewhere "hidden" within the BIOS variables region, the only way to read from that internal buffers is to use the ugly "OutdatedPadGetButtons()" function. For some strange reason it FFh-fills buf1/buf2, and does then call InitPad(buf1,22h,buf2,22) (which does immediately 00h-fill the previously FFh-filled buffers), and does then call StartPad().
Finally, it does memorize the "button_dest" address (see OutdatedPadGetButtons() for details on that value). The two unused parameters have no function, however, they are internally written back to the stack locations reserved for parameter 2 and 3, ie. at [SP+08h] and [SP+0Ch] on the caller's stack, so the function MUST be called with all four parameters allocated on stack. Return value is 2 (or 0 if type was disliked).

B(16h) - OutdatedPadGetButtons()
This is a very ugly function, using the internal "buf1/buf2" values from "OutdatedPadInitAndStart" and the "button_dest" value that was passed to that function.
If "button_dest" is non-zero, then this function is automatically called by the PadCardIrq handler, and stores it's return value at [button_dest] (where it may be read by the main program). If "button_dest" is zero, then it isn't called automatically, and it <can> be called manually (with return value in R2), however, it does additionally write the return value to [button_dest], ie. to [00000000h] in this case, destroying that memory location.
The return value itself is useless garbage: The lower 16bit contain the pad1 buttons, the upper 16bit the pad2 buttons, of which, both values have reversed byte-order (ie. the first button byte in upper 8bit). The function works only with controller IDs 41h (digital joypad) and 23h (nonstandard device). For ID=23h, the halfword is ORed with 07C7h, and bit6,7 are then cleared if the analogue inputs are greater than 10h. For ID=41h the data is left intact. Any other ID values, or disconnected joypads, cause the halfword to be set to FFFFh (same as when no buttons are pressed), that includes newer analogue pads (unless they are switched to "digital" mode).

BIOS GPU Functions

A(48h) - SendGP1Command(gp1cmd)
Writes [1F801814h]=gp1cmd. There's no return value (r2 is left unchanged).

A(49h) - GPU_cw(gp0cmd) ;send GP0 command word
Calls gpu_sync(), and does then write [1F801810h]=gp0cmd. Returns the return value from the gpu_sync() call.

A(4Ah) - GPU_cwp(src,num) ;send GP0 command word and parameter words
Calls gpu_sync(), and does then copy "num" words from [src and up] to [1F801810h], src should usually point to a command word, followed by num-1 parameter words. Transfer is done by software (without DMA). Always returns 0.

A(4Bh) - send_gpu_linked_list(src)
Transfer an OT via DMA. Calls gpu_sync(), and does then write [1F801814h]=4000002h, [1F8010F4h]=0, [1F8010F0h]=[1F8010F0h] OR 800h, [1F8010A0h]=src, [1F8010A4h]=0, [1F8010A8h]=1000401h. The function does additionally output a bunch of TTY status messages via printf. The function doesn't wait until the DMA is completed. There's no return value.

A(4Ch) - gpu_abort_dma()
Writes [1F8010A8h]=401h, [1F801814h]=4000000h, [1F801814h]=2000000h, [1F801814h]=1000000h. Ie. stops GPU DMA, and issues GP1(4), GP1(2), GP1(1). Returns 1F801814h, ie. the I/O address.

A(4Dh) - GetGPUStatus()
Reads [1F801814h] and returns that value.

A(46h) - GPU_dw(Xdst,Ydst,Xsiz,Ysiz,src)
Waits until GPUSTAT.Bit26 is set (unlike gpu_sync, which waits for Bit28), and does then [1F801810h]=A0000000h, [1F801810h]=YdstXdst, [1F801810h]=YsizXsiz, and finally transfers "N" words from [src and up] to [1F801810h], where "N" is "Xsiz*Ysiz/2". The data is transferred by software (without DMA) (by code executed in the uncached BIOS region with high waitstates, so the data transfer is very SLOW).
Caution: If "Xsiz*Ysiz" is odd, then the last halfword is NOT transferred, so the GPU stays waiting for the last data value.
Returns [SP+04h]=Ydst, [SP+08h]=Xsiz, [SP+0Ch]=Ysiz, [SP+10h]=src+N*4, and R2=src=N*4.

A(47h) - gpu_send_dma(Xdst,Ydst,Xsiz,Ysiz,src)
Calls gpu_sync(), writes [1F801810h]=A0000000h, [1F801814h]=4000002h, [1F8010F0h]=[1F8010F0h] OR 800h, [1F8010A0h]=src, [1F8010A4h]=N*10000h+10h (where N="Xsiz*Ysiz/32"), [1F8010A8h]=1000201h.
Caution: If "Xsiz*Ysiz" is not a multiple of 32, then the last halfword(s) are NOT transferred, so the GPU stays waiting for that values.
Returns R2=1F801810h, and [SP+04h]=Ydst, [SP+08h]=Xsiz, [SP+0Ch]=Ysiz.

A(4Eh) - gpu_sync()
If DMA is off (when GPUSTAT.Bit29-30 are zero): Waits until GPUSTAT.Bit28=1 (or until timeout).
If DMA is on: Waits until D2_CHCR.Bit24=0 (or until timeout), and does then wait until GPUSTAT.Bit28=1 (without timeout, ie. may hang forever), and does then turn off DMA via GP1(04h).
Returns 0 (or -1 in case of timeout, however, the timeout values are very big, so it may take a LOT of seconds before it returns).

BIOS Memory Allocation

A(33h) - malloc(size)
Allocates size bytes on the heap, and returns the memory handle (aka the address of the allocated memory block). The address of the block is guaranteed to by aligned to 4-byte memory boundaries. Size is rounded up to a multiple of 4 bytes. The address may be in KUSEG, KSEG0, or KSEG1, depending on the address passed to InitHeap.
Caution: The BIOS (tries to) initialize the heap size to 0 bytes (actually it accidently overwrites that initial setting by garbage during relocation), so any call to malloc will fail, unless InitHeap has been used to initialize the address/size of the heap.

A(34h) - free(buf)
Deallocates the memory block. There's no return value, and no error checking. The function simply sets [buf-4]=[buf-4] OR 00000001h, so if buf is an invalid handle it may destroy memory at [buf-4], or trigger a memory exception (for example, when buf=0).

A(37h) - calloc(sizx, sizy) ;SLOW!
Allocates xsiz*ysiz bytes by calling malloc(xsiz*ysiz), and, unlike malloc, it does additionally zerofill the memory via SLOW "bzero" function. Returns the address of the memory block (or zero if failed).

A(38h) - realloc(old_buf, new_size) ;SLOW!
If "old_buf" is zero, executes malloc(new_size), and returns r2=new_buf (or 0=failed). Else, if "new_size" is zero, executes free(old_buf), and returns r2=garbage. Else, executes malloc(new_size), bcopy(old_buf,new_buf,new_size), and free(old_buf), and returns r2=new_buf (or 0=failed).
Caution: The bcopy function is SLOW, and realloc does accidently copy "new_size" bytes from old_buf, so, if the old_size was smaller than new_size then it'll copy whatever garbage data - in worst case, if it exceeds the top of the 2MB RAM region, it may crash with a locked memory exception, although that'd happen only if SetMemSize(2) was used to restrict RAM to 2MBs.

A(39h) - InitHeap(addr, size)
Initializes the address and size of the heap - the BIOS does not automatically do this, so, before using the heap, InitHeap must be called by software. Usually, the heap would be memory region between the end of the boot executable, and the bottom of the executable's stack. InitHeap can be also used to deallocate all memory handles (eg. when a new exe file has been loaded, it may use it to deallocate all old memory).
The heap is used only by malloc/realloc/calloc/free, and by the "qsort" function.

B(00h) alloc_kernel_memory(size)
B(01h) free_kernel_memory(buf)
Same as malloc/free, but, instead of the heap, manages the 8kbyte control block memory at A000E000h..A000FFFFh. This region is used by the kernel to allocate ExCBs (4x08h bytes), EvCBs (N*1Ch bytes), TCBs (N*0C0h bytes), and the process control block (1x04h bytes). Unlike the heap, the BIOS does automatically initialize this memory region via SysInitMemory(addr,size), and does autimatically allocate the above data (where the number of EvCBs and TCBs is as specified in the SYSTEM.CNF file). Note: FCBs and DCBs are located elsewhere, at fixed locations in the kernel variables area.

Scratchpad Note
The kernel doesn't include any allocation functions for the scratchpad (nor do any kernel functions use that memory area), so the executable can freely use the "fast" memory at 1F800000h..1F8003FFh.

A(9Fh) - SetMemSize(megabytes)
Changes the effective RAM size (2 or 8 megabytes) by manipulating port 1F801060h, and additionally stores the size in megabytes in RAM at [00000060h].
Note: The BIOS bootcode accidently sets the RAM value to 2MB (which is the correct physical memory size), but initializes the I/O port to 8MB (which mirrors the physical 2MB within that 8MB region), so the initial values don't match up with each other.
Caution: Applying the correct size of 2MB may cause the "realloc" function to crash (that function may accidently access memory above 2MB).

BIOS Memory Fill/Copy/Compare (SLOW)

Like most A(NNh) functions, below functions are executed in uncached BIOS ROM, the ROM has very high waitstates, and the 32bit opcodes are squeezed through an 8bit bus. Moreover, below functions are restricted to process the data byte-by-byte. So, they are very-very-very slow, don't even think about using them.
Of course, that applies also for most other BIOS functions. But it's becoming most obvious for these small functions; memcpy takes circa 160 cycles per byte (reasonable would be less than 4 cycles), and bzero takes circa 105 cycles per byte (reasonable would be less than 1 cycles).

A(2Ah) - memcpy(dst, src, len)
Copies len bytes from [src..src+len-1] to [dst..dst+len-1]. Refuses to copy any data when dst=00000000h or when len>7FFFFFFFh. The return value is always the incoming "dst" value.

A(2Bh) - memset(dst, fillbyte, len)
Fills len bytes at [dst..dst+len-1] with the fillbyte value. Refuses to fill memory when dst=00000000h or when len>7FFFFFFFh. The return value is the incoming "dst" value (or zero, when len=0 or len>7FFFFFFFh).

A(2Ch) - memmove(dst, src, len) - bugged
Same as memcpy, but (attempts) to support overlapping src/dst regions, by using a backwards transfer when src<dst (and, for some reason, only when dst>=src+len).

A(2Dh) - memcmp(src1, src2, len) - bugged
Compares len bytes at [src1..src1+len-1] with [src2..src2+len-1], and (attempts) to return the difference between the first mismatching bytes, ie. [src1+N]-[src2+N], or 0 if there are no mismatches. Refuses to compare data when src1 or src2 is 00000000h, and returns 0 in that case.
BUG: Accidently returns the difference between the bytes AFTER the first mismatching bytes, ie. [src1+N+1]-[src2+N+1].
That means that a return value of 0 can mean absolutely anything: That the memory blocks are identical, or that a mismatch has been found (but that the NEXT byte after the mismatch does match), or that the function has failed (due to src1 or src2 being 00000000h).

A(2Eh) - memchr(src, scanbyte, len)
Scans [src..src+len-1] for the first occurence of scanbyte. Refuses to scan any data when src=00000000h or when len>7FFFFFFFh. Returns the address of that first occurence, or 0 if the scanbyte wasn't found.

A(27h) - bcopy(src, dst, len)
Same as "memcpy", but with "src" and "dst" exchanged. That is, the first parameter is "src", the refuse occurs when "src" is 00000000h, and, returns the incoming "src" value (whilst "memcpy" uses "dst" in that places).

A(28h) - bzero(dst, len)
Same as memset, but uses 00h as fixed fillbyte value.

A(29h) - bcmp(ptr1, ptr2, len) - bugged
Same as "memcmp", with exactly the same bugs.

BIOS String Functions

A(15h) - strcat(dst, src)
Appends src to the end of dst. Searches the ending 00h byte in dst, and copies src to that address, up to including the ending 00h byte in src. Returns the incoming dst value. Refuses to do anything if src or dst is 00000000h (and returns 0 in that case).

A(16h) - strncat(dst, src, maxlen)
Same as "strcat", but clipped to "MaxSrc=(min(0,maxlen)+1)" characters, ie. the total length is max "length(dst)+min(0,maxlen)+1". If src is longer or equal to "MaxSrc", then only the first "MaxSrc" chars are copied (with the last byte being replaced by 00h). If src is shorter, then everything up to the ending 00h byte gets copied, but without additional padding (unlike as in "strncpy").

A(17h) - strcmp(str1, str2)
Compares the strings up to including ending 00h byte. Returns 0 if they are identical, or otherwise [str1+N]-[str2+N], where N is the location of the first mismatch, the two bytes are sign-expanded to 32bits before doing the subtraction. The function rejects str1/str2 values of 00000000h (and returns 0=both are zero, -1=only str1 is zero, and +1=only str2 is zero).

A(18h) - strncmp(str1, str2, maxlen)
Same as "strcmp" but stops after comparing "maxlen" characters (and returns 0 if they did match). If the strings are shorter, then comparision stops at the ending 00h byte (exactly as for strcmp).

A(19h) - strcpy(dst, src)
Copies data from src to dst, up to including the ending 00h byte. Refuses to copy anything if src or dst is 00000000h. Returns the incoming dst address (or 0 if copy was refused).

A(1Ah) - strncpy(dst, src, maxlen)
Same as "strcpy", but clipped to "maxlen" characters. If src is longer or equal to maxlen, then only the first "maxlen" chars are copied (but without appending an ending 00h byte to dst). If src is shorter, then the remaining bytes in dst are padded with 00h bytes.

A(1Bh) - strlen(src)
Returns the length of the string up to excluding the ending 00h byte (or 0 when src is 00000000h).

A(1Ch) - index(src, char)
A(1Dh) - rindex(src, char)
A(1Eh) - strchr(src, char) ;exactly the same as "index"
A(1Fh) - strrchr(src, char) ;exactly the same as "rindex"
Scans for the first (index) or last (rindex) occurence of char in the string. Returns the memory address of that occurence (or 0 if there's no occurence, or if src is 00000000h). Char may be 00h (returns the end address of the string). Note that, despite of the function names, the return value is always a memory address, NOT an index value relative to src.

A(20h) - strpbrk(src, list)
Scans for the first occurence of a character that is contained in the list. The list contains whatever desired characters, terminated by 00h.
Returns the address of that occurence, or 0 if there was none. BUG: If there was no occurence, it returns 0 only if src[0]=00h, and otherwise returns the incoming "src" value (which is the SAME return value as when a occurence did occur on 1st character).

A(21h) - strspn(src, list)
A(22h) - strcspn(src, list)
Scans for the first occurence of a character that is (strspn), or that isn't (strcspn) contained in the list. The list contains whatever desired characters, terminated by 00h.
Returns the index (relative to src) of that occurence. If there was no occurence, then it returns the length of src. That silly return values do not actually indicate if an occurence has been found or not (unless one checks for [src+index]=00h or so).
***
"The strcspn() function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by s1 which consists entirely of bytes not from the string pointed to by s2."
"The strspn() function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by s1 which consists entirely of bytes from the string pointed to by s2."
***
Hmmmm, that'd be vice-versa?

A(23h) - strtok(src, list) ;first call
A(23h) - strtok(0, list) ;further call(s)
Used to split a string into fragments, list contains a list of characters that are to be treated as separators, terminated by 00h.
The first call copies the incoming string to a buffer in the BIOS variables area (the buffer size is 100h bytes, so the string should be max 255 bytes long, plus the

src=00000000h) are searching further fragments, starting at the buffer address from the previous call. The internal buffer is used only for strtok, so its contents (and the returned string fragments) remain intact until a new first call to strtok takes place.
The separate fragments are processed by searching the first separator, starting at the current buffer address, the separator is then replaced by a 00h byte, and the old buffer address is returned to the caller. Moreover, the function tries to skip all continously following separators, until reaching a non-separator, and does memorize that address for the next call (due to that skipping further calls won't return empty fragments, the first call may do so though). That skipping seems to be bugged, if list contains two or more different characters, then additional separators aren't skipped.

```
",,TEXT,,,END" with list=","  returns "", "TEXT", "END"
",,TEXT,,,END" with list=",." returns "", "", "TEXT", "", "", "END"
```

Once when there are no more fragments, then 00000000h is returned.

A(24h) - strstr(str, substr) - buggy
Scans for the first occurence of substr in the string. Returns the memory address of that occurence (or 0 if it was unable to find an occurence).
BUG: After rejecting incomplete matches, the function doesn't fallback to the old str address plus 1, but does rather continue at the current str address. Eg. it doesn't find substr="aab" in str="aaab" (in that example, it does merely realize that "aab"<>"aaa" and then that "aab"<>"b").

BIOS Number/String/Character Conversion

A(0Eh) - abs(val)
A(0Fh) - labs(val) ;exactly same as "abs"
Returns the absolute value (if val<0 then R2=-val, else R2=val).

A(0Ah) - todigit(char)
Takes the incoming character, ANDed with FFh, and returns 0..9 for characters "0..9" and 10..35 for "A..Z" or "a..z", or 0098967Fh (9,999,999 decimal) for any other 7bit characters, or garbage for characters 80h..FFh.

A(25h) - toupper(char)
A(26h) - tolower(char)
Returns the incoming character, ANDed with FFh, with letters "A..Z" converted to uppercase/lowercase format accordingly. Works only for char 00h..7Fh (some characters in range 80h..FFh are left unchanged, others are randomly "adjusted" by adding/subtracting 20h, and by sign-expanding the result to 32bits).

A(0Dh) - strtol(src, src_end, base)
Converts a string to a number. The function skips any leading "blank" characters (that are, 09h..0Dh, and 20h) (ie. TAB, CR, LF, SPC, and some others) (some characters in range 80h..FFh are accidently treated as "blank", too).
The incoming base value should be in range 2..11, although the function does also accept the buggy values in range of 12..36 (for values other than 2..36 it defaults to decimal/base10). The used numeric digits are "0..9" and "A..Z" (or less when base is smaller than 36).
The string may have a negative sign prefix "-" (negates the result) (a "+" is NOT recognized; and will be treated as the end of the string). Additionally, the string may contain prefixes "0b" (binary/base2), "0x" (hex/base16), or "o" (octal/base8) (only "o", not "0o"), allowing to override the incoming "base" value.
BUG: Incoming base values greater than 11 don't work due to the prefix feature (eg. base=16 with string "0b11" will be treated as 11 binary, and base=36 with string "o55" will be treated as 55 octal) (the only workaround would be to add/remove leading "0" characters, ie. "b11" or "00b11" or "0o55" would work okay).
Finally, the function initializes result=0, and does then process the digits as "result=result*base+digit" (without any overflow checks) unless/until it reaches an unknown digit (or when digit>=base) (ie. the string may end with 00h, or with any other unexpected characters).
The function accepts both uppercase and lowercase characters (both as prefixes, and as numeric digits). The function returns R2=result, and [src_end]=end_address (ie. usually the address of the ending 00h byte; or of any other unexpected end-byte). If src points to 00000000h, then the function returns r2=0, and leaves [src_end] unchanged.

A(0Ch) - strtoul(src, src_end, base)
Same as "strtol" except that it doesn't recognize the "-" sign prefix (ie. works only for unsigned numbers).

A(10h) - atoi(src)
A(11h) - atol(src) ;exactly same as "atoi" (but slightly slower)
Same as "strtol", except that it doesn't return the string end address in [src_end], and except that it defaults to base=10, but still supports prefixes, allowing to use base2,8,16. CAUTION: For some super bizarre reason, this function treats "0" (a leading ZERO digit) as OCTAL prefix (unlike strtol, which uses the "o" letter as octal prefix) (the "0x" and "0b" prefixes are working as usually).

A(12h) - atob(src, num_dst)
Calls "strtol(str,src_end,10)", and does then exchange the two return values (ie. sets R2=[src_end], and [num_dst]=value_32bit).

A(0Bh) - atof(src) ;USES (ABSENT) COP1 FPU !!!
A(32h) - strtod(src, src_end) ;USES (ABSENT) COP1 FPU !!!
These functions are intended to convert strings to floating point numbers, however, the functions are accidently compiled for MIPS processors with COP1 floating point unit (which is not installed in the PSX, nor does the BIOS support a COP1 software emulation), so calling these functions will produce a coprocessor exception, causing the PSX to lockup via A(40h) SystemErrorUnresolvedException.

Note
On other systems (eg. 8bit computers), "abs/atoi" (integer) and "labs/atol" (long) may act differently. However, on the Playstation, both use signed 32bit values.

BIOS Misc Functions

A(2Fh) - rand()
Advances the random generator as "x=x*41C64E6Dh+3039h" (aka plus 12345 decimal), and returns a 15bit random value "R2=(x/10000h) AND 7FFFh".

A(30h) - srand(seed)
Changes the current 32bit value of the random generator.

A(B4h) - GetSystemInfo(index) ;not supported by old CEX-1000 version
Returns a word, halfword, or string, depending on the selected index value:

```
00h      Get Kernel BCD Date       (eg. 19951204h) (YYYYMMDDh)
01h      Get Kernel Flags or so    (usually/always 000000003h)
02h      Get Kernel Version String (eg. "CEX-3000/1001/1002 by K.S.",0)
03h      Get whatever halfword     (usually 0)    ;PS2: returns cop0r15
04h      Get whatever halfword     (usually 0)
05h      Get RAM Size in kilobytes (usually 2048) ;=[00000060h] SHL 10
06h..0Eh Get whatever halfwords    (usually 0,400h,0,200h,0,0,1,1,1)
0Fh      N/A (returns zero) ;PS2: returns 0000h (effectively = same as zero)
```

Note: The Date/Version are referring to the Kernel (in the first half of the BIOS). The Intro and Bootmenu (in the second half of the BIOS) may have a different version, there's no function to retrieve info on that portion, however, a version string for it can be usually found at BFC7FF32h (eg. "System ROM Version 4.5 05/25/00 E",0) (in many bios versions, the last letter of that string indicates the region, but not in all versions) (the old SCPH1000 does not include that version string at all).

B(56h) GetC0Table()
B(57h) GetB0Table()
Retrieves the address of the jump lists for B(NNh) and C(NNh) functions, allowing to patch entries in that lists (however, the BIOS does often jump directly to the function addresses, rather than indirectly via the list, so patching may have little effect in such cases). Note: There's no function to retrieve the address of the A(NNh) jump list, however, that list is usually/always at 00000200h.

A(31h) - qsort(base, nel, width, callback)
Sorts an array, using a super-slow implementation of the "quick sort" algorithm. base is the address of the array, nel is the number of elements in the array, width is the size in bytes of each element, callback is a function that receives pointers to two elements which need to be compared; callback should return return zero if the elements are identical, or a positive/negative number to indicate which element is bigger.
The qsort function rearranges the contents of the array, ie. depending on the callback result, it may swap the contents of the two elements, for some bizarre reason it doesn't swap them directly, but rather stores one of the elements temporarily on the heap (that means, qsort works only if the heap was initialized with InitHeap, and only if "width" bytes are free). There's no return value.

A(35h) - lsearch(key, base, nel, width, callback)
A(36h) - bsearch(key, base, nel, width, callback)
Searches an element in an array (key is the pointer to the searched element, the other parameters are same as for "qsort"). "lsearch" performs a slow linear search in an unsorted array, by simply comparing one array element after each other. "bsearch" assumes that the array contains sorted elements (eg. via qsort), which is allowing to skip some elements, and to jump back and forth in the array, until it has found the desired element (or the location where it'd be, if it'd be in the array). Both functions return the address of the element (or 0 if it wasn't found).

C(19h) - ioabort(txt1,txt2)
Displays the two strings on the TTY (in some cases the BIOS does accidently pass garbage instead of the 2nd string though). And does then execute ioabort_raw(1), see there for more details.

A(B2h) - ioabort_raw(param) ;not supported by old CEX-1000 version
Executes "RestoreState(ioabortbuffer,param)". Internally used to recover from failed I/O operations, param should be nonzero to notify the SaveState caller that the abort has occurred.

A(13h) - SaveState(buf)
Stores some (not all) CPU registers in the specified buffer (30h bytes):
```
00h 4    r31 (ra) (aka caller's pc)
04h 4    r29 (sp)
08h 4    r30 (fp)
0Ch 4x8  r16..r23
2Ch 4    r28 (gp)
```
That type of buffer can be used with "ioabort", "RestoreState", and also "SetCustomExitFromException(addr)".
The "SaveState" function (initially) returns 0, however, it may return again - to the same return address - with another return value (which should be usually non-zero, to indicate that the state has been restored (eg. ioabort passes 1 as return value).

A(14h) - RestoreState(buf, param)
Restores the R16-R23,GP,SP,FP,RA registers from a previously recorded SaveState buffer, and "returns" to that new RA address (rather than to the caller of the RestoreState function), the "param" value is passed as "return value" to the code at RA, ie. usually to the caller of the original SaveState call) (since SaveState returns 0, "param" should be usually 1, or another non-zero value to inidicate that RestoreState has occurred). See SaveState for further details.

A(53h) - set_ioabort_handler(src) ;PS2 only ;PSX: SystemError
Normally the ioabort handler is changed only internally during booting, with this new function, games can install their own ioabort handler. src is pointer to a 30h-byte "savestate" structure, which will be copied to the actual ioabort structure.

A(06h) or B(38h) - exit(exitcode)
Terminates the program and returns control to the BIOS; which does then lockup itself via A(3Ah) SystemErrorExit.

A(A0h) - WarmBoot()
Performs a warmboot (resets the kernel and reboots from CDROM). Unlike the normal coldboot procedure, it doesn't display the "<S>" and "PS" intro screens (and doesn't verify the "PS" logo in the ISO System Area), and, doesn't enter the bootmenu (even if the disk drive is empty, or if it contains an Audio disk). And, it doesn't reload the SYSTEM.CNF file, so the function works only if the same disk is still inserted (or another disk with identical SYSTEM.CNF, such like Disk 2 of the same game).

A(B5h..BFh) B(11h,24h..29h,2Ch..31h,5Eh..FFh) C(1Eh..7Fh) - N/A - Jump 0
These functions jump to address 00000000h. For whatever reason, that address does usually contain a copy of the exception handler (ie. same as at address 80000080h). However, since there's no return address stored in EPC register, the functions will likely crash when returning from the exception handler.

A(57h..5Ah,73h..77h,79h..7Bh,7Dh,7Fh..80h,82h..8Fh,B0h..B1h,B3h), and
C(0Eh..11h,14h) - N/A - Returns 0
No function. Simply returns with r2=00000000h.
Reportedly, A(85h) is CdStop, but that seems to be nonsense?

SYS(00h) - NoFunction()
No function. Simply returns without changing any registers or memory locations (except that, of course, the exception handler destroys k0).

SYS(04h..FFFFFFFFh) - calls DeliverEvent(F0000010h,4000h)
These are syscalls with invalid function number in R4. For whatever reason that is handled by issuing DeliverEvent(F0000010h,4000h). Thereafter, the syscall returns to the main program (ie. it doesn't cause a SystemError).

A(3Ah) - SystemErrorExit(exitcode)
A(40h) - SystemErrorUnresolvedException()
A(A1h) - SystemErrorBootOrDiskFailure(type,errorcode) ;type "B"=Boot,"D"=Disk
These are used "SystemError" functions. The functions are repeatedly jumping to themselves, causing the system to hang. Possibly useful for debugging software which may hook that functions.

A(4Fh,50h,52h,53h,9Ah,9Bh) B(1Ah..1Fh,21h..23h,2Ah,2Bh,52h,5Ah) C(0Bh) - N/A
These are additional "SystemError" functions, but they are never used. The functions are repeatedly jumping to themselves, causing the system to hang.

BRK(1C00h) - Division by zero (commonly checked/invoked by software)
BRK(1800h) - Division overflow (-80000000h/-1, sometimes checked by software)
The CPU does not generate any exceptions upon divide overflows, because of that, the Kernel code and many games are commonly checking if the divider is zero (by software), and, if so, execute a BRK 1C00h opcode. The default BIOS exception handler doesn't handle BRK exceptions, and does simply redirect them to SystemErrorUnresolvedException().

BIOS Internal Boot Functions

A(45h) - init_a0_b0_c0_vectors
Copies the three default four-opcode handlers for the A(NNh),B(NNh),C(NNH) functions to A00000A0h..A00000CFh.

C(07h) - InstallExceptionHandlers() ;destroys/uses k0/k1
Copies the default four-opcode exception handler to the exception vector at 80000080h..8000008Fh, and, for whatever reason, also copies the same opcodes to 80000000h..8000000Fh.

C(08h) - SysInitMemory(addr,size)
Initializes the address (A000E000h) and size (2000h) of the allocate-able Kernel Memory region, and, seems to deallocate any memory handles which may have been allocated via B(00h).

C(09h) - SysInitKernelVariables()
Zerofills all Kernel variables; which are usually at [00007460h..0000891Fh].
Note: During the boot process, the BIOS accidently overwrites the first opcode of this function (by the last word of the A0h table), so, thereafter, this function won't work anymore (nor would it be of any use).

C(12h) - InstallDevices(ttyflag)
Initializes the size and address of the File and Device Control Blocks (FCBs and DCBs). Adds the TTY device by calling "KernelRedirect(ttyflag)", and the CDROM and Memory Card devices by calling "AddCDROMDevice()" and "AddMemCardDevice()".

C(1Ch) - AdjustA0Table()
Copies the B(32h..3Bh) and B(3Ch..3Fh) function addresses to A(00h..09h) and A(3Bh..3Eh). Apparently Sony's compiler/linker can't insert the addresses in the A0h table directly at compilation time, so this function is used to insert them during execution of the boot code.

BIOS More Internal Functions

Below are mainly internally used device related subfunctions.

Internal Device Stuff
```
A(5Bh) dev_tty_init()                                    ;PS2: SystemError
A(5Ch) dev_tty_open(fcb,and unused:"path\name",accessmode) ;PS2: SystemError
A(5Dh) dev_tty_in_out(fcb,cmd)                           ;PS2: SystemError
A(5Eh) dev_tty_ioctl(fcb,cmd,arg)                        ;PS2: SystemError
A(5Fh) dev_cd_open(fcb,"path\name",accessmode)
A(60h) dev_cd_read(fcb,dst,len)
A(61h) dev_cd_close(fcb)
A(62h) dev_cd_firstfile(fcb,"path\name",direntry)
A(63h) dev_cd_nextfile(fcb,direntry)
A(64h) dev_cd_chdir(fcb,"path")
A(65h) dev_card_open(fcb,"path\name",accessmode)
A(66h) dev_card_read(fcb,dst,len)
A(67h) dev_card_write(fcb,src,len)
A(68h) dev_card_close(fcb)
A(69h) dev_card_firstfile(fcb,"path\name",direntry)
A(6Ah) dev_card_nextfile(fcb,direntry)
A(6Bh) dev_card_erase(fcb,"path\name")
A(6Ch) dev_card_undelete(fcb,"path\name")
A(6Dh) dev_card_format(fcb)
A(6Eh) dev_card_rename(fcb1,"path\name1",fcb2,"path\name2")
A(6Fh) ?   ;card ;[r4+18h]=00000000h  ;card_clear_error(fcb) or so
A(96h) AddCDROMDevice()
A(97h) AddMemCardDevice()
A(98h) AddDuartTtyDevice()   ;PS2: SystemError
A(99h) AddDummyTtyDevice()
B(47h) AddDevice(device_info)  ;subfunction for AddXxxDevice functions
B(48h) RemoveDevice(device_name_lowercase)
B(5Bh) ChangeClearPad(int)   ;pad AND card (ie. used also for Card)
C(15h) tty_cdevinput(circ,char)
C(16h) tty_cdevscan()
C(17h) tty_circgetc(circ)    ;uses r5 as garbage txt for ioabort
C(18h) tty_circputc(char,circ)
```

Device Names
Device Names are case-sensitive (usually lowercase, eg. "bu" for memory cards). In filenames, the device name may be followed by a hexadecimal 32bit non-case-sensitive port number (eg. "bu00:" for selecting the first memory card slot). Accordingly, the device name should not end with a hexdigit (eg. "usb:" would be treated as device "us" with port number 0Bh).
Standard device names are "cdrom:", "bu00:", "bu10:", "tty00:". Other, nonstandard devices are:
```
Castlevania is trying to access an unknown device named "sim:".
Caetla (a firmware replacement for Cheat Devices) supports "pcdrv:" device.
```

BIOS PC File Server

DTL-H2000
Below BRK's are internally used in DTL-H2000 BIOS for two devices: "mwin:" (Message Window) and "sim:" (CDROM Sim).

Caetla Blurb
Caetla (a firmware replacement for Cheat Devices) supports "pcdrv:" device, the SN systems (=what?) device extension to access files on the drive of the pc. This fileserver can be accessed by using the kernel functions, with the "pcdrv:" device name prefix to the filenames or using the SN system calls.

The break functions have argument(s) in A1,A2,A3 (ie. unlike normal BIOS functions not in A0,A1,A2), and TWO return values (in V0, and V1).

BRK(101h) - PCInit() - Inits the fileserver
No parameters.

BRK(102h) - PCCreat(filename, fileattributes) - Creates a new file on PC
```
out: V0  0 = success, -1 = failure
     V1  file handle or error code if V0 is negative
```
Attributes Bits (standard MSDOS-style):
```
bit0     Read only file (R)
bit1     Hidden file    (H)
bit2     System file    (S)
bit3     Not used       (zero)
bit4     Directory      (D)
bit5     Archive file   (A)
bit6-31  Not used       (zero)
```

BRK(103h) - PCOpen(filename, accessmode) - Opens a file on the PC
```
out: V0  0 = success, -1 = failure
     V1  file handle or error code if V0 is negative
```

BRK(104h) - PCClose(filehandle) - Closes a file on the PC
```
out: V0  0 = success, -1 = failure
     V1  0 = success, error code if V0 is negative
```

BRK(105h) - PCRead(filehandle, length, memory_destination_address)
```
out: V0  0 = success, -1 = failure
     V1  number of read bytes or error code if V0 is negative.
```
Note: PCRead does not stop at EOF, so if you set more bytes to read than the filelength, the fileserver will pad with zero bytes. If you are not sure of the filelength obtain the filelength by PClSeek (A2=0, A3=2, V1 will return the length of the file, don't forget to reset the file pointer to the start before calling PCread!)

BRK(106h) - PCWrite(filehandle, length, memory_source_address)
```
out: V0  0 = success, -1 = failure
     V1  number of written bytes or error code if V0 is negative.
```

BRK(107h) - PClSeek(filehandle, file_offset, seekmode) - Change Filepos
seekmode may be from 0=Begin of file, 1=Current fpos, or 2=End of file.
```
out: V0  0 = success, -1 = failure
     V1  file pointer
```

BIOS TTY Console (std_io)

A(3Fh) - Printf(txt,param1,param2,etc.) - Print string to console
```
in:  A0                    Pointer to 0 terminated string
     A1,A2,A3,[SP+10h..]   Argument(s)
```
Prints the specified string to the TTY console. Printf does internally use "std_out_putchar" to output the separate characters (and expands char 09h and 0Ah accordingly).
The string can contain C-style escape codes (prefixed by "%" each):
```
c         display ASCII character
s         display ASCII string
i,d,D     display signed Decimal number (d/i=default32bit, D=force32bit)
u,U       display unsigned Decimal number (u=default32bit, U=force32bit)
o,O       display unsigned Octal number (o=default32bit, O=force32bit)
p,x,X     display unsigned Hex number (p=lower/force32bit, x=lower, X=upper)
n         write 32bit/16bit string length to [parameter] (default32bit)
```
Additionally, following prefixes (inserted between "%" and escape code):
```
+ or SPC  show leading plus or space character in positive signed numbers
NNN       fixed width (for padding or so) (first digit must be 1..9) (not 0)
.NNN      fixed width (for clipping or so)
*         variable width (using one of the parameters) (negative=ending_spc)
.*        variable width
-         force ending space padding (in case of width being specified)
#         show leading "0x" or "0X" (hex), or ensure 1 leading zero (octal)
0         show leading zero's
L         unknown/no effect?
h,l       force 16bit (h=halfword), or 32bit (l=long/word)
```
The force32bit codes (D,U,O,p,l) are kinda useless since the PSX defaults to 32bit parameters anyways. The force16bit code (h) may be useful as "%hn" (writeback 16bit value), otherwise it's rather useless, unless signed 16bit parameters have garbage in upper 16bit, for unsigned 16bit parameters it doesn't work at all (accidently sign-expands 16bit to 32bit, and then displays that signed 32bit value as giant unsigned value). Printf supports only octal, decimal, and hex (but not binary).

A(3Eh) or B(3Fh) std_out_puts(src) - Write string to TTY
```
in: R4=address of string (terminated by 00h)
```
Like "printf", but doesn't resolve any "%" operands. Empty strings are handled in a special way: If R4 points to a 00h character then nothing is output (as one would expect it), but, if R4 is 00000000h then "<NULL>" is output (only that six letters; without appending any CR or LF).

A(3Dh) or B(3Eh) std_in_gets(dst) - Read string from TTY (keyboard input)
```
in: r4=dst (pointer to a 128-byte buffer) - out: r2=dst (same is incoming r4)
```
Internally uses "std_in_getchar" to receive the separate characters (which are thus masked by 7Fh). The received characters are stored in the buffer, and are additionally sent back as echo to the TTY via std_out_putc.
The following characters are handled in a special way: 09h (TAB) is replaced by a single SPC. 08h or 7FH (BS or DEL) are removing the last character from the buffer (unless it is empty) and send 08h,20h,08h (BS,SPC,BS) to the TTY. 0Dh or 0Ah (CR or LF) do terminate the input (append 00h to the buffer, send 0Ah to the TTY, which is expanded to 0Dh,0Ah by the std_out_putc function, and do then return from the std_in_gets function).
The sequence 16h,NNh forces NNh to be stored in the buffer (even if NNh is a special character like 00h..1Fh or 7Fh). If the buffer is full (circa max 125 chars, plus one extra byte for the ending 00h), or if an unknown control code in range of 00h..1Fh is received without the 16h prefix, then 07h (BELL) is sent to the TTY.

A(3Bh) or B(3Ch) std_in_getchar() - Read character from TTY
Reads one character from the TTY console, by internally redirecting to "FileRead(0,tempbuf,1)". The returned character is ANDed by 7Fh (so, to read a fully intact 8bit character, "FileRead(0,tempbuf,1)" must be used instead of this function).

A(3Ch) or B(3Dh) std_out_putchar(char) - Write character to TTY
Writes the character to the TTY console, by internally redirecting to "FileWrite(1,tempbuf,1)". Char 09h (TAB) is expanded to one or more SPC characters, until

the remote terminal side) are 08h (BS, backspace, move cursor one position to the left), and 07h (BELL, produce a short beep sound).

C(13h) FlushStdInOutPut()
Closes and re-opens the std_in (fd=0) and std_out (fd=1) file handles.

C(1Bh) KernelRedirect(ttyflag) ;PS2: ttyflag=1 causes SystemError
Removes, re-mounts, and flushes the TTY device, the parameter selects whether to mount the real DUART-TTY device (r4=1), or a Dummy-TTY device (r4=0), the latter one sends any std_out to nowhere. Values other than r4=0 or r4=1 do remove the device, but do not re-mount it (which might result in problems).
Caution: Trying to use r4=1 on a PSX that does not has the DUART hardware installed causes the BIOS to hang (so one should first detect the DUART hardware, eg. by writing two different bytes to Port 1F802020h.1st/2nd access, and the read and verify that two bytes).

Activating std_io
The std_io functions can be enabled via C(1Bh) KernelRedirect(ttyflag), the BIOS is unable to detect the presence of the TTY hardware, by default the BIOS bootcode disables std_io by setting the initial KernelRedirect value at [A000B9B0h] to zero, this is hardcoded shortly after the POST(E) output:

```
call    output_post_r4        ;\output POST(E)
+mov    r4,0Eh                ;/
mov     r1,0A0010000h         ;\set [0A000B9B0h]=0 ;TTY=dummy/off
call    reset_cont_d_3        ; and call reset_cont_d_3
+mov    [r1-4650h],0          ;/
```

assuming that R28=A0010FF0h, the last 3 opcodes of above code can be replaced by:

```
mov     r1,1h                 ;\set [0A000B9B0h]=1 ;TTY=duart/on
call    reset_cont_d_3        ; and call reset_cont_d_3
+mov    [r28-4650h-0ff0h],r1  ;/
```

with that patch, the BIOS bootcode (and many games) are sending debug messages to the debug terminal, via expansion port, see:
EXP2 Dual Serial Port (for TTY Debug Terminal)
Note: The nocash BIOS automatically detects the DUART hardware, and activates TTY if it is present.

B(49h) PrintInstalledDevices()
Uses printf to display the long and short names from the DCB of the currently installed devices. Doesn't do anything else. There's no return value.

Note
Several BIOS functions are internally using printf to output status information, timeout, and error messages, etc. So, trying to close the TTY file handles (fd=0 and fd=1) would cause such functions to work unstable.

BIOS Character Sets

B(51h) Krom2RawAdd(shiftjis_code)

```
In: r4  = 16bit Shift-JIS character code
Out: r2 = address in BIOS ROM of the desired character (or -1 = error)
```

r4 should be 8140h..84BEh (charset 2), or 889Fh..9872h (charset 3).

B(53h) Krom2Offset(shiftjis_code)

```
In: r4  = 16bit Shift-JIS character code
Out: r2 = offset within charset (without charset base address)
```

This is a subfunction for B(51h) Krom2RawAdd(shiftjis_code).

Character Sets in ROM
The character sets are located at BFC64000h and up, intermixed with some other stuff:

```
BFC64000h  Charset 1 (16x15 pix, letters with accent marks)    (NOT in JAPAN)
BFC65CB6h  Garbage   (four-and-a-half reverb tables, ioports, printf strings)
BFC66000h  Charset 2 (16x15 pix, various alphabets, english, greek, etc.)
BFC69D68h  Charset 3 (16x15 pix, japanese or chinese symbols or so)
BFC7F8DEh  Charset 4 (8x15 pix, mainly ASCII letters)
BFC7FE6Fh  Charset 5 (8x15 pix, additional punctuation marks)    (NOT in PS2)
BFC7FF32h  Version   (Version and Copyright strings)        (NOT in SCPH1000)
BFC7FF8Ch  Charset 6 (8x15 pix, seven-and-a-half japanese chars) (NOT in PS2)
BFC80000h  End       (End of 512kBYTE BIOS ROM)
```

Charset 1 (and Garbage) is NOT included in japanese BIOSes (in the SCPH1000 version that region contains uncompressed program code, in newer japanese BIOSes that regions are zerofilled)
Charset 1 symbols are as defined in JIS-X-0212 char(2661h..2B77h), and EUC-JP char(8FA6E0h..8FABF7h).
Version (and Copyright) string is NOT included in SCPH1000 version (that BIOS includes further japanese 8x15 pix chars in that region).
For charset 2 and 3 it may be recommended to use the B(51h) Krom2RawAdd(shiftjis_code) to obtain the character addresses. Not sure if that BIOS function (or another BIOS function) allows to retrieve charset 1, 4, 5, and 6 addresses?

BIOS Control Blocks

Exception Control Blocks (ExCB) (4 blocks of 8 bytes each)

```
00h 4   ptr to first element of exception chain
04h 4   not used (zero)
```

Event Control Blocks (EvCB) (usually 16 blocks of 1Ch bytes each)

```
00h 4   class  (events are triggered when class and spec match)
04h 4   status (0=free,1000h=disabled,2000h=enabled/busy,4000h=enabled/ready)
08h 4   spec   (events are triggered when class and spec match)
0Ch 4   mode   (1000h=execute function/stay busy, 2000h=no func/mark ready)
10h 4   ptr to function to be executed when ready (or 0=none)
14h 8   not used (uninitialized)
```

Thread Control Blocks (TCB) (usually 4 blocks of 0C0h bytes each)

```
00h 4   status        (1000h=Free TCB, 4000h=Used TCB)
04h 4   not used      (set to 1000h by OpenThread) (not for boot executable?)
08h 80h r0..r31       (entries for r0/zero and r26/k0 are unused)
88h 4   cop0r14/epc   (aka r26/k0 and pc when returning from exception)
8Ch 8   hi,lo         (the mul/div registers)
94h 4   cop0r12/sr    (stored/restored by exception, NOT init by OpenThread)
98h 4   cop0r13/cause (stored when entering exception, NOT restored on exit)
9Ch 24h not used      (uninitialized)
```

Process Control Block (1 block of 4 bytes)

The PSX supports only one process, and thus only one Process Control Block.

File Control Blocks (FCB) (16 blocks of 2Ch bytes each)
```
  00h 4  status (0=Free FCB) (nonzero=accessmode)
  04h 4  cdrom: disk_id (checksum across path table of the corresponding disk),
         memory card: port number (00h=slot1, 10h=slot2)
  08h 4  transfer address (for dev_in_out function)
  0Ch 4  transfer length  (for dev_in_out function)
  10h 4  current file position
  14h 4  device flags (copy of DCB[04h])
  18h 4  error  ;used by B(55h) - GetLastFileError(fd)
  1Ch 4  Pointer to DCB for the file
  20h 4  filesize
  24h 4  logical block number (start of file) (for cdrom: at least)
  28h 4  file control block number (simply 0..15 for FCB number 0..15)
```

Device Control Blocks (DCB) (10 blocks of 50h bytes each)
```
  00h 4   ptr to lower-case short name ("cdrom", "bu", "tty") (or 0=Free DCB)
  04h 4   device flags (cdrom=14h, bu=14h, tty/dummy=1, tty/duart=3)
  08h 4   sector size  (cdrom=800h, bu=80h, tty=1)
  0Ch 4   ptr to upper-case long name  ("CD-ROM", "MEMORY CARD", "CONSOLE")
  10h 4   ptr to init()                                    (TTY only)
  14h 4   ptr to open(fcb,"path\name",accessmode)
  18h 4   ptr to in_out(fcb,cmd)                           (TTY only)
  1Ch 4   ptr to close(fcb)
  20h 4   ptr to ioctl(fcb,cmd,arg)                        (TTY only)
  24h 4   ptr to read(fcb,dst,len)
  28h 4   ptr to write(fcb,src,len)
  2Ch 4   ptr to erase(fcb,"path\name")
  30h 4   ptr to undelete(fcb,"path\name")
  34h 4   ptr to firstfile(fcb,"path\name",direntry)
  38h 4   ptr to nextfile(fcb,direntry)
  3Ch 4   ptr to format(fcb)
  40h 4   ptr to chdir(fcb,"path")                       (CDROM only)
  44h 4   ptr to rename(fcb1,"path\name1",fcb2,"path\name2")
  48h 4   ptr to remove()
  4Ch 4   ptr to testdevice(fcb,"path\name")
```

BIOS Versions

Kernel Versions
For the actual kernel, there seem to be only a few different versions. Most PSX/PSone's are containing the version from 1995 (which is kept 1:1 the same in all consoles; without any PAL/NTSC related customizations).
```
  28-Jul-1994  "DTL-H2000"                  ;v0.x (pre-retail devboard)
  22-Sep-1994  "CEX-1000 KT-3  by S.O."     ;v1.0 through v2.0
  no-new-date  "CEX-3000 KT-3  by K.S."     ;v2.1 only (old Port 1F801060h)
  04-Dec-1995  "CEX-3000/1001/1002 by K.S." ;v2.2 through v4.5 (except v4.0)
  29-May-1997  "CEX-7000/-7001 by K.S.    " ;v4.0 only (new Port 1F801010h)
  17-Jan-2000  "PS compatible mode by M.T." ;v5.0 (Playstation 2)
```
The date and version string can be retrieved via GetSystemInfo(index).
The "CEX-7000/-7001" version was only "temporarily" used (newer consoles switched back to the old version from 1995) (aside from the different date/version string, the only changed thing is the opcode at BFC00000h, which initializes port 1F801010h to BIOS ROM size of 1MB, instead of 512KB; no idea if that BIOS does actually contain additional data?).
The "CEX-3000 KT-3" version is already almost same as "CEX-3000/1001/1002", aside from version/date, the only differences are at offset BFC00014h..1Fh, and BFC003E0h (both related to Port 1F801060h).

Bootmenu/Intro Versions
This portion was updated more often. It's customized for PAL/NTSC displays, japanese/english language, and (maybe?) region/licence string checks. The SCPH1000 uses uncompressed Bootmenu/Intro code with "<S>" intro, but without "PS" intro (or, "PS" is shown only on region matches?), newer versions are using selfdecompressing code, with both intro screens. The GUI in older PSX models looks like a drawing program for children, the GUI in newer PSX models and in PSone's looks more like a modernized bathroom furniture, unknown how the PS2 GUI looks like?
Games are communicating only with the Kernel, so the differences in the Bootmenu/Intro part should have little or effect on compatibility (although some I/O ports might be initialized differently, and although some games might (accidently) read different (garbage) values from the ROM).
```
  Ver  CRC32    Used in                      System ROM Version  Kernel
  0.xj 18D0F7D8 DTL-H2000                    (no version string) dtlh2000
  1.0j 3B601FC8 SCPH-1000 and DTL-H1000      (no version string) cex1000
  1.1j 3539DEF6 SCPH-3000 and DTL-H1000H     "1.1 01/22/95"      ""
  2.0a 55847D8C DTL-H1001                    "2.0 05/07/95 A"    ""
  2.0e 9BB87C4B SCPH-1002 and DTL-H1002      "2.0 05/10/95 E"    ""
  2.1j BC190209 SCPH-3500                    "2.1 07/17/95 J"    cex3000
  2.1a AFF00F2F DTL-H1101                    "2.1 07/17/95 A"    ""
  2.1e 86C30531 SCPH-1002 and DTL-H1102      "2.1 07/17/95 E"    ""
  2.2j 24FC7E17 SCPH-5000 and DTL-H1200      "2.2 12/04/95 J"    cex3000/100x
  2.2a 37157331 SCPH-1001 and DTL-H1201/3001 "2.2 12/04/95 A"    ""
  2.2e 1E26792F SCPH-1002 and DTL-H1202/3002 "2.2 12/04/95 E"    ""
  2.2d DECB22F5 DTL-H1100                    "2.2 03/06/96 D"    ""
  3.0j FF3EEB8C SCPH-5500                    "3.0 09/09/96 J"    ""
  3.0a 8D8CB7E4 SCPH-5501/7003               "3.0 11/18/96 A"    ""
  3.0e D786F0B9 SCPH-5502/5552               "3.0 01/06/97 E"    ""
  4.0j EC541CD0 SCPH-7000/9000               "4.0 08/18/97 J"    cex7000
  4.1a 502224B6 SCPH-7001/7501/7503/9001     "4.1 12/16/97 A"    cex3000/100x
  4.1e 318178BF SCPH-7002/7502/9002          "4.1 12/16/97 E"    ""
  4.3j F2AF798B SCPH-100  (PSone)            "4.3 03/11/00 J"    ""
  4.4e 0BAD7EA9 SCPH-102  (PSone)            "4.4 03/24/00 E"    ""
  4.5a 171BDCEC SCPH-101  (PSone)            "4.5 05/25/00 A"    ""
  4.5e 76B880E5 SCPH-102  (PSone)            "4.5 05/25/00 E"    ""
  5.0t B7EF81A9 SCPH10000 (Playstation 2)    "5.0 01/17/00 T"    PS compatible
```
The System ROM Version string can be found at BFC7FF32h (except in v1.0).

v2.2j/a/e use exactly the same GUI as v2.1 (only the kernel was changed). v2.2d is almost same as v2.2j (but with some GUI patches or so).
v4.1 and v4.5 use exactly the same GUI code for "A" and "E" regions (the only difference is the last byte of the version string; which does specify whether the GUI shall use PAL or NTSC).
v5.0 is playstation 2 bios (4MB) with more or less backwards compatible kernel.

Character Set Versions

included only in non-japanese BIOSes, and in some newer japanese BIOSes (not included in v4.0j, but they are included in v4.3j).
The 8x15 pixel charset with characters 21h..7Fh is included in all BIOSes. In the SCPH1000, this region is followed by additional 8x15 punctuation marks at char 80h and up, however, this region is missing in PS2 BIOS. Moreover, some BIOSes include an incomplete 8x15 japanese character set (which ends abruptly at BF7FFFFFh), in newer BIOSes, some of theses chars are replaced by the version string at BFC7FF32h, and, the remaining 8x15 japanese chars were removed in the PS2 BIOS version.

BIOS Patches

The original PSX Kernel mainly consists of messy and unstable compiler generated code, and, to the worst, the <same> author seems to have attempted to use assembler code in some places. In result, most commercial games are causing a greater mess by inserting patches in the kernel code...
Which has been a nasty surprise when making the nocash PSX bios; which obviously wasn't compatible with these patches. The only solutions would have been to insert hundreds of NOPs to make my bios <exactly> as bloated as the original bios (which I really didn't want to do), or to create anti-patch-patches.

Patches and Anti-Patch-Patches
As shown below, all known patches are invoked by a B(56h) or B(57h) function call. In the nocash PSX bios, these two functions are examining the following opcodes, if the opcodes are a known patch, then the BIOS reproduces the desired behaviour, and does then continue normal execution after those opcodes. If the opcodes are unknown, then the BIOS simply locks up; and shows an error message with the address of that opcodes in the TTY window; info about any such unknown opcodes would be welcome!

Compatibility
If you want to (or need to) use patches, please use byte-identical opcodes as commercial games do (as shown below; only the "xxxx" address digits are don't care), so the nocash PSX bios (or other homebrewn BIOSes) can detect and reproduce them. Or alternately, don't use the BIOS, and access I/O ports directly, which is much better and faster anyways.

patch_missing_cop0r13_in_exception_handler:
In newer Kernel version, the exception handler reads cop0r13/cause to r2, examines the Excode value in r2, and if the exception was caused by an interrupt, and if the next opcode (at EPC) is a GTE/COP2 command, then it does increment EPC by 4. The GTE commands are executed even if an interrupt occurs simultaneously, so, without adjusting EPC, the command would be executed twice. With some commands that'd just waste some clock cycles, with other commands it may cause data to be written twice to the GTE FIFOs, or may re-use the result from the 1st command execution as input to the 2nd execution.
The old "CEX-1000 KT-3" Kernel version did examine r2, but it "forgot" to previously load cop0r13 to r2, so it did randomly examine a garbage value. The patch inserts the missing opcode, used in elo2 at 80033740h, and in Pandemonium II at 8007F3FCh:

```
  240A00B0 mov  r10,0B0h                    ;\   00000000 nop
  0140F809 call r10                         ;    00000000 nop
  24090056 +mov  r9,56h                     ;/   241A0100 mov k0,100h
  3C0Axxxx mov  r10,xxxx0000h               ;\   8F5A0008 mov k0,[k0+8h]
  3C09xxxx mov  r9,xxxx0000h                ;    00000000 nop
  8C420018 mov  r2,[r2+06h*4] ;=C(06h)      ;    8F5A0000 mov k0,[k0]
  254Axxxx add  r10,xxxxh ;=@@new_data      ;    00000000 nop
  2529xxxx add  r9,xxxxh  ;=@@new_data_end  ;/   235A0008 addt k0,8h
         @@copy_lop:                        ;\   AF410004 mov [k0+4h],r1
  8D430000 mov  r3,[r10]                    ;    AF420008 mov [k0+8h],r2
  254A0004 add  r10,4h                      ;    AF43000C mov [k0+0Ch],r3
  24420004 add  r2,4h                       ;    AF5F007C mov [k0+7Ch],ra
  1549FFFC jne  r10,r9,@@copy_lop           ;    40026800 mov r2,cop0r13
  AC43FFFC +mov [r2-4h],r3                  ;/   00000000 nop
```

Alternately, same as above, but using k0/k1 instead of r10/r9, used in Ridge Racer at 80047B14h:

```
  240A00B0 mov  r10,0B0h                    ;\     00000000 nop
  0140F809 call r10                         ;      00000000 nop
  24090056 +mov r9,56h                      ;/     241A0100 mov  k0,100h
  3C1Axxxx mov  k0,xxxx0000h                ;\     8F5A0008 mov  k0,[k0+8h]
  3C1Bxxxx mov  k1,xxxx0000h                ;      00000000 nop
  8C420018 mov  r2,[r2+06h*4] ;=C(06h)      ;      8F5A0000 mov  k0,[k0]
  275Axxxx add  k0,xxxxh  ;=@@new_data      ;      00000000 nop
  277Bxxxx add  k1,xxxxh  ;=@@new_data_end  ;/     235A0008 addt k0,8h
         @@copy_lop:                        ;\     AF410004 mov  [k0+4h],r1
  8F430000 mov  r3,[k0]                     ;      AF420008 mov  [k0+8h],r2
  275A0004 add  k0,4h                       ;      AF43000C mov  [k0+0Ch],r3
  24420004 add  r2,4h                       ;      AF5F007C mov  [k0+7Ch],ra
  175BFFFC jne  k0,k1,@@copy_lop            ;      40026800 mov  r2,cop0r13
  AC43FFFC +mov [r2-4h],r3                  ;/     00000000 nop
```

Alternately, slightly different code used in metal_gear_solid at 80095CC0h, and in alone1 at 800A3ECCh:

```
  24090056 mov  r9,56h                      ;\
  240A00B0 mov  r10,0B0h                    ; B(56h) GetC0Table
  0140F809 call r10                         ;
  00000000 +nop                             ;/
  8C420018 mov  r2,[r2+06h*4] ;=00000C80h = exception_handler = C(06h)
  00000000 nop
  24420028 add  r2,28h
  00407821 mov  r15,r2
  3C0Axxxx lui  r10,xxxxh ;\@@ori_data      ;\
  254Axxxx add  r10,xxxxh ;/                ;
  3C09xxxx lui  r9,xxxxh  ;\@@ori_data_end  ; @@ori_data:
  2529xxxx add  r9,xxxxh  ;/                ;  AF410004 mov [k0+4h],r1
         @@verify_lop:                      ;  AF420008 mov [k0+8h],r2
  8D430000 mov  r3,[r10]                    ;  AF43000C mov [k0+0Ch],r3
  8C4B0000 mov  r11,[r2]                    ;  AF5F007C mov [k0+7Ch],ra
  254A0004 add  r10,4h                      ;  40037000 mov r3,cop0r14
  146B000E jne  r3,r11,@@verify_mismatch    ;  00000000 nop
  24420004 +add r2,4h                       ;
  1549FFFA jne  r10,r9,@@verify_lop         ;
  00000000 +nop                             ;/
  01E01021 mov  r2,r15
  3C0Axxxx lui  r10,xxxxh ;\@@new_data      ;\
  254Axxxx add  r10,xxxxh ;/                ;
  3C09xxxx lui  r9,xxxxh  ;\@@new_data_end  ; @@new_data:
  2529xxxx add  r9,xxxxh  ;/                ;  AF410004 mov [k0+4h],r1
         @@copy_lop:                        ;  AF420008 mov [k0+8h],r2
  8D430000 mov  r3,[r10]                    ;  40026800 mov r2,cop0r13
  00000000 nop                              ;  AF43000C mov [k0+0Ch],r3
  AC430000 mov  [r2],r3                     ;  40037000 mov r3,cop0r14
  254A0004 add  r10,4h                      ;  AF5F007C mov [k0+7Ch],ra
  1549FFFB jne  r10,r9,@@copy_lop           ;
  24420004 +add r2,4h                       ;/
         @@verify_mismatch:
```

```
;BUG1: 8bit "movb r6" should be 32bit "mov r6"
;BUG2: @@copy_lop should transfer 6 words (not 7 words)
;BUG3: and, asides, the minimum demo works only with PAL BIOS (not NTSC)
0xxxxxxx call xxxxxxxxh              ;\B(56h) GetC0Table
00000000 +nop                        ;/(mov r8,0B0h, jmp r8, +mov r9,56h)
3C04xxxx mov  r4,xxxx0000h  ;\@@ori_data
2484xxxx add  r4,xxxxh      ;/
90460018 movb r6,[r2+06h*4] ;BUG1 ;exception_handler = C(06h)
24870018 add  r7,r4,18h ;@@ori_end     ;\
24C50028 add  r5,r6,28h ;C(06h)+28h    ;
00A03021 mov  r6,r5                  ;                  @@ori_data:
       @@verify_lop:                 ;  80086520 AF410004 mov [k0+4h],r1
8CA30000 mov  r3,[r5]                ;  80086524 AF420008 mov [k0+8h],r2
8C820000 mov  r2,[r4]                ;  80086528 AF43000C mov [k0+0Ch],r3
00000000 nop                         ;  8008652C AF5F007C mov [k0+7Ch],ra
1462000C jne  r3,r2,@@verify_mismatch  ;  80086530 40037000 mov r3,cop0r14
24840004 +add r4,4h                  ;  80086534 00000000 nop
1487FFFA jne  r4,r7,@@verify_lop      ;                  @@ori_end:
24A50004 +add r5,4h                  ;/
00C02821 mov  r5,r6                  ;\                 @@new_data:
3C04xxxx mov  r4,xxxx0000h ;\@@new_data;  80086538 AF410004 mov [k0+4h],r1
2484xxxx add  r4,xxxxh     ;/        ;  8008653C AF420008 mov [k0+8h],r2
2483001C add  r3,r4,1Ch ;@@bugged_end  ;  80086540 40026800 mov r2,cop0r13
       @@copy_lop:                   ;  80086544 AF43000C mov [k0+0Ch],r3
8C820000 mov  r2,[r4]                ;  80086548 40037000 mov r3,cop0r14
24840004 add  r4,4h                  ;  8008654C AF5F007C mov [k0+7Ch],ra
ACA20000 mov  [r5],r2                ;                  @@new_end:
1483FFFC jne  r4,r3,@@copy_lop       ;  80086550 00000000 nop  ;BUG2
24A50004 +add r5,4h                  ;/                 @@bugged_end:
       @@verify_mismatch:
```

early_card_irq_patch:
Because of a hardware glitch the card IRQ cannot be acknowledged while the external IRQ signal is still LOW, making it neccessary to insert a delay that waits until the signal gets HIGH before acknowledging the IRQ.
The original BIOS is so inefficient that it takes hundreds of clock cycles between the interrupt request and the IRQ acknowledge, so, normally, it doesn't require an additional delay.
However, the central mistake in the IRQ handler is that it doesn't memorize which IRQ has originally triggered the interrupt. For example, it may get triggered by a timer IRQ, but a newer card IRQ may occur during IRQ handling, in that case, the card IRQ may get processed and acknowledged without the required delay.
Used in Metal Gear Solid at 8009AA5Ch, and in alone1 at 800AE2F8h:

```
24090056 mov  r9,56h    ;\                 ;       @@new_data:
240A00B0 mov  r10,0B0h  ; B(56h) GetC0Table ;3C02A001 lui  r2,0A001h
0140F809 call r10       ;                  ;2442DFAC sub  r2,2054h
00000000 +nop           ;/                 ;00400008 jmp  r2 ;=@@new_cont_d
8C420018 mov  r2,[r2+06h*4] ;\get C(06h)   ;00000000 +nop    ;=A000DFACh
00000000 nop                ;/             ;00000000 nop
8C430070 mov  r3,[r2+70h]   ;\             ;       @@new_data_end:
00000000 nop                ; get          ;       @@new_cont_d:
3069FFFF and  r9,r3,0FFFFh  ; early_card   ;8C621074 mov  r2,[r3+1074h]
00094C00 shl  r9,10h        ; irq_handler  ;00000000 nop
8C430074 mov  r3,[r2+74h]   ;              ;30420080 and  r2,80h ;I_STAT.7
00000000 nop                ;              ;1040000B jz   r2,@@ret
306AFFFF and  r10,r3,0FFFFh ;/             ;00000000 +nop
012A1821 add  r3,r9,r10                    ;       @@wait_lop:
24620028 add  r2,r3,28h ;=early+28h        ;8C621044 mov  r2,[r3+1044h]
3C0Axxxx lui  r10,xxxxh ;\@@new_data       ;00000000 nop
254Axxxx sub  r10,xxxxh ;/                 ;30420080 and  r2,80h ;JOY_STAT.7
3C09xxxx lui  r9,xxxxh  ;\@@new_data_end   ;1440FFFC jnz  r2,@@wait_lop
2529xxxx sub  r9,xxxxh  ;/                 ;00000000 +nop
       @@copy_lop:                         ;3C020001 lui  r2,0001h
8D430000 mov  r3,[r10]                     ;8C42DFFC mov  r2,[r2-2004h]
00000000 nop                               ;00000000 nop
AC430000 mov  [r2],r3                      ;00400008 jmp  r2 ;=[0000DFFCh]
254A0004 add  r10,4h                       ;00000000 +nop
1549FFFB jne  r10,r9,@@copy_lop            ;       @@ret:
24420004 +add r2,4h                        ;03E00008 ret
3C010001 lui  r1,0001h      ;\[DFFCh]=r2   ;00000000 +nop
0xxxxxxx call xxxxxxxxh     ; and call ...  ;
AC22DFFC +mov [r1-2004h],r2 ;/             ;
```

Alternately, elo2 uses slightly different code at 8003961Ch:

```
240A00B0 mov  r10,0B0h  ;\                 ;       @@new_data:
0140F809 call r10       ; B(56h) GetC0Table ;3C02xxxx lui  r2,8xxxh
24090056 +mov r9,56h    ;/                 ;2442xxxx sub  r2,xxxxh
8C420018 mov  r2,[r2+06h*4] ;\get C(06h)   ;00400008 jmp  r2 ;=@@new_cont_d
00000000 nop                ;/             ;00000000 +nop    ;=8xxxxxxxh
8C430070 mov  r3,[r2+70h]   ;\             ;00000000 nop
00000000 nop                ; get          ;       @@new_data_end:
3069FFFF and  r9,r3,0FFFFh  ; early_card   ;       @@new_cont_d:
8C430074 mov  r3,[r2+74h]   ; irq_handler  ;8C621074 mov  r2,[r3+1074h]
00094C00 shl  r9,10h        ;              ;00000000 nop
306AFFFF and  r10,r3,0FFFFh ;              ;30420080 and  r2,80h ;I_STAT.7
012A1821 add  r3,r9,r10     ;/             ;1040000B jz   r2,@@ret
3C0Axxxx mov  r10,xxxx0000h                ;00000000 +nop
3C09xxxx mov  r9,xxxx0000h                 ;       @@wait_lop:
24620028 add  r2,r3,28h ;=early+28h        ;8C621044 mov  r2,[r3+1044h]
254Axxxx sub  r10,xxxxh ;=@@new_data       ;00000000 nop
2529xxxx sub  r9,xxxxh  ;=@@new_data_end   ;30420080 and  r2,80h ;JOY_STAT.7
       @@copy_lop:                         ;1440FFFC jnz  r2,@@wait_lop
8D430000 mov  r3,[r10]                     ;00000000 +nop
254A0004 add  r10,4h                       ;3C02xxxx lui  r2,8xxxh
24420004 add  r2,4h                        ;8C42xxxx mov  r2,[r2-xxxxh]
1549FFFC jne  r10,r9,@@copy_lop            ;00000000 nop
AC43FFFC +mov [r2-4h],r3                   ;00400008 jmp  r2 ;=[8xxxxxxxh]
3C018xxx mov  r1,8xxx0000h  ;\[...]=r2,    ;00000000 +nop
0xxxxxxx call xxxxxxxxh     ; and call ...  ;       @@ret:
AC22xxxx +mov [r1+xxxxh],r2 ;/             ;03E00008 ret
         ...                               ;00000000 +nop
```

Note: The above @@wait_lop's should be more preferably done with timeouts (else they may hang endless if a Sony Mouse is newly connected; the mouse does have /ACK stuck LOW on power-up).

patch_uninstall_early_card_irq_handler:

allow to uninstall it thereafter).
Used in Breath of Fire III (SLES-01304) at 8017E790, and also in Ace Combat 2 (SLUS-00404) at 801D23F4:

```
240A00B0 mov  r10,0B0h          ;\
0140F809 call r10               ; B(56h) GetC0Table
24090056 +mov r9,56h            ;/
3C0Axxxx mov  r10,xxxx0000h
3C09xxxx mov  r9,xxxx0000h
8C420018 mov  r2,[r2+06h*4] ;=00000C80h = exception_handler = C(06h)
254Axxxx add  r10,xxxxh ;@@new_data
2529xxxx add  r9,xxxxh  ;@@new_data_end
        @@copy_lop:             ;\  @@new_data:
8D430000 mov  r3,[r10]          ;    00000000 nop
254A0004 add  r10,4h            ;    00000000 nop
24420004 add  r2,4h             ;    00000000 nop
1549FFFC jne  r10,r9,@@copy_lop ;   @@new_data_end:
AC43006C +mov [r2+70h-4],r3     ;/
```

Alternately, more inefficient, used in Blaster Master-Blasting Again (SLUS-01031) at 80063FF4h, and Raiden DX at 80029694h:

```
24090056 mov  r9,56h            ;\
240A00B0 mov  r10,0B0h          ; B(56h) GetC0Table
0140F809 call r10               ;
00000000 +nop                   ;/
8C420018 mov  r2,[r2+06h*4] ;=00000C80h = exception_handler = C(06h)
3C0Axxxx mov  r10,xxxx0000h ;\@@new_data
254Axxxx add  r10,xxxxh     ;/
3C09xxxx mov  r9,xxxx0000h  ;\@@new_data_end
2529xxxx add  r9,xxxxh      ;/
        @@copy_lop:             ;\
8D430000 mov  r3,[r10]          ;   @@new_data:
00000000 nop                    ;    00000000 nop
AC430070 mov  [r2+70h],r3       ;    00000000 nop
254A0004 add  r10,4h  ;src      ;    00000000 nop
1549FFFB jne  r10,r9,@@copy_lop ;   @@new_data_end:
24420004 +add r2,4h   ;dst      ;/
```

Note: the above code is same as "patch_install_lightgun_irq_handler", except that it writes to r2+70h, instead of r2+80h.

patch_card_specific_delay:
Same purpose as the "early_card_irq_patch" (but for the command/status bytes rather than for the data bytes). The patch looks buggy since it inserts the delay AFTER the acknowledge, but it DOES work (the BIOS accidently acknowledges the IRQ twice; and the delay occurs PRIOR to 2nd acknowledge).
Used in Metal Gear Solid at 8009AAF0h, and in Legacy of Kain at 801A56D8h, and in alone1 at 800AE38Ch:

```
24090057 mov  r9,57h   ;\                   ;         @@new_data:
240A00B0 mov  r10,0B0h ; B(57h) GetB0Table  ; 3C08A001 lui  r8,0A001h
0140F809 call r10      ;/                   ; 2508DF80 sub  r8,2080h
00000000 +nop                               ; 0100F809 call r8 ;=A000DF80h
8C42016C mov  r2,[r2+5Bh*4] ;B(5Bh)         ; 00000000 +nop
00000000 nop                                ; 00000000 nop
8C4309C8 mov  r3,[r2+9C8h]  ;blah           ;         @@new_data_end:
3C0Axxxx lui  r10,xxxxh ;\@@new_data        ; 946F000A movh r15,[r3+0Ah]
254Axxxx sub  r10,xxxxh ;/                  ; 3C080000 mov  r8,0h
3C09xxxx lui  r9,xxxxh  ;\@@new_data_end    ; 01E2C025 or   r24,r15,r2
2529xxxx sub  r9,xxxxh  ;/                  ; 37190012 or   r25,r24,12h
        @@copy_lop:                         ; A479000A movh [r3+0Ah],r25
8D480000 mov  r8,[r10]                      ; 24080028 mov  r8,28h
00000000 nop                                ;         @@wait_lop:
AC4809C8 mov  [r2+9C8h],r8   ;B(5Bh)+9C8h.. ; 2508FFFF sub  r8,1h
254A0004 add  r10,4h                        ; 1500FFFE jnz  r8,@@wait_lop
1549FFFB jne  r10,r9,@@copy_lop             ; 00000000 +nop
24420004 +add r2,4h                         ; 03E00008 ret  ;above delay is
        ...                                 ; 00000000 +nop ;in UNCACHED RAM
```

Alternately, slightly different code used in elo2 at800396D4h, and in Resident Evil 2 at 800910E4h:

```
240A00B0 mov  r10,0B0h ;\                   ;         @@swap_begin:
0140F809 call r10      ; B(57h) GetB0Table  ; 3C088xxx lui  r8,8xxxh
24090057 +mov r9,57h   ;/                   ; 2508xxxx sub  r8,xxxxh
8C42016C mov  r2,[r2+5Bh*4] ;B(5Bh)         ; 0100F809 call r8 ;=8xxxxxxxh
3C0Axxxx mov  r10,xxxx0000h                 ; 00000000 +nop
3C09xxxx mov  r9,xxxx0000h                  ; 00000000 nop
8C4309C8 mov  r3,[r2+9C8h] ;blah            ;         @@swap_end:
254Axxxx sub  r10,xxxxh  ;=@@swap_begin     ;         ;- -  -
2529xxxx sub  r9,xxxxh   ;=@@swap_end       ; 00000000 nop
        @@swap_lop:                         ; 240800C8 mov  r8,0C8h
8C4309C8 mov  r3,[r2+9C8h] ;B(5Bh)+9C8h..   ;         @@wait_lop:
8D480000 mov  r8,[r10]                      ; 2508FFFF sub  r8,1h
254A0004 add  r10,4h                        ; 1500FFFE jnz  r8,@@wait_lop
AD43FFFC mov  [r10-4h],r3                   ; 00000000 +nop
24420004 add  r2,4h                         ; 03E00008 ret  ;above delay is
1549FFFA jne  r10,r9,@@swap_lop             ; 00000000 +nop ;in CACHED RAM
AC4809C4 +mov [r2+9C4h],r8                  ;
```

patch_card_info_step4:
The "card_info" function sends an incomplete read command to the card; in order to receive status information. After receiving the last byte, the function does accidently send a further byte to the card, so the card responds by another byte (and another IRQ7), which is not processed nor acknowledged by the BIOS. This patch kills the opcode that sends the extra byte.
Used in alone1 at 800AE214h:

```
24090057 mov  r9,57h                        ;\
240A00B0 mov  r10,0B0h                      ; B(57h) GetB0Table
0140F809 call r10                           ;
00000000 +nop                               ;/
240A0009 mov  r10,9h        ;=blah
8C42016C mov  r2,[r2+5Bh*4] ;=B(5Bh)
00000000 nop
20431988 addt r3,r2,1988h   ;=B(5Bh)+1988h  ;\store a NOP,
0xxxxxxx call xxxxxxxxh                     ; and call ...
AC600000 +mov [r3],0        ;=nop           ;/
```

patch_pad_error_handling_and_get_pad_enable_functions:
If a transmission error occurs (or if there's no controller connected), then the Pad handler handler does usually issue a strange chip select signal to the OTHER controller slot, and does then execute the bizarre_pad_delay function. The patch below overwrites that behaviour by NOPs. Purpose of the original (and patched) behaviour is unknown.
Used by Perfect Assassin at 800519D4h:

```
0140F809 call r10                          ; B(57h) GetB0Table
24090057 +mov r9,57h                       ;/
8C42016C mov  r2,[r2+5Bh*4] ;=B(5Bh)
3C01xxxx mov  r1,xxxx0000h
20430884 addt r3,r2,884h    ;B(5Bh)+884h
AC23xxxx mov  [r1+xxxxh],r3 ;<--- SetPadEnableFlag()
3C01xxxx mov  r1,xxxx0000h
20430894 addt r3,r2,894h    ;B(5Bh)+894h
2409000B mov  r9,0Bh        ;len
AC23xxxx mov  [r1+xxxxh],r3 ;<--- ClearPadEnableFlag()
        @@fill_lop:                        ;\
2529FFFF sub  r9,1h                        ;
AC400594 mov  [r2+594h],0   ;B(5Bh)+594h..  ; erase error handling
1520FFFD jnz  r9,@@fill_lop                ;
24420004 +add r2,4h                        ;/
```

Alternately, same as above, but with inefficient nops, used by Sporting Clays at 8001B4B4h:

```
24090057 mov  r9,57h       ;\
240A00B0 mov  r10,0B0h     ; B(57h) GetB0Table
0140F809 call r10          ;
00000000 +nop              ;/
8C42016C mov  r2,[r2+5Bh*4]
2409000B mov  r9,0Bh ;len
20430884 addt r3,r2,884h
3C01xxxx mov  r1,xxxx0000h
AC23xxxx mov  [r1+xxxxh],r3 ;<--- SetPadEnableFlag()
20430894 addt r3,r2,894h
3C01xxxx mov  r1,xxxx0000h
AC23xxxx mov  [r1+xxxxh],r3 ;<--- ClearPadEnableFlag()
        @@fill_lop:        ;\
AC400594 mov  [r2+594h],0   ;
24420004 add  r2,4h         ; erase error handling
2529FFFF sub  r9,1h         ;
1520FFFC jnz  r9,@@fill_lop ;
00000000 +nop              ;/
```

Alternately, same as above, but without getting PadEnable functions, used in Pandemonium II (at 80083C94h and at 8010B77Ch):

```
240A00B0 mov  r10,0B0h              ;\
0140F809 call r10                   ; B(57h) GetB0Table
24090057 +mov r9,57h                ;/
8C42016C mov  r2,[r2+5Bh*4] ;=B(5Bh)
2409000B mov  r9,0Bh        ;len            ;\
        @@fill_lop:                         ;
2529FFFF sub  r9,1h                         ; erase error handling
AC400594 mov  [r2+594h],0   ;B(5Bh)+594h..  ;
1520FFFD jnz  r9,@@fill_lop                 ;
24420004 +add r2,4h                         ;/
```

patch_optional_pad_output:

The normal BIOS functions are only allowing to READ from the controllers, but not to SEND data to them (which would be required to control Rumble motors, and to auto-activate Analog mode without needing the user to press the Analog button). Internally, the BIOS does include some code for sending data to the controller, but it doesn't offer a function vector for setting up the data source address, and, even if that would be supported, it clips the data bytes to 00h or 01h. The patch below retrieves the required SetPadOutput function address (in which only the src1/src2 addresses are relevant, the blah1/blah2 values aren't used), and suppresses clipping (ie. allows to send any bytes in range 00h..FFh).

Used in Resident Evil 2 at 80091914h:

```
240A00B0 mov  r10,0B0h                      ;\
0140F809 call r10                           ; B(57h) GetB0Table
24090057 +mov r9,57h                        ;/
8C42016C mov  r2,[r2+5Bh*4] ;B(5Bh)
3C0Axxxx mov  r10,xxxx0000h
3C09xxxx mov  r9,xxxx0000h
3C01xxxx mov  r1,xxxx0000h
204307A0 addt r3,r2,7A0h    ;B(5Bh)+7A0h
254Axxxx add  r10,xxxxh  ;=@@new_data
2529xxxx add  r9,xxxxh   ;=@@new_data_end
AC23xxxx mov  [r1-xxxxh],r3 ;<--- SetPadOutput(src1,blah1,src2,blah2)
        @@double_copy_lop:                  ;\
8D430000 mov  r3,[r10]                      ;          @@new_data:
254A0004 add  r10,4h                        ;   00551024 and     r2,r21
AC4303D8 mov  [r2+3D8h],r3  ;<--- here      ;   00000000 nop
24420004 add  r2,4h                         ;   00000000 nop
1549FFFB jne  r10,r9,@@double_copy_lop      ;   00000000 nop
AC4304DC +mov [r2+4DCh],r3  ;<--- here      ;/          @@new_data_end:
```

Alternately, more inefficient (with NOPs), used in Lemmings at 80036618h:

```
24090057 mov  r9,57h                        ;\
240A00B0 mov  r10,0B0h                      ; B(57h) GetB0Table
0140F809 call r10                           ;
00000000 +nop                               ;/
3C0Axxxx mov  r10,xxxx0000h
254Axxxx add  r10,xxxxh    ;=@@new_data
3C09xxxx movp r9,xxxx0000h
2529xxxx add  r9,xxxxh     ;=@@new_data_end
8C42016C mov  r2,[r2+5Bh*4] ;B(5Bh)
00000000 nop
204307A0 addt r3,r2,7A0h    ;B(5Bh)+7A0h
3C01xxxx mov  r1,xxxx0000h
AC23xxxx mov  [r1+xxxxh],r3 ;<--- SetPadOutput(src1,blah1,src2,blah2)
        @@double_copy_lop:                  ;\
8D430000 mov  r3,[r10]                      ;          @@new_data:
00000000 nop                                ;   00551024 and     r2,r21
AC4303D8 mov  [r2+3D8h],r3                  ;   00000000 nop
AC4304E0 mov  [r2+4E0h],r3                  ;   00000000 nop
24420004 add  r2,4h                         ;   00000000 nop
254A0004 add  r10,4h                        ;          @@new_data_end:
1549FFF9 jne  r10,r9,@@double_copy_lop      ;
00000000 +nop                               ;/
```

patch_no_pad_card_auto_ack:

This patch suppresses automatic IRQ0 (vblank) acknowleding in the Pad/Card IRQ handler, that, even if auto-ack is enabled. Obviously, one could as well disable auto-ack via B(5Bh) ChangeClearPad(int), so this patch is total nonsense. Used in Resident Evil 2 at 800919ACh:

```
240A00B0 mov   r10,0B0h                     ;\
0140F809 call  r10                          ; B(57h) GetB0Table
```

```
8C42016C mov   r2,[r2+5Bh*4] ;=B(5Bh)
240A0009 mov   r10,9h        ;len            ;\
2043062C addt  r3,r2,62Ch    ;=B(5Bh)+62Ch   ;
        @@fill_lop:                          ;
254AFFFF sub   r10,1h                        ;
AC600000 mov   [r3],0                        ;
1540FFFD jnz   r10,@@fill_lop                ;
24630004 +add  r3,4h                         ;/
```

Alternately, same as above, but more inefficient, used in Sporting Clays at 8001B53Ch:

```
24090057 mov   r9,57h                        ;\
240A00B0 mov   r10,0B0h                      ; B(57h) GetB0Table
0140F809 call  r10                           ;
00000000 +nop                                ;/
240A0009 mov   r10,9h    ;len
8C42016C mov   r2,[r2+5Bh*4]
00000000 nop
2043062C addt  r3,r2,62Ch
        @@fill_lop:                          ;\
AC600000 mov   [r3],0                        ;
24630004 add   r3,4h                         ;
254AFFFF sub   r10,1h                        ;
1540FFFC jnz   r10,@@fill_lop                ;
00000000 +nop                                ;/
```

Either way, no matter if using the patch or if using ChangeClearPad(int), having auto-ack disabled allows to install a custom vblank IRQ0 handler, which is probably desired for most games, however, mind that the PSX BIOS doesn't actually support the same IRQ to be processed by two different IRQ handlers, eg. the custom handler may acknowledge the IRQ even when the Pad/Card handler didn't process it, so pad input may become bumpy.

patch_install_lightgun_irq_handler:

Used in Sporting Clays at 80027D68h (when Konami Lightgun connected):

```
240A00B0 mov  r10,0B0h     ;\
0140F809 call r10          ; B(56h) GetC0Table
24090056 +mov r9,56h       ;/
3C0Axxxx mov  r10,xxxx0000h ;src
3C09xxxx mov  r9,xxxx0000h  ;src.end
8C420018 mov  r2,[r2+06h*4] ;C(06h)
254Axxxx add  r10,xxxxh     ;src
2529xxxx add  r9,xxxxh      ;src.end (=src+10h)
        @@copy_lop:              ;\    ;        @@src:
8D430000 mov  r3,[r10]           ;     ;3C02xxxx mov  r2,xxxx0000h
254A0004 add  r10,4h             ;     ;2442xxxx add  r2,xxxxh
24420004 add  r2,4h              ;     ;0040F809 call r2  ;lightgun_proc
1549FFFC jne  r10,r9,@@copy_lop  ;     ;00000000 +nop
AC43007C +mov [r2+80h-4],r3      ;/            @@src_end:
```

Alternately, same as above, but more inefficient, used in DQM (Dragon Quest Monsters 1&2) at 80089390h (install) and 800893F8h (uninstall):

```
24090056 mov  r9,56h        ;\
240A00B0 mov  r10,0B0h      ; B(56h) GetC0Table
0140F809 call r10           ;
00000000 +nop               ;/
8C420018 mov  r2,[r2+06h*4] ;=00000C80h = exception_handler = C(06h)
3C0Axxxx mov  r10,xxxx0000h ;\@@new_data (3xNOP)
254Axxxx add  r10,-xxxxh    ;/
3C09xxxx mov  r9,xxxx0000h  ;\@@new_data_end
2529xxxx add  r9,-xxxxh     ;/
        @@copy_lop:             ;\
8D430000 mov  r3,[r10]          ; @@new_data: ;for (un-)install...
00000000 nop                    ; 00000000 nop / 3C02xxxx mov r2,xxxx0000h
AC430080 mov  [r2+80h],r3       ; 00000000 nop / 2442xxxx add r2,-xxxxh
254A0004 add  r10,4h            ; 00000000 nop / 0040F809 call r2  ;proc
1549FFFB jne  r10,r9,@@copy_lop ; @@new_data_end:
24420004 +add r2,4h             ;/
```

Some lightgun games (eg. Project Horned Owl) do (additionally to above stuff) hook the exception vector at 00000080h, the hook copies the horizontal coordinate (timer0) to a variable in RAM, thus getting the timer0 value "closest" to the actual IRQ execution. Doing that may eliminate some unpredictable timing offsets that could be caused by cache hits/misses during later IRQ handling (and may also eliminate a rather irrelevant 1-cycle inaccuracy depending on whether EPC was pointing to a GTE opcode, and also eliminates constant cycle offsets depending on whether early_card_irq_handler was installed and enabled, and might eliminate timing differences for different BIOS versions).

set_conf_without_realloc:

Used in Spec Ops Airborne Commando at 80070AE8h, and also in the homebrew game Roll Boss Rush at 80010B68h and 8001B85Ch. Purpose is unknown (maybe to override improperly defined .EXE headers).

```
8C030474 mov   r3,[200h+(9Dh*4)]      ;\get ptr to A(9Dh) GetConf (done so,
00000000 nop                          ;/as there's no "GetA0Table" funtion)
94620000 movh  r2,[r3+0h] ;lui msw    ;\
84630004 movhs r3,[r3+4h] ;lw lsw+8   ; extract ptr to "boot_cnf_values"
00021400 shl   r2,10h     ;msw*10000h ; (from first 2 opcodes of GetConf)
2442FFF8 sub   r2,8h      ;undo +8    ;
00431021 add   r2,r3      ;lsw        ;/
AC450000 mov   [r2+0h],r5 ;num_TCB    ;\set num_EvCB,num_TCB,stacktop
AC440004 mov   [r2+4h],r4 ;num_EvCB   ; (unlike A(9Ch) SetConf, without
03E00008 ret                          ; actually reallocting anything)
AC460008 +mov  [r2+8h],r6 ;stacktop   ;/
```

Cheat Devices

CAETLA detects the PSX BIOS version by checksumming BFC06000h..BFC07FFFh and does then use some hardcoded BIOS addresses based on that checksum. The reason for doing that is probably that the Pre-Boot Expansion ROM vector is called with the normal A0h/B0h/C0h vectors being still uninitialized.
Problems are that the hardcoded addresses won't work with all BIOSes (eg. not with the no$psx bios clone, probably also not with the newer PS2 BIOS), moreover, the checksumming can fail with patched original BIOSes (eg. no$psx allows to enable TTY debug messages and to skip the BIOS intro).
The Cheat Firmwares are probably also hooking the Vblank handler, and maybe also some other functions.
ACTION REPLAY (at least later versions like 2.81) uses the Pre-Boot handler to set a COP0 hardware breakpoint at 80030000h and does then resume normal BIOS booting (which will then initialize important things like A0h/B0h/C0h tables, and will then break when starting the GUI code at 80030000h).
XPLORER searches opcode 24040385h at BFC06000h and up, and does then place a COP0 opcode fetch breakpoint at the opcode address+10h (note: this is within a branch delay slot, which makes COP0 emulation twice as complicated). XPLORER does also require space in unused BIOS RAM addresses (eg. Xplorer v3.20: addr 7880h at 1F002280h, addr 017Fh at 1F006A58h).

Note

Most games include two or three patches. The only game that I've seen so far that does NOT use any patches is Wipeout 2097.

Arcade Cabinets

PSX-Based Arcade Boards
```
Namco System 11, System 12 (and System 10?)
Capcom/Sony ZN-1, ZN-2
Konami GV, Konami GQ
Taito FX-1A, Taito FX-1B
Atlus PSX, PS Arcade 95, Tecmo TPS
```

CPU
Same as in PSX. Except, one board is said to be having the CPU clocked at 48MHz, instead of 33MHz???

GPU
Same as in PSX. Except, most or all boards are said to have 2MB VRAM instead of 1MB. Unknown how the extra VRAM is accessed... maybe as Y=512..1023... (though the PSX VRAM transfer commands are limited to 9bit Ysiz values, but maybe Y coordinates can be 10bit wide).

ROM vs CDROM
Arcade games are stored on ROM (or FLASH) instead of using CDROM drives.

SPU
Most PSX-based arcade boards are having the PSX-SPU replaced by a different sound chip (with each arcade manufacturer using their own custom sound chip, often controlled by a separate sound CPU).

Controls
Arcade boards are typically having digital joysticks instead of joypads, with differently named buttons (instead of /\,[],(),><), probably accessed via custom I/O ports (instead of serial transmission)? Plus additional coin inputs and DIP switches.

Cheat Devices

Cheat Devices - Datel Code Format
Cheat Devices - Xplorer Code Format
Cheat Devices - Xplorer Cheat Code and ROM-Image Decryption
Cheat Devices - Datel I/O
Cheat Devices - Xplorer I/O

Two families
Differences
Clones/hacks
Comm Link
Libraries

http://gamehacking.org/faqs/hackv500c.html - cheat code formats
http://doc.kodewerx.org/hacking_psx.html - cheat code formats
http://xianaix.net/museum.htm - around 64 bios versions
http://www.murraymoffatt.com/playstation-xplorer.html - xplorer bioses

Separating between Gameshark and Xplorer Codes
```
First Digit  Usage
3,8          Same for Gameshark & Xplorer (for Xplorer: can be encrypted)
1,2,C,D,E    Gameshark
4,6,7,9,B,F  Xplorer
0,5          Meaning differs for Gameshark & Xplorer
A            Unused
```

The Gold Finger is compatible with Game Shark, Pro Action Replay, and Gamebuster codes.

GameShark
Codebreaker
Megacom Power Replay III Game Enhancer

Cheat Devices - Datel Code Format

PSX Gameshark Code Format
```
30xxxxxx 00dd   ;-8bit Write  [aaaaaa]=dd
80aaaaaa dddd   ;-16bit Write [aaaaaa]=dddd
```
Below for v2.2 and up only
```
D0aaaaaa dddd   ;-16bit/Equal     If dddd=[aaaaaa] then (exec next code)
D1aaaaaa dddd   ;-16bit/NotEqual  If dddd<>[aaaaaa] then (exec next code)
D2aaaaaa dddd   ;-16bit/Less      If dddd<[aaaaaa] then (exec next code)
D3aaaaaa dddd   ;-16bit/Greater   If dddd>[aaaaaa] then (exec next code)
E0aaaaaa 00dd   ;-8bit/Equal      If dd=[aaaaaa] then (exec next code)
E1aaaaaa 00dd   ;-8bit/NotEqual   If dd<>[aaaaaa] then (exec next code)
E2aaaaaa 00dd   ;-8bit/Less       If dd<[aaaaaa] then (exec next code)
E3aaaaaa 00dd   ;-8bit/Greater    If dd>[aaaaaa] then (exec next code)
10aaaaaa dddd   ;-16bit Increment [aaaaaa]=[aaaaaa]+dddd
11aaaaaa dddd   ;-16bit Decrement [aaaaaa]=[aaaaaa]-dddd
20aaaaaa 00dd   ;-8bit Increment  [aaaaaa]=[aaaaaa]+dd
21aaaaaa 00dd   ;-8bit Decrement  [aaaaaa]=[aaaaaa]-dd
```
Below for v2.41 and up only
```
D4000000 dddd   ;-Buttons/If  If dddd=JoypadButtons then (exec next code)
D5000000 dddd   ;-Buttons/On  If dddd=JoypadButtons then (turn on all codes)
D6000000 dddd   ;-Buttons/Off If dddd=JoypadButtons then (turn off all codes)
C0aaaaaa dddd   ;-If/On       If dddd=[aaaaaa] (turn on all codes)
```
Below probably v2.41, too (though other doc claims for v2.2)
```
5000nnbb dddd   ;\Slide Code aka Patch Code aka Serial Repeater
aaaaaaaa ??ee   ;/for i=0 to nn-1, [aaaaaaaa+(i*bb)]=dddd+(i*??ee), next i
00000000 0000   ;-Dummy (do nothing?) needed between slides (CD version only)
```

```
C1000000 nnnn   ;-Delays activation of codes by nnnn (4000-5000 = 20-30 sec)
C2ssssss nnnn   ;\Copy ssss bytes from 80ssssss to 80tttttt
80tttttt 0000   ;/
```

Below from Caetla .341 release notes

These are probably caetla-specific, not official Datel-codes. In fact, Caetla .341 itself might be an inofficial hacked version of Caetla .34 (?) so below might be totally inofficial stuff:

```
C3aaaaaa 0000     ;\Indirect 8bit Write  [[aaaaaa]+bbbb]=dd
9100bbbb 000000dd ;/
C3aaaaaa 0001     ;\Indirect 16bit Write [[aaaaaa]+bbbb]=dddd (Tomb Raider 2)
9100bbbb 0000dddd ;/
C3aaaaaa 0002     ;\Indirect 32bit Write [[aaaaaa]+bbbb]=dddddddd
9100bbbb dddddddd ;/
FFFFFFFF 0001     ;-Optional prefix for GameShark 2.2 codes(force non-caetla)
12aaaaaa dddddddd ;-32bit Increment [aaaaaa]=[aaaaaa]+dddddddd
22aaaaaa dddddddd ;-32bit Decrement [aaaaaa]=[aaaaaa]-dddddddd
```

Notes

A maximum of 30 increment/decrement codes can be used at a time.
A maximum of 60 conditionals can be used at a time (this includes Cx codes).
Increment/decrement codes should (must?) be used with conditionals.
Unknown if greater/less conditionals are signed or unsigned.
Unclear if greater/less compare dddd by [aaaaaa], or vice-versa.
Unknown if 16bit codes do require memory alignment.

Cheat Devices - Xplorer Code Format

PSX Xplorer/Xploder Code Format

```
3taaaaaa 00dd  ;-8bit write  [aaaaaa]=dd
8taaaaaa dddd  ;-16bit write [aaaaaa]=dddd
00aaaaaa dddd  ;-32bit write [aaaaaa]=0000dddd  <-- not "0taaaaaa dddd" ?
4t000000 000x  ;-Slow Motion (delay "x" whatever/ns,us,ms,frames?)
7taaaaaa dddd  ;-IF [aaaaaa]=dddd then <execute following code>
9taaaaaa dddd  ;-IF [aaaaaa]<>dddd then <execute following code>
Ftaaaaaa dddd  ;-IF [aaaaaa]=dddd then activate "other selected" codes (uh?)
5taaaaaa ?nnn  ;\
d0d1d2d3 d4d5  ; write "?nnn" bytes to [aaaaaa]  ;ordered d0,d1,d2... ?
d6d7d8.. ....  ;/
6t000000 nnnn  ;\COP0 hardware breakpoint
aaaaaaaa cccc  ; aaaaaaaa=break_address, mmmmmmmm=break_mask
mmmmmmmm d0d1  ; nnnn=num_bytes (d0,d1,d2,etc.), cccc=break_type (see below)
d2d3d4.. ....  ;/
B?nnbbbb eeee  ;\Slide/Patch Code, with unclear end: "end=?nn+/-1" ?
10aaaaaa dddd  ;/for i=0 to end, [aaaaaa+(i*bbbb)]=dddd+(i*eeee), next i
C0aaaaaa dddd  ;-garbage/mirror of 70aaaaaa dddd ?  ;\or maybe meant to be
D0aaaaaa dddd  ;-garbage/mirror of 70aaaaaa dddd ?  ;/same as on GameShark?
```

The second code digit (t) contains encryption type (bit0-2), and a "default on/off" flag (bit3: 0=on, 1=off; whatever that means, it does probably require WHATEVER actions to enable codes that are "off"; maybe via the Ftaaaaaa dddd code).

break_type (cccc) (aka MSBs of cop0r7 DCIC register)

```
E180 (instruction gotton by CPU but not yet implemented) (uh, gotton what?)
EE80 (data to be read or written)  ;<--looks okay
E680 (data to be read)             ;<--looks okay
EA80 (data to be wrtten)           ;<--looks okay
EF80 (instruction)   ;<-- looks crap, should be probably E180
```

The CPU supports one data breakpoint and one instruction breakpoint (though unknown if the Xplorer does support to use both simultaneously, or if it does allow only one of them to be used).
If the break_type/address/mask to match up with CPU's memory access actions... then "something" does probably happen (maybe executing a sub-function that consists of the d0,d1,d2,etc-bytes, if so, maybe at a fixed/unknown memory address, or maybe at some random address; which would require relocatable code).

Notes

The "Slide" code shall be used only with even addresses, unknown if other 16bit/32bit codes do also require aligned addresses.

Cheat Devices - Xplorer Cheat Code and ROM-Image Decryption

decrypt_xplorer_cheat_code:

```
key  = x[0] and 07h              ;'''''''' AABBCCDD EEFF ''''''''
x[0] = x[0] xor key              ;         / / /  \  \ \         ;
if key=0                         ; x[0] --' / /    \  \ '-- x[5] ;
  ;unencrypted (keep as is)      ; x[1] ---' /      \  '--- x[4] ;
elseif key=4                     ; x[2] ----'        '----- x[3] ;
  x[1] = x[1] xor (025h)         ;,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,;
  x[2] = x[2] xor (0FAh + (x[1] and 11h))
  x[3] = x[3] xor (0C0h + (x[2] and 11h) + (x[1] xor 12h))
  x[4] = x[4] xor (07Eh + (x[3] and 11h) + (x[2] xor 12h) + x[1])
  x[5] = x[5] xor (026h + (x[4] and 11h) + (x[3] xor 12h) + x[2] + x[1])
elseif key=5
  x[1] = (x[1] + 057h)  ;"W"ayne
  x[2] = (x[2] + 042h)  ;"B"eckett
  x[3] = (x[3] + 031h)  ;"1"
  x[4] = (x[4] + 032h)  ;"2"
  x[5] = (x[5] + 033h)  ;"3"
elseif key=6
  x[1] = (x[1] + 0ABh) xor 01h
  x[2] = (x[2] + 0ABh) xor 02h
  x[3] = (x[3] + 0ABh) xor 03h
  x[4] = (x[4] + 0ABh) xor 04h
  x[5] = (x[5] + 0ABh) xor 05h
elseif key=7
  x[5] = x[5] + 0CBh
  x[4] = x[4] + 0CBh + (x[5] and 73h)
  x[3] = x[3] + 05Ah + (x[4] and 73h) - (x[5] xor 90h)
  x[2] = x[2] + 016h + (x[3] and 73h) - (x[4] xor 90h) + x[5]
  x[1] = x[1] + 0F5h + (x[2] and 73h) - (x[3] xor 90h) + x[4] + x[5]
```

```
    error ;(key=1,2,3)
  endif
```

decrypt_xplorer_fcd_rom_image:

```
  for i=0 to romsize-1
    x=45h
    y=(i and 37h) xor 2Ch
    if (i and 001h)=001h then x=x xor 01h
    if (i and 002h)=002h then x=x xor 01h
    if (i and 004h)=004h then x=x xor 06h
    if (i and 008h)=008h then x=x xor 04h
    if (i and 010h)=010h then x=x xor 18h
    if (i and 020h)=020h then x=x xor 30h
    if (i and 040h)=040h then x=x xor 60h
    if (i and 080h)=080h then x=x xor 40h
    if (i and 100h)=100h then x=x xor 80h
    if (i and 006h)=006h then x=x xor 0ch
    if (i and 00Eh)=00Eh then x=x xor 08h
    if (i and 01Fh)>016h then x=x-10h
    rom[i]=(rom[i] XOR x)+y
  next i
```

Cheat Devices - Datel I/O

Datel Memory and I/O Map

```
  1F000000h-1F01FFFFh R/W Flash (first 128K)
  1F020010h           R   Comms Link STB pin state (bit0)
  1F020018h           R   Switch Setting (bit0: 0=Off, 1=On)
  1F040000h-1F05FFFFh R/W Flash (second 128K) + feedback area (see below)
  1F060000h           R   Comms Link data in (byte)
  1F060008h           W   Comms Link data out (byte, pulses ACK to Comms Link)
```

Cheat Devices - Xplorer I/O

Xplorer Memory and I/O Map

```
  1F000000h.RW Xplorer Firmware (256K, SST 29EE020) (or 384K? in Xplorer FX)
  1F060000h.R  Xplorer I/O - Switch Setting (bit0: 0=Off, 1=On)
  1F060001h.R  Xplorer I/O - 8bit Data from PC (bit0-7)
  1F060001h.W  Xplorer I/O - 3bit Data to PC (bit0-2)
  1F060001h.W  Xplorer I/O - Handshake to PC (bit3)
  1F060001h.W  Xplorer I/O - Unknown/to PAL (bit4-5)
  1F060001h.W  Xplorer I/O - Unused (bit6-7)
  1F060002h.R  Xplorer I/O - Handshake from PC (bit0)
  1F060006h.R  Xplorer I/O - Unknown (used by Xplorer v4.52)
  1F060007h.R  Xplorer I/O - Unknown (used by Xplorer v4.52)
```

PSX outputs data to PC 3 bits at a time on the following lines slct, paper end, busy and uses /ack for handshake YOU MUST INVERT BUSY WHEN READING FROM PC, ALSO PRINTER SELECT FROM PC IS INVERTED.

To send a byte to psx

```
  out data to psx
  set printer select line high
  wait for ack from psx to go high
  set printer select line low
  wait for ack from psx to go low
```

To get a byte from psx

```
  set printer select line high
  psx sets ack high and puts d6 on slct, d7 on pe
  read data
  set printer select line low
  psx sets ack low and puts d5 on busy, d4 on pe, d3 on slct
  read data
  set printer select line high
  psx sets ack high and puts d2 on busy, d1 on pe, d0 on slct
  read data
  set printer select line low
  psx sets ack low and puts low on busy, low on pe, high on slct
  read data
  set printer select line high
  reassemble byte
```

READ DATA FROM PSX

```
  DATA FLOW              STEP1   STEP2   STEP3   STEP4
  from PC: PRINTER SEL   HIGH    LOW     HIGH    LOW
  from PSX: SLCT         D6      D3      D0      HIGH
  from PSX: PE           D7      D4      D1      LOW
  from PSX: /BUSY        HIGH    D5      D2      LOW
  from PSX: ACK          HIGH    LOW     HIGH    LOW
```

XPLODER COMMANDS:

not all commands here yet.

send bin file to psx

```
  send 0x57
  send 0x53 updload bin file
  send address 4 bytes
  send file size 3 bytes
  send data
  send checksum 2 bytes
```

Jump to address

```
  send 0x57
  send 0x0f Jump to address
  send address 4 bytes
```

PSX Dev-Board Chipsets

Sony DTL-H2000 CPU Board

```
CL825 20pin pin test points (2x10 pins)
CL827 20pin pin test points (2x10 pins)
U83   64pin SEC KM4216V256G-60 (DRAM 256Kx16) ;VRAM
U84   64pin SEC KM4216V256G-60 (DRAM 256Kx16) ;VRAM
CL828 20pin pin test points (2x10 pins)
CL826 20pin pin test points (2x10 pins)
X10    4pin JC53.20 (PAL, 53.203425MHz)
X2    ? pin ?? (NTSC?, 5?.???MHz)
U62   20pin logic?
U27   64pin Sony CXD2923AR    ;GPU'b
CL813 20pin pin test points (2x10 pins)
CL814 20pin pin test points (2x10 pins)
U16  160pin Sony CXD8514Q     ;GPU'a
X?    ? pin 67.73760 MHz
CL807 20pin pin test points (2x10 pins)
CL809 20pin pin test points (2x10 pins)
CL?4? 20pin pin test points (2x10 pins)
U801 208pin Sony CXD8530BQ    ;CPU
U11   28pin SEC KM48V2104AJ-6 (DRAM 2Mx8) ;Main RAM
U10   28pin SEC KM48V2104AJ-6 (DRAM 2Mx8) ;Main RAM
U9    28pin SEC KM48V2104AJ-6 (DRAM 2Mx8) ;Main RAM
U8    28pin SEC KM48V2104AJ-6 (DRAM 2Mx8) ;Main RAM
?    100pin Blue connector (to other ISA board)
U66   48pin ?
U65   48pin ?
U34   48pin ?
U?   100pin Sony CXD2922Q     ;SPU
U63   14pin logic?
U32   44pin SEC KM416V256B1-8 (DRAM 256Kx16)  ;SoundRAM
CL801 20pin pin test points (2x10 pins)
CL802 20pin pin test points (2x10 pins)
?      3pin voltage stuff?
U31   20pin logic? 74ACT2xx?
U35   18pin OKI M6538-01 (aka MSM6538-01?) (audio related?)
U36   20pin Sanyo LC78815  ;16bit D/A Converter
U36    8pin ?
J806   8pin solder pads...
J805   9pin solder pads...
J804  10pin solder pads... (11pins, with only 10 contacts?)
?     48pin solder pads (12x4pin config jumpers or so)
U26   20pin logic?
U?    24pin Sony CXA1xxxx?  ;RGB?
?      9pin PAL/NTSC Jumpers (three 3pin jumpers)
J801  24pin solder pads...
J803   9pin rear connector: Serial Port (3.3V) (aka "J308") (DB9) (5+4pin)
J802  15pin rear connector: AV Multi-out   (5+5+5pin)
CN881 98pin ISA Bus Cart-edge (2x31 basic pins, plus 2x18 extended pins)
```

Sony DTL-H2000 PIO Board

```
?     68pin Black connector (maybe equivalent to 68pin PSX expansion port?)
?      5pin solder pads...
U371? 40pin HN27C4000G-12 (512Kx8 / 256Kx16 EPROM) (sticker: "94/7/27")
U370  84pin Altera EPM7160ELC84-12 (sticker: "U730, cntl 1")
U3    14pin logic?
U43   44pin Altera EPM7032?LC44-10 (sticker: "U43, add 1" ?)
U716  28pin Sharp LH5498D-35 (FIFO 2Kx9)
U717  28pin Sharp LH5498D-35 (FIFO 2Kx9)
U7?   28pin Sharp LH5498D-35 (FIFO 2Kx9)
U719  28pin Sharp LH5498D-35 (FIFO 2Kx9)
U724  20pin logic?
U?    20pin logic?
U?    20pin 74FCT244Axx ? (dual 4-bit 3-state noninverting buffer/line driver)
U732  48pin ...?
U711? 20pin logic?
U712  20pin logic?
U713  20pin 74HC244AP(dual 4-bit 3-state noninverting buffer/line driver)
U714  20pin 74HC244AP(dual 4-bit 3-state noninverting buffer/line driver)
U721  20pin logic?
U55   14pin logic?
U726  20pin logic?
U715  20pin 74HC244AP(dual 4-bit 3-state noninverting buffer/line driver)
?    100pin Blue connector (to other ISA board)
U738  20pin logic? (SMD)
U734  32pin KM684000G-7 (SRAM 512Kx8)         ;\maybe 1Mbyte EXP3 RAM ?
U733  32pin KM684000G-7 (SRAM 512Kx8)         ;/
U721  20pin logic? (U721 or U710 or U725 ?)
S700  24pin 12bit DIP switch  (select I/O Address bits A15..A4)
JP700  8pin Jumper (4x2 pins) (select IRQ15/IRQ12/IRQ11/IRQ10)
JP7xx 12pin Jumper (3x4 pins) (select DMA7/DMA6/DMA5)
U64   48pin ...?
U65   48pin ...?
U66   48pin ...?
U737  48pin ...?
U710? 20pin logic?
U709  20pin logic?
U?    14pin logic?
U2    14pin logic?
U1     8pin Dallas DS1232 (MicroMonitor Chip) ;power-good-detect ?
U708  20pin logic?
X?     ?pin 4.1900
U42   80pin P823, U01Q (Sony CXP82300 SPC700 CPU with piggyback EPROM socket)
U42'  32pin 27C256A-15 (EPROM 32Kx8) (sticker: "94/11/28")
U706   ?pin slim chip with 1xN pins?
BT700  2pin battery (!) (not installed)
U729?  5pin voltage stuff?
U40    8pin Dallas Dxxxxx (maybe RTC ???)
X4     2pin small crystal (maybe 32.768kHz?)
```

```
U736  28pin Sony CXK58257ASP-70L (SRAM 32Kx8)         ;CDROM Sector Buffer?
U735 100pin Sony CXD1199BQ    ;CDROM Decoder/FIFO
?     40pin Blue connector... to external DTL-H2010 CDROM drive?
?      9pin rear connector: Joypad/Memcard 2 (DB9)
?      9pin rear connector: Joypad/Memcard 1 (DB9)
?      -    rear hole for cable to Blue 40pin connector?
J70x  98pin ISA Bus Cart-edge (2x31 basic pins, plus 2x18 extended pins)
```

Sony DTL-H2500 Dev board (PCI bus)
Newer revision of the DTL-H2000 board. Consists of a single PCI card (plus tiny daughterboard with Controller ports). Exact chipset is unknown (there are components on both sides of the PCI card, but no high resolution photos/scans exist).

Sony DTL-H2700 Dev board (ISA bus) (CPU, ANALYZER ...?)
Another revision of the DTL-H2000 board. Consists of a single ISA card stacked together with a huge daughterboard. Exact chipset is unknown (there might be components on both sides of the PCBs, and half of the components aren't visible due to the stacked PCBs (ie. taking pictures/scans of the PCBs would require using advanced techniques with screwdrivers).
One uncommon feature is an extra connector for a "trigger switch" or "foot pedal" (with unknown purpose; maybe to start/stop CPU execution or so).

Sony DTL-H201A / DT-HV - Graphic Artist Board (IBM PC/ATs to NTSC video)

```
X2     xpin TXC-2 OSC 66.000MHz
X1     xpin TXC-2AOSC 53.693MHz
U16   14pin 74F74 (dual flipflop)
U29   14pin 74AS04 (hex inverters)
U14   20pin LVT244 (dual 4-bit 3-state noninverting buffer/line driver)
U18   20pin LVT244 (dual 4-bit 3-state noninverting buffer/line driver)
U15   20pin ACT244 (dual 4-bit 3-state noninverting buffer/line driver)
U11   84pin Altera EPM7096LC84-12 (sticker: "artpc13" or "ARTPC13")
U13  160pin Sony CXD8514Q     ;GPU'a
U5    14pin ALS38A   ? (quad open-collector NAND gates with buffered output)
U27   20pin ALS244AJ ? (dual 4bit tristate noninverting buffer/line driver)
Q1     3pin T B596
U23   64pin KM4216V256G-60 (DRAM 256Kx16) ;VRAM
U22   64pin KM4216V256G-60 (DRAM 256Kx16) ;VRAM
U28   64pin Sony CXD2923AR    ;GPU'b
S1    16pin 8bit DIP switch (select I/O address A15..A8)
S2     8pin 4bit DIP switch (select I/O address A7..A4)
U1    20pin SN74ALS688N (8bit inverting identity comparator with enable)
U2    20pin SN74ALS688N (8bit inverting identity comparator with enable)
U3    20pin ALS245A (8bit tristate noninverting bus transceiver)
JP9   12pin Jumper (6x2 pins) (select IRQ15/IRQ11/IRQ10/IRQ9/IRQ5/IRQ3)
U26   24pin Sony CXA1145M ?  ;RGB?
JP10   3pin Jumper   ;\
JP12   3pin Jumper   ; select "S" or "O" (?)
JP11   3pin Jumper   ;/
J3    2pin? Yellow connector (composite video out?)
J2?    pin? Mini DIN? connector (maybe S-video out?)
J1    15pin High Density SubD (maybe video multi out?)
CJx   98pin ISA Bus Cart-edge (2x31 basic pins, plus 2x18 extended pins)
```

DTL-S2020 aka Psy-Q CD Emu

```
Yellow PCB "CD Emulator System, (C) Cirtech & SN Systems Ldt, 1994 v1.2"
IC    24pin GAL20V8B
IC    68pin Analog Devices ADSP-2101 (16bit DSP Microprocessor)
IC    20pin HD74HC244P
IC15  20pin HD74HC244P
IC14  20pin CD74HCT245E
IC7   28pin 27C512-10     (EPROM 64Kx8) (yellow sticker, without text)
IC    28pin HY62256ALP-70 (SRAM 32Kx8)
IC12  28pin HY62256ALP-70 (SRAM 32Kx8)
IC    28pin HY62256ALP-70 (SRAM 32Kx8)
IC13  84pin Emulex/QLogic FAS216 (Fast Architecture SCSI Processor)
IC5   84pin Emulex/QLogic FAS216 (Fast Architecture SCSI Processor)
IC4   24pin GAL20V8B (near IO Addr jumpers)
IC    20pin 74LS244B1   (near lower 8bit of ISA databus)
IC    20pin SN74LS245N? (near lower 8bit of ISA databus)
IC    20pin SN74LS245N  (near upper 8bit of ISA databus)
DMA   12pin Jumpers (select DMA7/6/5)
IRQ   12pin Jumpers (select IRQ15/12/11/10/7/5)
IO    16pin Jumpers (select IO Addr 300/308/310/318/380/388/390/398)
SCSI   6pin Jumpers (select SCSI ID 4/2/1) (aka 3bit 0..7 ?)
PL3   34pin Connector to DTL-H2000 ?
PL1   50pin Connector to INTERNAL SCSI hardware ?
PL2  50pin? Connector to EXTERNAL SCSI hardware ? (25pin plug/50pin cable?)
Jx    98pin ISA Bus Cart-edge (2x31 basic pins, plus 2x18 extended pins)
```

Note: There's also a similar ISA cart (DTL-S510B) with less chips and less connectors.
Note: The SN Systems carts seem to have been distributed by Sony (with "DTL-Sxxxx" numbers), and also distributed by Psygnosis. The external SCSI connectors can be possibly also used with Psy-Q Development Systems for SNES and Sega Saturn?

PSY-Q Development System (Psygnosis 1994)

```
32pin GM76C8128ALLFW85 (SRAM 128Kx8)
44pin ALTERA EPM7032LC44-15T
34pin EMULEX FAS101 (SCSI Interface Processor)
28pin 27C64 (EPROM 8Kx8) (green sticker, without text)
20pin LCX245 (=74245?)
 8pin 2112, CPA, H9527 (?)
 3pin transistor?
20pin DIP socket (containing two 10pin resistor networks)
20pin DIP socket (containing two 10pin resistor networks)
 2pin CR2032 Battery 3V
68pin Connector to PSX "Parallel I/O" expansion port
25pin Connector to SCSI hardware (probably to DTL-S510B ISA cart?)
```

Sony DTL-H800 Sound Artist Board (with optical fibre audio out)

```
U15  24pin ?
U5   28pin 27C256 (EPROM 32Kx8) (not installed)
U7    4pin 67.7376MHz oscillator
U8   14pin ?
U11  44pin SEC KM416V256B1-8 (DRAM 256Kx16)  ;SoundRAM
             (44pin package with middle 4pin missing, 40pins used)
```

```
U4  160pin Lattice IspLSI 3256 (sticker: "VER3")
U6  128pin Lattice IspLSI xxxx ?
U12  48pin ?
U13  48pin ?
U3   20pin 74ACT244
U14   5pin "LM25755, -3.3 P+" ?
U2   54pin ?
U1   54pin ?
U9    ?pin GP1F31T (light transmitting unit for optical fibre cable)
?   124pin PCI bus cart edge connector
?     8pin internal jumper/connector? (7pin installed, 1pin empty)
```

Note: There's also a similar board (DTL-H700) for MAC/NuBus instead of PCI bus.

Sony COH-2000 (unknown purpose)

```
U1   14pin SN74ALS388N  ?
U2   20pin SN74ALS688N (8bit inverting identity comparator with enable)
U3   20pin SN74ALS688N (8bit inverting identity comparator with enable)
U4   24pin PALxxx ?
U5   20pin SN74ALS245AN
U6   20pin SN74ALS245AN
U7   20pin SN74ALS244N
U8   20pin SN74ALS244N
U9   20pin SN74ALS245AN
U10  20pin SN74ALS245AN
U11  20pin SN74ALS244N
S2   16pin 8bit DIP switch (ISA 15/14/13/12/11/10/9/8) ;I/O address bit15-8
S1    8pin 4bit DIP switch (ISA 7/6/5/4)               ;I/O address bit7-4
S3    8pin 4bit DIP switch (BISO? 3/2/1/0)         ;BISO? or BISD? or 8150?
JPxx  .... several jumpers (unknown purpose)
Jx    98pin ISA Bus Cart-edge (2x31 basic pins, plus 2x18 extended pins)
J5    68pin Connector on rear side (unknown purpose)
```

Unknown what COH-2000 was used for. One theory was that it's related to PSX-based arcade cabinets. The 68pin connector might be also related to the 68pin PSX "Parallel I/O" expansion port.

Sony DTL-H2010 (Black External CDROM Drive for DTL-H2000, CD-R compatible)

Some sort of external front loading CDROM drive with Eject button. Components are unknown. Probably includes CXD2510Q and CXA1782BR chips and PSX drive unit. Connects to the blue 40pin connector on DTL-H2000 boards.

Sony DTL-H2510 (Gray Internal CDROM Drive)

```
IC309 80pin Sony CXD2510Q (CDROM Signal Processor)
ICxx   ?pin Unknown if there are further ICs (eg. CXA1782BR should exist?)
CN1   10pin Connector to daughterboard (with drive unit)
CN2    4pin Connector to PC power supply (12V/5V and 2xGND)
CN3   50pin Connector to DTL-H2500 or so? (need "PCS-E50FC" plug?)
```

This is some sort of a mimmicked front loading PC CDROM drive. The thing doesn't simply eject the disc - it does also eject the whole PSX drive unit (including sled/spin motors and laser optics). There is no eject button. Unknown if there's some eject motor, or if one does need to push/pull drive tray manually.

Sony SCPH-9903 (Gray SCEx-free Playstation)

A rare SCEx-free Playstation that can boot from CDR's without SCEx strings; maybe intended for beta-testers. Marked "Property of Sony Computer Entertainment", "U/C".

Hardware Numbers

Sony's own hardware (for PSX) (can be also used with PSone)

```
SCPH-1000 PlayStation (1994) (NTSC-J)   (with S-Video)
SCPH-1001 PlayStation (1995) (NTSC-U/C) (without S-Video)
SCPH-1002 PlayStation (199x) (PAL)      (without S-Video)
SCPH-1010 Digital joypad (with short cable) (1994)
SCPH-1020 Memory Card 1Mbits (1994)
SCPH-1030 2-button Mouse (with short cable) (1994)
SCPH-1040 Serial Link Cable
SCPH-1050 RGB Cable (21-pin RGB Connector)
SCPH-1060 RFU Cable/Adaptor (antennae connector) (NTSC-JP?) (1995)
SCPH-1061 RFU Cable/Adaptor (antennae connector) (NTSC-US?)
SCPH-1062 RFU Cable/Adaptor (antennae connector) (PAL)
SCPH-1070 Multitap adaptor (four controllers/memory cards on one slot) (1995)
SCPH-1080 Digital joypad (with longer cable) (1996)
SCPH-1090 2-button Mouse (with longer cable) (1998)
SCPH-1100 S Video Cable (1995)
SCPH-1110 Analog Joystick (1996)
SCPH-1120 RFU Adaptor (antennae connector) (NTSC-JP?) (1996)
SCPH-1121 RFU Adaptor (antennae connector) (NTSC-US?)
SCPH-1122 RFU Adaptor (antennae connector) (PAL)
SCPH-1130 AC Power Cord (1996)
SCPH-1140 AV Cable (1997)
SCPH-1150 Analog Joypad (with one vibration motor, with red/green led) (1997)
SCPH-1160 AV Adaptor (1997)
SCPH-1170 Memory Card Triple Pack (three Memory Cards) (1996)
SCPH-1180 Analog Joypad (without vibration motors, with red/green led)
SCPH-119X Memory Card (X=different colors) (1997)
SCPH-1200 Analog Joypad (with two vibration motors) (dualshock) (1997)
SCPH-1210 Memory Card Case (1998)
SCPH-2000 Prototype Keyboard/Mouse adapter (PS/2 to PSX controller port)
SCPH-3000 PlayStation (1995) (NTSC-J) (with the S-video output removed)
SCPH-3500 PlayStation Fighting Box (console bundled with 2 controllers)(1996)
SCPH-4000 PocketStation (Memory Card with LCD-screen) (1999)
SCPH-4010 VPick (guitar-pick controller) (for Quest for Fame, Stolen Song)
SCPH-4020 Long Strap for PocketStation (1999)
SCPH-4030 Wrist Strap for PocketStation (1999)
SCPH-5000 PlayStation (cost reduced) (1996)
SCPH-5003 PlayStation "Asian revision"
SCPH-5500 PlayStation without Cinch sockets (ie. AV Multi Out only) (1996)(J)
SCPH-5501 "" North American version of the 5500
SCPH-5502 "" European version of the 5500
SCPH-5552 "" European revision
SCPH-5903 PlayStation with built-in MPEG Video-CD decoder (Asia-only)
SCPH-7000 PlayStation with Dualshock (1997) (Japan)
```

```
SCPH-7002 PlayStation with Dualshock (199x) (Europe)
SCPH-7003 PlayStation with Dualshock (199x) (Asia)
SCPH-7500 PlayStation with Dualshock, cost reduced (1999) (Japan)
SCPH-7501 PlayStation with Dualshock, cost reduced (199x) (North America)
SCPH-7502 PlayStation with Dualshock, cost reduced (199x) (Europe)
SCPH-7503 PlayStation with Dualshock, cost reduced (199x) (Asia)
SCPH-9000 PlayStation without Parallel I/O port (1999) (Japan)
SCPH-9001 PlayStation without Parallel I/O port (199x) (North America)
SCPH-9002 PlayStation without Parallel I/O port (199x) (Europe)
SCPH-9003 PlayStation without Parallel I/O port (199x) (Asia)
SCPH-9903 Rare SCEx-free PSX (Property of Sony Computer Entertainment, U/C)
```

Sony's own hardware (for PSone)

```
SCPH-100 PSone (miniaturized PlayStation) (2000) (Japan)
SCPH-101 PSone (miniaturized PlayStation) (200x) (North America)
SCPH-102 PSone (miniaturized PlayStation) (200x) (Europe)
SCPH-103 PSone (miniaturized PlayStation) (200x) (Asia)
SCPH-110 Dual Analog Pad (for PSone) (Dualshock) (2000)
SCPH-111 Multitap for PSone (seems to be quite rare, except in brazil)
SCPH-112 AC adapter for PSone (In: 110-220VAC,  Out: 7.5VDC, 2.0A)
SCPH-113 AC adapter for PSone (In: 120VAC/60Hz, Out: 7.5VDC, 2.0A)
SCPH-114 AC adapter for PSone (In: 220-240VAC,  Out: 7.5VDC, 2.0A) (Europe)
SCPH-115 AC adapter for PSone (In: 220-240VAC,  Out: 7.5VDC, 2.0A)
SCPH-116 AC adapter for PSone (australia?)
SCPH-117 AC adapter for PSone (In: 110VAC,      Out: 7.5VDC, 2.0A) (Asia?)
SCPH-120 AC adapter for PSone with LCD Screen (In: 100VAC, Out: 7.5VDC, 3.0A)
SCPH-130 LCD Screen for PSone (to be attached to the console) (2001)
SCPH-140 PSone and LCD screen combo (2001)
SCPH-152 LCD screen for PSone (PAL SCPH-152C)
SCPH-162 PSone and LCD screen (PAL SCPH-162C)
SCPH-170 Car Adapter for PSone from car cigarette lighter (2001)
SCPH-180 AV Connection Cable for LCD-screen's AV IN
SCPH-10180K DoCoMo I-Mode Adaptor Cable (for internet via mobile phones)
```

Sony's own devkits

```
DTL-H201A Graphic Artist Board (ISA bus) (with NTSC video out)
DTL-H240  PS-X RGB Cable
DTL-H505  PS-X (Code Name) Target Box ?
DTL-H700  Sound Artist Board (NuBus for Mac)
DTL-H800  Sound Artist Board (PCI Bus for IBM) (with optical fibre sound out)
DTL-H1000 Debugging Station (CD-R compatible PSX console) (Japan)
DTL-H1001 Debugging Station (CD-R compatible PSX console) (North America)
DTL-H1002 Debugging Station (CD-R compatible PSX console) (Europe)
DTL-H1030 Mouse ?
DTL-H1040 Link Cable ?
DTL-H1050 RGB Cable ?
DTL-H110x Debugging Station revision? (DC-powered)
DTL-H120x Debugging Station revision? (AC-powered)
DTL-H1500 Stand-Alone Box ?
DTL-H2000 Dev board v1 (PSX on two ISA carts) (old pre-retail)
DTL-H2010 Black External CDROM Drive for DTL-H2000 (CD-R compatible)
DTL-H2040 Memory Box ?
DTL-H2050 Adaptor for Controller port ?
DTL-H2060 Serial Link cable
DTL-H2070 RGB Cable ?
DTL-H2080 Controller Box (joypad/memcard adaptor for DTL-H2000/DTL-xxxx?)
DTL-H2500 Dev board (PCI bus)
DTL-H2510 Gray Internal CDROM Drive for DTL-H2500/DTL-H2700 (CD-R compatible)
DTL-H2700 Dev board (ISA bus) (CPU, ANALYZER ...?)
DTL-H3000 Net Yaroze (hobby programmer dev kit) (Japan)
DTL-H3001 Net Yaroze (hobby programmer dev kit) (North America)
DTL-H3002 Net Yaroze (hobby programmer dev kit) (Europe)
DTL-H3050 Communication Cable (for yaroze?)
DTL-D2020 Documentation: BUILD CD (Manual of Programmer's Tool)
DTL-D2120 Documentation: (Manual of Programmer's Tool)
DTL-D2130 Documentation: PsyQ (Manual of Programmer's Tool)
DTL-D2130 Documentation: SdevTC (Manual of Programmer's Tool)
DTL-D2140A Documentation: Ver.1.0 (Manual of Programmer's Tool)
DTL-D2150A Documentation: Ver.2.0 (Manual of Programmer's Tool)
```

SN System / Psy-Q devkit add-ons / SCSI cards

```
DTL-S510B Unknown (another CDROM emulator version?)
DTL-S2020 CD-ROM EMULATOR for DTL-H2000/DTL-H2500/DTL-H2700
```

Sony Licensed Hardware (Japan)

```
SLPH-00001 Namco NegCon, Twist controller (SLEH-0003)
SLPH-00002 Hori Fighting stick, digital stick with autofire/slowmotion/rumble
SLPH-00003 ASCII Fighter stick V, psx-shaped digital stick (SLEH-0002)
SLPH-00004 Sunsoft Sunstation pad, digital pad with autofire/slowmotion
SLPH-00005 ASCII ASCIIPAD V, digital pad with autofire/slowmotion
SLPH-00006 Imagineer Sandapaddo ThunderPad
SLPH-00007 SANKYO N.ASUKA aka Nasca Pachinco Handle, bizarre paddle
SLPH-00008 Spital SANGYO Programmable joystick
SLPH-00009 Hori Fighting commander 2way controller
SLPH-00010 Super Pro Commander
SLPH-00011 Extended memo repack memory
SLPH-00012 Hori Fighting Commander 10B Pad, digital pad with extras
SLPH-00013
SLPH-00014 Konami Hyper blaster, IRQ10-based Lightgun (SLEH-0005/SLUH-00017)
SLPH-00015 Namco Volume controller, paddle with 2 buttons
SLPH-00016 [chiyo] clean! peripheral equipment
SLPH-00017 Hori Fighting Commander
SLPH-00018 Namco Arcade Stick, digital stick, small L1/L2 buttons (SLEH-0004)
SLPH-00019 Konami Hyperstick
SLPH-00020
SLPH-00021 Imagegun
SLPH-00022 Optec A.I. Commander Pro, digital pad with extras / lcd display
SLPH-00023 Namco Joystick
SLPH-00024 Optec Cockpit Wheel, analog joystick/analog pedals or so
SLPH-00025 Extended memo repack ZERO2 version memory
SLPH-00026
```

SLPH-00028 Hori Grip (note: other Hori Grips are SLPH-00086..00088)
SLPH-00029 Hori Horipad (clear), digital pad
SLPH-00030 Hori Horipad (black), digital pad
SLPH-00031 Hori Horipad (gray), digital pad
SLPH-00032 Hori Horipad (white), digital pad
SLPH-00033 Hori Horipad (blue), digital pad
SLPH-00034 Namco G-CON 45, Cinch-based Lightgun (SLEH-0007/SLUH-00035)
SLPH-00035 Fighter stick V Jr.
SLPH-00036 Optec Wireless Dual Shot, digital pad with turbo button
SLPH-00037
SLPH-00038 ASCII pad V Jr., digital pad without any extras
SLPH-00039 ASCII pad V2 (gray), digital pad with turbo switches
SLPH-00040
SLPH-00041
SLPH-00042 ASCII Grip V plus (Derby Stallion'99 supplement set), single-hand
SLPH-00043 ASCII pad V2 (pink)
SLPH-00044 ASCII pad V2 (white)
SLPH-00045 ASCII pad V2 (?)
SLPH-00046 ASCII pad V2 (green)
SLPH-00047 ASCII pad V2 (black)
SLPH-00048 ASCII pad V2 (lead)
SLPH-00049 ASCII pad V2 (yellow)
SLPH-00050 ASCII pad V2 (orange)
SLPH-00051 Taito Streetcar GO! Controller 2 steering wheel tie toe strange
SLPH-00052 KOEI/ERGOSOFT EGWord 2.0 and Parallel Printer bundle
SLPH-00053
SLPH-00054 Hori Zerotech Steering Controller
SLPH-00055
SLPH-00056
SLPH-00057
SLPH-00058 ASCII pad V2 (gold)
SLPH-00059 ASCII pad V2 (silver)
SLPH-00060 ASCII Biohazard, digital pad with re-arranged buttons (SLEH-0011)
SLPH-00061 ASCII pad V2 (pearl white) (or ASCII ARCADE STICK V2 ?)(SLEH-0009)
SLPH-00062 ASCII pad V2 (pearl blue)
SLPH-00063 ASCII pad V2 (pearl pink)
SLPH-00064 ASCII pad V2 (pearl green)
SLPH-00065 ASCII pad V Pro, with lcd for button-combinations
SLPH-00066 ASCII Stick 3 Ultimate
SLPH-00067 ASCII pad V2 (purple metallic)
SLPH-00068 ASCII pad V2 (cancer/gun metallic)
SLPH-00069 Screw kelp rack controller
SLPH-00070 something supported by irem titles that do also use SLPH-00007
SLPH-00071 Hori Commandstick
SLPH-00072 Command pack controller
SLPH-00073 Wireless digital set (light/write gray)
SLPH-00074 Wireless digital set (black)
SLPH-00075 Wireless digital set (clear)
SLPH-00076 Wireless digital set (clear blue)
SLPH-00077 Wireless digital set (clear black)
SLPH-00078 Wireless digital shot (light/write gray)
SLPH-00079 Wireless digital shot (black)
SLPH-00080 Wireless digital shot (clear)
SLPH-00081 Wireless digital shot (clear blue)
SLPH-00082 Wireless digital shot (clear black)
SLPH-00083 ASCII stick justice controller
SLPH-00084
SLPH-00085 Compact joystick controller
SLPH-00086 Hori Grip Controller (blue)
SLPH-00087 Hori Grip Controller (yellow)
SLPH-00088 Hori Grip Controller (pink)
SLPH-00089
SLPH-00090 Hori Multi Analogue Pad (clear)
SLPH-00091
SLPH-00092 ASCII pad V2 (margin green)
SLPH-00093 ASCII pad V2 (margin blue)
SLPH-00094 ASCII pad V2 (margin pink)
SLPH-00095 ASCII pad V2 (margin orange)
SLPH-00096 High pass tear ring V controller
SLPH-00097
SLPH-00098 Pachinko slot controller
SLPH-00099 ASCII pad V2 (rainbow)
SLPH-00100 ASCII 'Hanging' Fishing Controller, controller for fishing games
SLPH-00101 Cockpit big shock
SLPH-00102 ASCII grip V (Mars story corresponding set)
SLPH-00103
SLPH-00104 Hori Horipad II
SLPH-00105
SLPH-00106
SLPH-00107 Hori Compact Joystick Meisai
SLPH-00108
SLPH-00109
SLPH-00110 ASCII pad V2 (marble)
SLPH-00111
SLPH-00112 ASCII pad V3
SLPH-00113 ASCII pad V3 with cable reel
SLPH-00114 ASCII pad V3 with (white)
SLPH-00115 ASCII pad V3 with (pink)
SLPH-00116 ASCII pad V3 with (blue)
SLPH-00117 ASCII pad V3 (blue) with V2 (pearl green)
SLPH-00118
SLPH-00119
SLPH-00120
SLPH-00121 Hori Analog Sindou Pad (clear)
SLPH-00122
SLPH-00123 Hori Analog Sindou Pad (red)
SLPH-00124
SLPH-00125
SLPH-00126 Namco Jogcon, digital pad, steering dial (SLEH-0020/SLUH-00059)
SLPH-00127
SLPH-00128 ASCII stick ZERO3
SLPH-00129 ASCII pad V2 (wood grain pitch)
SLPH-00130

```
SLPH-00132 ASCII pad V3 (blue)
SLPH-00123
SLPH-00134 ASCII pad V3 (blue) with cable reel
SLPH-00135 ASCII pad V3 (blue) with V2 silver
SLPH-00136 ASCII pad V3 with V2 (purple metallic)
SLPH-00137 ASCII pad V3 with V2 (gold)
SLPH-00138 ASCII pad V3 with Vpro.
SLPH-00139
SLPH-00140
SLPH-00141
```

And, maybe unlicensed (at least, they are without official SLPH number):

```
ASC-05158B ASCII Beatmania Junk (similar to SLEH-0021)
BANC-0001 Bandai Fishing Controller
BANC-0002 Bandai Kids Station
```

Sony Licensed Hardware (Europe)

```
SLEH-00001 Ascii Specialized Pad (similar to SLPH-00005: ASCII ASCIIPAD V)
SLEH-00002 Ascii Arcade Stick, psx-shaped digital stick (SLPH-00003)
SLEH-00003 Namco Negcon, Twist controller (SLPH-00001)
SLEH-00004 Namco Arcade Stick (SLPH-00018)
SLEH-00005 Konami Hyper blaster, IRQ10-based Lightgun (SLPH-00014/SLUH-00017)
SLEH-00006 Mad Catz Steering Wheel (SLPH-?)
SLEH-00007 Namco G-Con 45, Cinch-based Lightgun (SLPH-00034/SLUH-00035)
SLEH-00008 ASCII Grip, single-handed digital pad (SLPH-00027/SLUH-00038)
SLEH-00009 Ascii Arcade Stick v2 (SLPH-00061... or is that a pad?)
SLEH-00010 Ascii Enhanced Control Pad (similar as SLEH-00001) (SLPH-00039)
SLEH-00011 Resident Evil Pad (aka SLPH-00060 ASCII Biohazard)
SLEH-00012 Reality-Quest The Glove (right-handed only) (SLUH-00045/SLPH-?)
SLEH-00013 CD Case (small nylon bag for fourteen CDs) (SLPH-?)
SLEH-00014 ?
SLEH-00015 PlayStation Case (bigger bag for the console) (SLPH-?)
SLEH-00016 PlayStation Case + Digital Joypad + Memory Card
SLEH-00017 ?
SLEH-00018 Ascii Sphere 360 (SLUH-00028/SLPH-?)
SLEH-00019 Interact V3 Racing Wheel (SLPH-?)
SLEH-00020 Namco JogCon, digital pad, steering dial (SLPH-00126/SLUH-00059)
SLEH-00021 Beatmania Controller (SLPH-?)
SLEH-00022 ?
SLEH-00023 Official Dance Mat (SLPH-?) (for PSone and PS2)
SLEH-00024 Fanatec Speedster 2 (wheel with pedals) (for PSone and PS2)
SLEH-00025 Mad Catz 8MB Memory Card (for PS2)
SLEH-00026 Olympus Eye-Trek FMD-20P Game/DVD glasses (for PS2)
SLEH-00027 ... Eye-Trek FMD-20P Game/DVD glasses, too? (for PS2)
SLEH-00028 ?
SLEH-00029 Fanatec Speedster 3 (for PS2)
SLEH-00030 Logitech Eye Toy (camera?) (for PS2)
```

Sony Licensed Hardware (USA)

```
SLUH-00001 Specialized Joystick (single-axis, digital?)
SLUH-00002 Control Pad (redesigned joypad)
SLUH-00003 InterAct Piranha Pad, digital pad, autofire/slowmotion
SLUH-00017 Konami Justifier, IRQ10-based Lightgun (Hyperblaster/SLPH-00014)
SLUH-00018 Enhanced Pad (joypad with whatever extra functions)
SLUH-00022 Analog and Digital Steering Wheel with pedals (for testdrive 4?)
SLUH-00026 Optec Mach 1 (gray steering/flight controller with pedals)
SLUH-00028 Ascii Sphere 360 (SLEH-00018)
SLUH-00029 Namco NPC-102 Joystick (single-axis, digital?)
SLUH-00031 Interact Program Pad
SLUH-00033 Piranha Pad (redesigned joypad)
SLUH-00034 NUBY Manufacturing The Heater, white lightgun (irq10 or cinch?)
SLUH-00035 Namco G-CON 45, Cinch-based Lightgun (SLEH-0007/SLPH-00034)
SLUH-00037 Arcade Stick (single-axis, digital?)
SLUH-00038 ASCII Grip V, single-handed digital pad (SLPH-00027/SLEH-00008)
SLUH-00040 System Organizer (huh? looks like... a black storage box?)
SLUH-00041 V3 Racing Wheel with pedals
SLUH-00043 GunCon (bundled with Time Crisis 1)
SLUH-00044 Remote Wizard (looks like wireless joypad or so)
SLUH-00045 Reality-Quest The Glove (right-handed only) (SLEH-00012/SLPH-?)
SLUH-00046 GunCon (bundled with Point Blank)
SLUH-00055 Aftershock Wheel with pedals
SLUH-00056 UltraRacer Steering Controller (grip-style)
SLUH-00057 EA Sports Game Pad (redesigned joypad)
SLUH-00058 something for point blank 2 (?) (maybe a lightgun)
SLUH-00059 Namco Jogcon, digital pad, steering dial (SLEH-0020/SLPH-00126)
SLUH-00061 MadCatz MC2 Racing Wheel (black/gray)
SLUH-00063 Bass Landing Fishing Reel controller
SLUH-00066 Sportster racing wheel
SLUH-00068 Jungle Book Rhythm N Groove Dance Pack
SLUH-00071 Konami Dance Pad (DDR Dance Pad)
SLUH-00072 GunCon (bundled with Point Blank 3)
SLUH-00073 GunCon (bundled with Time Crisis 2 - Project Titan)
SLUH-00077 Logitech Cordless Controller, analog pad (ps1/ps2)
SLUH-00081 Logitech NetPlay Controller, pad with keyboard (usb/ps2)
SLUH-00083 Konami Dance Dance Revolution Controller (for PS1 and PS2)
SLUH-00084 NYKO iType2, pad with keyboard (usb/ps2)
SLUH-00085 Logitech Cordless Action Controller (for PS2)
SLUH-00086 Namco/Taiko Drum Master (Taiko Controller Pack) (for PS2)
SLUH-00088 RedOctane In the Groove Dance Pad Controller ?
SLUH-00090 Dance Pad (bundled with Pump It Up) (for PS2)
```

Sony Licensed Hardware (Asia)

```
Unknown (if any)
```

Newer hardware add-ons?

```
SCEH-0001 SingStar (USB to Microfon) (for PS2)
```

Note

Early SLEH/SLUH devices used 4-digit numbers (eg. the "official" name for SLEH-00003 is SLEH-0003; unlike as shown in the above list).

Software (CDROM Game Codes)

```
SCED-NNNNN Sony Computer Europe Demo
SLES-NNNNN Sony Licensed Europe Software
SLED-NNNNN Sony Licensed Europe Demo
SCPS-NNNNN Sony Computer Japan Software
SLPS-NNNNN Sony Licensed Japan Software
SLPM-NNNNN Sony Licensed Japan ...?
SCUS-NNNNN Sony Computer USA Software
SLUS-NNNNN Sony Licensed USA Software
PAPX-NNNNN Demo ...?
LSP-NNNNNN Lightspan series (non-retail educational games)
```

Note: Multi-disc games have more than one game code. The game code for Disc 1 is also printed on the CD cover, and used in memory card filenames. The per-disk game codes are printed on the discs, and are used as boot-executable name in SYSTEM.CNF file. There is no fixed rule for the multi-disc numbering; some games are using increasing numbers of XNNNN or NNNNX (with X increasing from 0 upwards), and some are randomly using values like NNNXX and NNNYY for different discs.

Pinouts

External Connectors

Pinouts - Controller Ports and Memory-Card Ports
Pinouts - Audio, Video, Power, Expansion Ports
Pinouts - SIO Pinouts

Internal Pinouts

Pinouts - Chipset Summary
Pinouts - CPU Pinouts
Pinouts - GPU Pinouts
Pinouts - SPU Pinouts
Pinouts - DRV Pinouts
Pinouts - HC05 Pinouts
Pinouts - MEM Pinouts
Pinouts - CLK Pinouts
Pinouts - PWR Pinouts
Pinouts - Component List and Chipset Pin-Outs for Digital Joypad
Pinouts - Component List and Chipset Pin-Outs for Analog Joypad
Pinouts - Component List and Chipset Pin-Outs for Multitap
Pinouts - Memory Cards

Mods/Upgrades

Mods - Nocash PSX-XBOO Upload

Pinouts - Controller Ports and Memory-Card Ports

Controller Ports and Memory-Card Ports

```
1 In  JOYDAT Data from joypad/card (data in)    _____________________
2 Out JOYCMD Data to joypad/card (command out) |       |       |       |
3 -   +7.5V  +7.5VDC supply (eg. for Rumble)   | 9 7 6 | 5 4 3 |  2 1  | CARD
4 -   GND    Ground                            |_______|_______|_______|
5 -   +3.5V  +3.5VDC supply (normal supply)    _____________________
6 Out /JOYn  Select joypad/card in Slot 1/2    |       |       |       |
7 Out JOYCLK Data Shift Clock                  | 9 8 7 | 6 5 4 | 3 2 1 | PAD
8 In  /IRQ10 IRQ10 (Joy only, not mem card)     \_____|_____|_____/
9 In  /ACK   IRQ7 Acknowledge (each eight CLKs)
Shield       Ground (Joypad only, not memory card)
```

/JOYn are two separate signals (/JOY1 for left card/pad, /JOY2 for right card/pad) (whether it is an card or pad access depends on the first CMD bit). All other signals are exactly the same on all four connectors (except that pin8 and shield are missing on the card slots).

Pin8 (/IRQ10)

Most or all controllers leave pin8 unused, the pin can be used as lightpen input (not sure if the CPU is automatically latching a timer somewhere?), if there's no auto-latched timer, then the interrupt would be required to be handled as soon as possible; ie. don't disable interrupts, and don't "halt" the CPU for longer periods (as far as I understood, the GTE can halt the CPU when trying to read results of incomplete operations; to avoid that, one could wait by software, eg. inserting NOPs, before reading GTE results...?)
(Some (or maybe all?) existing psx lightguns are reportedly connected to the Video output on the Multiout port for determining the current cathode ray position though).

Pinouts - Audio, Video, Power, Expansion Ports

AV Multi Out (Audio/Video Port)

```
1      RGB-Video Green
2      RGB-Video Red
3      Supply +5.0V (eg. supply for external RF adaptor)
4      RGB-Video Blue
5      Supply Ground
6      S-Video C (chrominance)
7      Composite Video (yellow cinch)
8      S-Video Y (luminance)                  _____________________________
9      Audio Left      (white cinch)         |                             |
10     Audio Left Ground                     | 12 11 10 9 8 7 6 5 4 3 2 1 |
11     Audio Right     (red cinch)           |_____________________________|
12     Audio Right Ground
Shield Video Ground
```

The standard AV-cable connects only to Pins 7,9,10,11,12,Shield (with pin 1 and 3 and Shield shortcut with each other, used for both audio and video ground).
The plug on that cable does have additional sparings for pin 1,3,5 (though without any metal-contacts installed in there) (pin 3,5 would be used as supply for external RF modulators) (no idea what pin 1 could be used for though?).
RGB displays may (or may not) be able to extract /SYNC from the Composite signal, if that doesn't work, note that /SYNC (and separate /VSYNC, /HSYNC signals) are found on the GPU pinouts, moreover, the GPU outputs 24bit digital RGB.
Not sure if a VGA monitor can be connected? The SYNC signals are there (see GPU pinputs), but the vertical resolution is only 200/240 lines... standard VGA displays probably support only 400/480 lines (or higher resolutions for newer multisync SVGA displays) (as far as I know, the classic 200 lines VGA mode is

Parallel Port (PIO) (Expansion Port) (CN103)
This port exists only on older PSX boards (not on newer PSX boards, and not on PSone boards).
The parallel port is used by Gameshark, Game Enhancer II, and Gold Finger cheat devices (not used by the Code Breaker CDROM cheat software).

```
          _______                                Console Rear View
         |       |                         .-------------------------.
     GND ==| 1   35 |== GND                |1  2  3  ... ... 32 33 34|
  /RESET  =| 2   36 |=  DACK5              |35 36 37 ... ... 66 67 68|
   DREQ5  =| 3   37 |=  /IRQ10             |__.-------------------.__|
   /EXP?  =| 4   38 |=  /WR1? (CPU99)      |
 NC?GND?  =| 5   39 |=  GND?NC?
      D0  =| 6   40 |=  D1
      D2  =| 7   41 |=  D3
      D4  =| 8   42 |=  D5
      D6  =| 9   43 |=  D7
      D8  =|10   44 |=  D9
     D10  =|11   45 |=  D11
     D12  =|12   46 |=  D13
     D14  =|13   47 |=  D15
      A0  =|14   48 |=  A1
      A2  =|15   49 |=  A3
 NC?GND?  =|16   50 |=  GND?NC?
  +3.5V ==|17   51 |== +3.5V
  +7.5V ==|18   52 |== +7.5V
   GND?  =|19   53 |=  GND?NC?
      A4  =|20   54 |=  A5
      A6  =|21   55 |=  A7
      A8  =|22   56 |=  A9
     A10  =|23   57 |=  A11
     A12  =|24   58 |=  A13
     A14  =|25   59 |=  A15
     A16  =|26   60 |=  A17
     A18  =|27   61 |=  A19
     A20  =|28   62 |=  A21
     A22  =|29   63 |=  A23
     /RD  =|30   64 |=  /WR0
   NC!X?  =|31   65 |=  X?NC!
  SYSCK?  =|32   66 |=  LRCK (44.1kHz)
   SCLK?  =|33   67 |=  SDATA?
     GND ==|34   68 |== GND
         |_______|
```

Lots of pins are still unknown?
EDIT: see http://cgfm2.emuviews.com/new/psx-pio.png
apparently, many of the "unknown" pins are just GROUND, is that possible?

Internal Power Supply (PSX)
The PSX contains an internal power supply, however, like the PSone, it's only having a "Standby" button, which merely disconnects 3.5V and 7.9V from the mainboard. The actual power supply remains powered, and wastes energy day and night, thanks Sony!

External Power Supply (PSone)
```
Inner +7.5V DC 2.0A   (inside diameter 0.8mm)
Outer GND             (outside diameter 5.0mm)
```

Pinouts - SIO Pinouts

Serial Port
That port exists only on original Playstation (not on the PSone). The shape of the Serial Port is identical to the 12pin Multiout (audio/video) port, but with only 8pins.

```
1 SIO1 In  RXD receive data    (from remote TXD)
2 SIO2 -   VCC +3.5VDC         (supply, eg. for voltage conversion)
3 SIO3 In  DSR                 (from remote CTS)          _________________
4 SIO4 Out TXD transmit data   (to remote RXD)           |                 |
5 SIO5 In  CTS clear to send   (from remote RTS)         | 8 7 6 5 4 3 2 1 |
6 SIO6 Out DTR                 (to remote DSR)           |_________________|
7 SIO7 -   GND Ground          (supply, eg. for voltage conversion)
8 SIO8 Out RTS request to send (to remote CTS)
Shield     GND Ground          (to/from remote GND)
```

Can be used to communicate with another PSX via simple cable connection. With an external RS232 adaptor (for voltage conversion) it could be also used to communicate with a PC, a mouse, a modem, etc.

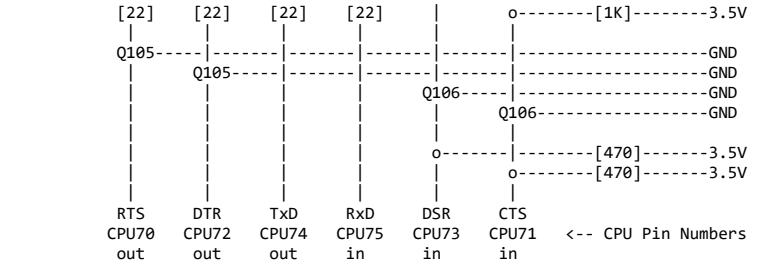
PSone Serial Port
The PSone doesn't have an external serial connector, however, easy to use soldering points for serial port signals are found as cluster of 5 soldering points (below CPU pin52), and a single soldering point (below CPU pin100), arranged like so (on PM-41 boards) (might be different on PM-41(2) boards):

```
              CPU70.RTS
         CPU71.CTS   CPU74.TxD
              CPU72.DTR   CPU75.RxD                    CPU73.DSR
```

The three outputs (RTS,DTR,TXD) are left floating, the RXD input is wired via a 1K ohm pull-up resistor to 3.5V, the other two inputs (CTS,DSR) are wired via 1K ohm pull-down resistors to GND.
If you want to upgrade the PSone, remove that resistors, and then install the PSX-style serial circuit (as shown below), or, think of a more simplified circuit without (dis-)inverted signals.

PSX Serial Port Connection (PU-23 board) (missing on PM-41 board)
The PSX serial circuit basically consists of a few transistors, diodes, and resistors. The relevant part is that most of the signals are inverted - compared with RS232 signals, the CPU uses normal high/low levels (of course with 0V and 3.5V levels, not -12V and +12V), and the signals at the serial port socket are inverted. Ie. if you want to built a RS232 adaptor, you must either externally undo the inversion, or, disconnect the transistors, and wire your circuit directly to the CPU signals.

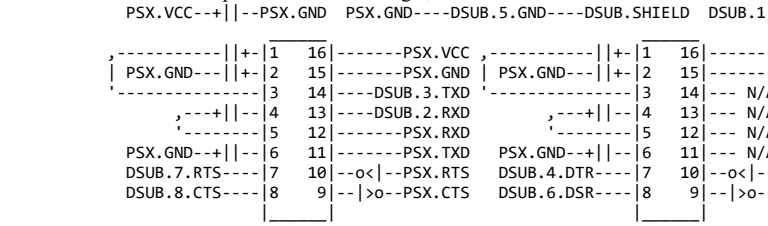
```
SIO8    SIO6    SIO4    SIO1    SIO3    SIO5    SIO2    SIO7---GND
  |       |       |       |       |       |       |
FB112   FB114   FB116   FB115   FBnnn   FBnnn    o--L102-------3.5V
  |       |       |       |       |       |
  |       |       o-------|-------|-------|--------diode-------GND
  |       |       |       o-------|-------|--------diode-------GND
  |       |       |       |       o-------|--------diode-------GND
  |       |       |       |       |       o--------diode-------GND
  |       |       |       |       |       |
  |       |       |       o-------|-------|--------[1K]--------3.5V
```



All six signals are passed through fuses (or loops or so). The three inputs have 1K ohm pull-ups, and diodes as protection against negative voltages, two of the inputs are inverted via transistors, with 470 ohm pull-ups at the CPU side, the other input is passed through 22 ohm to the CPU. The three outputs are also passed through 22 ohm, one of them having a diode as negative voltage protection, the other two are inverted via transistors (which may also serve as negative voltage protection). Note that there is no positive voltage protection (ie. +12V inputs would do no good, also strong -12V inputs might overheat the diodes/fuses, so if you want to use RS232 voltages, better use a circuit for voltage conversion).

Serial RS232 Adaptor

The PSX serial port uses 0V/3.5V logic, whilst RS232 uses -5V/+5V...-15V/+15V logic. An example circuit for converting the logic levels would be:



Parts List: 1 or 2 MAX232 chips (voltage conversion), 0 or 1 7400 (NAND, used as inverter), 4 or 8 1uF/16V capacitors, 1x 10uF/16V capacitor, 1x 9pin male SubD plug.

The four inverters are needed only for external adapters (which need to undo the transistor inversion on the PSX mainboard) (ie. the inverters are not needed when when connecting the circuit directly to the PSX CPU).

The second MAX232 chip is needed only if DTR/DSR "not ready" conditions are required (for an "always ready" condition: DSUB.4.DTR can be wired to -8.5V, which is available at Pin6 of the first MAX232 chip, and PSX.DSR can be wired to +3.5V).

With the above DSUB pin numbers, peripherals like mice or modems can be connected directly to the circuit. For connection to another computer, use a "null modem" cable (with crossed RXD/TXD, RTS/CTS, DTR/DSR wires).

The circuit works with both VCC=5V (default for MAX232) and with VCC=3.5V (resulting in slightly weaker signals, but still strong enough even for serial mice; which are mis-using the RS232 signals as power supply).

Pinouts - Chipset Summary

PSX/PSone Mainboards

```
Board    Expl.
PU-7     PSX, with AV multiout+cinch+svideo, GPU in two chips (160+64pins)
PU-8     PSX, with AV multiout+cinch, four 8bit Main RAM chips
          EARLY-PU-8: "PU-8 1-658-467-11, N4" --> old chipset, resembles PU-7
          LATE-PU-8:  "PU-8 1-658-467-22, N6" --> new chipset, other as PU-7
PU-9     PSX, without SCPH-number (just sticker saying "NOT FOR SALE, SONY)
PU-16    PSX, with extra Video CD daughterboard (for SCPH-5903)
PU-18    PSX, with AV multiout only, single 32bit Main RAM (instead 4x8bit)
PU-20    PSX, unknown if/how it differs from PU-18
PU-22    PSX, unknown if/how it differs from PU-18
PU-23    PSX, with serial port, but without expansion port
PM-41    PSone, older PSone, for GPU/SPU with RAM on-board (see revisions)
PM-41(2) PSone, newer PSone, for GPU/SPU with RAM on-chip
```

There are at least two revisions of the "PM-41" board:

```
PM-41, 1-679-335-21  PSone with incomplete RGB signals on multiout port
PM-41, 1-679-335-51  PSone with complete RGB signals on multiout port
```

The "incomplete" board reportedly requires to solder one wire to the multiout port to make it fully functional... though no idea which wire... looks like the +5V supply? Also, the capacitors near multiout are arranged slightly differently.

CPU chips

```
IC103 - 208pin - "SONY CXD8530BQ"  ;seen on PU-7 board
IC103 - 208pin - "SONY CXD8530CQ"  ;seen on PU-7 and PU-8 boards
IC103 - 208pin - "SONY CXD8606Q"   ;seen in PU-18 schematic
IC103 - 208pin - "SONY CXD8606AQ"  ;seen on PU-xx? board
IC103 - 208pin - "SONY CXD8606BQ"  ;seen on PM-41, PU-23, PU-20 boards
IC103 - 208pin - "SONY CXD8606CQ"  ;seen on PM-41 board, too
```

These chips contain the MIPS CPU, COP0, and COP2 (aka GTE). And maybe MDEC?

GPU chips

```
IC203 - 160pin - "SONY CXD8514Q"   ;\seen on PU-7 and EARLY-PU-8 boards
IC207 -  64pin - "SONY CXD2923AR"  ;/(GPU divided into two chips?!)
IC203 - 208pin - "SONY CXD8561Q"   ;seen on LATE-PU-8 board
IC203 - 208pin - "SONY CXD8561BQ"  ;seen on PU-18, PU-20 boards
IC203 - 208pin - "SONY CXD8561CQ"  ;seen on PM-41 board
IC203 - 208pin - "SONY <unknown>"  ;with on-chip RAM ;for PM-41(2) board
```

SPU chips - Sound Processing Unit

```
IC308 - 100pin - "SONY CXD2922Q" (SPU)               ;PU-7 and EARLY-PU-8
IC308 - 100pin - "SONY CXD2925Q" (SPU)               ;LATE-PU-8, PU-18, PU-20
IC732 - 208pin - "SONY CXD2938Q" (SPU+CDROM)         ;PSone/PM-41 Board
IC732 - 176pin - "SONY CXD2941R" (SPU+CDROM+SPU_RAM) ;PSone/PM-41(2) Board
IC402 - 24pin  - "AKM AK4309VM"  (Serial 2x16bit DAC);older boards only
IC405 - 8pin   - "NJM2100E (TE2)" Audio Amplifier    ;PU-8 and PU-22 boards
IC405 - 14pin  - "NJM2174" Audio Amplifier with Mute ;later boards
```

IC106 CPU-RAM / Main RAM chips

```
IC106/IC107/IC108/IC109 - NEC 424805AL-A60 (28pin, 512Kx8) (PU-8 board)
IC106 - "Samsung K4Q153212M-JC60" (70pin, 512Kx32) (newer boards)
```

GPU-RAM / Video RAM chips

```
IC201 - 64pin NEC D482445LGW-A70-S   ;VRAM  ;\on PU-7 and EARLY-PU-8 board
IC202 - 64pin NEC D482445LGW-A70-S   ;VRAM  ;/split into 2 chips !
IC201 - 64pin SEC KM4216Y256G-60     ;VRAM  ;\on other PU-7 board
IC202 - 64pin SEC KM4216Y256G-60     ;VRAM  ;/split into 2 chips !
IC201 - 100pin - Samsung KM4132G271BQ-10 (128Kx32x2)  ;-on later boards
IC201 - 100pin - Samsung K4G163222A-PC70 (256Kx32x2)  ;-on PM-41
```

Note: The PM-41 board uses a 2MB VRAM chip (but allows to access only 1MB)
Note: The PM-41(2) board has on-chip RAM in the GPU (no external memory chip)

IC310 - SPU-RAM - Sound RAM chips

```
IC310 - 40pin - "TOSHIBA TC51V4260DJ-70"  ;seen on PU-8 board
IC310 - 40pin - EliteMT M11B416256A-35J (256K x 16bit)
```

Note: The PM-41(2) board has on-chip RAM in the SPU (no external memory chip)

BIOS ROM

```
IC102 - 40pin - "SONY ..."          ;seen on PU-7 & early-PU-8 board (40pin!)
IC102 - 32pin - "SONY M534031C-25"  ;seen on later-PU-8 board
IC102 - 32pin - "SONY 2030"         ;seen on PU-18 board
IC102 - 32pin - "SONY M534031E-47"  ;seen on PM-41 board
IC102 - 32pin - "SONY M27V401D-41"  ;seen on PM-41 board, too
```

Oscillators and Clock Multiplier/Divider

```
X101 - 4pin - "67.737" (NTSC, presumably)         ;PSX only
X201 - 2pin - "17.734" (PAL) or "14.318" (NTSC)   ;PSone only
IC204 - 8pin - "2294A" (PAL) or <unknown?> (NTSC) ;PSone only
```

Voltage Converter (for +7.5V to +5.0V conversion)

```
IC601 - 3pin - "78M05" or "78005"  ;used in PSone
```

Pulse-Width-Modulation Power-Control Chip

```
IC606 16pin/10mm "TL594CD" (alternately to IC607) ;seen on PM-41 board
IC607 16pin/5mm  "T594"    (alternately to IC606) ;seen on PM-41 board, too
```

The PM-41 board has locations for both IC606 and IC607, some boards have the bigger IC606 (10mm) installed, others the smaller IC607 (5mm), both chips have exactly the same pinouts, the only difference is the size.

Reset Generator

```
IC002 - 8pin - <not installed> (would be alternately to IC003) ;\on PSone
IC003 - 5pin - <usually installed>                             ;/
IC101 - 5pin - M51957B (Reset Generator) (on PSX-power supply boards)
```

CDROM Chips

```
U42   80pin    SUB-CPU (CXP82300) with piggyback EPROM ;DTL-H2000
IC304 80pin    SUB-CPU (MC68HC05L16) 80pin package     ;PU-7 and EARLY-PU-8
IC304 52pin    SUB-CPU (MC68HC05G6) 52pin package      ;LATE-PU-8 and up
IC305 - 100pin SONY CXD1199BQ (Decoder/FIFO)           ;PU-7
IC305 - 100pin SONY CXD1815Q  (Decoder/FIFO)           ;PU-8, PU-18
IC309 - 100pin SONY CXD2516Q  (Signal Processor)       ;PU-7 (100pin!)
IC309 - 80pin  SONY CXD2510Q  (Signal Processor)       ;PU-8 and DTL-H2510
IC702 - 48pin  SONY CXA1782BR (Servo Amplifier)        ;PU-7, PU-8
IC701 - 100pin SONY CXD2545Q  (=CXD2510Q+CXA1782BR)    ;PU-18
IC720 - 144pin SONY CXD1817R  (=CXD2545Q+CXD1815Q)     ;PU-20
IC722 - 28pin - "BA5947FP"           ;seen on PM-41 and various boards
IC722 - 28pin - "Panasonic AN8732SB"  ;seen on PM-41 board
IC703 - 20pin  SONY CXA1791N    (RF Amplifier) (on PU-18 boards)
IC723 - 20pin  SONY CXA2575N-T4 (RF Matrix Amplifier) (later boards)
```

Note: The SUB-CPU contains an on-chip BIOS (which does exist in at least seven versions, plus US/JP/PAL-region variants, plus region-free debug variants).

RGB Chips

```
IC202 44pin "Philips TDA8771H" Digital to Analog RGB         ;\older boards
IC501 24pin "SONY CXA1645M" Analog RGB to Composite          ;/
IC502 48pin "SONY CXA2106R-T4" 24bit RGB to Analog+Composite ;-newer boards
```

MISC

```
CDROM Drive: "KSM-440BAM" ;seen used with PM-41 board
IC602 5pin           "L/\1B" or "<symbol> 3DR"
```

Pinouts - CPU Pinouts

CPU Pinouts (IC103)

```
1-3.5V  27-GND   53-3.5V  79-3.5V   105-3.5V 131-3.5V   157-3.5V   183-3.5V
2-3.5V  28-DQ12  54-3.5V  80-/JOY1  106-3.5V 132-A5     158-3.5V   184-GD19
3-NC    29-DQ11 55-A11:A8 81-JOYCLK 107-D0   133-A6     159-HBLANK 185-GD20
4-67MHz 30-DQ10 56-A10:NC 82-/IRQ7  108-D1   134-A7     160-DOTCLK 186-GD21
5-DQ31  31-DQ9   57-A9    83-JOYCMD 109-D2   135-A8     161-GD0    187-GD22
6-DQ30  32-DQ8   58-A8:NC 84-JOYDAT 110-D3   136-A9     162-GD1    188-GD23
7-DQ29  33-DQ7   59-A7    85-DACK5  111-D4   137-A10    163-GD2    189-GD24
8-DQ28  34-DQ6   60-A6    86-DREQ5  112-D5   138-A11    164-GD3    190-GD25
9-DQ27  35-DQ5   61-A5    87-DMA4   113-D6   139-A12    165-GD4    191-GD26
10-DQ26 36-DQ4   62-A4    88-/SPUW  114-D7   140-A13    166-GD5    192-GD27
11-DQ25 37-DQ3   63-A3    89-/IRQ10 115-D8   141-A14    167-GD6    193-GD28
12-DQ24 38-3.5V  64-A2    90-/IRQ9  116-D9   142-A15    168-GD7    194-GD29
13-DQ23 39-GND   65-GND   91-GND    117-GND  143-GND    169-GD8    195-GND
14-3.5V 40-DQ2   66-3.5V  92-3.5V   118-3.5V 144-3.5V   170-GND    196-3.5V
15-GND  41-DQ1   67-A1    93-GND    119-D10  145-A16    171-3.5V   197-GD30
16-DQ22 42-DQ0   68-A0    94-/IRQ2  120-D11  146-A17    172-GD9    198-GD31
17-DQ21 43-/W    69-3.5V  95-/CD    121-D12  147-A18    173-GD10   199-VBLANK
18-DQ20 44-NC?   70-RTS   96-/SPU   122-D13  148-A19    174-GD11   200-GPU12
19-DQ19 45-/RAS  71-CTS   97-/BIOS  123-D14  149-A20    175-GD12   201-33MHzG
20-DQ18 46-/CAS3 72-DTR   98-/EXP   124-D15  150-A21    176-GD13   202-GPU5
21-DQ17 47-/CAS2 73-DSR   99- CPU99 125-A0   151-A22    177-GD14   203-/GWR
22-DQ16 48-/CAS1 74-TxD   100-/WR   126-A1   152-A23    178-GD15   204-/GRD
23-DQ15 49-/CAS0 75-RxD   101-/RD   127-A2   153-GPU.A2 179-GD16   205-/GPU
24-DQ14 50-3.5V  76-/RES  102-/IRQ1 128-A3   154-33MHzS 180-GD17   206-67MHzG
```

```
  26-3.5V 52-GND   78-GND   104-GND   130-GND  156-GND    182-GND    208-GND
```
Pin5-68 = Main RAM bus. Pin 95-152 = System bus. Pin 102,153,159-206 = Video bus.
```
  85=DACK5  93=GND=/CSHTST  199=/INT0   44=/RAS1:NC
  86=DREQ5  99=/SWR1=NC     200=DREQ2   45=/RAS0
  87=DACK4  100=/SWR0       201=SYSCLK0
  88=DREQ4  154=SYSCLK1     202=DACK2
```

CPU Pinout Notes
Pin 43,45..49,100,101,125(A0!),201,203..206 are connected via 22 ohm.
Pin 77,80,81,83 are connected via 470 ohm.
Pin 82,84,89 are connected via 47 ohm.
Pin 95,96,97 are connected via 100 ohm.
Pin 44: goes LOW for a short time once every N us (guessed: maybe /REFRESH ?)
Pin 4: 67MHz (from IC204.pin5)
Pin 87/88: SPU-DMA related (/SPUW also permanent LOW for Manual SPU-RAM Write)
Pin 154: 33MHzS (via 22ohm and FB102 to SPU) (and TESTPOINT near MainRAM pin70)
Pin 160: DOTCLK (via 22ohm), and IC502.Pin41 (without 22ohm)
Pin 56,58 are maybe additional address lines for the addressable 8MB RAM.
The System Bus address lines are latched outputs (containing the most recently used /BIOS /EXP /SPU /CD address) (not affected by Main RAM and GPU addressing).

Pinouts - GPU Pinouts

GPU Pinouts (IC203)
```
  1-/GPU    27-GD28  53-GD10   79-D29  105-GND  131-CLK   157-/PAL   183-R3
  2-GPU.A2  28-GD27  54-GD9    80-3.5V 106-3.5V 132-GND   158-/VSYNC 184-GND
  3-/GRD    29-3.5V  55-GD8    81-GND  107-D17  133-3.5V  159-/HSYNC 185-3.5V
  4-/GWR    30-GND   56-GD7    82-D28  108-D16  134-CLK   160-B0     186-R4
  5-CPU202  31-GD26  57-GD6    83-D27  109-D7   135-GND   161-B1     187-R5
  6-/RES    32-GD25  58-GD5    84-D26  110-D6   136-3.5V  162-B2     188-R6
  7-3.5V    33-GD24  59-GD4    85-D25  111-D5   137-(A10) 163-B3     189-R7
  8-GND     34-GD23  60-GND    86-D24  112-D4   138-A9/AP 164-GND    190-GND
  9-33MHzG  35-GD22  61-3.5V   87-3.5V 113-GND  139-A7    165-3.5V   191-3.5V
  10-3.5V   36-GD21  62-GD3    88-GND  114-3.5V 140-A6    166-B4     192-53MHz
  11-GND    37-3.5V  63-GD2    89-D15  115-D3   141-3.5V  167-B5     193-3.5V
  12-CPU200 38-GND   64-GD1    90-D14  116-D0   142-GND   168-B6     194-GND
  13-/IRQ1  39-GD20  65-GD0    91-D13  117-D1   143-A5    169-B7     195-3.5V
  14-HBLANK 40-GD19  66-GND    92-D12  118-D2   144-A4    170-G0     196-53MHz
  15-GND    41-GD18  67-3.5V   93-D11  119-GND  145-A3    171-G1     197-3.5V
  16-3.5V   42-GD17  68-(high) 94-D10  120-3.5V 146-GND   172-G2     198-GND
  17-VBLANK 43-3.5V  69-(high) 95-D9   121-NC   147-3.5V  173-G3     199-DOTCLK
  18-(pull) 44-GND   70-(high) 96-GND  122-/CS  148-A2    174-GND    200-GND
  19-(low)  45-GD16  71-3.5V   97-3.5V 123-DSF  149-A1    175-3.5V   201-3.5V
  20-GND    46-GD15  72-3.5V   98-D8   124-/RAS 150-A0    176-G4     202-BLANK
  21-(low)  47-GD14  73-3.5V   99-D18  125-/CAS 151-3.5V  177-G5     203-(low)
  22-3.5V   48-GD13  74-3.5V   100-D19 126-/WE  152-GND   178-G6     204-GND
  23-3.5V   49-GD12  75-3.5V   101-D20 127-DQM1 153-NC    179-G7     205-3.5V
  24-GD31   50-GD11  76-GND    102-D21 128-DQM0 154-3.5V  180-R0     206-67MHzG
  25-GD30   51-3.5V  77-D31    103-D22 129-GND  155-GND   181-R1     207-GND
  26-GD29   52-GND   78-D30    104-D23 130-3.5V 156-/SYNC 182-R2     208-3.5V
```
Pin 77..150 = Video RAM Bus. Pin 156..189 = Video Out Bus. Other = CPU Bus.

GPU Pinout Notes
Pin 1,3,4,9,122..128,199,206 are connected via 22 ohm.
Pin 18 has a 4K7 ohm pullup to 3.5V
Pin 77..118 data lines (DQ0..DQ31) are connected via 82 ohm.
Pin 192/196: via 220 ohm to IC204.pin1 (53MHz)
At RAM Side: CKE via 4K7 to 3.5V, and, A8 is GROUNDED!
DQM0 is wired to both DQM0 and DQM2, DQM1 is wired to both DQM1 and DQM3.
CLK is wired to both GPU pin 131 and 134.
RGBnn = IC502 pin nn
/VSYNC, /HSYNC, (and BLANK?) are test points (not connected to any components).
/SYNC = (/VSYNC AND /HSYNC). BLANK = (VBLANK OR HBLANK).

IC202 44pin "Philips TDA8771H" Digital to Analog RGB (older boards only)
Region Japan+Europe: TDA8771AN
Region America+Asia: MC151854FLTEG or so?
```
  1-IREF  6-GNDd1  11-R1   16-G4   21-B7  26-B2     31-CLK    36-OUTB  41-NC
  2-GNDa1 7-VDDd1  12-R0   17-G3   22-B6  27-VDDd2  32-VDDa1  37-NC    42-GNDa2
  3-R7    8-R4     13-G7   18-G2   23-B5  28-GNDd2  33-VREF   38-NC    43-VDDa4
  4-R6    9-R3     14-G6   19-G1   24-B4  29-B1     34-NC     39-VDDa3 44-OUTR
  5-R5    10-R2    15-G5   20-G0   25-B3  30-B0     35-VDDa2  40-OUTG
```
Used only on older boards (eg. PU-8), newer boards generate analog RGB via 48pin IC502.

IC501 24pin "SONY CXA1645M" Analog RGB to Composite (older boards only)
```
  1-GND1  4-BIN   7-NPIN   10-SYNCIN  13-IREF  16-YOUT   19-VCC2   22-GOUT
  2-RIN   5-NC    8-BFOUT  11-BC      14-VREF  17-YTRAP  20-CVOUT  23-ROUT
  3-GIN   6-SCIN  9-YCLPC  12-VCC1    15-COUT  18-FO     21-BOUT   24-GND2
```
Used only on older boards (eg. PU-7, PU-8), newer boards generate composite signal via 48pin IC502.
Pin7 (NPIN): NTSC=VCC, PAL=GND.

IC502 48pin "SONY CXA2106R-T4" - 24bit RGB video D/A converter
```
  1-(cap) 7-Comp.  13-/PAL   19-R4     25-G7     31-G1     37-B3     43-NC
  2-GND   8-Chro.  14-/SYNC  20-5.0V   26-G6     32-G0     38-B2     44-(cap)
  3-Red   9-5.0V   15-4.4MHz 21-R3     27-G5     33-B7     39-B1     45-GND
  4-Green 10-(2K7) 16-R7     22-R2     28-G4     34-B6     40-B0     46-(cap)
  5-Blue  11-NC    17-R6     23-R1     29-G3     35-B5     41-DOTCLK 47-5.0V
  6-Lum.  12-NC    18-R5     24-R0     30-G2     36-B4     42-GND    48-(cap)
```
Pin 3..8 (analogue outputs) are passed via external 75 ohm resistors.
Pin 6,7 additionally via 220uF. Pin 8 additionally via smaller capacitor.
Pin 10 wired via 2K7 to 5.0V, dunno why (the signal is just high).
Pin 1,44,46,48 (can) connect via capacitors to ground (only installed for 44).

The /PAL pin can be reportedly GROUNDED to force PAL colors in NTSC mode, when doing that, you may first want to disconnect the pin from the GPU.

Beware
Measuring in the region near GPU Pin10 is the nocash number one source for blowing up components on the mainboard. If you want to measure that signals while power is on, better measure them at the CPU side.

Pinouts - SPU Pinouts

IC308 - SONY CXD2925Q (SPU) (on PU-8, PU-18 boards)

1-D0	14-D11	27-A8	40-GND	53-3.5V	66-A15	79-5V	92-LRIA
2-D1	15-GND	28-3.5V	41-SYSCK	54-GND	67-A14	80-A3	93-DTIA
3-3.5V	16-D12	29-GND	42-GND	55-D7	68-A13	81-A2	94-BCIB
4-GND	17-D13	30-A9	43-TEST	56-D6	69-A12	82-A1	95-LRIB
5-D2	18-D14	31-/SPU	44-TES2	57-D5	70-A11	83-A0	96-DTIB
6-D3	19-D15	32-/RD	45-D15	58-D4	71-A10	84-/WE0	97-BCKO
7-D4	20-A1	33-/WR	46-D14	59-D3	72-A9	85-/OE0	98-LRCO
8-D5	21-A2	34-DACK	47-D13	60-D2	73-A8	86-/WE1	99-DATO
9-D6	22-A3	35-/IRQ	48-D12	61-D1	74-A7	87-/OE1	100-WCKO
10-D7	23-A4	36-DREQ	49-D11	62-D0	75-A6	88-GND	
11-D8	24-A5	37-MUTE	50-D10	63-/RAS	76-A5	89-XCK	
12-D9	25-A6	38-/RST	51-D9	64-/CAS	77-A4	90-GND	
13-D10	26-A7	39-NC	52-D8	65-GND	78-GND	91-BCIA	

Pin 1..36 = MIPS-CPU bus. Pin 45..87 = SPU-RAM bus (A0,A10-A15,/WE1,OE1=NC). Pin 91..99 = Digital serial audio in/out.

IC732 - SONY CXD2941R (SPU+CDROM+SPU_RAM) (on PM-41(2) boards)

1-DA16	23-FILO	45-LOCK	67-FSTO	89-SCSY	111-XCS	133-HD9	155-VSS5
2-DA15	24-FILI	46-SSTP	68-COUT	90-SCLK	112-XRD	134-HD8	156-HA1
3-DA14	25-PCO	47-SFDR	69-XDRST	91-SQSO	113-XWR	135-HD7	157-HA0
4-VDDM0	26-CLTV	48-SRDR	70-DA11	92-SENS	114-HINT	136-HD6	158-VDDM3
5-DA13	27-AVSSO	49-TFDR	71-DA10	93-DATA	115-XIRQ	137-VDD4	159-XCK
6-DA12	28-RFAC	50-TRDR	72-DA09	94-XLAT	116-VDDM2	138-HD5	160-DTIB
7-LRCK	29-BIAS	51-VSSM1	73-DA08	95-CLOK	117-XSCS	139-HD4	161-BCKO
8-WDCK	30-ASYI	52-FFDA	74-AVSMO	96-XINT	118-XHCS	140-HD3	162-LRCO
9-VDD0	31-AVDDO	53-FRDA	75-AVDMO	97-A4	119-XHRD	141-HD2	163-DAVDD0
10-VSS0	32-ASYO	54-MDP	76-DA07	98-A3	120-XHWR	142-VSS4	164-DAREFL
11-PSSL	33-VC	55-MDS	77-DA06	99-A2	121-DACK	143-HD1	165-AOUTL
12-ASYE	34-CE	56-VDD2	78-VDDM1	100-A1	122-DREQ	144-HD0	166-DAVSS0
13-GND	35-CEO	57-VSS2	79-DA05	101-A0	123-XRST	145-VSSM3	167-DAVSS1
14-C4M	36-CEI	58-MIRR	80-DA04	102-D7	124-VDD3	146-HA9	168-AOUTR
15-C16M	37-RFDC	59-DFCT	81-DA03	103-D6	125-SYSCK	147-HA8	169-DAREFR
16-FSOF	38-ADIO	60-AVSM1	82-DA02	104-D5	126-VSS3	148-HA7	170-DAVDD1
17-XTSL	39-AVDD1	61-AVDM1	83-DA01	105-D4	127-HD15	149-HA6	171-MUTO
18-VDD1	40-IGEN	62-FOK	84-WFCK	106-VSSM2	128-HD14	150-HA5	172-DATO
19-GND	41-AVSS1	63-PWMI	85-SCOR	107-D3	129-HD13	151-HA4	173-MTS3
20-VPCO1	42-TE	64-FSW	86-SBSO	108-D2	130-HD12	152-VDD5	174-MTS2
21-VPCO2	43-SE	65-MON	87-EXCK	109-D1	131-HD11	153-HA3	175-MTS1
22-VCTL	44-FE	66-ATSK	88-SQCK	110-D0	132-HD10	154-HA2	176-MTS0

IC732 - SONY CXD2938Q (SPU+CDROM) (on newer boards) (PM-41 boards)

1-	27-	53-TrckR	79-/XINT	105-A0	131-3.5V	157-(tst)	183-A8
2-GND	28-GND	54-TrckF	80-SQCK	106-3.5V	132-D9	158-(tst)	184-A7
3-GND	29-	55-FocuR	81-SQSO	107-A1	133-D8	159-GND	185-A6
4-	30-	56-3.5V	82-SENSE	108-A2	134-D7	160-D15	186-A5
5-	31-	57-FocuF	83-GND	109-A3	135-D6	161-D0	187-GND
6-GND	32-	58-SledR	84-GND	110-A4	136-D5	162-D14	188-A4
7-	33-	59-SledF	85-CD.D7	111-A5	137-3.5V	163-D1	189-A3
8-3.5V	34-	60-NC	86-CD.D6	112-3.5V	138-D4	164-D13	190-A2
9-	35-	61-GND	87-CD.D5	113-A6	139-D3	165-3.5V	191-A1
10-GND	36-	62-NC	88-CD.D4	114-A7	140-D2	166-D2	192-A0
11-4.3MHz	37-	63-GND	89-CD.D3	115-A8	141-D1	167-D12	193-3.5V
12-12MHz	38-	64-(tst)	90-CD.D2	116-A9	142-D0	168-D3	194-NC
13-	39-see39	65-(tst)	91-CD.D1	117-/IRQ2	143-GND	169-D11	195-(tst)
14-	40-	66-note	92-CD.D0	118-/IRQ9	144-33MHzS	170-D10	196-GND
15-	41-	67-note	93-3.5V	119-/RD	145-	171-D4	197-(tst)
16-	42-	68-(tst)	94-CD/CS	120-/WR	146-3.48V	172-D9	198-NC
17-3.5V	43-	69-3.5V	95-CD/WR	121-DMA4	147-ZZ11	173-GND	199-NC
18-	44-	70-(tst)	96-CD/RD	122-GND	148-GND	174-D5	200-NC
19-GND	45-GND	71-(tst)	97-CD.A0	123-GND	149-GND	175-D8	201-3.5V
20-	46-	72-(tst)	98-CD.A1	124-/SPUW	150-ZZ7	176-D6	202-NC
21-Spin	47-	73-(tst)	99-CD.A2	125-D15	151-3.48V	177-D7	203-NC
22-3.5V	48-GND	74-DATA	100-GND	126-D14	152-/RES	178-/CAS	204-NC
23-	49-GND	75-XLAT	101-CDA3	127-D13	153-3.5V	179-/WE	205-GND
24-	50-GND	76-CLOK	102-CDA4	128-D12	154-ZZ5	180-3.5V	206-(tst)
25-	51-GND	77-SCOR	103-/CD	129-D11	155-(tst)	181-/OE	207-(tst)
26-	52-GND	78-GND	104-/SPU	130-D10	156-(tst)	182-/RAS	208-GND

Pin 74..102 = SubCPU. Pin 103..144 = MainCPU. Pin 160..192 = Sound RAM Bus.
Pin 21 and 53..59 = Drive Motor Control (IC722).
Pin 1..47 are probably mainly CDROM related.
Pin 39 "see39" = IC723.Pin16 - CL709, and via 15K to SPU.39
Pin 66 connects via 4K7 to IC723.Pin19.
Pin 67 not connected (but there's room for an optional capacitor or resistor)
The (tst) pins are wired to test points (but not connected to any components)

CXD2938Q SPU Pinout Notes
Pin 74,75,76,119,120 are connected via 22 ohm.
Pin 103,104 are connected via 100 ohm.
ZZnn = IC405 Pin nn (analog audio related, L/R/MUTE).
Pin 103..142 = System Bus (BIOS,CPU). Pin 160..192 = Sound RAM Bus.
Pin 178 used for both /CASL and /CASH (which are shortcut with each other).
Pin 146 and 151 are 3.48V (another supply, not 3.5V).
Pin 147 and 150 are connected via capacitors.
Pin 195 and 197 testpoints are found below of the pin 206/207 testpoints.

```
SPU155 (tst) always low         ;=maybe external audio (serial) this?
SPU156 (tst) 45kHz  (22us)      ;=probably 44.1kHz (ext audio sample-rate)
SPU157 (tst) 2777kHz  (0.36us)  ;=probably 64*44.1kHz (ext audio bit-rate)
```

SPU.Pin5 connects to MANY modchips
SPU.Pin42 connects to ALL modchips
SPU.Pin42 via capacitor to SPU.Pin41, and via resistor?/diode? to IC723.10

CXD2938Q CDROM clocks

```
SPU197  (*) 7.35kHz (44.1kHz/6) (stable clock, maybe DESIRED drive speed)
SPU5    (*) 7.35kHz (44.1kHz/6) (unstable clock, maybe ACTUAL drive speed)
SPU15   (*) 44.1kHz (44.1kHz*1)
SPU16   (*) 88.2kHz (44.1kHz*2)
SPU206  (*) circa 2.27MHz
SPU70   (*) whatever clock (with SHORT low pulses)
```

(*) these frequencies are twice as fast in double speed mode.

CXD2938Q CDROM signals

```
SPU207  fastsignal?
SPU195  slowsignal?
SPU18   usually high, low during seek or spinup or so
SPU44   superslow hi/lo with superfast noise on it
SPU73   mainly LOW with occasional HIGH levels...
SPU71   LOW=SPIN_OK, PULSE=SPIN_UP/DOWN_OR_STOPPED
SPU72   similar as SPU71
SPU64   LOW=STOP, HI=SPIN
SPU68   always low...?
SPU65   whatever?
SPU75   mainly HIGH, short LOW pulses when changing speed up/down/break
```

CXD2938Q CDROM/SPU Testpoints (on PM-41 board)

```
                |                                       | SPU73
                |          CXD2938Q (SPU)               |       SPU72
                |          (on PM-41 board)             | SPU70 SPU71
                |                                       | SPU64 SPU65 SPU68
  SPU206 SPU207 |_______________________________________|
   SPU197
    SPU195                    SPU16                   SPU44
                  SPU18 SPU5 SPU15
                         SPU12
```

IC402 - 24pin AKM AK4309VM (or AK4309AVM/AK4310VM) - Serial 2x16bit DAC

```
1-TST?  4-/PD   7-CKS    10-LRCK  13-NC?    16-AOUTL  19-GNDa  22-VREFH
2-VCCd  5-/RST  8-BICK   11-NC?   14-NC?    17-VCOM   20-NC?   23-VREFL
3-GNDd  6-MCLK  9-SDATA  12-NC?   15-AOUTR  18-VCCa   21-NC?   24-DZF?
```

Used only on older boards (eg. PU-8), newer boards seem to have the DAC in the 208pin SPU.
No 24pin AK4309VM datasheet exists (however it seems to be same as 20pin AK4309B's, with four extra NC pins at pin10-14).

IC405 - "2174, 1047C, JRC" or "3527, 0A68" (on newer boards)

Called "NJM2174" in service manual. Audio Amplifier with Mute.

```
1  GND
2  NC     ? via 100ohm to multiout pin 9    ;Audio Left (white cinch)
3  OUT-R  ?
4  MUTE1      ;specified as LOW = Mute
5  MUTE2      ;specified as HIGH = Mute
6  MUTEC      ;unspecified, maybe capacitor, or output based on MUTE1+MUTE2?
7  IN-R     via capacitor to SPU.150
8  BIAS
9  NC
10 NC
11 IN-L     via capacitor to SPU.147
12 OUT-L  ?
13 NC     ? via 100ohm to multiout pin 11   ;Audio Right (red cinch)
14 VCC      +5.0V (via L401)
```

Audio amplifier, for raising the signals to 5V levels.

IC405 - "NJM2100E (TE2)" Audio Amplifier (on older PU-8 and PU-22 boards)

```
1-ROUT
2-RIN- IC732.SPU.150
3-RIN+
4-GND
5-LIN+
6-LIN- IC732.SPU.147
7-LOUT
8-VCC 4.9V (+5.0V via L401)
```

Pinouts - DRV Pinouts

IC304 - 52pin/80pin - Motorola HC05 8bit CPU

Pinouts - HC05 Pinouts

IC305 - SONY CXD1815Q - CDROM Decoder/FIFO (used on PU-8, PU-18)

```
1-D0    14-/XINT 27-/HRD  40-GND   53-VDD   66-/MWR  79-GND   92-LRCO
2-D1    15-GND   28-VDD   41-HDRQ  54-GND   67-MDB0  80-CLK   93-WCKO
3-VDD   16-A0    29-GND   42-/HAC  55-MA8   68-MDB1  81-HCLK  94-BCKO
4-GND   17-A1    30-/HWR  43-MA0   56-MA9   69-MDB2  82-CKSL  95-MUTE
5-D2    18-A2    31-HD0   44-MA1   57-MA10  70-MDB3  83-RMCK  96-TD7
6-D3    19-A3    32-HD1   45-MA2   58-MA11  71-MDB4  84-LRCK  97-TD6
7-D4    20-A4    33-HD2   46-T01   59-MA12  72-MDB5  85-DATA  98-TD5
8-D5    21-TD0   34-HD3   47-T02   60-MA13  73-MDB6  86-BCLK  99-TD4
9-D6    22-/HRS  35-HD4   48-MA3   61-MA14  74-MDB7  87-C2PO  100-TD3
10-D7   23-/HCS  36-HD5   49-MA4   62-MA15  75-MDBP  88-EMP
11-/CS  24-HA0   37-HD6   50-MA5   63-MA16  76-XTL2  89-/RST
12-/RD  25-HA1   38-HD7   51-MA6   64-/MOE  77-XTL1  90-GND
13-/WR  26-HINT  39-HDP   52-MA7   65-GND   78-VDD   91-DATO
```

Pin 1..20 to HC05 CPU, pin 22..42 to MIPS cpu, pin 43..75 to SRAM cd-buffer.
The pinouts/registers in CXD1199AQ datasheet are about 99% same as CXD1815Q.
Note: Parity on the 8bit data busses is NC. SRAM is 32Kx8 (A15+A16 are NC). Later boards have this integrated in the SPU.

```
1-FEO    7-FE_M   13-RA_O  19-CLK    25-FOK   31-RF_O  37-FE_BIAS 43-LPFI
2-FEI    8-SRCH   14-SL_P  20-XLT    26-CC2   32-RF_M  38-F       44-TEI
3-FDFCT  9-TGU    15-SL_M  21-DATA   27-CC1   33-LD    39-E       45-ATSC
4-FGD    10-TG2   16-SL_O  22-XRST   28-CB    34-PD    40-EI      46-TZC
5-FLB    11-FSET  17-ISET  23-C.OUT  29-CP    35-PD1   41-GND     47-TDFCT
6-FE_O   12-TA_M  18-VCC   24-SENS   30-RF_I  36-PD2   42-TEO     48-VC
```

Datasheet exists. Later boards have CXA1782BR+CXD2510Q integrated in CXD2545Q, and even later boards have it integrated in the SPU.

IC309 - SONY CXD2510Q - CDROM Signal Processor (used on PU-8 boards)

```
1-FOK   11-PDO   21-GNDa 31-WDCK        41-DA09-XPLCK 51-APTL 61-EMPH 71-DATA
2-FSW   12-GND   22-VLTV 32-LRCK        42-DA08-GFS   52-GND  62-WFCK 72-XLAT
3-MON   13-TEST0 23-VDDa 33-VDD 5V      43-DA07-RFCK  53-XTAI 63-SCOR 73-VDD
4-MDP   14-NC    24-RF   34-DA16-SDTA48 44-DA06-C2PO  54-XTAO 64-SBSO 74-CLOK
5-MDS   15-NC    25-BIAS 35-DA15-SCLK48 45-DA05-XRAOF 55-XTSL 65-EXCK 75-SEIN
6-LOCK  16-VPCO  26-ASYI 36-DA14-SDTA64 46-DA04-MNT3  56-FSTT 66-SQSO 76-CNIN
7-NC    17-VCKI  27-ASYO 37-DA13-SCLK64 47-DA03-MNT2  57-FSOF 67-SQCK 77-DATO
8-VCOO  18-FILO  28-ASYE 38-DA12-LRCK64 48-DA02-MNT1  58-C16M 68-MUTE 78-XLTO
9-VCOI  19-FILI  29-NC   39-DA11-GTOP   49-DA01-MNT0  59-MD2  69-SENS 79-CLKO
10-TEST 20-PCO   30-PSSL 40-DA10-XUGF   50-APTR       60-DOUT 70-XRST 80-MIRR
```

Datasheet exists. Later boards have CXA1782BR+CXD2510Q integrated in CXD2545Q, and even later boards have it integrated in the SPU.

IC701 - SONY CXD2545Q - Signal Processor + Servo Amp (used on PU-18 boards)

```
1-SRON  14-TEST 27-TE   40-VDDa        53-DA09-XPLCK 66-FSTI 79-MUTE 92-DFCT
2-SRDR  15-GND  28-SE   41-VDD         54-DA08-GFS   67-FSTO 80-SENS 93-FOK
3-SFON  16-TES2 29-FE   42-ASYE        55-DA07-RFCK  68-FSOF 81-XRST 94-FSW
4-TFDR  17-TES3 30-VC   43-PSSL        56-DA06-C2PO  69-C16M 82-DIRC 95-MON
5-TRON  18-PDO  31-FILO 44-WDCK        57-DA05-XRAOF 70-MD2  83-SCLK 96-MDP
6-TRDR  19-VPCO 32-FILI 45-LRCK        58-DA04-MNT3  71-DOUT 84-DFSW 97-MDS
7-TFON  20-VCKI 33-PCO  46-DA16-SDTA48 59-DA03-MNT2  72-EMPH 85-ATSK 98-LOCK
8-FFDR  21-VDDa 34-CLTV 47-DA15-SCLK48 60-DA02-MNT1  73-WFCK 86-DATA 99-SSTP
9-DRON  22-IGEN 35-GNDa 48-DA14-SDTA64 61-DA01-MNT0  74-SCOR 87-XLAT 100-SFDR
10-FRDR 23-GNDa 36-RFAC 49-DA13-SCLK64 62-XTAI       75-SBSO 88-CLOK
11-FFON 24-ADIO 37-BIAS 50-DA12-LRCK64 63-XTAO       76-EXCK 89-COUT
12-VCOO 25-RFC  38-ASYI 51-DA11-GTOP   64-XTSL/GNDed 77-SQSO 90-VDD
13-VCOI 26-RFDC 39-ASYO 52-DA10-XUGF   65-GND        78-SQCK 91-MIRR
```

Datasheet exists. The CXD2545Q combines the functionality of CXA1782BR+CXD2510Q from older boards (later boards have it integrated in the SPU).
XTAI/XTAO input is 16.9344MHz (44.1kHz*180h), with XTSL=GND. Clock outputs are FSTO=16.9344MHz/3, FSOF=16.9344MHz/4, C16M=16.9344MHz/1.

IC720 - 144pin SONY CXD1817R (=CXD2545Q+CXD1815Q) ;PU-20

```
1..144 - unknown
```

IC701 - 8pin chip (on bottom side, but NOT installed) (PU-7 and EARLY-PU-8)

```
1-8 Unknown (maybe CDROM related, at least it's near other CDROM chips)
```

IC722 "BA5947FP" or "Panasonic AN8732SB" - IC for Compact Disc Players

Drive Motor related.

```
1 to pin24,27
2 SPINDLE           - via 15K to SPU21
3 SW (ON/OFF)       - IC304.27
4 TRACKING FORWARD
5 TRACKING REVERSE
6 FOCUS FORWARD
7 FOCUS REVERSE
8 GND               - CN702 pin 11
9 NC (INTERNAL)     - via C731 (10uF) to GND
10 +7.5V (Pow VCC ch1,2)
11 FOCUS COIL (1)   - CN702 pin 15
12 FOCUS COIL (2)   - CN702 pin 14
13 TRACKING COIL (1) - CN702 pin 16
14 TRACKING COIL (2) - CN702 pin 13
15 SPINDLE MOTOR (1) - CN701 pin 4
16 SPINDLE MOTOR (2) - CN701 pin 3
17 SLED MOTOR (1)   - CN701 pin 1
18 SLED MOTOR (2)   - CN701 pin 2
19 +7.5V (Pow VCC ch3,4)
20 MUTE             - /RES (via 5K6)
21 GND
22 SLED REVERSE
23 SLED FORWARD
24 to pin1
25 via capacitors to pin1
26 BIAS 1.75V
27 to pin1
28 +7.5V (Pre VCC)
```

Additionally to the above 28pins, the chip has two large grounded pins (between pin 7/8 and 21/22) for shielding or cooling purposes.

IC703 - 20pin - "SONY CXA1791N" (RF Amplifier) (on PU-18 boards)

```
1 LD       O APC amplifier output
2 PD       I APC amplifier input
3 PD1      I Input 1 for RF I-V amplifiers
4 PD2      I Input 2 for RF I-V amplifiers
5 GND/VEE  - Supply Ground
6 F        I Input F for I-V amplifier
7 E        I Input E for I-V amplifier
8 VR       O DC Voltage Output (VCC+VEE)/2
9 VC       I Center Voltage Input
10 NC      - NC
11 NC      - NC
12 EO      O Monitoring Output for I-V amplifier E
13 EI      - Gain Adjust for I-V amplifier E
14 TE      O Tracking Error Amplifier Output
15 FE_BIAS I BIAS Adjustment for Focus Error
16 FE      O Focus Error Amplifier Output
17 RFO     O RF Amplifier Output
18 RFI     I RF Amplifier Input
19 /LD_ON  I APC amplifier ON=GND, OFF=VCC
20 VCC     - Supply
```

Datasheet for CXA1791N does exist. Later boards have IC703 replaced by IC723. Older PU-7/PU-8 boards appear to have used a bunch of smaller components (8pin chips and/or transistors) instead of 20pin RF amplifiers.

IC723 - 20pin - "SONY CXA2575N-T4" (RF (Matrix?) Amplifier)
XXX below may be wrong... or maybe 1..10 and 11..20 are reversed...?

```
1-TEIM
2-TEIG
3-VEE     GND
4-E       via 33K to CN702 pin 4
5-F       via 33K to CN702 pin 8
6-PD2     via 36K to CN702 pin 6
7-PD1     via 36K to CN702 pin 7
8-PD      to CN702 pin 9
9-LD
10-VC     CL710, and CN702.Pin3, and via resistor?/diode? to SPU42
11-VCC      IC304.Pin49 "LDON"  ..... XXX or is that Pin 20 "LD_ON" ?
12-MIRR                                                    ;or AL/TE?
13-COMP+  CL704, and...
14-TE0
15-TE     CL708, and...                (maybe tracking error?)  ;or FE ?
16-FE     CL709, and via 15K to SPU.39  (maybe focus error?)    ;or TE ?
17-RFM                                                     ;or TEO ?
18-RF0
19-G_CONT via 4K7 to SPU66
20-LD_ON  3.48V (not 3.5V)
```

Used only on newer boards (PU-18 boards used IC703 "CXA1791N", and even older boards... maybe had this in CXA1782BR... or maybe had it in a bunch of 8pin NJMxxxx chips?).
There is no CXA2575N datasheet (but maybe some signals do resemble CXA2570N/CXA2571N/CXA1791N datasheets).

CN702 CDROM Data Signal socket (PU-23 and PM-41 board)

```
1-LD     to Q701
2-VCC    to Q701
3-VC     to IC723.Pin10 (and CL710)
4-F-     to IC723.Pin4 (via 33K ohm)
5-NC     to CL776
6-PD2    to IC723.Pin6 (via 33K ohm)
7-PD1    to IC723.Pin7 (via 33K ohm)
8-E-     to IC723.Pin5 (via 33K ohm)
9-M1     to IC723.Pin8
10-VR    via 91 ohm to GND
11-GND   GND
12-LS    /POS0 (switch, GNDed when at head is at inner-most position)
13-FCS+  TRACKING COIL (2)   ;\
14-TRK+  FOCUS COIL (2)      ; or swapped?
15-TRK   FOCUS COIL (1)      ;
16-FCS   TRACKING COIL (1)   ;/
```

PU-23 and PM-41 board seem to be using exactly the same Drive, the only difference is the length (and folding) of the attached cable.

CN701 CDROM Motor socket (PU-8, PU-18, PU-23, PM-41 boards)

```
1-SL-   SLED MOTOR (1)
2-SL+   SLED MOTOR (2)
3-SP+   SPINDLE MOTOR (2)
4-SP-   SPINDLE MOTOR (1)
```

CLnnn - Calibration Points (PU-23 and PM-41 boards)

```
CL616 +7.5V (PM-41 only, not PM-23) (before power switch)
CL617 GND   (PM-41 only, not PM-23)
CL316 to IC304 pin 21
CL704 to IC723.Pin13
CL706 GND
CL708 to IC723.Pin15
CL709 to IC723.Pin16
CL710 to IC723.Pin10, and CN702.Pin3
CL711 via 1K to IC723.Pin15
CL776 to CN702.Pin5
```

Probably test points for drive calibration or so.

Pinouts - HC05 Pinouts

Motorola HC05 chip versions for PSX cdrom control

```
80pin "4246xx" - MC68HC05L16, on-chip ROM (DTL-H120x & old retail consoles)
80pin "MC68HC705L16CFU" - MC68HC705L16, on-chip ROM (DTL-H100x, and PU-9)
52pin "SC4309xx" - MC68HC05G6, on-chip ROM (newer retail consoles)
```

The early DTL-H2000 devboard is also using a 80pin CPU (with piggyback EPROM socket), but that CPU is a Sony CXP82300 SPC700 CPU, not a Motorola HC05 CPU.

IC304 - "C 3060, SC430943PB, G63C 185" (PAL/PSone) - CDROM Controller
Called "MC68HC05G6PB" in service manual (=8bit CPU).

```
1  NC     NC          ;-Port F (lower bits on Pin 50-52)        ;PortF.Bit3
2  VDD    3.5V
3  NC     NC          ;\                                  ;maybe PortE.Bit7?
4  NC     NC          ; maybe MSBs of Port E              ;maybe PortE.Bit6?
5  NC     NC          ;/                                  ;maybe PortE.Bit5?
6  DECA4  SPU102      ;\                                        ;PortE.Bit4
7  DECA3  SPU101      ; Port E [04h], aka Address/Index         ;PortE.Bit3
8  DECA2  SPU99       ;                                         ;PortE.Bit2
9  DECA1  SPU98       ;                                         ;PortE.Bit1
10 DECA0  SPU97       ;/                                        ;PortE.Bit0
11 VSS    GND
12 NDLY   GND     reserved for factory test, should be wired to VDD, not GND?
13 /RES   /RES (via 5K6)
14 OSC1   4.3MHz (SPU11)(used as external clock for some modchips)(low volts)
15 OSC2   NC
16 F-BIAS         aka FOK=NC (in SCPH-5500)                   ;PortB.Bit0
17 CG     NC      aka CG=CG (in SCPH-5500)     ;this IS portb.1! ;PortB.Bit1
18 LMTSW  /POS0 (switch, GNDed when head at inner-most position) ;PortB.Bit2
19 DOOR   SHELL_OPEN                                          ;PortB.Bit3
20 TEST2  NC                                                  ;PortB.Bit4
21 TEST1  to CL316                                            ;PortB.Bit5
22 COUT              NC                                       ;PortB.Bit6
```

```
24 SUBQ   SPU81  ;CXD2510Q.66                                  ;PortC.Bit0
25 NC     NC     ;NC                                          ;PortC.Bit1
26 SQCK   SPU80  ;CXD2510Q.67                                  ;PortC.Bit2
27 SPEED  IC722.Pin3 (SW)                                      ;PortC.Bit3
28 AL/TE         ;transisor aka MIRROR=..   (in SCPH-5500);ISN'T PortB.Bit1 !
29 ROMSEL        ;NC        aka ROMSEL=SCLK (in SCPH-5500)      ;PortC.Bit5
30 /XINT  SPU79  ;CXD1815Q.14                                  ;PortC.Bit6
31 SCOR   SPU77  ;CXD2510Q.63                                  ;PortC.Bit7
32 VDD    3.5V
33 DECD0  CD.D0       ;\                                       ;PortA.Bit0
34 DECD1  CD.D1       ;                                        ;PortA.Bit1
35 DECD2  CD.D2       ;                                        ;PortA.Bit2
36 DECD3  CD.D3       ; Port A [00h], aka Data                 ;PortA.Bit3
37 DECD4  CD.D4       ;                                        ;PortA.Bit4
38 DECD5  CD.D5       ;                                        ;PortA.Bit5
39 VSS    GND         ;
40 DECD6  CD.D6       ;                                        ;PortA.Bit6
41 DECD7  CD.D7       ;/                                       ;PortA.Bit7
42 NC     NC                                          ;maybe PortD.Bit0?
43 DATA   SPU74 (via 22 ohm)                                   ;PortD.Bit1
44 XLAT   SPU75 (via 22 ohm)                                   ;PortD.Bit2
45 CLOK   SPU76 (via 22 ohm)                                   ;PortD.Bit3
46 DECCS  SPU94                                                ;PortD.Bit4
47 DECWR  SPU95                                                ;PortD.Bit5
48 DECRD  SPU96                                                ;PortD.Bit6
49 LDON   IC723.Pin11                                          ;PortD.Bit7
50 NC     NC                  ;\                               ;PortF.Bit0
51 NC     NC                  ; PortF (fourth bit on Pin1)     ;PortF.Bit1
52 NC     NC                  ;/                               ;PortF.Bit2
```

This chip isn't connected directly to the CPU, but rather to a Fifo Interface, which is then forwarding data to/from the CPU. On older PSX boards, that Fifo Interface seems to have been located in a separate chip, on newer PSX boards and PSone boards, the Fifo stuff is contained in the SPU chip. The CDROM has a 32K buffer, which is also implemeted at the Fifo Interface side.

OSC input (internally HC05 is running at OSC/2, ie. around 2MHz):

```
PU-8      4.0000MHz from separate 4.000MHz oscillator (X302)
DTL-H2000 4.1900MHz from separate 4.1900MHz oscillator (SPC700, not HC05)
PU-18     4.2336MHz from CXD2545Q.pin68 (Servo+Signal) (FSOF=16.9344MHz/4)
PU-20     4.2xxxMHz from CXD1817R.pin?  (Servo+Signal+Decoder)
PM-41     4.2xxxMHz from CXD2938Q.pin11 (Servo+Signal+Decoder+SPU)
```

HC05 - 80pin version (pinout from MC68HC05L16 datasheet)

```
1 VDD
2 FP28/PE6    ;\
3 FP29/PE5    ;
4 FP30/PE4    ;
5 FP31/PE3    ; Port E LSBs
6 FP32/PE2    ;
7 FP33/PE1    ;
8 FP34/PE0    ;/
9 FP35/PD7    ;\
10 FP36/PD6   ; Port D MSBs
11 FP37/PD5   ;
12 FP38/PD4   ;/
13 VLCD3
14 VLCD2
15 VLCD1
16 VSS
17 NDLY
18 XOSC1
19 XOSC2
20 /RESET
---
21 OSC1
22 OSC2
23 PA0        ;\
24 PA1        ;
25 PA2        ;
26 PA3        ; Port A
27 PA4        ;
28 PA5        ;
29 PA6        ;
30 PA7        ;/
31 PB0/KWI0   ;\
32 PB1/KWI1   ;
33 PB2/KWI2   ;
34 PB3/KWI3   ; Port B
35 PB4/KWI4   ;
36 PB5/KWI5   ;
37 PB6/KWI6   ;
38 PB7/KWI7   ;/
39 PC0/SDI    ;\
40 PC1/SDO    ;
---           ;
41 PC2/SCK    ; Port C
42 PC3/TCAP   ;
43 PC4/EVI    ;
44 PC5/EVO    ;
45 PC6/IRQ2   ;
46 PC7/IRQ1   ;/
47 VDD
48 BP3/PD3            ;\
49 BP2/PD2            ; Port D LSBs
50 BP1/PD1            ;
51 BP0 (no "PD0")     ;/
52 FP0
53 FP1
54 FP2
55 FP3
56 FP4
57 FP5
58 FP6
59 FP7
60 VSS
```

```
61 FP8
62 FP9
63 FP10
64 FP11
65 FP12
66 FP13
67 FP14
68 FP15
69 FP16
70 FP17
71 FP18
72 FP19
73 FP20
74 FP21
75 FP22
76 FP23
77 FP24
78 FP25
79 FP26
80 FP27/PE7   ;- Port E MSB
```

HC05 - 32pin Version
Sony's Digital Joypad and Mouse contain 32pin CPUs, which are probably also HC05's:
Pinouts - Component List and Chipset Pin-Outs for Digital Joypad

Pinouts - MEM Pinouts

IC102 - BIOS (32pin, used on LATE-PU-8 boards, and newer boards)

```
1-A19  5-A7  9-A3   13-D0   17-D3  21-D7   25-A11  29-A14
2-A16  6-A6  10-A2  14-D1   18-D4  22-/CE  26-A9   30-A17     ;/CE=/BIOS
3-A15  7-A5  11-A1  15-D2   19-D5  23-A10  27-A8   31-A18
4-A12  8-A4  12-A0  16-GND  20-D6  24-/OE  28-A13  32-3.5V    ;/OE=/RD
```

Uses standard EPROM pinouts, VCC is 3.5V though, when replacing the ROM by an EPROM, it may be required to replace the supply by 5V. Note that, on PM-41 boards at least, Pin 1 is connected to A19 (allowing to install a 1MB BIOS chip on that board, however, normally, a 512KB BIOS chip is installed, and, the CPU is generating an exception when trying to access more than 512KB, but that 512K limit can be disabled via memory control registers).

IC102 - BIOS ROM (40pin, used on PU-7 boards, and EARLY-PU-8 boards)
The 40pin pin-out was found in a ROM-replacement circuit in the internet. It's no joke: 40pin ROMs can be actually seen on photos of PU-7 and PU-8 mainboards.

```
1-A18   6-A4   11-GND  16-    21-VCC  26-D6  31-     36-A13
2-A8    7-A3   12-/OE  17-D2  22-D4   27-    32-A17  37-A12
3-A7    8-A2   13-D0   18-    23-     28-D7  33-A16  38-A11
4-A6    9-A1   14-     19-D3  24-D5   29-A0  34-A15  39-A10
5-A5    10-/CS 15-D1   20-    25-     30-    35-A14  40-A9
```

The nine "blank" pins seem to be unused?

CPU-RAM (four 28pin chips) (older boards)
Unknown.
Note: The newer 70pin RAM comes up without external /REFRESH signal, but maybe the 28pin RAMs required refresh (the CPU has some odd delays once and when).

IC106 - CPU-RAM (single 70pin chip, on newer boards)
"Samsung K4Q153212M-JC60" (70pin, 512Kx32) (newer boards)
"Toshiba T7X16" (70pin, 512Kx32) (newer boards, too)

```
1-VCC   11-N.C   21-DQ15  31-A3   41-N.C   51-DQ17  61-DQ24
2-DQ0   12-VCC   22-N.C   32-A4   42-N.C   52-DQ18  62-DQ25
3-DQ1   13-DQ8   23-N.C!  33-A5   43-/OE   53-DQ19  63-DQ26
4-DQ2   14-DQ9   24-N.C   34-A6   44-/W    54-VSS   64-DQ27
5-DQ3   15-DQ10  25-N.C   35-VCC  45-/CAS3 55-DQ20  65-VSS
6-VCC   16-DQ11  26-N.C   36-VSS  46-/CAS2 56-DQ21  66-DQ28
7-DQ4   17-VCC   27-/RAS  37-A7   47-/CAS1 57-DQ22  67-DQ29
8-DQ5   18-DQ12  28-A0    38-A8   48-/CAS0 58-DQ23  68-DQ30
9-DQ6   19-DQ13  29-A1    39-A9   49-N.C   59-VSS   69-DQ31
10-DQ7  20-DQ14  30-A2    40-N.C  50-DQ16  60-N.C   70-VSS
```

Notes: Pin23 must NC or VSS. In the PSone, /OE is wired to GND.
Datasheet for K4Q153212M-JC60 does exist (the chip supports 27ns Hyper Page mode access, which seems to be used for DMA).

IC106/IC107/IC108/IC109 - CPU-RAM (four 28pin chips, on PU-8, PU-18 boards)
SEC KM48V514BJ-6 (DRAM 512Kx8) (four pieces = 512Kx32)

```
1-VCC  5-DQ3   9-A9     13-A3     17-A5  21-NC    25-DQ5
2-DQ0  6-NC    10-A0    14-VCC    18-A6  22-/OE   26-DQ6
3-DQ1  7-/W    11-A1    15-GND    19-A7  23-/CAS  27-DQ7
4-DQ2  8-/RAS  12-A2    16-A4     20-A8  24-DQ4   28-GND
```

Datasheet for KM48V514B-6 and BL-6 exist (though none for BJ-6). The chips support 25ns Hyper Page mode access.

IC310 - SPU-RAM (512Kbyte)
EliteMT M11B416256A-35J (256K x 16bit) (40pin SOJ, PM-41 boards)
Nippon Steel NN514256ALTT-50 (256K x 16bit) (40pin TSOP-II, PU-23 boards)
Toshiba TC51V4260DJ-70 (40pin, PU-8 board) (PseudoSRAM)

```
1-5.0V  6-5.0V   11-NC   16-A0    21-VSS  26-A8    31-I/O8   36-I/O12
2-I/O0  7-I/O4   12-NC   17-A1    22-A4   27-/OE   32-I/O9   37-I/O13
3-I/O1  8-I/O5   13-/WE  18-A2    23-A5   28-/CASH 33-I/O10  38-I/O14
4-I/O2  9-I/O6   14-/RAS 19-A3    24-A6   29-/CASL 34-I/O11  39-I/O15
5-I/O3  10-I/O7  15-NC   20-5.0V  25-A7   30-NC    35-VSS    40-VSS
```

Note: SPU-RAM supply can be 3.5V (PU-8), or 5.0V (PU-22 and PM-41).
Note: The /CASL and /CASH pins are shortcut with each other on the mainboard, both wired to the /CAS pin of the SPU (ie. always accessing 16bit data at once).
Note: The TSOP-II package (18mm length, super-flat and with spacing between pin 10/11 and 30/31) is used on PU-23 boards. The pinouts and connections are identical for SOJ and TSOP-II.
Note: Nippon Steels NN514256-series is normally 256Kx4bit, nethertheless, for some bizarre reason, their 256Kx16bit chip is marked "NN514256ALTT"... maybe that happened accidently in the manufacturing process.
Note: The PM-41(2) board has on-chip RAM in the SPU (no external memory chip).

IC303 - CDROM Buffer (32Kbyte)

"SONY CXK5V8257BTM" 32Kx8 SRAM (PU-18)

```
1-A14  4-A6  7-A3  10-A0  13-D2   16-D4  19-D7   22-/OE  25-A8   28-VCC
2-A12  5-A5  8-A2  11-D0  14-GND  17-D5  20-/CS  23-A11  26-A13
3-A7   6-A4  9-A1  12-D1  15-D3   18-D6  21-A10  24-A9   27-/WE
```

Used only on older boards (eg. PU-8, PU-18), newer boards seem to have that RAM included in the 208pin SPU chip.

IC201 - GPU-RAM (1MByte) (or 2MByte, of which, only 1MByte is used though)
Samsung KM4132G271BQ-10 (128K x 32bit x 2 Banks, Synchronous Graphic RAM) 1MB
Samsung K4G163222A-PC70 (256K x 32bit x 2 Banks, Synchronous Graphic RAM) 2MB

```
1-DQ3   13-DQ19  25-/WE     37-N.C 49-A6    61-DQ9   73-VDDQ  85-VSS  97-DQ0
2-VDDQ  14-VDDQ  26-/CAS    38-N.C 50-A7    62-VSSQ  74-DQ24  86-N.C  98-DQ1
3-DQ4   15-VDD   27-/RAS    39-N.C 51-A8    63-DQ10  75-DQ25  87-N.C  99-VSSQ
4-DQ5   16-VSS   28-/CS     40-N.C 52-N.C   64-DQ11  76-VSSQ  88-N.C  100-DQ2
5-VSSQ  17-DQ20  29-A9(BA)  41-N.C 53-DSF   65-VDD   77-DQ26  89-N.C
6-DQ6   18-DQ21  30-NC(GND) 42-N.C 54-CKE   66-VSS   78-DQ27  90-N.C
7-DQ7   19-VSSQ  31-A0      43-N.C 55-CLK   67-VDDQ  79-VDDQ  91-N.C
8-VDDQ  20-DQ22  32-A1      44-N.C 56-DQM1  68-DQ12  80-DQ28  92-N.C
9-DQ16  21-DQ23  33-A2      45-N.C 57-DQM3  69-DQ13  81-DQ29  93-N.C
10-DQ17 22-VDDQ  34-A3      46-VSS 58-NC    70-VSSQ  82-VSSQ  94-N.C
11-VSSQ 23-DQM0  35-VDD     47-A4  59-VDDQ  71-DQ14  83-DQ30  95-N.C
12-DQ18 24-DQM2  36-N.C     48-A5  60-DQ8   72-DQ15  84-DQ31  96-VDD
```

Newer boards often have 2MB VRAM installed (of which only 1MB is used, apparently the 2MB chips became cheaper than the 1MB chips). At the chip side, the only difference is that Pin30 became an additional address line (that, called A8, and, accordingly, the old A8,A9 pins were renamed to A9,A10). At the mainboard side, the connection is exactly the same for both 1MB and 2MB chips; Pin30 is grounded on both PU-23 boards (which typically have 1MB) and PM-41 boards (which typically have 2MB).
Note: The PM-41(2) board has on-chip RAM in the GPU (no external memory chip).

Pinouts - CLK Pinouts

The "should-be" CPU clock is 33.868800 Hz (ie. the 44100Hz CDROM/Audio clock, multiplied by 300h). However, the different PSX/PSone boards are using different oscillators, multipliers and dividers, which aren't exactly reaching that "should-be" value. The PSone are using a single oscillator for producing CPU/GPU clocks, and for producing the TV/color signal:

```
For PAL,  Fsc=4.43361875MHz (5^6*283.75Hz+25Hz) --> 4*Fsc=17.734MHz
For NTSC, Fsc=3.579545MHz   (4.5*455/572 MHz)   --> 4*Fsc=14.318MHz
```

PSone/PAL - IC204 8pin - "2294A, 1913" - Clock Multiplier/Divider

```
1 53MHz          ;17.734MHz*3 = 53.202 MHz (?)
2 GND
3 X1 17.734MHz
4 X2 17.734MHz
5 67MHz          ;17.734MHz*3*2*7/11 = 67.711636 MHz (?)
6 4.4Mhz         ;17.734MHz/4 = 4.4335MHz  (?)  ;via 2K2 to IC502.pin15
7 3.5V
8 3.5V
```

PSone/NTSC - IC204 8pin
Unknown. Uses a 14.318MHz oscillator, so multiply/divide factors must be somehow different.

```
3*3*7*5/2/11 = 14.3181818
3*3*7*7*100  = 44100
```

The "optimal" conversion would be (hardware is barely able to do that):

```
14.3181818 * 3*7*11*64 / (5*5*5*5*5) = 67.737600
```

So, maybe it's doing

```
14.3181818 * 2*2*13/11   ... or so?
```

PSX/PAL
PU-7 and PU-8 boards are using three separate oscillators:

```
X101: 67.737MHz (div2 = CPU Clock = 33.8685MHz) (div600h = 44.1kHz audio)
X201: 53.20MHz (GPU Clock) (div12 = PAL color clock)
X302: 4.000MHz (for CDROM SUB CPU)
```

PU-18 does have same X101/X201 as above, but doesn't seem to have X302.

PSX/NTSC
PU-7 and PU-8 boards are using three separate oscillators:

```
X101: 67.737MHz (div2 = CPU Clock = 33.8685MHz) (div600h = 44.1kHz audio)
X201: 53.69MHz (GPU Clock) (div15 = NTSC color clock)
X302: 4.000MHz (for CDROM SUB CPU)
```

Pinouts - PWR Pinouts

Voltage Summary

```
+7.5V  Used to generate other voltages and CDROM/Joypad/MemoryCard/Expansion
+5.0V  Used for Multiout, IC405, and IC502, and IC602
+3.5V  Used for most ICs, and for Joypad/MemoryCard/Expansion
+3.48V Used for SPU and CDROM
GND    Ground, shared for all voltages
```

Fuses
There are a lot of SMD elements marked FBnnn, these are NOT fuses (at least they don't seem to blow-up whatever you do). The actual fuses are marked PSnnn, found near the power switch and near the power socket.

IC601 3pin +5.0V "78M05, RZ125, (ON)"

```
1 +7.5V
2 GND
3 +5.0V   (used for Multiout, IC405, and IC502)
```

IC602 - Audio/CDROM Supply
Called "LP29851MX-3.5" in service manual.

```
1 VIN    5.0V (in)
2 GND    GND
3 ON/OFF 5.0V (in)
```

```
5 VOUT   3.48V (out)
```

IC002/IC003 - Reset Generator (PM-41 board)

```
IC002  IC003  Expl.
2      2      connected to Q002 (reset input?)
5      5      connected via capacitor to GND
6      1      reset-output (IC002=wired to /RES, IC003: via Q004 to /RES)
7      -      7.5V
4      3      GND
1,3,8  4      NC
```

/RES is connected via 330 ohm to GPU/CPU, and via 5K6 SPU/IC722/IC304.
Note: Either IC002 or IC003/Q004 can be installed on PM-41 boards. Most or all boards seem to contain IC003/Q004.
Note: PSX consoles have something similar on the Power Supply boards (IC101: M51957B).

IC606/IC607 - TL594CD - Pulse-Width-Modulation Power-Control Chip

```
1 1IN+
2 1IN-
3 FEEDBACK
4 DTC
5 CT
6 RT
7 GND
8 C1
9  E1
10 E2
11 C2
12 VCC
13 OUTPUT CTRL
14 REF
15 2IN-
16 2IN+
```

Q602

```
x +7.5V
y +3.5V
z REG
```

CN602 - PU-8, PU-9 board Power Socket (to internal power supply board)

```
1 Brown   7.5V (actually 7.69V)
2 Red     GND  Ground
3 Orange  3.5V (actually 3.48V)
4 Yellow  GND  Ground
5 White   STAND-BY (3.54V, always ON, even if power switch is off)
6 Blue    GND  Ground
7 Magenta /RES Reset input (from power-on logic and reset button)
```

Purpose of the standy-by voltage is unknown... maybe to expansion port?

CN602 - PU-18, PU-23 board Power Socket (to internal power supply board)

```
1 Brown   7.5V (actually 7.92V or so) (ie. higher than in PSone)
2 Red     GND  Ground
3 Orange  3.5V (actually 3.53V or so) (ie. quite same as PSone)
4 Yellow  GND  Ground
5 White   /RES Reset input (from power-on logic and reset button)
```

CN102 - Controller/memory card daughter-board connector (PU-23 board)

```
1 /IRQ10 (/IRQ10)
2 /ACK (/IRQ7)
3 /JOY2
4 7.5V (or actually 7.92V)
5 /JOY1
6 DAT
7 GND
8 CMD
9 3.5V
10 CLK
```

Pinouts - Component List and Chipset Pin-Outs for Digital Joypad

Digital Joypad Component List

```
Case: "SONY, CONTROLLER, Sony Computer Entertainment Inc. H"
Case: "SCPH-1080 Made in China"
PCB: "CMK-PIHB /\, CFS8121-200010-01"
U?: 32pin "(M), SC401800, FB C37B, JSJD520C" (Motorola) (TQFP-32 package)
U?: 14pin "BA10339F, 528 293" (Quad Comparator) (/ACK,JOYDAT,and reset or so)
X?:  3pin "4.00G1f" (on PCB bottom side)
Z1:  2pin z-diode or so (on PCB bottom side) (+1.7V VREF for BA10339F)
CN?: 7pin cable to controller port (plus shield; but not connected to PCB)
C1   2pin   to GND and R5
C2   2pin capacitor for power supply input (between +3.5V and GND)
C3   2pin   between BA.pin8 and (via R6) BA.pin15
R1   2pin 1M ohm (for X1)
R2   2pin 2.7K
R3   2pin 8xK ohm?
R4   2pin 100K
R5   2pin 22K ohm
R6   2pin 56K ohm
RN1  8pin 4x200 ohm (/JOYn,JOYCMD,JOYCLK)
RN2  8pin 4x22K ohm (pull-ups for button bit0..3)
RN3  8pin 4x22K ohm (pull-ups for button bit12..15)
RN4  8pin 4x22K ohm (pull-ups for button bit8..11)
RN5  8pin 4x22K ohm (pull-ups for button bit4..7)
```

Digital Joypad Connection Cable:

```
PSX.1 -------brown---- PAD.2 JOYDAT
PSX.2 -------orange--- PAD.6 JOYCMD
PSX.3 ---              NC    +7.5V
PSX.4 -------black---- PAD.3 GND
```

```
PSX.6 -------yellow--- PAD.5 /JOYn
PSX.7 -------blue----- PAD.7 JOYCLK
PSX.8 ---              NC    /IRQ10
PSX.9 -------green---- PAD.1 /ACK
PSX.Shield --shield--- NC    (cable is shielded but isn't connected in joypad)
```

Digital Joypad 32pin SC401800 Chip Pin-Outs

```
1 Bit14 SW-X
2 Bit13 SW-O
3 Bit12 SW-/\
4 Bit11 SW-R1 (via cable pin1, white wire)
5 Bit10 SW-L1 (via cable pin1, white wire)
6 Bit9  SW-R2 (via cable pin3, black wire)
7 Bit8  SW-L2 (via cable pin3, black wire)
8 via BA10339F.pin7 to cn.2 JOYDAT (PSX.1)
---
9  via RN1 (200 ohm) to cn.5 /JOYn  (PSX.6)
10 via RN1 (200 ohm) to cn.6 JOYCMD (PSX.2)
11 via RN1 (200 ohm) to cn.7 JOYCLK (PSX.7)
12 GND to cn.3 (PSX.4)
13 Bit7 SW-LEFT
14 Bit6 SW-DOWN
15 Bit5 SW-RIGHT
16 via BA10339F.pin5 to cn.1 /ACK (PSX.9)
---
17 Bit4 SW-UP
18 Bit3 SW-START
19 Bit2 (HI) (would be R3 on Analog Pads) ;\unused, but working button inputs
20 Bit1 (HI) (would be L3 on Analog Pads) ;/(each fitted with a RN2 pullup)
21 Bit0 SW-SELECT
22
23
24 wired to SC401800.pin25
---
25 wired to SC401800.pin24
26 4.00MHz'a
27 4.00MHz'b
28 +3.5V to cn.4 (PSX.5)
29 wired to SC401800.pin32, and via 22K ohm to +3.5V, and to BA.14
30
31 Bit15 SW-[]
32 wired to SC401800.pin29
```

Digital Joypad 14pin BA10339F Chip Pin-Outs

```
1 OUT2  CN.2 JOYDAT (PSX.1)
2 OUT1  CN.1 /ACK (PSX.9)
3 VCC   +3.5V
4 -IN1  +1.7V VREF via Z1 to GND
5 +IN1  CXD.16 /ACK
6 -IN2  +1.7V VREF via Z1 to GND
7 +IN2  CXD.8  JOYDAT
---
8 -IN3  +1.7V VREF via Z1 to GND
9 +IN3      C3,R3,R4
10 -IN4     C1 to +3.5V
11 +IN4 GND
12 GND  GND
13 OUT4     NC ??
14 OUT3 CXD.29/32
```

Pinouts - Component List and Chipset Pin-Outs for Analog Joypad

Analog Joypad Component List

```
Case "SONY, ANALOG, CONTROLLER, SonyCompEntInc. H, SCPH-1200 MADE IN CHINA"
PCB1 "01, /\YG-H2, (r)RU" (mainboard with digital buttons)
PCB2 "M-29-01, YG-H3, (r)RU" (daughterboard with analog joysticks)
PCB3 "E, /\YG-H2, (r)RU, 01" (daughterboard with R-1, R-2 buttons) (J1)
PCB4 "01, W, /\YG-H2, (r)RU" (daughterboard with L-1, L-2 buttons) (J2)
U1 44pin "SONY, CXD8771Q 4A03, JAPAN 9840 HAL, 148896"
U2  4pin ",\\ 29" (PST9329) (System Reset with 2.9V detection voltage)
U3  8pin "2904, 8346G, JRC" (NJM2904) (Dual Operational Amplifier)
Q1  3pin ".Y S'" (big transistor for big M1 rumble motor)
Q2  3pin "Z" (small transistor for small M2 rumble motor)
Y1  3pin "800CMLX" or so (hides underneath of the CN2 ribbon cable)
CN1 8pin cable to PSX controller port
CN2 8pin ribbon cable to analog-joystick daughterboard
J1  3pin ribbon cable to R-1, R-2 button daughterboard
J2  3pin ribbon cable to L-1, L-2 button daughterboard
M1  2pin wires to big rumble motor
M2  2pin wires to small rumble motor
ZD1,ZD2 some Z-diodes
D1,D2 diodes near M1,M2 motors (these diodes aren't installed)
LED1  red analog mode LED (with transparent optics/light direction mirror)
plus resistors/capacitors
```

Analog Joypad Connection Cables

CN1 (cable to PSX controller port)

```
PSX.1 -------brown---- PAD.2 JOYDAT
PSX.2 -------orange--- PAD.6 JOYCMD
PSX.3 -------magenta-- PAD.8 +7.5V
PSX.4 -------black---- PAD.3 GND
PSX.5 -------red------ PAD.4 +3.5V
PSX.6 -------yellow--- PAD.5 /JOYn
PSX.7 -------blue----- PAD.7 JOYCLK
PSX.8 ---              NC    /IRQ10
PSX.9 -------green---- PAD.1 /ACK
PSX.Shield --shield--- NC    (cable is shielded but isn't connected in joypad)
```

CN2 (ribbon cable to analog-joystick daughterboard)

```
1 +3.5V to POT pins
```

```
  3 GND to POT pins and Button L3/R3 pins A,B
  4 Button R3 pins C,D
  5 Axis R_Y middle POT pin (CXD.20)
  6 Axis R_X middle POT pin (CXD.19)
  7 Axis L_X middle POT pin (CXD.21)
  8 Axis L_Y middle POT pin (CXD.22)
```

J1 (ribbon cable to R-1, R-2 button daughterboard)

```
  1 (red)  R1
  2 (gray) GND
  3 (gray) R2
```

J2 (ribbon cable to L-1, L-2 button daughterboard)

```
  1 (red)  L1
  2 (gray) GND
  3 (gray) L2
```

M1 wires to big rumble motor

```
  + (red)   Q1.E
  - (black) GND
```

M2 wires to small rumble motor

```
  + (red)   +7.5V
  - (black) Q2.C
```

Analog Joypad Chipset Pin-Outs

U1 SONY CXD8771Q

```
  1 PSX.7/CN1.7 JOYCLK (via 220 ohm R2)
  2 via R10 to U3.3 (for big M1 motor)
  3 via R15 to Q2.B (for small M2 motor)
  4 GND
  5 BUTTON Bit15 []
  6 BUTTON Bit14 ><
  7 BUTTON Bit13 ()
  8 BUTTON Bit12 /\
  9 BUTTON Bit11 R1 (via J1.1)
  10 BUTTON Bit10 L1 (via J2.1)
  11 BUTTON Bit9 R2 (via J1.3)
  ---
  12 BUTTON Bit8 L2 (via J2.3)
  13 GND
  14 U2.Pin3 (reset)
  15 Y1'a
  16 Y1'b
  17 GND
  18 +3.5V
  19 Analog Axis R_X via CN2.6
  20 Analog Axis R_Y via CN2.5
  21 Analog Axis L_X via CN2.7
  22 Analog Axis L_Y via CN2.8
  ---
  23 GND
  24 GND
  25 GND
  26 GND
  27 GND
  28 +3.5V
  29 BUTTON Bit0 SELECT
  30 BUTTON Bit1 L3 (via CN2.2)
  31 BUTTON Bit2 R3 (via CN2.4)
  32 BUTTON Bit3 START
  33 BUTTON Bit4 UP
  ---
  34 BUTTON Bit5 RIGHT (aka spelled RIHGT on the PCB)
  35 BUTTON Bit6 DOWN
  36 BUTTON Bit7 LEFT
  37 PSX.6/CN1.5./JOYn  (via 220 ohm R1)
  38 ANALOG BUTTON
  39 GND
  40 +3.5V
  41 /LED (to LED1, and from there via 300 ohm R6 to +3.5V)
  42 PSX.9/CN1.1./ACK   (via 22 ohm R5)
  43 PSX.1/CN1.2.JOYDAT (via 22 ohm R3)
  44 PSX.2/CN1.6 JOYCMD (via 220 ohm R4)
```

U2 PST9329 (System Reset with 2.9V detection voltage)

```
  1 NC   GND
  2 GND  GND
  3 Vout U1.14
  4 VCC  +3.5V
```

U3 NJM2904 (Dual Operational Amplifier)

```
  1 A.OUTPUT  Q1.B (big motor M1 transistor)
  2 A.INPUT-  to R11/R12
  3 A.INPUT+  to R10/R17
  4 GND       PSX.4/CN1.3 GND
  5 B.INPUT+  GND
  6 B.INPUT-  NC?
  7 B.OUTPUT  NC?
  8 VCC       PSX.3/CN1.8 +7.5V
```

Q1 (transistor for big M1 motor)

```
  E M1+
  B U3.1 (NJM2904)
  C +7.5V
```

Q2 (transistor for small M2 motor)

```
  E GND
  B via 1K ohm R15 to U1.3 (CXD), and via 100K ohm R16 to GND
  C M2-
```

Pinouts - Component List and Chipset Pin-Outs for Multitap

Multitap Component List

```
  Case "SONY, MULTITAP, SonyComputerEntertainmentInc, SCPH-1070 MADE IN CHINA"
  PCB1 "SONY 1-659-343-11" (mainboard with Slot A,B, ICs, X1, PSX-cable)
```

```
IC? 64pin "SONY JAPAN, CXD103, -166Q, 550D66E"           (smd/back side)
IC02 8pin "7W14, 5K" some tiny SMD chip (for JOYCLK)     (smd/back side)
X1   2pin "4.00G CMj" oscillator                         (front side)
J34  2pin fuse or 1 ohm resistor or so (for +3.5V input) (front side)
Jxx  2pin normal wire bridges (except: J34 is NOT a wire) (front side)
```

Cables from Multitap PCB1 to PCB2:

```
1pin black wire Shield/GND (lower edge)
1pin black wire Shield/GND (upper edge)
2x8pin red/gray ribbon cable (side edge)
2x2pin red/gray ribbon cable (lower edge)
2pin red/gray ribbon cable (upper middle) (gray=+3.3V, red=+7.5V)
```

plus a bunch of SMD capacitors and around 70 SMD resistors.

Multitap PSX Controller Port Cable

```
PSX.1 -------brown------ TAP.1 JOYDAT ;via  47 ohm (R57) to CXD.35
PSX.2 -------orange----- TAP.2 JOYCMD ;via 220 ohm (R58) to CXD.37
PSX.3 -------magenta---- TAP.3 +7.5V  ;directly to +7.5V on JOY/CARD's
PSX.4 -------black------ TAP.4 GND    ;directly to GND
PSX.5 -------red-------- TAP.5 +3.5V  ;via 1 ohm or so (J34) to +3.3V
PSX.6 -------yellow----- TAP.6 /JOYn  ;via 220 ohm (R59) to CXD.46
PSX.7 -------blue------- TAP.7 JOYCLK ;via 220 ohm (R60) to IC02.pin6
PSX.8 -------gray------- TAP.8 /IRQ10 ;via 47 ohm (R02/R16/R30/R44) to JOY's
PSX.9 -------green------ TAP.9 /ACK   ;via 47 ohm (R61) to CXD.51
PSX.Shield --shield----- TAP.shielding.plate (GND)
```

Multitap CARD A/B/C/D Slots

```
1 JOYDAT Via  47 ohm (R11/R25/R38/R5x) to CXD.18/29/60/5 (and to JOY slot)
2 JOYCMD Via 220 ohm (R10/R24/R39/R52) to CXD.19/30/62/6
3 +7.5V  Directly to PSX.3
4 GND    Directly to PSX.4
5 +3.3V  Via J34 to PSX.5 (+3.5V)
6 /JOYn  Via 220 ohm (R09/R2x/Rxx/R51) to CXD.11/22/52/61
7 JOYCLK Via 220 ohm (R08/R2x/Rxx/R50) to CXD.33/33/47/47
9 /ACK   Via  47 ohm (R07/R2x/Rxx/R49) to CXD.12/21/45/64
```

Multitap JOY A/B/C/D Slots

```
1 JOYDAT Via  47 ohm (R06/Rxx/R34/R5x) to CXD.18/29/60/5 (and to CARD slot)
2 JOYCMD Via 220 ohm (R05/R19/R35/R5x) to CXD.17/28/59/4
3 +7.5V  Directly to PSX.3
4 GND    Directly to PSX.4
5 +3.3V  Via 1 ohm or so (J34) to PSX.5 (+3.5V)
6 /JOYn  Via 220 ohm (R04/R18/R32/R4x) to CXD.16/20/55/63
7 JOYCLK Via 220 ohm (R03/R17/R31/R45) to CXD.15/23/56/2
8 /IRQ10 Via  47 ohm (R02/R16/R30/R44) to PSX.8
9 /ACK   Via  47 ohm (R01/R15/R29/R43) to CXD.13/27/54/7
Shield   Directly to Shield/GND
```

Multitap IC02 8pin "7W14, 5K" some tiny SMD chip

```
1
2
3
4 GND
5
6 via 220 ohm (R60) to PSX.7 (JOYCLK)
7 to CXD.Pin48
8 +3.3V, aka via 1 ohm (J34) to PSX.5 (+3.5V)
```

Multitap "SONY CXD103-166Q" Chip Pin-Outs (Multitap CPU)

```
1 via to 10K (R63) to +3.3V, and via C13 to GND (probably power-on reset)
2 JOY.D.7.JOYCLK
3
4 JOY.D.2.JOYCMD
5 JOY/CARD.D.1.JOYDAT
6 CARD.D.2.JOYCMD
7 JOY.D.9./ACK
8 4MHz X1/C12
9 4MHz X1/C11
10 GND
11 CARD.A.6./JOYn
12 CARD.A.9./ACK
13 JOY.A.9./ACK
14
15 JOY.A.7.JOYCLK
16 JOY.A.6./JOYn
17 JOY.A.2.JOYCMD
18 JOY/CARD.A.1.JOYDAT
19 CARD.A.2.JOYCMD
---
20 JOY.B.6./JOYn
21 CARD.B.9./ACK
22 CARD.B.6./JOYn
23 JOY.B.7.JOYCLK
24
25 GND
26 +3.3V
27 JOY.B.9./ACK
28 JOY.B.2.JOYCMD
29 JOY/CARD.B.1.JOYDAT
30 CARD.B.2.JOYCMD
31 GND
32
---
33 CARD.A/B.7.JOYCLK
34
35 PSX.1.JOYDAT
36
37 PSX.2.JOYCMD
38
39
40
41
```

```
43
44 GND
45 CARD.C.9./ACK
46 PSX.6./JOYn
47 CARD.C/D.7.JOYCLK
48 IC02.Pin7.PSX.JOYCLK
49
50
51 PSX.9./ACK
---
52 CARD.C.6./JOYn
53
54 JOY.C.9./ACK
55 JOY.C.6./JOYn
56 JOY.C.7.JOYCLK
57 GND
58 +3.3V
59 JOY.C.2.JOYCMD
60 JOY/CARD.C.1.JOYDAT
61 CARD.D.6./JOYn
62 CARD.C.2.JOYCMD
63 JOY.D.6./JOYn
64 CARD.D.9./ACK
```

Pinouts - Memory Cards

Sony Playstation Memory Card (SCPH-1020)
The "SONY CXD8732AQ" chip is installed on memory cards with "SPC02K1020B" boards, however, the text layer on the board says that it's an "LC86F8604A" chip. So, the CXD8732AQ is most probably a standard LC86F8604A chip (more on that below) with a Sony Memory Card BIOS ROM on it.
The "SONY CXD8732AQ" comes in a huge 64pin package, but it connects only to:

```
5 = /IRQ7   (via 22 ohm)         2 = /RESET (from U2)
6 = JOYCLK  (via 220 ohm)        30,31 = CF1,CF2 (12 clock pulses per 2us)
7 = /JOYn   (via 220 ohm)        14,16,25,32,38,39,61 = 3.5V (via 3.3 ohm)
12 = JOYCMD (via 220 ohm)        8,15,28,29 = GND
13 = JOYDAT (via 22 ohm)         All other pins = Not connected
```

Aside from that chip, the board additionally contains some resistors, capacitors, z-diodes (for protection against too high voltages), a 6MHz oscillator (for the CPU), and a 5pin reset generator (on the cart edge connector, the supply pins are slightly longer than the data signal pins, so when inserting the cartridge, power/reset gets triggered first; the 7.5V supply pin is left unconnected, only 3.5V are used).
Caution: The "diagonal edge" at the upper-left of the CXD8732AQ chip is Pin 49 (not pin 1), following the pin numbers on the board (and the Sanyo datasheet pinouts), pin 1 is at the lower-left.

Sanyo LC86F8604A
8bit CPU with 132Kbyte EEPROM, 4Kbyte ROM, 256 bytes RAM, 2 timers, serial port, and general purpose parallel ports. The 132K EEPROM is broken into 128K plus 4K, the 4K might be internally used by the CPU, presumably containing the BIOS (not too sure if it's really containing 4K EEPROM plus 4K ROM, or if it's meant to be only either one).

```
1=P40/A0  9=P13   17=TP0  25=VDD  33=A11  41=NC   49=A7    57=NC
2=/RES    10=P14  18=TP1  26=NC   34=A9   42=NC   50=A6    58=NC
3=TEST2   11=P15  19=TP2  27=NC   35=A8   43=NC   51=A5    59=NC
4=TEST1   12=P16  20=TP3  28=NC   36=A13  44=NC   52=A4    60=NC
5=P10     13=P17  21=TP4  29=VSS  37=A14  45=A17  53=NC    61=NC61
6=P11     14=/CE  22=TP5  30=CF1  38=/WE  46=A16  54=NC    62=P43/A3
7=P12     15=A10  23=TP6  31=CF2  39=VDD  47=A15  55=NC    63=P42/A2
8=VSS     16=/OE  24=TP7  32=VDD  40=EP   48=A12  56=NC    64=P41/A1
```

Ports P10..P17 have multiple functions (I/O port, data bus, serial, etc):

```
P10/DQ0/SEPMOD       P12/DQ2/FSI0  P14/DQ4    P16/DQ6/SI0/FSTART
P11/DQ1/SCLK0/FSCLK  P13/DQ3       P15/DQ5    P17/DQ7/SO0/FRW
```

In March 1998, Sanyo has originally announced the LC86F8604A as an 8bit CPU with "2.8V FLASH, achieved for the first time in the industry", however, according to their datasheet, what they have finally produced is an 8bit CPU with "3.5V EEPROM". Although, maybe the 3.5V EEPROM version came first, and the 2.8V FLASH was announced to be a later low-power version of the old chip; namely, otherwise, it'd be everyones guess what kind of memory Sony used in memory cards before 1998?

Note
For the actual pin-outs of the cart-edge connector, see
Pinouts - Controller Ports and Memory-Card Ports

Mods - Nocash PSX-XBOO Upload

Nocash PSX-XBOO Connection (required)

```
GND (BOARD)       --------- GND    (SUBD.18-25, CNTR.19-30)
A16 (ROM.2)       --------- SLCT   (SUBD.13, CNTR.13)     ;\
A17 (ROM.30)      --------- PE     (SUBD.12, CNTR.12)     ; 4bit.dta.out
A18 (ROM.31)      --------- /ACK   (SUBD.10, CNTR.10)     ;
A19 (ROM.1)       --------- BUSY   (SUBD.11, CNTR.11)     ;/
/RESET            ---|>|--- /INIT  (SUBD.16, CNTR.31)     ;-reset.in
D0..D7 (74HC541)  --------- DATA   (SUBD.2-9, CNTR.2-9)   ;\
Y0..Y7 (74HC541)  --------- D0..D7 (ROM.13-15,17-21)      ; 7bit.dta.in, and
/OE1 (74HC541.1)  --------- /EXP   (CPU.98)               ; 1bit.dta.clk.in
/OE2 (74HC541.19) --------- /OE    (ROM.24)               ;
GND  (74HC541.10) --------- GND    (BOARD)                ;
VCC  (74HC541.20) --------- +5V    (BOARD)                ;/
```

Nocash PSX-BIOS Connection (required)

```
A0..A19 (ROM) --------- A0..A19 (EPROM)
D0..D7  (ROM) --------- D0..D7  (EPROM)
/BIOS (CPU.97)--------- /CS  (EPROM.22)
/OE (ROM.24)  --------- /OE  (EPROM.24)
+5V (BOARD)   --------- VCC  (EPROM.32)
GND (BOARD)   --------- GND  (EPROM.16)
/CS (ROM.22)  --/cut/-- /BIOS (CPU.97)
/CS (ROM.22)  --------- +5V  (BOARD) (direct, or via 100k ohm)
```

Nocash BIOS "Modchip" Feature (optional)

```
SPU.Pin42 "data" -------|>|------ CPU.Pin149 (A20)
```

The nocash PSX bios outputs the "data" signal on the A20 address line, so (aside from the BIOS chip) one only needs to install a 1N4148 diode and two wires to unlock the CDROM.

Composite NTSC/PAL Mod (optional)

```
/PAL (IC502.13) --/cut/-- /PAL (GPU.157)
/PAL (IC502.13) --------- GND (for PAL) or VCC (for NTSC)
```

This forces the console to be always producing the desired composite color format (regardless of whether the GPU is in 50Hz or 60Hz mode).

Component List

```
32pin socket for EPROM
EPROM (or FLASH)
74HC541 (8-bit 3-state noninverting buffer/line driver)
1N4148 diode (for reset signal)
1N4148 diode (for optional "modchip" feature)
36pin Centronics socket for printer cable (or 25pin dsub)
```

PSX-XBOO Upload BIOS

The required BIOS is contained in no$psx (built-in in the no$psx.exe file), the Utility menu contains a function for creating a standalone ROM-image (file PSX-XBOO.ROM in no$psx folder; which can be then burned to FLASH or EPROM).

Pinouts

```
            ______  ______                      ____  ____
           |      \/      |                    |    \/    |
A19,VPP12  | 1         32 |  VCC6         /OE1 |1       20| VCC
      A16  | 2         31 |  A18,/PGM       D0 |2       19| /OE2
      A15  | 3         30 |  A17            D1 |3       18| Y0
      A12  | 4         29 |  A14            D2 |4       17| Y1
       A7  | 5         28 |  A13            D3 |5 74541 16| Y2
       A6  | 6         27 |  A8             D4 |6       15| Y3
       A5  | 7         26 |  A9,IDENT12     D5 |7       14| Y4
       A4  | 8         25 |  A11            D6 |8       13| Y5
       A3  | 9         24 |  /OE,VPP12      D7 |9       12| Y6
       A2  | 10        23 |  A10           GND |10      11| Y7
       A1  | 11        22 |  /CE,(/PGM)        |__________|
       A0  | 12        21 |  D7
       D0  | 13        20 |  D6
       D1  | 14        19 |  D5
       D2  | 15        18 |  D4
      GND  | 16        17 |  D3
           |______________|
```

About & Credits

Credits

GPU.TXT by doomed/padua; based on info from K-communications & Nagra/Blackbag
GTE.TXT by doomed@c64.org / psx.rules.org
SPU.TXT by doomed@c64.org / psx.rules.org
CDINFO.TXT by doomed with big thanks to Barubary, who rewrote a large part
SYSTEM.TXT by doomed with thanx to Herozero for breakpoint info
PS_ENG.TXT PlayStation PAD/Memory Interface Protocol by HFB03536
IDT79R3041 Hardware User's Manual by Integrated Device Technology, Inc.
IDTR3051, R3052 RISController User's Manual by Integrated Device Technology
PSX.* by Joshua Walker (additional details in various distorted file formats)
LIBMIRAGE by Rok; info/source code for various cdrom-image formats
psxdev.ru; cdrom sub-cpu decapping

PSXSPX homepage

http://problemkaputt.de/psx.htm no$psx emulator/debugger
http://problemkaputt.de/psx-spx.htm psx specs in html formal
http://problemkaputt.de/psx-spx.txt psx specs in text formal

Contact

http://problemkaputt.de/email.htm (spam-shielded)

Index

Contents
Memory Map
I/O Map
Graphics Processing Unit (GPU)
GPU I/O Ports, DMA Channels, Commands, VRAM
GPU Render Polygon Commands
GPU Render Line Commands
GPU Render Rectangle Commands
GPU Rendering Attributes
GPU Memory Transfer Commands
GPU Other Commands
GPU Display Control Commands (GP1)
GPU Status Register
GPU Depth Ordering
GPU Video Memory (VRAM)
GPU Texture Caching
GPU Timings
GPU (MISC)
Geometry Transformation Engine (GTE)
GTE Overview

GTE Saturation
GTE Opcode Summary
GTE Coordinate Calculation Commands
GTE General Purpose Calculation Commands
GTE Color Calculation Commands
GTE Division Inaccuracy
Macroblock Decoder (MDEC)
MDEC I/O Ports
MDEC Commands
MDEC Decompression
MDEC Data Format
Sound Processing Unit (SPU)
SPU Overview
SPU ADPCM Samples
SPU ADPCM Pitch
SPU Volume and ADSR Generator
SPU Voice Flags
SPU Noise Generator
SPU Control and Status Register
SPU Memory Access
SPU Interrupt
SPU Reverb Registers
SPU Reverb Formula
SPU Reverb Examples
SPU Unknown Registers
Interrupts
DMA Channels
Timers
CDROM Drive
CDROM Controller I/O Ports
CDROM Controller Command Summary
CDROM - Control Commands
CDROM - Seek Commands
CDROM - Read Commands
CDROM - Status Commands
CDROM - CD Audio Commands
CDROM - Test Commands
CDROM - Test Commands - Version, Switches, Region, Chipset, SCEx
CDROM - Test Commands - Test Drive Mechanics
CDROM - Test Commands - Prototype Debug Transmission
CDROM - Test Commands - Read/Write Decoder RAM and I/O Ports
CDROM - Test Commands - Read HC05 SUB-CPU RAM and I/O Ports
CDROM - Secret Unlock Commands
CDROM - Mainloop/Responses
CDROM - Response Timings
CDROM - Response/Data Queueing
CDROM Disk Format
CDROM Subchannels
CDROM Sector Encoding
CDROM XA Subheader, File, Channel, Interleave
CDROM XA Audio ADPCM Compression
CDROM ISO Volume Descriptors
CDROM ISO File and Directory Descriptors
CDROM ISO Misc
CDROM File Formats
CDROM Protection - SCEx Strings
CDROM Protection - Bypassing it
CDROM Protection - Modchips
CDROM Protection - LibCrypt
CDROM Disk Images CCD/IMG/SUB (CloneCD)
CDROM Disk Images CDI (DiscJuggler)
CDROM Disk Images CUE/BIN/CDT (Cdrwin)
CDROM Disk Images MDS/MDF (Alcohol 120%)
CDROM Disk Images NRG (Nero)
CDROM Disk Image/Containers CDZ
CDROM Disk Image/Containers ECM
CDROM Subchannel Images
CDROM Disk Images Other Formats
CDROM Internal Info on PSX CDROM Controller
CDROM Internal HC05 Instruction Set
CDROM Internal HC05 On-Chip I/O Ports
CDROM Internal HC05 On-Chip I/O Ports - Extras
CDROM Internal HC05 I/O Port Usage in PSX
CDROM Internal HC05 Motorola Selftest Mode
CDROM Internal HC05 Motorola Selftest Mode (52pin chips)
CDROM Internal HC05 Motorola Selftest Mode (80pin chips)
CDROM Internal CXD1815Q Sub-CPU Configuration Registers
CDROM Internal CXD1815Q Sub-CPU Sector Status Registers
CDROM Internal CXD1815Q Sub-CPU Address Registers
CDROM Internal CXD1815Q Sub-CPU Misc Registers
CDROM Internal Commands CX(0x..3x) - CXA1782BR Servo Amplifier
CDROM Internal Commands CX(4x..Ex) - CXD2510Q Signal Processor
CDROM Internal Commands CX(0x..Ex) - CXD2545Q Servo/Signal Combo
CDROM Internal Commands CX(0x..Ex) - CXD2938Q Servo/Signal/SPU Combo
CDROM Internal Commands CX(xx) - Notes
CDROM Internal Commands CX(xx) - Summary of Used CX(xx) Commands

Inflate
Inflate - Core Functions
Inflate - Initialization & Tree Creation
Inflate - Headers and Checksums
Controllers and Memory Cards
Controller and Memory Card I/O Ports
Controller and Memory Card Misc
Controller and Memory Card Signals
Controller and Memory Card Multitap Adaptor
Controllers - Communication Sequence
Controllers - Standard Digital/Analog Controllers
Controllers - Mouse
Controllers - Racing Controllers
Controllers - Lightguns
Controllers - Lightguns - Namco (GunCon)
Controllers - Lightguns - Konami Justifier/Hyperblaster (IRQ10)
Controllers - Lightguns - PSX Lightgun Games
Controllers - Rumble Configuration
Controllers - Dance Mats
Controllers - Fishing Controllers
Controllers - I-Mode Adaptor (Mobile Internet)
Controllers - Additional Inputs
Controllers - Misc
Memory Card Read/Write Commands
Memory Card Data Format
Memory Card Images
Memory Card Notes
Other Hardware that connects to Memory Card slot
Pocketstation
Pocketstation Overview
Pocketstation I/O Map
Pocketstation Memory Map
Pocketstation IO Video and Audio
Pocketstation IO Interrupts and Buttons
Pocketstation IO Timers and Real-Time Clock
Pocketstation IO Infrared
Pocketstation IO Memory-Control
Pocketstation IO Communication Ports
Pocketstation IO Power Control
Pocketstation SWI Function Summary
Pocketstation SWI Misc Functions
Pocketstation SWI Communication Functions
Pocketstation SWI Execute Functions
Pocketstation SWI Date/Time/Alarm Functions
Pocketstation SWI Flash Functions
Pocketstation SWI Useless Functions
Pocketstation BU Command Summary
Pocketstation BU Standard Memory Card Commands
Pocketstation BU Basic Pocketstation Commands
Pocketstation BU Custom Pocketstation Commands
Pocketstation File Header/Icons
Pocketstation File Images
Pocketstation XBOO Cable
Serial Port (SIO)
Expansion Port (PIO)
EXP1 Expansion ROM Header
EXP2 Dual Serial Port (for TTY Debug Terminal)
EXP2 DTL-H2000 I/O Ports
EXP2 Post Registers
EXP2 Nocash Emulation Expansion
Memory Control
Unpredictable Things
CPU Specifications
CPU Registers
CPU Opcode Encoding
CPU Load/Store Opcodes
CPU ALU Opcodes
CPU Jump Opcodes
CPU Coprocessor Opcodes
CPU Pseudo Opcodes
COP0 - Register Summary
COP0 - Exception Handling
COP0 - Misc
COP0 - Debug Registers
Kernel (BIOS)
BIOS Overview
BIOS Memory Map
BIOS Function Summary
BIOS File Functions
BIOS File Execute and Flush Cache
BIOS CDROM Functions
BIOS Memory Card Functions
BIOS Interrupt/Exception Handling
BIOS Event Functions
BIOS Event Summary
BIOS Thread Functions

BIOS Joypad Functions
BIOS GPU Functions
BIOS Memory Allocation
BIOS Memory Fill/Copy/Compare (SLOW)
BIOS String Functions
BIOS Number/String/Character Conversion
BIOS Misc Functions
BIOS Internal Boot Functions
BIOS More Internal Functions
BIOS PC File Server
BIOS TTY Console (std_io)
BIOS Character Sets
BIOS Control Blocks
BIOS Versions
BIOS Patches
Arcade Cabinets
Cheat Devices
Cheat Devices - Datel Code Format
Cheat Devices - Xplorer Code Format
Cheat Devices - Xplorer Cheat Code and ROM-Image Decryption
Cheat Devices - Datel I/O
Cheat Devices - Xplorer I/O
PSX Dev-Board Chipsets
Hardware Numbers
Pinouts
Pinouts - Controller Ports and Memory-Card Ports
Pinouts - Audio, Video, Power, Expansion Ports
Pinouts - SIO Pinouts
Pinouts - Chipset Summary
Pinouts - CPU Pinouts
Pinouts - GPU Pinouts
Pinouts - SPU Pinouts
Pinouts - DRV Pinouts
Pinouts - HC05 Pinouts
Pinouts - MEM Pinouts
Pinouts - CLK Pinouts
Pinouts - PWR Pinouts
Pinouts - Component List and Chipset Pin-Outs for Digital Joypad
Pinouts - Component List and Chipset Pin-Outs for Analog Joypad
Pinouts - Component List and Chipset Pin-Outs for Multitap
Pinouts - Memory Cards
Mods - Nocash PSX-XBOO Upload
About & Credits

[extracted from no$psx v1.9 documentation]