Pan Docs

The single, most comprehensive technical reference to Game Boy available to the public.

0.0.1 Table of Contents

Chapter 1

About the Pan Docs

This is a new, experimental version of Pan Docs, mantained in the Markdown format.

To learn more about the history and the mission of the project, check the README.

This document version was produced from git commit 59d00b0 (2020-06-28 03:26:02 +0200).

1.1 Authors

This document is the product of 25 years of community effort: it's hard to keep track of every contribution. If we missed something, contact us.

Non-exhaustive, list of content contributors:

Pan of ATX, Marat Fayzullin, Felber Pascal, Robson Paul, Korth Martin.

Antonio Niño Díaz, Antonio Vivace, Beannaich, Elizafox, endrift, exezin, Furrtek, Gekkio, ISSOtm, Jeff Frohwein, John Harrison, kOOPa, LIJI32, jrra, Mantidactyle, mattcurrie, nitro2k01, Pascal Felber, pinobatch, T4g1, TechFalcon

Chapter 2

Specifications

	Game Boy	Game Boy Pocket	Game Boy Color	Super Game Boy
CPU	8-bit Sharp LR35902			
Clock	4.194304 MHz		8.388608 MHz	4.295454MHz (S
Work RAM	8 KB		32 KB	
Video RAM	8 KB		16 KB	additional 10KB
Screen Size	LCD 4,7 x 4,3 cm	LCD 4,8 x 4,4 cm	TFT 4,4 x 4 cm	
Resolution	160x144			256x224 (includi
Sprites	Max 40 per screen, 10 per line			
Palettes	1x4 BG, 2x3 OBJ		8x4 BG, 8x3 OBJ	1+4x3, 4x15 (bo
Colors	4 grayshades		32768 colors	
Horizontal Sync	9.198 KHz			9.41986 KHz
Vertical Sync	59.73 Hz			61.1679 Hz
Sound	4 channels with stereo sound			
Power	DC6V 0.7W	DC3V 0.7W	DC3V 0.6W	

Chapter 3

Memory Map

The Game Boy has a 16bit address bus, that is used to address ROM, RAM and I/O

3.1 General Memory Map

Start	End	Description	Notes
0000	3FFF	16KB ROM bank 00	From cartridge, usually a fixed bank
4000	7FFF	16KB ROM Bank 01~NN	From cartridge, switchable bank via MB (if any)

Start	End	Description	Notes
8000	9FFF	8KB Video RAM (VRAM)	Only bank 0 in Non-CGB mode Switch-able bank 0/1 in CGB mode
A000	BFFF	8KB External RAM	In car-tridge, switch-able bank if any
C000	CFFF	4KB Work RAM (WRAM) bank 0	
D000	DFFF	4KB Work RAM (WRAM) bank 1~N	Only bank 1 in Non-CGB mode Switch-able bank 1~7 in CGB mode
E000	FDFF	Mirror of C000~DDFF (ECHO RAM)	Typically not used
FE00	FE9F	Sprite attribute table (OAM	
FEA0	FEFF	Not Usable	
FF00	FF7F	I/O Registers	
FF80	FFFE	High RAM (HRAM)	
FFFF	FFFF	Interrupts Enable Register (IE)	

3.2 Jump Vectors in first ROM bank

The following addresses are supposed to be used as jump vectors:

- RST commands: 0000, 0008,0010, 0018, 0020, 0028, 0030, 0038
- Interrupts: 0040, 0048, 0050, 0058, 0060

However, the memory may be used for any other purpose in case that your program doesn't use any (or only some) RST commands or interrupts. RST commands are 1-byte opcodes that work similar to CALL opcodes, except that the destination address is fixed. Since they are only 1 byte large, they are also slightly faster.

3.3 Cartridge Header in first ROM bank

The memory at 0100-014F contains the cartridge header. This area contains information about the program, its entry point, checksums, information about the used MBC chip, the ROM and RAM sizes, etc. Most of the bytes in this area are required to be specified correctly. For more information read the chapter about The Cartridge Header.

3.4 External Memory and Hardware

The areas from 0000-7FFF and A000-BFFF may be used to connect external hardware. The first area is typically used to address ROM (read only, of course), cartridges with Memory Bank Controllers (MBCs) are additionally using this area to output data (write only) to the MBC chip. The second area is often used to address external RAM, or to address other external hardware (Real Time Clock, etc). External memory is often battery buffered, and may hold saved game positions and high score tables (etc.) even when the Game Boy is turned off, or when the cartridge is removed. For specific information read the chapter about Memory Bank Controllers.

3.5 Echo RAM

The memory range E000-FDFF is a mirror (or "echo") of WRAM, both for reading and writing. For example, writing to $E123 will modify both $C123 and $E123. It is recommended to avoid using this memory range anyways. This memory range's behavior has been confirmed on all grey GBs as well as on CGB and GBA. Some emulators (such as VisualBoyAdvance <1.8) don't emulate Echo RAM. It is possible to check if Echo RAM is properly emulated by writing to WRAM (avoid values 00 and FF) and checking if said value is mirrored in Echo RAM.

3.6 FEA0-FEFF range

This range is very poorly documented. It doesn't even have a name ! From my experience, this stays 00 on DMG, and alternates between 00 and seemingly random values on CGB.

Chapter 4

Video Display

4.1 LCD Control Register

LCDC is the main **LCD C**ontrol register. Its bits toggle what elements are displayed on the screen, and how.

```
Bit 7 - LCD Display Enable             (0=Off, 1=On)
Bit 6 - Window Tile Map Display Select (0=9800-9BFF, 1=9C00-9FFF)
Bit 5 - Window Display Enable          (0=Off, 1=On)
Bit 4 - BG & Window Tile Data Select   (0=8800-97FF, 1=8000-8FFF)
Bit 3 - BG Tile Map Display Select     (0=9800-9BFF, 1=9C00-9FFF)
Bit 2 - OBJ (Sprite) Size              (0=8x8, 1=8x16)
Bit 1 - OBJ (Sprite) Display Enable    (0=Off, 1=On)
Bit 0 - BG/Window Display/Priority     (0=Off, 1=On)
```

4.1.1 LCDC.7 - LCD Display Enable

This bit controls whether the LCD is on and the PPU is active. Setting it to 0 turns both off, which grants immediate and full access to VRAM, OAM, etc.

Stopping LCD operation (Bit 7 from 1 to 0) may be performed during `VBlank` ONLY, disabling the display outside of the V-Blank period may damage the hardware by burning in a black horizontal line similar to that which appears when the GB is turned off. This appears to be a serious issue, Nintendo is reported to reject any games that do not follow this rule.

When the display is disabled the screen is blank, which on DMG is displayed as a white “whiter” than color #0.

On SGB, the screen doesn’t turn white, it appears that the previous picture sticks to the screen. (TODO: research this more.)

When re-enabling the LCD, the PPU will immediately start drawing again, but the screen will stay blank during the first frame.

4.1.2 LCDC.6 - Window Tile Map Display Select

This bit controls which background map the Window uses for rendering. When it's reset, the $9800 tilemap is used, otherwise it's the $9C00 one.

4.1.3 LCDC.5 - Window Display Enable

This bit controls whether the window shall be displayed or not. (TODO : what happens when toggling this mid-scanline ?) This bit is overridden on DMG by bit 0 if that bit is reset.

Note that on CGB models, setting this bit to 0 then back to 1 mid-frame may cause the second write to be ignored. (TODO : test this.)

4.1.4 LCDC.4 - BG & Window Tile Data Select

This bit controls which addressing mode the BG and Window use to pick tiles.

Sprites aren't affected by this, and will always use $8000 addressing mode.

4.1.5 LCDC.3 - BG Tile Map Display Select

This bit works similarly to LCDC-6: if the bit is reset, the BG uses tilemap $9800, otherwise tilemap $9C00.

4.1.6 LCDC.2 - OBJ Size

This bit controls the sprite size (1 tile or 2 stacked vertically).

Be cautious when changing this mid-frame from 8x8 to 8x16 : “remnants” of the sprites intended for 8x8 could “leak” into the 8x16 zone and cause artifacts.

4.1.7 LCDC.1 - OBJ Display Enable

This bit toggles whether sprites are displayed or not.

This can be toggled mid-frame, for example to avoid sprites being displayed on top of a status bar or text box.

(Note: toggling mid-scanline might have funky results on DMG? Investigation needed.)

4.1.8 LCDC.0 - BG/Window Display/Priority

LCDC.0 has different meanings depending on Game Boy type and Mode:

Monochrome Gameboy, SGB and CGB in Non-CGB Mode: BG Display

When Bit 0 is cleared, both background and window become blank (white), and the Window Display Bit is ignored in that case. Only Sprites may still be displayed (if enabled in Bit 1).

CGB in CGB Mode: BG and Window Master Priority

When Bit 0 is cleared, the background and window lose their priority - the sprites will be always displayed on top of background and window, independently of the priority flags in OAM and BG Map attributes.

4.1.9 Using LCDC

LCDC is a powerful tool: each bit controls a lot of behavior, and can be modified at any time during the frame.

One of the important aspects of LCDC is that unlike VRAM, the PPU never locks it. It's thus possible to modify it mid-scanline!

4.1.10 Faux-layer textbox/status bar

A problem often seen especially in NES games is sprites rendering on top of the textbox/status bar. It's possible to prevent this using LCDC if the textbox/status bar is "alone" on its scanlines:

- Set LCDC.1 to 1 for gameplay scanlines
- Set LCDC.1 to 0 for textbox/status bar scanlines

Usually, these bars are either at the top or bottom of the screen, so the bit can be set by the VBlank handler

4.2 LCD Status Register

FF41 - STAT - LCDC Status (R/W)

```
Bit 6 - LYC=LY Coincidence Interrupt (1=Enable) (Read/Write)
Bit 5 - Mode 2 OAM Interrupt         (1=Enable) (Read/Write)
Bit 4 - Mode 1 V-Blank Interrupt     (1=Enable) (Read/Write)
Bit 3 - Mode 0 H-Blank Interrupt     (1=Enable) (Read/Write)
Bit 2 - Coincidence Flag  (0:LYC<>LY, 1:LYC=LY) (Read Only)
Bit 1-0 - Mode Flag       (Mode 0-3, see below) (Read Only)
          0: During H-Blank
          1: During V-Blank
          2: During Searching OAM
          3: During Transferring Data to LCD Driver
```

The two lower STAT bits show the current status of the LCD controller.

The LCD controller operates on a 2^{22} Hz = 4.194 MHz dot clock. An entire frame is 154 scanlines, 70224 dots, or 16.74 ms. On scanlines 0 through 143, the LCD controller cycles through modes 2, 3, and 0 once every 456 dots. Scanlines 144 through 153 are mode 1.

The following are typical when the display is enabled:

```
Mode 2  2_____2_____2_____2_____2_____2___________________2____
Mode 3  _33____33____33____33____33____33__________________3___
Mode 0  ___000___000___000___000___000___000________________000
Mode 1  ____________________________________11111111111111_____
```

When the LCD controller is reading a particular part of video memory, that memory is inaccessible to the CPU.

- During modes 2 and 3, the CPU cannot access OAM (FE00h-FE9Fh).
- During mode 3, the CPU cannot access VRAM or CGB Palette Data (FF69,FF6B).

Mode	Action	Duration	Accessible video memory
2	Scanning OAM for (X, Y) coordinates of sprites that overlap this line	80 dots (19 us)	VRAM, CGB palettes
3	Reading OAM and VRAM to generate the picture	168 to 291 dots (40 to 60 us) depending on sprite count	None
0	Horizontal blanking	85 to 208 dots (20 to 49 us) depending on previous mode 3 duration	VRAM, OAM, CGB palettes
1	Vertical blanking	4560 dots (1087 us, 10 scanlines)	VRAM, OAM, CGB palettes

Properties of STAT modes

Unlike most game consoles, the Game Boy can pause the dot clock briefly, adding dots to mode 3's duration. It routinely takes a 6 to 11 dot break to fetch sprite patterns between background tile pattern fetches. On DMG and GBC in DMG mode, mid-scanline writes to `BGP` allow observing this behavior, as a sprite delay shifts the effect of a write to the left by that many dots.

Three things are known to pause the dot clock:

- Background scrolling : If `SCX mod 8` is not zero at the start of the scanline, rendering is paused for that many dots while the shifter discards that many pixels from the leftmost tile.
- Window : An active window pauses for at least 6 dots, as the background fetching mechanism starts over at the left side of the window.

- Sprites : Each sprite usually pauses for `11 - min(5, (x + SCX) mod 8)` dots. Because sprite fetch waits for background fetch to finish, a sprite's cost depends on its position relative to the left side of the background tile under it. It's greater if a sprite is directly aligned over the background tile, less if the sprite is to the right. If the sprite's left side is over the window, use `255 - WX` for `SCX` in this formula.

::: warning Not fully understood The exact pause duration for window start is not confirmed; it may have the same background fetch finish delay as a sprite. If two sprites' left sides are over the same background or window tile, the second may pause for fewer dots. :::

A hardware quirk in the monochrome Game Boy makes the LCD interrupt sometimes trigger when writing to STAT (including writing $00) during OAM scan, H-Blank, V-Blank, or LY=LYC. It behaves as if $FF were written for one cycle, and then the written value were written the next cycle. Because the GBC in DMG mode does not have this quirk, two games that depend on this quirk (Ocean's *Road Rash* and Vic Tokai's *Xerd no Densetsu*) will not run on a GBC.

4.3 LCD Interrupts

INT 40 - V-Blank Interrupt

The V-Blank interrupt occurs ca. 59.7 times a second on a handheld Game Boy (DMG or CGB) or Game Boy Player and ca. 61.1 times a second on a Super Game Boy (SGB). This interrupt occurs at the beginning of the V-Blank period (LY=144). During this period video hardware is not using VRAM so it may be freely accessed. This period lasts approximately 1.1 milliseconds.

INT 48 - LCDC Status Interrupt

There are various reasons for this interrupt to occur as described by the STAT register ($FF40). One very popular reason is to indicate to the user when the video hardware is about to redraw a given LCD line. This can be useful for dynamically controlling the SCX/SCY registers (FF43/FF42) to perform special video effects.

Example application : set LYC to WY, enable LY=LYC interrupt, and have the handler disable sprites. This can be used if you use the window for a text box (at the bottom of the screen), and you want sprites to be hidden by the text box.

The interrupt is triggered when transitioning from "No conditions met" to "Any condition met", which can cause the interrupt to not fire. Example : the Mode 0 and LY=LYC interrupts are enabled ; since the latter triggers during Mode 2 (right after Mode 0), the interrupt will trigger for Mode 0 but fail to for LY=LYC.

4.4 LCD Position and Scrolling

These registers can be accessed even during Mode 3, but they have no effect until the end of the current scanline.

FF42 - SCY - Scroll Y (R/W), FF43 - SCX - Scroll X (R/W)

Specifies the position in the 256x256 pixels BG map (32x32 tiles) which is to be displayed at the upper/left LCD display position. Values in range from 0-255 may be used for X/Y each, the video controller automatically wraps back to the upper (left) position in BG map when drawing exceeds the lower (right) border of the BG map area.

FF44 - LY - LCDC Y-Coordinate (R)

The LY indicates the vertical line to which the present data is transferred to the LCD Driver. The LY can take on any value between 0 through 153. The values between 144 and 153 indicate the V-Blank period.

FF45 - LYC - LY Compare (R/W)

The Game Boy permanently compares the value of the LYC and LY registers. When both values are identical, the coincident bit in the STAT register becomes set, and (if enabled) a STAT interrupt is requested.

FF4A - WY - Window Y Position (R/W), FF4B - WX - Window X Position minus 7 (R/W)

Specifies the upper/left positions of the Window area. (The window is an alternate background area which can be displayed above of the normal background. OBJs (sprites) may be still displayed above or behind the window, just as for normal BG.)

The window becomes visible (if enabled) when positions are set in range WX=0..166, WY=0..143. A position of WX=7, WY=0 locates the window at upper left, it is then completely covering normal background.

WX values 0-6 and 166 are unreliable due to hardware bugs. If WX is set to 0, the window will “stutter” horizontally when SCX changes. (Depending on SCX modulo 8, behavior is a little complicated so you should try it yourself.)

4.5 LCD Monochrome Palettes

FF47 - BGP - BG Palette Data (R/W) - Non CGB Mode Only

This register assigns gray shades to the color numbers of the BG and Window tiles.

```
Bit 7-6 - Shade for Color Number 3
Bit 5-4 - Shade for Color Number 2
Bit 3-2 - Shade for Color Number 1
Bit 1-0 - Shade for Color Number 0
```

The four possible gray shades are:

```
0  White
1  Light gray
2  Dark gray
3  Black
```

In CGB Mode the Color Palettes are taken from CGB Palette Memory instead.

FF48 - OBP0 - Object Palette 0 Data (R/W) - Non CGB Mode Only

This register assigns gray shades for sprite palette 0. It works exactly as BGP (FF47), except that the lower two bits aren't used because sprite data 00 is transparent.

FF49 - OBP1 - Object Palette 1 Data (R/W) - Non CGB Mode Only

This register assigns gray shades for sprite palette 1. It works exactly as BGP (FF47), except that the lower two bits aren't used because sprite data 00 is transparent.

4.6 LCD Color Palettes (CGB only)

FF68 - BCPS/BGPI - CGB Mode Only - Background Palette Index

This register is used to address a byte in the CGBs Background Palette Memory. Each two byte in that memory define a color value. The first 8 bytes define Color 0-3 of Palette 0 (BGP0), and so on for BGP1-7.

```
Bit 0-5   Index (00-3F)
Bit 7     Auto Increment  (0=Disabled, 1=Increment after Writing)
```

Data can be read/written to/from the specified index address through Register FF69. When the Auto Increment bit is set then the index is automatically incremented after each **write** to FF69. Auto Increment has no effect when **reading** from FF69, so the index must be manually incremented in that case. Writing to FF69 during rendering still causes auto-increment to occur.

Unlike the following, this register can be accessed outside V-Blank and H-Blank.

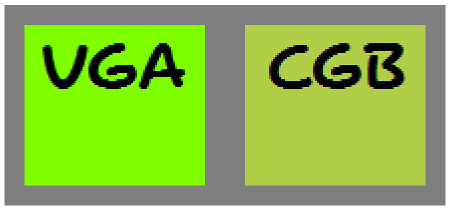


Figure 4.1: sRGB versus CGB color mixing

FF69 - BCPD/BGPD - CGB Mode Only - Background Palette Data

This register allows to read/write data to the CGBs Background Palette Memory, addressed through Register FF68. Each color is defined by two bytes (Bit 0-7 in first byte).

```
Bit 0-4   Red Intensity   (00-1F)
Bit 5-9   Green Intensity (00-1F)
Bit 10-14 Blue Intensity  (00-1F)
```

Much like VRAM, data in Palette Memory cannot be read/written during the time when the LCD Controller is reading from it. (That is when the STAT register indicates Mode 3). Note: All background colors are initialized as white by the boot ROM, but it's a good idea to initialize at least one color yourself (for example if you include a soft-reset mechanic).

FF6A - OCPS/OBPI - CGB Mode Only - Sprite Palette Index, FF6B - OCPD/OBPD - CGB Mode Only - Sprite Palette Data

These registers are used to initialize the Sprite Palettes OBP0-7, identically as described above for Background Palettes. Note that four colors may be defined for each OBP Palettes - but only Color 1-3 of each Sprite Palette can be displayed, Color 0 is always transparent, and can be initialized to a don't care value or plain never initialized.

Note: All sprite colors are left uninitialized by the boot ROM, and are somewhat random.

RGB Translation by CGBs

When developing graphics on PCs, note that the RGB values will have different appearance on CGB displays as on VGA/HDMI monitors calibrated to sRGB color. Because the GBC is not lit, the highest intensity will produce Light Gray color rather than White. The intensities are not linear; the values 10h-1Fh will all appear very bright, while medium and darker colors are ranged at 00h-0Fh.

The CGB display's pigments aren't perfectly saturated. This means the colors mix quite oddly; increasing intensity of only one R,G,B color will also influence the other two R,G,B colors. For example, a color setting of 03EFh

(Blue=0, Green=1Fh, Red=0Fh) will appear as Neon Green on VGA displays, but on the CGB it'll produce a decently washed out Yellow. See image on the right.

RGB Translation by GBAs

Even though GBA is described to be compatible to CGB games, most CGB games are completely unplayable on older GBAs because most colors are invisible (black). Of course, colors such like Black and White will appear the same on both CGB and GBA, but medium intensities are arranged completely different. Intensities in range 00h..07h are invisible/black (unless eventually under best sunlight circumstances, and when gazing at the screen under obscure viewing angles), unfortunately, these intensities are regularly used by most existing CGB games for medium and darker colors.

Newer CGB games may avoid this effect by changing palette data when detecting GBA hardware (see how). Based on measurement of GBC and GBA palettes using the 144p Test Suite ROM, a fairly close approximation is GBA = GBC * 3/4 + 8h for each R,G,B intensity. The result isn't quite perfect, and it may turn out that the color mixing is different also; anyways, it'd be still ways better than no conversion.

This problem with low brightness levels does not affect later GBA SP units and Game Boy Player. Thus ideally, the player should have control of this brightness correction.

4.7 LCD OAM DMA Transfers

FF46 - DMA - DMA Transfer and Start Address (R/W)

Writing to this register launches a DMA transfer from ROM or RAM to OAM memory (sprite attribute table). The written value specifies the transfer source address divided by 100h, ie. source & destination are:

```
Source:      XX00-XX9F   ;XX in range from 00-F1h
Destination: FE00-FE9F
```

The transfer takes 160 machine cycles: 152 microseconds in normal speed or 76 microseconds in CGB Double Speed Mode. On DMG, during this time, the CPU can access only HRAM (memory at FF80-FFFE); on CGB, the bus used by the source area cannot be used (this isn't understood well at the moment, it's recommended to assume same behavior as DMG). For this reason, the programmer must copy a short procedure into HRAM, and use this procedure to start the transfer from inside HRAM, and wait until the transfer has finished:

```
 run_dma:
  ld a, start address / 100h
  ldh  (FF46h),a ;start DMA transfer (starts right after instruction)
  ld  a,28h      ;delay...
```

```
wait:            ;total 4x40 cycles, approx 160  s
 dec a           ;1 cycle
 jr  nz,wait     ;3 cycles
 ret
```

Because sprites are not displayed while OAM DMA is in progress, most programs are executing this procedure from inside of their VBlank procedure. But it is also possible to execute it during display redraw also, allowing to display more than 40 sprites on the screen (ie. for example 40 sprites in upper half, and other 40 sprites in lower half of the screen), at the cost of a couple lines that lack sprites.

A more compact procedure is

```
run_dma:  ; This part is in ROM
 ld a, start address / 100h
 ld bc, 2946h  ; B: wait time; C: OAM trigger
 jp run_dma_hrampart

run_dma_hrampart:
 ldh ($FF00+c), a
wait:
 dec b
 jr nz,wait
 ret
```

which should be called with a = start address / 100h, bc = 2946h. This saves 5 bytes of HRAM, but is slightly slower in most cases because of the jump into the HRAM part.

4.8 LCD VRAM DMA Transfers (CGB only)

FF51 - HDMA1 - CGB Mode Only - New DMA Source, High

FF52 - HDMA2 - CGB Mode Only - New DMA Source, Low

These two registers specify the address at which the transfer will read data from. Normally, this should be either in ROM, SRAM or WRAM, thus either in range 0000-7FF0 or A000-DFF0. [Note : this has yet to be tested on Echo RAM, OAM, FEXX, IO and HRAM]. Trying to specify a source address in VRAM will cause garbage to be copied.

The four lower bits of this address will be ignored and treated as 0.

FF53 - HDMA3 - CGB Mode Only - New DMA Destination, High

FF54 - HDMA4 - CGB Mode Only - New DMA Destination, Low

These two registers specify the address within 8000-9FF0 to which the data will be copied. Only bits 12-4 are respected; others are ignored. The four lower bits of this address will be ignored and treated as 0.

FF55 - HDMA5 - CGB Mode Only - New DMA Length/Mode/Start

These registers are used to initiate a DMA transfer from ROM or RAM to VRAM. The Source Start Address may be located at 0000-7FF0 or A000-DFF0, the lower four bits of the address are ignored (treated as zero). The Destination Start Address may be located at 8000-9FF0, the lower four bits of the address are ignored (treated as zero), the upper 3 bits are ignored either (destination is always in VRAM).

Writing to this register starts the transfer, the lower 7 bits of which specify the Transfer Length (divided by 10h, minus 1), ie. lengths of 10h-800h bytes can be defined by the values 00h-7Fh. The upper bit indicates the Transfer Mode:

Bit7=0 - General Purpose DMA

When using this transfer method, all data is transferred at once. The execution of the program is halted until the transfer has completed. Note that the General Purpose DMA blindly attempts to copy the data, even if the LCD controller is currently accessing VRAM. So General Purpose DMA should be used only if the Display is disabled, or during V-Blank, or (for rather short blocks) during H-Blank. The execution of the program continues when the transfer has been completed, and FF55 then contains a value of FFh.

Bit7=1 - H-Blank DMA

The H-Blank DMA transfers 10h bytes of data during each H-Blank, ie. at LY=0-143, no data is transferred during V-Blank (LY=144-153), but the transfer will then continue at LY=00. The execution of the program is halted during the separate transfers, but the program execution continues during the 'spaces' between each data block. Note that the program should not change the Destination VRAM bank (FF4F), or the Source ROM/RAM bank (in case data is transferred from bankable memory) until the transfer has completed! (The transfer should be paused as described below while the banks are switched)

Reading from Register FF55 returns the remaining length (divided by 10h, minus 1), a value of 0FFh indicates that the transfer has completed. It is also possible to terminate an active H-Blank transfer by writing zero to Bit 7 of FF55. In that case reading from FF55 will return how many $10 "blocks" remained (minus 1) in the lower 7 bits, but Bit 7 will be read as "1". Stopping the transfer doesn't set HDMA1-4 to $FF.

H-Blank DMA should not be started (write to FF55) during a H-Blank period (STAT mode 0).

If the transfer's destination address overflows, the transfer stops prematurely. [Note : what's the state of the registers if this happens ?]

Confirming if the DMA Transfer is Active

Reading Bit 7 of FF55 can be used to confirm if the DMA transfer is active (1=Not Active, 0=Active). This works under any circumstances - after completion of General Purpose, or H-Blank Transfer, and after manually terminating a H-Blank Transfer.

Transfer Timings

In both Normal Speed and Double Speed Mode it takes about 8 s to transfer a block of 10h bytes. That are 8 tstates in Normal Speed Mode, and 16 ‘fast’ tstates in Double Speed Mode. Older MBC controllers (like MBC1-4) and slower ROMs are not guaranteed to support General Purpose or H-Blank DMA, that’s because there are always 2 bytes transferred per microsecond (even if the itself program runs it Normal Speed Mode).

4.9 VRAM Tile Data

Tile Data is stored in VRAM at addresses $8000-97FF; with one tile being 16 bytes large, this area defines data for 384 Tiles. In CGB Mode, this is doubled (768 tiles) because of the two VRAM banks.

Each tile is sized 8x8 pixels and has a color depth of 4 colors/gray shades. Tiles can be displayed as part of the Background/Window map, and/or as OAM tiles (foreground sprites). Note that foreground sprites don’t use color 0 - it’s transparent instead.

There are three “blocks” of 128 tiles each:

- Block 0 is $8000-87FF
- Block 1 is $8800-8FFF
- Block 2 is $9000-97FF

Tiles are always indexed using a 8-bit integer, but the addressing method may differ. The “8000 method” uses $8000 as its base pointer and uses an unsigned addressing, meaning that tiles 0-127 are in block 0, and tiles 128-255 are in block 1. The “8800 method” uses $9000 as its base pointer and uses a signed addressing. To put it differently, “8000 addressing” takes tiles 0-127 from block 0 and tiles 128-255 from block 1, whereas “8800 addressing” takes tiles 0-127 from block 2 and tiles 128-255 from block 1. (You can notice that block 1 is shared by both addressing methods)

Sprites always use 8000 addressing, but the BG and Window can use either mode, controlled by LCDC bit 4.

Each Tile occupies 16 bytes, where each 2 bytes represent a line:

```
Byte 0-1  First Line (Upper 8 pixels)
Byte 2-3  Next Line
etc.`
```

For each line, the first byte defines the least significant bits of the color numbers for each pixel, and the second byte defines the upper bits of the color numbers. In either case, Bit 7 is the leftmost pixel, and Bit 0 the rightmost.

For example : let's say you have $57 $36 (in this order in memory). To obtain the color index for the leftmost pixel, you take bit 7 of both bytes : 0, and 0. Thus the index is 00b = 0. For the second pixel, repeat with bit 6 : 1, and 0. Thus the index is 01b = 1 (remember to flip the order of the bits !). If you repeat the operation you'll find that the indexes for the 8 pixels are 0 1 2 3 0 3 3 1.

A more visual explanation can be found here.

So, each pixel is having a color number in range from 0-3. The color numbers are translated into real colors (or gray shades) depending on the current palettes. The palettes are defined through registers BGP, OBP0 and OBP1 (Non CGB Mode), and BCPS/BGPI, BCPD/BGPD, OCPS/OBPI and OCPD/OBPD (CGB Mode).

4.10 VRAM Background Maps

The Game Boy contains two 32x32 tile background maps in VRAM at addresses 9800h-9BFFh and 9C00h-9FFFh. Each can be used either to display "normal" background, or "window" background.

BG Map Tile Numbers

An area of VRAM known as Background Tile Map contains the numbers of tiles to be displayed. It is organized as 32 rows of 32 bytes each. Each byte contains a number of a tile to be displayed.

Tile patterns are taken from the Tile Data Table using either of the two addressing modes (described above), which can be selected via LCDC register.

As one background tile has a size of 8x8 pixels, the BG maps may hold a picture of 256x256 pixels, and an area of 160x144 pixels of this picture can be displayed on the LCD screen.

BG Map Attributes (CGB Mode only)

In CGB Mode, an additional map of 32x32 bytes is stored in VRAM Bank 1 (each byte defines attributes for the corresponding tile-number map entry in VRAM Bank 0, ie. 1:9800 defines the attributes for the tile at 0:9800):

```
Bit 0-2  Background Palette number  (BGP0-7)
Bit 3    Tile VRAM Bank number      (0=Bank 0, 1=Bank 1)
Bit 4    Not used
Bit 5    Horizontal Flip            (0=Normal, 1=Mirror horizontally)
Bit 6    Vertical Flip              (0=Normal, 1=Mirror vertically)
Bit 7    BG-to-OAM Priority         (0=Use OAM priority bit, 1=BG Priority)
```

When Bit 7 is set, the corresponding BG tile will have priority above all OBJs (regardless of the priority bits in OAM memory). There's also a Master

Priority flag in LCDC register Bit 0 which overrides all other priority bits when cleared.

Note that, if the map entry at 0:9800 is tile $2A, the attribute at 1:9800 doesn't define properties for ALL tiles $2A on-screen, but only the one at 0:9800 !

Normal Background (BG)

The SCY and SCX registers can be used to scroll the background, allowing to select the origin of the visible 160x144 pixel area within the total 256x256 pixel background map. Background wraps around the screen (i.e. when part of it goes off the screen, it appears on the opposite side.)

The Window

Besides background, there is also a "window" overlaying the background. The window is not scrollable, that is, it is always displayed starting from its left upper corner. The location of a window on the screen can be adjusted via WX and WY registers. Screen coordinates of the top left corner of a window are WX-7,WY. The tiles for the window are stored in the Tile Data Table. Both the Background and the window share the same Tile Data Table.

Both background and window can be disabled or enabled separately via bits in the LCDC register.

4.11 VRAM Banks (CGB only)

The CGB has twice the VRAM of the DMG, but it is banked and either bank has a different purpose.

FF4F - VBK - CGB Mode Only - VRAM Bank (R/W)

This register can be written to to change VRAM banks. Only bit 0 matters, all other bits are ignored.

VRAM bank 1

VRAM bank 1 is split like VRAM bank 0 ; 8000-97FF also stores tiles (just like in bank 0), which can be accessed the same way as (and at the same time as) bank 0 tiles. 9800-9FFF contains the attributes for the corresponding Tile Maps.

Reading from this register will return the number of the currently loaded VRAM bank in bit 0, and all other bits will be set to 1.

4.12 VRAM Sprite Attribute Table (OAM)

Gameboy video controller can display up to 40 sprites either in 8x8 or in 8x16 pixels. Because of a limitation of hardware, only ten sprites can be displayed per scan line. Sprite patterns have the same format as BG tiles, but they are taken from the Sprite Pattern Table located at $8000-8FFF and have unsigned numbering.

Sprite attributes reside in the Sprite Attribute Table (OAM - Object Attribute Memory) at $FE00-FE9F. Each of the 40 entries consists of four bytes with the following meanings:

Byte0 - Y Position

Specifies the sprites vertical position on the screen (minus 16). An off-screen value (for example, Y=0 or Y>=160) hides the sprite.

Byte1 - X Position

Specifies the sprites horizontal position on the screen (minus 8). An off-screen value (X=0 or X>=168) hides the sprite, but the sprite still affects the priority ordering - a better way to hide a sprite is to set its Y-coordinate off-screen.

Byte2 - Tile/Pattern Number

Specifies the sprites Tile Number (00-FF). This (unsigned) value selects a tile from memory at 8000h-8FFFh. In CGB Mode this could be either in VRAM Bank 0 or 1, depending on Bit 3 of the following byte. In 8x16 mode, the lower bit of the tile number is ignored. IE: the upper 8x8 tile is “NN AND FEh”, and the lower 8x8 tile is “NN OR 01h”.

Byte3 - Attributes/Flags:

```
Bit7   OBJ-to-BG Priority (0=OBJ Above BG, 1=OBJ Behind BG color 1-3)
       (Used for both BG and Window. BG color 0 is always behind OBJ)
Bit6   Y flip           (0=Normal, 1=Vertically mirrored)
Bit5   X flip           (0=Normal, 1=Horizontally mirrored)
Bit4   Palette number  **Non CGB Mode Only** (0=OBP0, 1=OBP1)
Bit3   Tile VRAM-Bank  **CGB Mode Only**     (0=Bank 0, 1=Bank 1)
Bit2-0 Palette number  **CGB Mode Only**     (OBP0-7)
```

Sprite Priorities and Conflicts

During each scanline’s OAM scan, the LCD controller compares LY to each sprite’s Y position to find the 10 sprites on that line that appear first in OAM ($FE00-$FE03 being the first). It discards the rest, allowing only 10 sprites to be displayed on any one line. When this limit is exceeded, sprites appearing later in OAM won’t be displayed. To keep unused sprites from affecting onscreen sprites, set their Y coordinate to Y = 0 or Y >= 160 (144 + 16) (Note : Y <= 8 also works if sprite size is set to 8x8). Just setting the X coordinate to X = 0 or X >= 168 (160 + 8) on a sprite will hide it, but it will still affect other sprites sharing the same lines.

If using `BGB`, in the VRAM viewer - OAM tab, hover your mouse over the small screen to highlight the sprites on a line. Sprites hidden due to the limitation will be highlighted in red.

When these 10 sprites overlap, the highest priority one will appear above all others, etc. (Thus, no Z-fighting.) In CGB mode, the first sprite in OAM ($FE00-$FE03) has the highest priority, and so on. In Non-CGB mode, the smaller the X coordinate, the higher the priority. The tie breaker (same X coordinates) is the same priority as in CGB mode.

The priority calculation between sprites disregards OBJ-to-BG Priority (attribute bit 7). Only the highest-priority nonzero sprite pixel at any given point is compared against the background. Thus if a sprite with a higher priority (based on OAM index) but with OBJ-to-BG Priority turned on overlaps a sprite with a lower priority and a nonzero background pixel, the background pixel is displayed regardless of the lower-priority sprite's OBJ-to-BG Priority.

Writing Data to OAM Memory

The recommended method is to write the data to normal RAM first, and to copy that RAM to OAM by using the DMA transfer function, initiated through DMA register (FF46). Beside for that, it is also possible to write data directly to the OAM area by using normal LD commands, this works only during the H-Blank and V-Blank periods. The current state of the LCD controller can be read out from the STAT register (FF41).

4.13 Accessing VRAM and OAM

When the LCD Controller is drawing the screen it is directly reading from Video Memory (VRAM) and from the Sprite Attribute Table (OAM). During these periods the Game Boy CPU may not access the VRAM and OAM. That means, any attempts to write to VRAM/OAM are ignored (the data remains unchanged). And any attempts to read from VRAM/OAM will return undefined data (typically a value of FFh).

For this reason the program should verify if VRAM/OAM is accessible before actually reading or writing to it. This is usually done by reading the Mode Bits from the STAT Register (FF41). When doing this (as described in the examples below) you should take care that no interrupts occur between the wait loops and the following memory access - the memory is guaranteed to be accessible only for a few cycles directly after the wait loops have completed.

VRAM (memory at 8000h-9FFFh) is accessible during Mode 0-2

```
Mode 0 - H-Blank Period,
Mode 1 - V-Blank Period, and
Mode 2 - Searching OAM Period
```

A typical procedure that waits for accessibility of VRAM would be:

```
ld   hl,0FF41h    ;-STAT Register
@@wait:           ;
bit  1,(hl)       ; Wait until Mode is 0 or 1
jr   nz,@@wait    ;
```

Even if the procedure gets executed at the *end* of Mode 0 or 1, it is still proof to assume that VRAM can be accessed for a few more cycles because in either case the following period is Mode 2 which allows access to VRAM either. However, be careful about STAT LCD interrupts or other interrupts that could cause the LCD to be back in mode 3 by the time it returns. In CGB Mode an alternate method to write data to VRAM is to use the HDMA Function (FF51-FF55).

If you're not using LCD interrupts, another way to synchronize to the start of mode 0 is to use `halt` with IME turned off (`di`). This allows use of the entire mode 0 on one line and mode 2 on the following line, which sum to 165 to 288 dots. For comparison, at single speed (4 dots per machine cycle), a copy from stack that takes 9 cycles per 2 bytes can push 8 bytes (half a tile) in 144 dots, which fits within the worst case timing for mode 0+2.

OAM (memory at FE00h-FE9Fh) is accessible during Mode 0-1

```
Mode 0 - H-Blank Period
Mode 1 - V-Blank Period
```

Aside from that, OAM can be accessed at any time by using the DMA Function (FF46). When directly reading or writing to OAM, a typical procedure that waits for accessibility of OAM Memory would be:

```
 ld   hl,0FF41h    ;-STAT Register
@@wait1:           ;
 bit  1,(hl)       ; Wait until Mode is -NOT- 0 or 1
 jr   z,@@wait1    ;
@@wait2:           ;
 bit  1,(hl)       ; Wait until Mode 0 or 1 -BEGINS-
 jr   nz,@@wait2   ;
```

The two wait loops ensure that Mode 0 or 1 will last for a few clock cycles after completion of the procedure. In V-Blank period it might be recommended to skip the whole procedure - and in most cases using the above mentioned DMA function would be more recommended anyways.

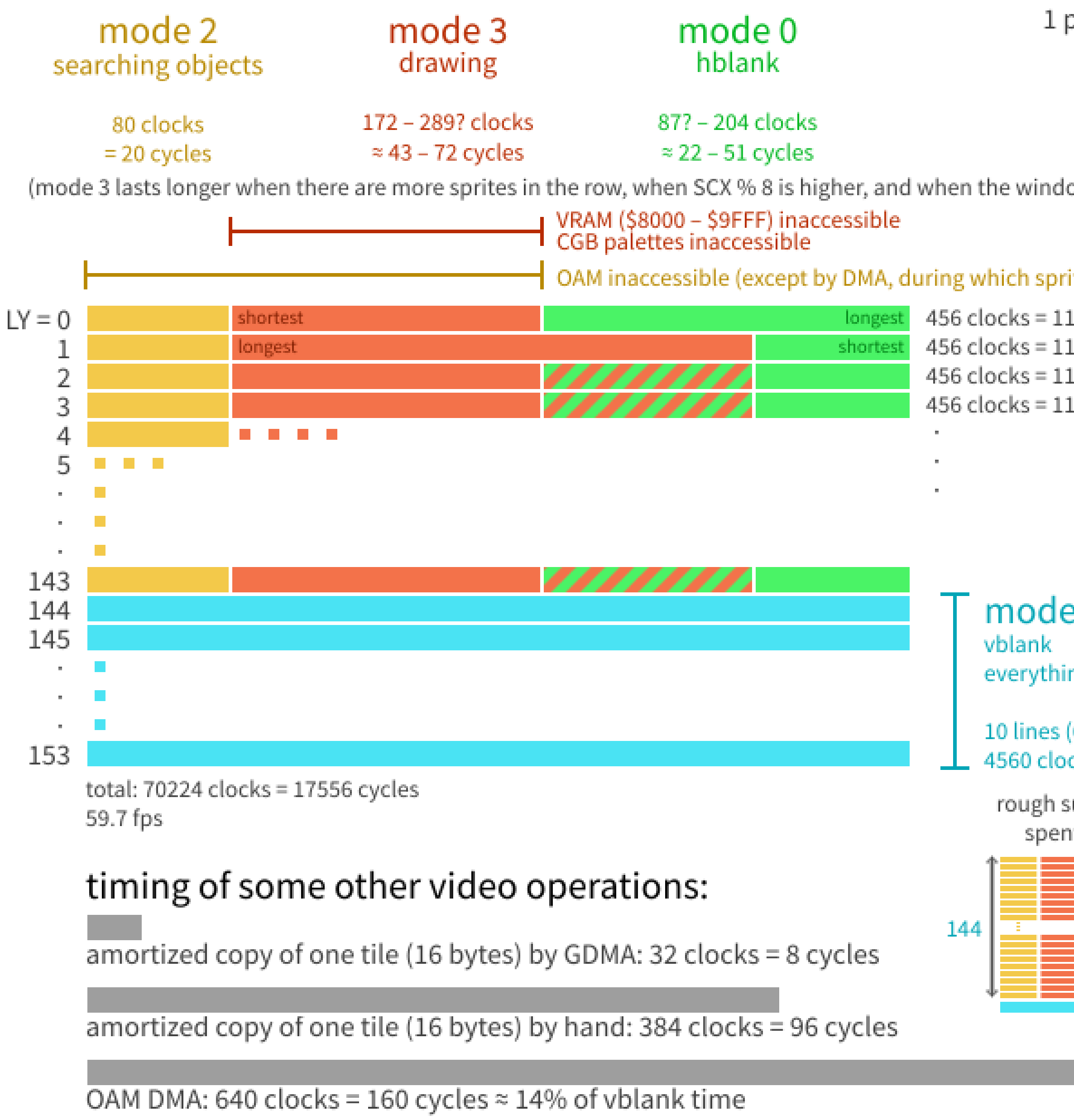
::: tip NOTE

When the display is disabled, both VRAM and OAM are accessible at any time. The downside is that the screen is blank (white) during this period, so that disabling the display would be recommended only during initialization.

:::

4.14 Pixel FIFO

::: warning Terminology All references to a cycle are meant as T-cycles (4.19 MHz) and cycle counts are doubled on CGB in double speed mode. When it is stated that a certain action "lengthens mode 3" this means that mode 0 (hblank) is shortened. The extra time has to be made up somewhere and the lengthening of mode 3 eats into the mode that comes afterwards, which happens to be mode 0 as shown in the following diagram. :::



mode 2
searching objects
80 clocks
= 20 cycles
mode 3
drawing
172 – 289? clocks
≈ 43 – 72 cycles
mode 0
hblank
87? – 204 clocks
≈ 22 – 51 cycles
1 p
(mode 3 lasts longer when there are more sprites in the row, when SCX % 8 is higher, and when the windo
VRAM ($8000 – $9FFF) inaccessible
CGB palettes inaccessible
OAM inaccessible (except by DMA, during which spri
LY = 0
1
2
3
4
5
143
144
145
153
shortest
longest
longest
shortest
456 clocks = 11
456 clocks = 11
456 clocks = 11
456 clocks = 11
mode
vblank
everythin
10 lines (
4560 cloc
total: 70224 clocks = 17556 cycles
59.7 fps
rough su
spen
144
timing of some other video operations:
amortized copy of one tile (16 bytes) by GDMA: 32 clocks = 8 cycles
amortized copy of one tile (16 bytes) by hand: 384 clocks = 96 cycles
OAM DMA: 640 clocks = 160 cycles ≈ 14% of vblank time

4.14.1 Introduction

FIFO stands for *First In, First Out*. The first pixel to be pushed to the FIFO is the first pixel to be popped off. In theory that sounds great, in practice there are a lot of intricacies.

There are two pixel FIFOs. One for background pixels and one for OAM (sprite) pixels. These two FIFOs are not shared. They are independent of each other. The two FIFOs are mixed only when popping items. Sprites take priority unless they're transparent (color 0) which will be explained in detail later. Each FIFO can hold up to 16 pixels. The FIFO and Pixel Fetcher work together to ensure that the FIFO always contains at least 8 pixels at any given time, as 8 pixels are required for the Pixel Rendering operation to take place. Each FIFO is manipulated only during mode 3 (pixel transfer).

Each pixel in the FIFO has four properties: - Color: a value between 0 and 3 - Palette: on CGB a value between 0 and 7 and on DMG this only applies to sprites - Sprite Priority: on CGB this is the OAM index for the sprite and on DMG this doesn't exist - Background Priority: holds the value of the OBJ-to-BG Priority bit

4.14.2 FIFO Pixel Fetcher

The fetcher fetches a row of 8 background or window pixels and queues them up to be mixed with sprite pixels. The pixel fetcher has 5 steps. The first four steps take 2 cycles each and the fifth step is attempted every cycle until it succeeds. The order of the steps are as follows:

- Get tile
- Get tile data low
- Get tile data high
- Sleep
- Push

Get Tile:

This step determines which background/window tile to fetch pixels from. By default the tilemap used is the one at $9800 but certain conditions can change that.

When LCDC.3 is enabled and the X coordinate of the current scanline is not inside the window then tilemap $9C00 is used.

When LCDC.6 is enabled and the X coordinate of the current scanline is inside the window then tilemap $9C00 is used.

The fetcher keeps track of which X and Y coordinate of the tile it's on:

If the current tile is a window tile, the X coordinate for the window tile is used, otherwise the following formula is used to calculate the X coordinate: ((SCX / 8) + fetcher's X coordinate) & $1F. Because of this formula, fetcherX can be between 0 and 31.

If the current tile is a window tile, the Y coordinate for the window tile is used, otherwise the following formula is used to calculate the Y coordinate: (currentScanline + SCY) & 255. Because of this formula, fetcherY can be between 0 and 159.

The fetcher's X and Y coordinate can then be used to get the tile from VRAM. However, if the PPU's access to VRAM is blocked then the value for the tile is read as $FF.

CGB can access both tile index and the attributes in the same clock cycle.

Get Tile Data Low:

Check LCDC.4 for which tilemap to use. At this step CGB also needs to check which VRAM bank to use and check if the tile is flipped vertically. Once the tilemap, VRAM and vertical flip is calculated the tile data is retrieved from VRAM. However, if the PPU's access to VRAM is blocked then the tile data is read as $FF.

The tile data retrieved in this step will be used in the push steps.

Get Tile Data High:

Same as Get Tile Data Low except the tile address is incremented by 1.

The tile data retrieved in this step will be used in the push steps.

This also pushes a row of background/window pixels to the FIFO. This extra push is not part of the 8 steps, meaning there's 3 total chances to push pixels to the background FIFO every time the complete fetcher steps are performed.

Push:

Pushes a row of background/window pixels to the FIFO. Since tiles are 8 pixels wide, a "row" of pixels is 8 pixels from the tile to be rendered based on the X and Y coordinates calculated in the previous steps.

Pixels are only pushed to the background FIFO if it's empty.

This is where the tile data retrieved in the two Tile Data steps will come in handy. Depending on if the tile is flipped horizontally the pixels will be pushed to the background FIFO differently. If the tile is flipped horizontally the pixels will be pushed LSB first. Otherwise they will be pushed MSB first.

Sleep:

Do nothing.

VRAM Access

At various times during PPU operation read access to VRAM is blocked and the value read is $FF: - LCD turning off - At scanline 0 on CGB when not in double speed mode - When switching from mode 3 to mode 0 - On CGB when searching OAM and index 37 is reached

At various times during PPU operation read access to VRAM is restored: - At scanline 0 on DMG and CGB when in double speed mode - On DMG when searching OAM and index 37 is reached - After switching from mode 2 (oam search) to mode 3 (pixel transfer)

NOTE: These conditions are checked only when entering STOP mode and the PPU's access to VRAM is always restored upon leaving STOP mode.

4.14.3 Mode 3 Operation

As stated before the pixel FIFO only operates during mode 3 (pixel transfer). At the beginning of mode 3 both the background and OAM FIFOs are cleared.

The Window

When rendering the window the background FIFO is cleared and the fetcher is reset to step 1. When WX is 0 and the SCX & 7 > 0 mode 3 is shortened by 1 cycle.

When the window has already started rendering there is a bug that occurs when WX is changed mid-scanline. When the value of WX changes after the window has started rendering and the new value of WX is reached again, a pixel with color value of 0 and the lowest priority is pushed onto the background FIFO.

Sprites

The following is performed for each sprite on the current scanline if LCDC.1 is enabled (this condition is ignored on CGB) and the X coordinate of the current scanline has a sprite on it. If those conditions are not met then sprite fetching is aborted.

At this point the fetcher is advanced one step until it's at step 5 or until the background FIFO is not empty. Advancing the fetcher one step here lengthens mode 3 by 1 cycle. This process may be aborted after the fetcher has advanced a step.

When SCX & 7 > 0 and there is a sprite at X coordinate 0 of the current scanline then mode 3 is lengthened. The amount of cycles this lengthens mode 3 by is whatever the lower 3 bits of SCX are. After this penalty is applied object fetching may be aborted. Note that the timing of the penalty is not confirmed. It may happen before or after waiting for the fetcher. More research needs to be done.

After checking for sprites at X coordinate 0 the fetcher is advanced two steps. The first advancement lengthens mode 3 by 1 cycle and the second advancement lengthens mode 3 by 3 cycles. After each fetcher advancement there is a chance for a sprite fetch abortion to occur.

The lower address for the row of pixels of the target object tile is now retrieved and lengthens mode 3 by 1 cycle. Once the address is retrieved this is the last chance for sprite fetch abortion to occur. Exiting object fetch lengthens mode 3 by 1 cycle. The upper address for the target object tile is now retrieved and does not shorten mode 3.

At this point VRAM Access is checked for the lower and upper addresses for the target object. Before any mixing is done, if the OAM FIFO doesn't have at least 8 pixels in it then transparent pixels with the lowest priority are pushed onto the OAM FIFO. Once this is done each pixel of the target object row is checked. On CGB, horizontal flip is checked here. If the target object pixel is not white and the pixel in the OAM FIFO *is* white, or if the pixel in the OAM FIFO has higher priority than the target object's pixel, then the pixel in the OAM FIFO is replaced with the target object's properties.

Now it's time to render a pixel! The same process described in Sprite Fetch Abortion is performed: a pixel is rendered and the fetcher is advanced one step. This advancement lengthens mode 3 by 1 cycle if the X coordinate of the current scanline is not 160. If the X coordinate is 160 the PPU stops processing sprites (because they won't be visible).

Everything in this section is repeated for every sprite on the current scanline unless it was decided that fetching should be aborted or the X coordinate is 160.

Pixel Rendering

This is where the background FIFO and OAM FIFO are mixed. There are conditions where either a background pixel or a sprite pixel will have display priority.

If there are pixels in the background and OAM FIFOs then a pixel is popped off each. If the OAM pixel is not transparent and LCDC.1 is enabled then the OAM pixel's background priority property is used if it's the same or higher priority as the background pixel's background priority.

Pixels won't be pushed to the LCD if there is nothing in the background FIFO or the current pixel is pixel 160 or greater.

If LCDC.0 is disabled then the background is disabled on DMG and the background pixel won't have priority on CGB. When the background pixel is disabled the pixel color value will be 0, otherwise the color value will be whatever color pixel was popped off the background FIFO. When the pixel popped off the background FIFO has a color value other than 0 and it has priority then the sprite pixel will be discarded.

At this point, on DMG, the color of the pixel is retrieved from the BGP register and pushed to the LCD. On CGB when palette access is blocked a black pixel is pushed to the LCD.

When a sprite pixel has priority the color value is retrieved from the popped pixel from the OAM FIFO. On DMG the color for the pixel is retrieved from either the OBP1 or OBP0 register depending on the pixel's palette property. If the palette property is 1 then OBP1 is used, otherwise OBP0 is used. The pixel is then pushed to the LCD. On CGB when palette access is blocked a black pixel is pushed to the LCD.

The pixel is then finally pushed to the LCD.

CGB Palette Access

At various times during PPU operation read access to the CGB palette is blocked and a black pixel pushed to the LCD when rendering pixels: - LCD turning off - First HBlank of the frame - When searching OAM and index 37 is reached -

After switching from mode 2 (oam search) to mode 3 (pixel transfer) - When entering HBlank (mode 0) and not in double speed mode, blocked 2 cycles later no matter what

At various times during PPU operation read access to the CGB palette is restored and pixels are pushed to the LCD normally when rendering pixels: - At the end of mode 2 (oam search) - For only 2 cycles when entering HBlank (mode 0) and in double speed mode

::: warning Note These conditions are checked only when entering STOP mode and the PPU's access to CGB palettes is always restored upon leaving STOP mode. :::

Sprite Fetch Abortion

Sprite fetching may be aborted if LCDC.1 is disabled while the PPU is fetching an object from OAM. This abortion lengthens mode 3 by the amount of cycles the previous instruction took plus the residual cycles left for the PPU to process. When OAM fetching is aborted a pixel is rendered, the fetcher is advanced one step. This advancement lengthens mode 3 by 1 cycle if the current pixel is not 160. If the current pixel is 160 the PPU stops processing sprites because they won't be visible.

Chapter 5

Sound Controller

There are two sound channels connected to the output terminals SO1 and SO2. There is also a input terminal Vin connected to the cartridge. It can be routed to either of both output terminals. Game Boy circuitry allows producing sound in four different ways:

```
Quadrangular wave patterns with sweep and envelope functions
Quadrangular wave patterns with envelope functions
Voluntary wave patterns from wave RAM
White noise with an envelope function
```

These four sounds can be controlled independantly and then mixed separately for each of the output terminals.
Sound registers may be set at all times while producing sound.
(Sounds will have a 2.4% higher frequency on Super GB.)

5.1 Sound Channel 1 - Tone & Sweep

FF10 - NR10 - Channel 1 Sweep register (R/W)

```
 Bit 6-4 - Sweep Time
 Bit 3   - Sweep Increase/Decrease
            0: Addition    (frequency increases)
            1: Subtraction (frequency decreases)
 Bit 2-0 - Number of sweep shift (n: 0-7)
```

Sweep Time:

```
000: sweep off - no freq change
001: 7.8 ms  (1/128Hz)
010: 15.6 ms (2/128Hz)
011: 23.4 ms (3/128Hz)
100: 31.3 ms (4/128Hz)
```

```
101: 39.1 ms (5/128Hz)
110: 46.9 ms (6/128Hz)
111: 54.7 ms (7/128Hz)
```

The change of frequency (NR13,NR14) at each shift is calculated by the following formula where X(0) is initial freq & X(t-1) is last freq:
X(t) = X(t-1) +/- X(t-1)/2^n‘

FF11 - NR11 - Channel 1 Sound length/Wave pattern duty (R/W)

```
Bit 7-6 - Wave Pattern Duty (Read/Write)
Bit 5-0 - Sound length data (Write Only) (t1: 0-63)
```

Wave Duty:

```
00: 12.5% ( _--------_--------_-------- )
01: 25%   ( __-------__-------__------- )
10: 50%   ( ____----____----____---- ) (normal)
11: 75%   ( ______--______--______-- )
```

Sound Length = (64-t1)*(1/256) seconds The Length value is used only if Bit 6 in NR14 is set.

FF12 - NR12 - Channel 1 Volume Envelope (R/W)

```
 Bit 7-4 - Initial Volume of envelope (0-0Fh) (0=No Sound)
 Bit 3   - Envelope Direction (0=Decrease, 1=Increase)
 Bit 2-0 - Number of envelope sweep (n: 0-7)
           (If zero, stop envelope operation.)
```

Length of 1 step = n*(1/64) seconds

FF13 - NR13 - Channel 1 Frequency lo (Write Only)

Lower 8 bits of 11 bit frequency (x). Next 3 bit are in NR14 ($FF14)

FF14 - NR14 - Channel 1 Frequency hi (R/W)

```
Bit 7   - Initial (1=Restart Sound)     (Write Only)`
Bit 6   - Counter/consecutive selection (Read/Write)`
          (1=Stop output when length in NR11 expires)`
Bit 2-0 - Frequency's higher 3 bits (x) (Write Only)`
```

Frequency = 131072/(2048-x) Hz

5.2 Sound Channel 2 - Tone

This sound channel works exactly as channel 1, except that it doesn’t have a Tone Envelope/Sweep Register.

FF16 - NR21 - Channel 2 Sound Length/Wave Pattern Duty (R/W)

```
Bit 7-6 - Wave Pattern Duty (Read/Write)`
Bit 5-0 - Sound length data (Write Only) (t1: 0-63)`
```

Wave Duty:

```
00: 12.5% ( _--------_--------_-------- )`
01: 25%   ( __-------__-------__------- )`
10: 50%   ( ____----____----____---- ) (normal)`
11: 75%   ( ______--______--______-- )`
```

Sound Length = (64-t1)*(1/256) seconds The Length value is used only if Bit 6 in NR24 is set.

FF17 - NR22 - Channel 2 Volume Envelope (R/W)

```
Bit 7-4 - Initial Volume of envelope (0-0Fh) (0=No Sound)`
Bit 3   - Envelope Direction (0=Decrease, 1=Increase)`
Bit 2-0 - Number of envelope sweep (n: 0-7)`
          (If zero, stop envelope operation.)`
```

Length of 1 step = n*(1/64) seconds

FF18 - NR23 - Channel 2 Frequency lo data (W)

Frequency's lower 8 bits of 11 bit data (x). Next 3 bits are in NR24 ($FF19).

FF19 - NR24 - Channel 2 Frequency hi data (R/W)

```
Bit 7   - Initial (1=Restart Sound)     (Write Only)`
Bit 6   - Counter/consecutive selection (Read/Write)`
          (1=Stop output when length in NR21 expires)`
Bit 2-0 - Frequency's higher 3 bits (x) (Write Only)`
```

Frequency = 131072/(2048-x) Hz

5.3 Sound Channel 3 - Wave Output

This channel can be used to output digital sound, the length of the sample buffer (Wave RAM) is limited to 32 digits. This sound channel can be also used to output normal tones when initializing the Wave RAM by a square wave. This channel doesn't have a volume envelope register.

FF1A - NR30 - Channel 3 Sound on/off (R/W)

```
Bit 7 - Sound Channel 3 Off  (0=Stop, 1=Playback)  (Read/Write)`
```

FF1B - NR31 - Channel 3 Sound Length

```
Bit 7-0 - Sound length (t1: 0 - 255)`
```

Sound Length = (256-t1)*(1/256) seconds This value is used only if Bit 6 in NR34 is set.

FF1C - NR32 - Channel 3 Select output level (R/W)

Bit 6-5 - Select output level (Read/Write)‘
Possible Output levels are:

```
0: Mute (No sound)`
1: 100% Volume (Produce Wave Pattern RAM Data as it is)`
2:  50% Volume (Produce Wave Pattern RAM data shifted once to the right)`
3:  25% Volume (Produce Wave Pattern RAM data shifted twice to the right)`
```

FF1D - NR33 - Channel 3 Frequency’s lower data (W)

Lower 8 bits of an 11 bit frequency (x).

FF1E - NR34 - Channel 3 Frequency’s higher data (R/W)

```
Bit 7   - Initial (1=Restart Sound)     (Write Only)`
Bit 6   - Counter/consecutive selection (Read/Write)`
          (1=Stop output when length in NR31 expires)`
Bit 2-0 - Frequency's higher 3 bits (x) (Write Only)`
```

Frequency = 4194304/(64*(2048-x)) Hz = 65536/(2048-x) Hz

FF30-FF3F - Wave Pattern RAM

Contents - Waveform storage for arbitrary sound data‘
This storage area holds 32 4-bit samples that are played back, upper 4 bits first.

Wave RAM should only be accessed while CH3 is disabled (NR30 bit 7 reset), otherwise accesses will behave weirdly.

On almost all models, the byte will be written at the offset CH3 is currently reading. On GBA, the write will simply be ignored.

5.4 Sound Channel 4 - Noise

This channel is used to output white noise. This is done by randomly switching the amplitude between high and low at a given frequency. Depending on the frequency the noise will appear ‘harder’ or ‘softer’.

It is also possible to influence the function of the random generator, so the that the output becomes more regular, resulting in a limited ability to output Tone instead of Noise.

FF20 - NR41 - Channel 4 Sound Length (R/W)

```
Bit 5-0 - Sound length data (t1: 0-63)
```

Sound Length = (64-t1)*(1/256) seconds The Length value is used only if Bit 6 in NR44 is set.

FF21 - NR42 - Channel 4 Volume Envelope (R/W)

```
 Bit 7-4 - Initial Volume of envelope (0-0Fh) (0=No Sound)
 Bit 3   - Envelope Direction (0=Decrease, 1=Increase)
 Bit 2-0 - Number of envelope sweep (n: 0-7)`
           (If zero, stop envelope operation.)
```

Length of 1 step = n*(1/64) seconds

FF22 - NR43 - Channel 4 Polynomial Counter (R/W)

The amplitude is randomly switched between high and low at the given frequency. A higher frequency will make the noise to appear 'softer'. When Bit 3 is set, the output will become more regular, and some frequencies will sound more like Tone than Noise.

```
Bit 7-4 - Shift Clock Frequency (s)
Bit 3   - Counter Step/Width (0=15 bits, 1=7 bits)
Bit 2-0 - Dividing Ratio of Frequencies (r)
```

Frequency = 524288 Hz / r / 2^(s+1) ;For r=0 assume r=0.5 instead

FF23 - NR44 - Channel 4 Counter/consecutive; Inital (R/W)

```
Bit 7   - Initial (1=Restart Sound)     (Write Only)
Bit 6   - Counter/consecutive selection (Read/Write)
          (1=Stop output when length in NR41 expires)
```

5.5 Sound Control Registers

FF24 - NR50 - Channel control / ON-OFF / Volume (R/W)

The volume bits specify the "Master Volume" for Left/Right sound output. SO2 goes to the left headphone, and SO1 goes to the right.

```
Bit 7   - Output Vin to SO2 terminal (1=Enable)
Bit 6-4 - SO2 output level (volume)  (0-7)
Bit 3   - Output Vin to SO1 terminal (1=Enable)
Bit 2-0 - SO1 output level (volume)  (0-7)
```

The Vin signal is an analog signal received from the game cartridge bus, allowing external hardware in the cartridge to supply a fifth sound channel, additionally to the Game Boy's internal four channels. No licensed games used this feature, and it was omitted from the Game Boy Advance.

(Despite rumors, *Pocket Music* does not use Vin. It blocks use on the GBA for a different reason: the developer couldn't figure out how to silence buzzing associated with the wave channel's DAC.)

FF25 - NR51 - Selection of Sound output terminal (R/W)

Each channel can be panned hard left, center, or hard right.

```
Bit 7 - Output sound 4 to SO2 terminal
Bit 6 - Output sound 3 to SO2 terminal
Bit 5 - Output sound 2 to SO2 terminal
Bit 4 - Output sound 1 to SO2 terminal
Bit 3 - Output sound 4 to SO1 terminal
Bit 2 - Output sound 3 to SO1 terminal
Bit 1 - Output sound 2 to SO1 terminal
Bit 0 - Output sound 1 to SO1 terminal
```

FF26 - NR52 - Sound on/off

If your GB programs don't use sound then write 00h to this register to save 16% or more on GB power consumption. Disabeling the sound controller by clearing Bit 7 destroys the contents of all sound registers. Also, it is not possible to access any sound registers (execpt FF26) while the sound controller is disabled.

```
 Bit 7 - All sound on/off  (0: stop all sound circuits) (Read/Write)
 Bit 3 - Sound 4 ON flag (Read Only)
 Bit 2 - Sound 3 ON flag (Read Only)
 Bit 1 - Sound 2 ON flag (Read Only)
 Bit 0 - Sound 1 ON flag (Read Only)
```

Bits 0-3 of this register are read only status bits, writing to these bits does NOT enable/disable sound. The flags get set when sound output is restarted by setting the Initial flag (Bit 7 in NR14-NR44), the flag remains set until the sound length has expired (if enabled). A volume envelopes which has decreased to zero volume will NOT cause the sound flag to go off.

5.6 Pitfalls

- Enabling or disabling a DAC (resetting NR30 bit 7 or writing %0000 0XXX to NRx2 for other channels), adding or removing it using NR51, or changing the volume in NR50, will cause an audio pop. (This causes a change in DC offset, which is smoothed out by a high-pass circuit over time, but still creates a pop)

- The final output goes through a high-pass filter, which is more aggressive on GBA than on GBC, which is more aggressive than on DMG. (What this means is that the output is “pulled” towards 0V with various degrees of “aggressiveness”)
- When first starting up a pulse channel, it will *always* output a (digital) zero.
- The pulse channels’ “duty step” (at which position in the duty cycle they are) can’t be reset. The exception to this is turning off the APU, which causes them to start over from 0 when turning it on.
- Restarting a pulse channel causes its “duty step timer” to reset, meaning that “tickling” a pulse channel regularly enough will cause its “duty step” to never advance.
- When restarting CH3, it resumes playing the last 4-bit sample it read from wave RAM, or 0 if no sample has been read since APU reset. (Sample latching is independent of output level control in NR32.) After the latched sample completes, it starts with the second sample in wave RAM (low 4 bits of $FF30). The first sample (high 4 bits of $FF30) is played last.
- CH3 output level control does not, in fact, alter the output level. It shifts the **digital** value CH3 is outputting (read below), not the analog value.
- On GBA, CH3 is inverted. This causes the channel to output a loud spike when disabled; it’s a good idea to “remove” the channel using NR51 before refreshing wave RAM.

5.7 APU technical explanation

Game Boy, Game Boy Color

Each of the 4 channels work pretty identically. First, there’s a “generation” circuit, which usually outputs either a 0 or another value (CH3 differs in that it can output multiple values, but regardless). That value is digital, and can range between 0 and 0xF. This is then fed to a DAC, which maps this to an analog value; 7 maps to the lowest (negative) voltage, 0 to the highest (positive) one. Finally, all channels are mixed through NR51, scaled through NR50, and sent to the output.

Each DAC is controlled independently from the generation circuit. For CH3, the DAC is controlled by NR30 bit 7; for other channels, the DAC is turned on unless bits 3-7 of NRx2 are reset, and the envelope will be set to `[NRx2] >> 4`. (Note: the envelope sweep function changes the envelope, but not the value in NRx2! It won’t disable the DAC, either.) The generation circuits are turned on by restarting them for the first time, and this is what sets the corresponding bit in NR52. Yes, it’s possible to turn on a DAC but not the generation circuit. Finally, disabling a DAC also kills the generation circuit.

Note that each DAC has a DC offset, so enabling, disabling, adding to or removing from NR51, will all cause an audio pop; changing the volume in NR50 will as well.

Finally, all the output goes through a high-pass filter to remove the DC offsets from the DACs.

Game Boy Advance

The APU was reworked pretty heavily for the GBA. Instead of mixing being done analogically, it's instead done digitally; then, sound is converted to an analog signal and an offset is added (see SOUNDBIAS in GBATEK for more details.

This means that the APU has no DACs, or if modelling the GBA as a GB, they're always on. # Joypad Input

FF00 - P1/JOYP - Joypad (R/W)

The eight Game Boy buttons/direction keys are arranged in form of a 2x4 matrix. Select either button or direction keys by writing to this register, then read-out bit 0-3.

```
Bit 7 - Not used
Bit 6 - Not used
Bit 5 - P15 Select Button Keys      (0=Select)
Bit 4 - P14 Select Direction Keys   (0=Select)
Bit 3 - P13 Input Down  or Start    (0=Pressed) (Read Only)
Bit 2 - P12 Input Up    or Select   (0=Pressed) (Read Only)
Bit 1 - P11 Input Left  or Button B (0=Pressed) (Read Only)
Bit 0 - P10 Input Right or Button A (0=Pressed) (Read Only)
```

::: tip NOTE Most programs are repeatedly reading from this port several times (the first reads used as short delay, allowing the inputs to stabilize, and only the value from the last read actually used). :::

Usage in SGB software

Beside for normal joypad input, SGB games mis-use the joypad register to output SGB command packets to the SNES, also, SGB programs may read out gamepad states from up to four different joypads which can be connected to the SNES. See SGB description for details.

INT 60 - Joypad Interrupt

Joypad interrupt is requested when any of the above Input lines changes from High to Low. Generally this should happen when a key becomes pressed (provided that the button/direction key is enabled by above Bit4/5), however, because of switch bounce, one or more High to Low transitions are usually produced both when pressing or releasing a key.

Using the Joypad Interrupt

It's more or less useless for programmers, even when selecting both buttons and direction keys simultaneously it still cannot recognize all keystrokes, because in that case a bit might be already held low by a button key, and pressing the corresponding direction key would thus cause no difference. The only meaningful purpose of the keystroke interrupt would be to terminate STOP (low power) standby state. GBA SP, because of the different buttons used, seems to not be affected by switch bounce.

Chapter 6

Serial Data Transfer

Communication between two Gameboys happens one byte at a time. One Gameboy acts as the master, uses its internal clock, and thus controls when the exchange happens. The other one uses an external clock (i.e., the one inside the other Gameboy) and has no control over when the transfer happens. If it hasn't gotten around to loading up the next data byte at the time the transfer begins, the last one will go out again. Alternately, if it's ready to send the next byte but the last one hasn't gone out yet, it has no choice but to wait.

FF01 - SB - Serial transfer data (R/W)

Before a transfer, it holds the next byte that will go out.

During a transfer, it has a blend of the outgoing and incoming bytes. Each cycle, the leftmost bit is shifted out (and over the wire) and the incoming bit is shifted in from the other side:

```
o7 o6 o5 o4 o3 o2 o1 o0
o6 o5 o4 o3 o2 o1 o0 i7
o5 o4 o3 o2 o1 o0 i7 i6
o4 o3 o2 o1 o0 i7 i6 i5
o3 o2 o1 o0 i7 i6 i5 i4
o2 o1 o0 i7 i6 i5 i4 i3
o1 o0 i7 i6 i5 i4 i3 i2
o0 i7 i6 i5 i4 i3 i2 i1
i7 i6 i5 i4 i3 i2 i1 i0
```

FF02 - SC - Serial Transfer Control (R/W)

```
Bit 7 - Transfer Start Flag (0=No transfer is in progress or requested, 1=Transfer in progre
Bit 1 - Clock Speed (0=Normal, 1=Fast) ** CGB Mode Only **
Bit 0 - Shift Clock (0=External Clock, 1=Internal Clock)
```

The Game Boy acting as master will load up a data byte in SB and then set SC to 0x81 (Transfer requested, use internal clock). It will be notified that the transfer is complete in two ways: SC's Bit 7 will be cleared (i.e., SC will be set up 0x01), and also the Serial Interrupt handler will be called (i.e., the CPU will jump to 0x0058).

The other Game Boy will load up a data byte and can optionally set SC's Bit 7 (i.e., SC=0x80). Regardless of whether or not it has done this, if and when the master Game Boy wants to conduct a transfer, it will happen (pulling whatever happens to be in SB at that time). The passive gameboy will have its serial interrupt handler called at the end of the transfer, and if it bothered to set SC's Bit 7, it will be cleared.

Internal Clock

In Non-CGB Mode the Game Boy supplies an internal clock of 8192Hz only (allowing to transfer about 1 KByte per second). In CGB Mode four internal clock rates are available, depending on Bit 1 of the SC register, and on whether the CGB Double Speed Mode is used:

```
   8192Hz -  1KB/s - Bit 1 cleared, Normal
  16384Hz -  2KB/s - Bit 1 cleared, Double Speed Mode
 262144Hz - 32KB/s - Bit 1 set,     Normal
 524288Hz - 64KB/s - Bit 1 set,     Double Speed Mode
```

External Clock

The external clock is typically supplied by another gameboy, but might be supplied by another computer (for example if connected to a PCs parallel port), in that case the external clock may have any speed. Even the old/monochrome Game Boy is reported to recognizes external clocks of up to 500KHz. And there is no limitation into the other direction - even when suppling an external clock speed of "1 bit per month", then the gameboy will still eagerly wait for the next bit(s) to be transferred. It isn't required that the clock pulses are sent at an regular interval either.

Timeouts

When using external clock then the transfer will not complete until the last bit is received. In case that the second Game Boy isn't supplying a clock signal, if it gets turned off, or if there is no second gameboy connected at all) then transfer will never complete. For this reason the transfer procedure should use a timeout counter, and abort the communication if no response has been received during the timeout interval.

Delays and Synchronization

The Game Boy that is using internal clock should always execute a small delay between each transfer, in order to ensure that the opponent gameboy has enough time to prepare itself for the next transfer, ie. the gameboy with external clock must have set its transfer start bit before the Game Boy with internal clock starts the transfer. Alternately, the two gameboys could switch between internal and external clock for each transferred byte to ensure synchronization.

Transfer is initiated by setting the master Game Boy setting its Transfer Start Flag, regardless of the value of this flag on the other device. This bit is automatically set to 0 (on both) at the end of Transfer. Reading this bit can be used to determine if the transfer is still active.

INT 58 - Serial Interrupt

When the transfer has completed (ie. after sending/receiving 8 bits, if any) then an interrupt is requested by setting Bit 3 of the IF Register (FF0F). When that interrupt is enabled, then the Serial Interrupt vector at 0058 is called.

XXXXXX...

Transmitting and receiving serial data is done simultaneously. The received data is automatically stored in SB.

The serial I/O port on the Game Boy is a very simple setup and is crude compared to standard RS-232 (IBM-PC) or RS-485 (Macintosh) serial ports. There are no start or stop bits.

During a transfer, a byte is shifted in at the same time that a byte is shifted out. The rate of the shift is determined by whether the clock source is internal or external. The most significant bit is shifted in and out first.

When the internal clock is selected, it drives the clock pin on the game link port and it stays high when not used. During a transfer it will go low eight times to clock in/out each bit.

The state of the last bit shifted out determines the state of the output line until another transfer takes place.

If a serial transfer with internal clock is performed and no external GameBoy is present, a value of $FF will be received in the transfer.

The following code initiates the process of shifting $75 out the serial port and a byte to be shifted into $FF01:

```
ld   a,$75
ld  ($FF01),a
ld   a,$81
ld  ($FF02),a
```

The Game Boy does not support wake-on-LAN. Completion of an externally clocked serial transfer does not exit STOP mode.

Chapter 7

Timer and Divider Registers

FF04 - DIV - Divider Register (R/W)

This register is incremented at rate of 16384Hz (~16779Hz on SGB). Writing any value to this register resets it to 00h.

Note: The divider is affected by CGB double speed mode, and will increment at 32768Hz in double speed.

FF05 - TIMA - Timer counter (R/W)

This timer is incremented by a clock frequency specified by the TAC register ($FF07). When the value overflows (gets bigger than FFh) then it will be reset to the value specified in TMA (FF06), and an interrupt will be requested, as described below.

FF06 - TMA - Timer Modulo (R/W)

When the TIMA overflows, this data will be loaded.

FF07 - TAC - Timer Control (R/W)

```
Bit  2   - Timer Enable
Bits 1-0 - Input Clock Select
           00: CPU Clock / 1024 (DMG, CGB:   4096 Hz, SGB:   ~4194 Hz)
           01: CPU Clock / 16   (DMG, CGB: 262144 Hz, SGB: ~268400 Hz)
           10: CPU Clock / 64   (DMG, CGB:  65536 Hz, SGB:  ~67110 Hz)
           11: CPU Clock / 256  (DMG, CGB:  16384 Hz, SGB:  ~16780 Hz)
```

::: tip NOTE The “Timer Enable” bit only affects the timer, the divider is **always** counting :::

INT 50 - Timer Interrupt

Each time when the timer overflows (ie. when TIMA gets bigger than FFh), then an interrupt is requested by setting Bit 2 in the IF Register (FF0F). When that interrupt is enabled, then the CPU will execute it by calling the timer interrupt vector at 0050h.

Timer Obscure Behaviour

Read this page for a more detailed description of what the registers do: Timer Obscure Behaviour

::: tip NOTE The above described Timer is the built-in timer in the gameboy. It has nothing to do with the MBC3s battery buffered Real Time Clock - that's a completely different thing, described in the chapter about Memory Banking Controllers. ::: # Interrupts

IME - Interrupt Master Enable Flag (Write Only)

```
0 - Disable all Interrupts
1 - Enable all Interrupts that are enabled in IE Register (FFFF)
```

The IME flag is used to disable all interrupts, overriding any enabled bits in the IE Register. It isn't possible to access the IME flag by using a I/O address, instead IME is accessed directly from the CPU, by the following opcodes/operations:

```
EI     ;Enable Interrupts  (ie. IME=1)
DI     ;Disable Interrupts (ie. IME=0)
RETI   ;Enable Ints & Return (same as the opcode combination EI, RET)
<INT>  ;Disable Ints & Call to Interrupt Vector
```

where <INT> means the operation which is automatically executed by the CPU when it executes an interrupt.

The effect of EI is delayed by one instruction. This means that EI followed immediately by DI does not allow interrupts between the EI and the DI.

FFFF - IE - Interrupt Enable (R/W)

```
Bit 0: V-Blank  Interrupt Enable  (INT 40h)  (1=Enable)
Bit 1: LCD STAT Interrupt Enable  (INT 48h)  (1=Enable)
Bit 2: Timer    Interrupt Enable  (INT 50h)  (1=Enable)
Bit 3: Serial   Interrupt Enable  (INT 58h)  (1=Enable)
Bit 4: Joypad   Interrupt Enable  (INT 60h)  (1=Enable)
```

FF0F - IF - Interrupt Flag (R/W)

```
Bit 0: V-Blank  Interrupt Request (INT 40h)  (1=Request)
Bit 1: LCD STAT Interrupt Request (INT 48h)  (1=Request)
Bit 2: Timer    Interrupt Request (INT 50h)  (1=Request)
Bit 3: Serial   Interrupt Request (INT 58h)  (1=Request)
Bit 4: Joypad   Interrupt Request (INT 60h)  (1=Request)
```

When an interrupt signal changes from low to high, then the corresponding bit in the IF register becomes set. For example, Bit 0 becomes set when the LCD controller enters into the V-Blank period.

Interrupt Requests

Any set bits in the IF register are only **requesting** an interrupt to be executed. The actual **execution** happens only if both the IME flag, and the corresponding bit in the IE register are set, otherwise the interrupt 'waits' until both IME and IE allow its execution.

Interrupt Execution

When an interrupt gets executed, the corresponding bit in the IF register becomes automatically reset by the CPU, and the IME flag becomes cleared (disabling any further interrupts until the program re-enables the interrupts, typically by using the RETI instruction), and the corresponding Interrupt Vector (that are the addresses in range 0040h-0060h, as shown in IE and IF register decriptions above) becomes called.

Manually Requesting/Discarding Interrupts

As the CPU automatically sets and clears the bits in the IF register, it is usually not required to write to the IF register. However, the user may still do that in order to manually request (or discard) interrupts. As for real interrupts, a manually requested interrupt isn't executed unless/until IME and IE allow its execution.

Interrupt Priorities

In the following three situations it might happen that more than 1 bit in the IF register are set, requesting more than one interrupt at once:

1. More than one interrupt signal changed from Low to High at the same time.
2. Several interrupts have been requested during a time in which IME/IE didn't allow these interrupts to be executed directly.
3. The user has written a value with several "1" bits (for example 1Fh) to the IF register.

Provided that IME and IE allow the execution of more than one of the requested interrupts, then the interrupt with the highest priority becomes executed first. The priorities are ordered as the bits in the IE and IF registers, Bit 0 (V-Blank) having the highest priority, and Bit 4 (Joypad) having the lowest priority.

Nested Interrupts

The CPU automatically disables all other interrupts by setting IME=0 when it executes an interrupt. Usually IME remains zero until the interrupt procedure returns (and sets IME=1 by the RETI instruction). However, if you want any other interrupts of lower or higher (or same) priority to be allowed to be executed from inside of the interrupt procedure, then you can place an EI instruction into the interrupt procedure.

Interrupt Service Routine

According to Z80 datasheets, the following occurs when control is being transferred to an interrupt handler:

1. Two wait states are executed (2 machine cycles pass while nothing occurs, presumably the CPU is executing NOPs during this time).
2. The current PC is pushed onto the stack, this process consumes 2 more machine cycles.
3. The high byte of the PC is set to 0, the low byte is set to the address of the handler ($40,$48,$50,$58,$60). This consumes one last machine cycle.

The entire ISR **should** consume a total of 5 machine cycles. This has yet to be tested, but is what the Z80 datasheet implies.

Chapter 8

CGB Registers

This chapter describes only CGB (Color Gameboy) registers that didn't fit into normal categories - most CGB registers are described in the chapter about Video Display (Color Palettes, VRAM Bank, VRAM DMA Transfers, and changed meaning of Bit 0 of LCDC Control register). Also, a changed bit is noted in the chapter about the Serial/Link port.

8.1 Unlocking CGB functions

When using any CGB registers (including those in the Video/Link chapters), you must first unlock CGB features by changing byte 0143h in the cartridge header. Typically use a value of 80h for games which support both CGB and monochrome gameboys, and C0h for games which work on CGBs only. Otherwise, the CGB will operate in monochrome 'Non CGB' compatibility mode.

8.2 Detecting CGB (and GBA) functions

CGB hardware can be detected by examing the CPU accumulator (A-register) directly after startup. A value of 11h indicates CGB (or GBA) hardware, if so, CGB functions can be used (if unlocked, see above). When A=11h, you may also examine Bit 0 of the CPUs B-Register to separate between CGB (bit cleared) and GBA (bit set), by that detection it is possible to use 'repaired' color palette data matching for GBA displays.

8.3 Documented registers

FF4D - KEY1 - CGB Mode Only - Prepare Speed Switch

```
Bit 7: Current Speed     (0=Normal, 1=Double) (Read Only)
Bit 0: Prepare Speed Switch (0=No, 1=Prepare) (Read/Write)
```

This register is used to prepare the Game Boy to switch between CGB Double Speed Mode and Normal Speed Mode. The actual speed switch is performed by executing a STOP command after Bit 0 has been set. After that Bit 0 will be cleared automatically, and the Game Boy will operate at the 'other' speed. The recommended speed switching procedure in pseudo code would be:

```
IF KEY1_BIT7 <> DESIRED_SPEED THEN
  IE=00H       ;(FFFF)=00h
  JOYP=30H     ;(FF00)=30h
  KEY1=01H     ;(FF4D)=01h
  STOP         ;STOP
ENDIF
```

The CGB is operating in Normal Speed Mode when it is turned on. Note that using the Double Speed Mode increases the power consumption, it would be recommended to use Single Speed whenever possible. However, the display will flicker (white) for a moment during speed switches, so this cannot be done permanentely. In Double Speed Mode the following will operate twice as fast as normal:

```
The CPU (2.10 MHz, 1 Cycle = approx. 0.5us)
Timer and Divider Registers
Serial Port (Link Cable)
DMA Transfer to OAM
```

And the following will keep operating as usual:

- LCD Video Controller
- HDMA Transfer to VRAM
- All Sound Timings and Frequencies

FF56 - RP - CGB Mode Only - Infrared Communications Port

This register allows to input and output data through the CGBs built-in Infrared Port. When reading data, bit 6 and 7 must be set (and obviously Bit 0 must be cleared - if you don't want to receive your own gameboys IR signal). After sending or receiving data you should reset the register to 00h to reduce battery power consumption again.

```
Bit 0:   Write Data   (0=LED Off, 1=LED On)             (Read/Write)
Bit 1:   Read Data    (0=Receiving IR Signal, 1=Normal) (Read Only)
Bit 6-7: Data Read Enable (0=Disable, 3=Enable)         (Read/Write)
```

Note that the receiver will adapt itself to the normal level of IR pollution in the air, so if you would send a LED ON signal for a longer period, then the receiver would treat that as normal (=OFF) after a while. For example, a Philips TV Remote Control sends a series of 32 LED ON/OFF pulses (length 10us ON, 17.5us OFF each) instead of a permanent 880us LED ON signal. Even though being generally CGB compatible, the GBA does not include an infra-red port.

FF70 - SVBK - CGB Mode Only - WRAM Bank

In CGB Mode 32 KBytes internal RAM are available. This memory is divided into 8 banks of 4 KBytes each. Bank 0 is always available in memory at C000-CFFF, Bank 1-7 can be selected into the address space at D000-DFFF.

```
 Bit 0-2  Select WRAM Bank (Read/Write)
```

Writing a value of 01h-07h will select Bank 1-7, writing a value of 00h will select Bank 1 either.

8.4 Undocumented registers

These are undocumented CGB Registers. Purpose of these registers is unknown (if any). It isn't recommended to use them in your software, but you could, for example, use them to check if you are running on an emulator or on DMG hardware.

FF6C - Bit 0 (Read/Write) - CGB Mode Only

Only the least significant bit of this register can be written to. It defaults to 0, so this register's initial value is $FE.

In non-CGB mode, it isn't writable, and its value is locked at $FF.

FF72 - Bits 0-7 (Read/Write)

FF73 - Bits 0-7 (Read/Write)

Both of these registers are fully read/write. Their initial value is $00.

FF74 - Bits 0-7 (Read/Write) - CGB Mode Only

In CGB mode, this register is fully readable and writable. Its initial value is $00.

Otherwise, this register is read-only, and locked at value $FF.

FF75 - Bits 4-6 (Read/Write)

Only bits 4, 5 and 6 of this register are read/write enabled. Their initial value is 0.

FF76 - PCM12 - PCM amplitudes 1 & 2 (Read Only)

This register is read-only. The low nibble is a copy of sound channel #1's PCM amplitude, the high nibble a copy of sound channel #2's.

FF77 - PCM34 - PCM amplitudes 3 & 4 (Read Only)

Same, but with channels 3 and 4.

Chapter 9

SGB Functions

9.1 SGB Description

General Description

Basically, the SGB (Super Gameboy) is an adapter cartridge that allows to play Game Boy games on a SNES (Super Nintendo Entertainment System) gaming console. In detail, you plug the Game Boy cartridge into the SGB cartridge, then plug the SGB cartridge into the SNES, and then connect the SNES to your TV Set. In result, games can be played and viewed on the TV Set, and are controlled by using the SNES joypad(s).

More Technical Description

The SGB cartridge just contains a normal Game Boy CPU and normal gameboy video controller. Normally the video signal from this controller would be sent to the LCD screen, however, in this special case the SNES read out the video signal and displays it on the TV set by using a special SNES BIOS ROM which is located in the SGB cartridge. Also, normal gameboy sound output is forwared to the SNES and output to the TV Set, vice versa, joypad input is forwared from the SNES controller(s) to the gameboy joypad inputs.

Normal Monochrome Games

Any Game Boy games which have been designed for normal monochrome handheld gameboys will work with the SGB hardware as well. The SGB will apply a four color palette to these games by replacing the normal four grayshades. The 160x144 pixel gamescreen is displayed in the middle of the 256x224 pixel SNES screen (the unused area is filled by a screen border bitmap). The user may access built-in menues, allowing to change color palette data, to select between several pre-defined borders, etc.

Games that have been designed to support SGB functions may also access the following additional features:

Colorized Game Screen

There's limited ability to colorize the gamescreen by assigning custom color palettes to each 20x18 display characters, however, this works mainly for static display data such like title screens or status bars, the 20x18 color attribute map is non-scrollable, and it is not possible to assign separate colors to moveable foreground sprites (OBJs), so that animated screen regions will be typically restricted to using a single palette of four colors only.

SNES Foreground Sprites

Up to 24 foreground sprites (OBJs) of 8x8 or 16x16 pixels, 16 colors can be displayed. When replacing (or just overlaying) the normal gameboy OBJs by SNES OBJs it'd be thus possible to display OBJs with other colors than normal background area. This method doesn't appear to be very popular, even though it appears to be quite easy to implement, however, the bottommost character line of the gamescreen will be masked out because this area is used to transfer OAM data to the SNES.

The SGB Border

The possibly most popular and most impressive feature is to replace the default SGB screen border by a custom bitmap which is stored in the game cartridge.

Multiple Joypads

Up to four joypads can be conected to the SNES, and SGB software may read-out each of these joypads separately, allowing up to four players to play the same game simultaneously. Unlike for multiplayer handheld games, this requires only one game cartridge and only one SGB/SNES, and no link cables are required, the downside is that all players must share the same display screen.

Sound Functions

Beside for normal Game Boy sound, a number of digital sound effects is pre-defined in the SNES BIOS, these effects may be accessed quite easily. Programmers whom are familiar with SNES sounds may also access the SNES sound chip, or use the SNES MIDI engine directly in order to produce other sound effects or music.

Taking Control of the SNES CPU

Finally, it is possible to write program code or data into SNES memory, and to execute such program code by using the SNES CPU.

SGB System Clock

Because the SGB is synchronized to the SNES CPU, the Game Boy system clock is directly chained to the SNES system clock. In result, the gameboy CPU, video controller, timers, and sound frequencies will be all operated approx 2.4% faster as by normal gameboys. Basically, this should be no problem, and the game will just run a little bit faster. However sensitive musicians may notice that sound frequencies are a bit too high, programs that support SGB functions may avoid this effect by reducing frequencies of Game Boy sounds when having detected SGB hardware. Also, I think that I've heard that SNES models which use a 50Hz display refresh rate (rather than 60Hz) are resulting in respectively slower SGB/gameboy timings ???

9.2 SGB Unlocking and Detecting SGB Functions

Cartridge Header

SGB games are required to have a cartridge header with Nintendo and proper checksum just as normal Game Boy games. Also, two special entries must be set in order to unlock SGB functions:

```
146h - SGB Flag - Must be set to 03h for SGB games 14Bh - Old Licensee
Code - Must be set 33h for SGB games
```

When these entries aren't set, the game will still work just like all 'monochrome' Game Boy games, but it cannot access any of the special SGB functions.

Detecting SGB hardware

The recommended detection method is to send a MLT_REQ command which enables two (or four) joypads. A normal handheld Game Boy will ignore this command, a SGB will now return incrementing joypad IDs each time when deselecting keyboard lines (see MLT_REQ description for details). Now readout joypad state/IDs several times, and if the ID-numbers are changing, then it is a SGB (a normal Game Boy would typically always return 0Fh as ID). Finally, when not intending to use more than one joypad, send another MLT_REQ command in order to re-disable the multi-controller mode. Detection works regardless of whether and how many joypads are physically connected to the SNES. However, detection works only when having unlocked SGB functions in the cartridge header, as described above.

Separating between SGB and SGB2

It is also possible to separate between SGB and SGB2 models by examining the inital value of the accumulator (A-register) directly after startup.

```
01h  SGB or Normal Gameboy (DMG) FFh  SGB2 or Pocket Gameboy 11h
CGB or GBA
```

Because values 01h and FFh are shared for both handhelds and SGBs, it is still required to use the above MLT_REQ detection procedure. As far as I know the SGB2 doesn't have any extra features which'd require separate SGB2 detection except for curiosity purposes, for example, the game "Tetris DX" chooses to display an alternate SGB border on SGB2s.

Reportedly, some SGB models include link ports (just like handheld gameboy) (my own SGB does not have such an port), possibly this feature is available in SGB2-type models only ???

9.3 SGB Command Packet Transfers

Command packets (aka Register Files) are transferred from the Game Boy to the SNES by using P14 and P15 output lines of the JOYPAD register (FF00h), these lines are normally used to select the two rows in the gameboy keyboard matrix (which still works).

Transferring Bits

A command packet transfer must be initiated by setting both P14 and P15 to LOW, this will reset and start the SNES packet receiving program. Data is then transferred (LSB first), setting P14=LOW will indicate a "0" bit, and setting P15=LOW will indicate a "1" bit. For example:

```
   RESET 0    0    1    1    0    1    0 P14  --_---_---_-----------_-------_--...
P15  --_-----------_---_-------_------...
```

Data and reset pulses must be kept LOW for at least 5us. P14 and P15 must be kept both HIGH for at least 15us between any pulses. Obviously, it'd be no good idea to access the JOYPAD register during the transfer, for example, in case that your VBlank interrupt procedure reads-out joypad states each frame, be sure to disable that interrupt during the transfer (or disable only the joypad procedure by using a software flag).

Transferring Packets

Each packet is invoked by a RESET pulse, then 128 bits of data are transferred (16 bytes, LSB of first byte first), and finally, a "0"-bit must be transferred as stop bit. The structure of normal packets is:

```
   1 PULSE Reset 1 BYTE  Command Code*8+Length 15 BYTES Parameter Data
1 BIT   Stop Bit (0)
```

The above 'Length' indicates the total number of packets (1-7, including the first packet) which will be sent, ie. if more than 15 parameter bytes are used, then further packet(s) will follow, as such:

```
   1 PULSE Reset 16 BYTES Parameter Data 1 BIT   Stop Bit (0)
```

By using all 7 packets, up to 111 data bytes (15+16*6) may be sent. Unused bytes at the end of the last packet don't matter. A 60ms (4 frames) delay should be invoked between each packet transfer.

9.4 SGB VRAM Transfers

Overview

Beside for the packet transfer method, larger data blocks of 4KBytes can be transferred by using the video signal. These transfers are invoked by first sending one of the commands with the ending _TRN (by using normal packet transfer), the 4K data block is then read-out by the SNES from gameboy display memory during the next frame.

Transfer Data

Normally, transfer data should be stored at 8000h-8FFFh in Game Boy VRAM, even though the SNES receives the data in from display scanlines, it will automatically re-produce the same ordering of bits and bytes, as being originally stored at 8000h-8FFFh in Game Boy memory.

Preparing the Display

The above method works only when recursing the following things: BG Map must display unsigned characters 00h-FFh on the screen; 00h..13h in first line, 14h..27h in next line, etc. The Game Boy display must be enabled, the display may not be scrolled, OBJ sprites should not overlap the background tiles, the BGP palette register must be set to E4h.

Transfer Time

Note that the transfer data should be prepared in VRAM **before** sending the transfer command packet. The actual transfer starts at the beginning of the next frame after the command has been sent, and the transfer ends at the end of the 5th frame after the command has been sent (not counting the frame in which the command has been sent). The displayed data must not be modified during the transfer, as the SGB reads it in multiple chunks.

Avoiding Screen Garbage

The display will contain 'garbage' during the transfer, this dirt-effect can be avoided by freezing the screen (in the state which has been displayed before the transfer) by using the MASK_EN command. Of course, this works only when actually executing the game on a SGB (and not on normal handheld gameboys), it'd be thus required to detect the presence of SGB hardware before blindly sending VRAM data.

9.5 SGB Command Summary

SGB System Command Table

Code	Name	Expl.
00	PAL01	Set SGB Palette 0,1 Data
01	PAL23	Set SGB Palette 2,3 Data
02	PAL03	Set SGB Palette 0,3 Data
03	PAL12	Set SGB Palette 1,2 Data
04	ATTR_BLK	"Block" Area Designation Mode
05	ATTR_LIN	"Line" Area Designation Mode
06	ATTR_DIV	"Divide" Area Designation Mode
07	ATTR_CHR	"1CHR" Area Designation Mode
08	SOUND	Sound On/Off
09	SOU_TRN	Transfer Sound PRG/DATA
0A	PAL_SET	Set SGB Palette Indirect
0B	PAL_TRN	Set System Color Palette Data
0C	ATRC_EN	Enable/disable Attraction Mode
0D	TEST_EN	Speed Function
0E	ICON_EN	SGB Function
0F	DATA_SND	SUPER NES WRAM Transfer 1
10	DATA_TRN	SUPER NES WRAM Transfer 2
11	MLT_REG	Controller 2 Request
12	JUMP	Set SNES Program Counter
13	CHR_TRN	Transfer Character Font Data
14	PCT_TRN	Set Screen Data Color Data
15	ATTR_TRN	Set Attribute from ATF
16	ATTR_SET	Set Data to ATF
17	MASK_EN	Game Boy Window Mask
18	OBJ_TRN	Super NES OBJ Mode

9.6 SGB Color Palettes Overview

Available SNES Palettes

The SGB/SNES provides 8 palettes of 16 colors each, each color may be defined out of a selection of 34768 colors (15 bit). Palettes 0-3 are used to colorize the gamescreen, only the first four colors of each of these palettes are used. Palettes 4-7 are used for the SGB Border, all 16 colors of each of these palettes may be used.

Color format

Colors are encoded as 16-bit RGB numbers, in the following way:

```
FEDC BA98 7654 3210 0BBB BBGG GGGR RRRR
```

Here's a formula to convert 24-bit RGB into SNES format: `(color & 0xF8) << 7 | (color & 0xF800) >> 6 | (color & 0xF80000) >> 19`

The palettes are encoded **little-endian**, thus, the R/G byte comes first in memory.

Color 0 Restriction

Color 0 of each of the eight palettes is transparent, causing the backdrop color to be displayed instead. The backdrop color is typically defined by the most recently color being assigned to Color 0 (regardless of the palette number being used for that operation). Effectively, gamescreen palettes can have only three custom colors each, and SGB border palettes only 15 colors each, additionally, color 0 can be used for for all palettes, which will then all share the same color though.

Translation of Grayshades into Colors

Because the SGB/SNES reads out the Game Boy video controllers display signal, it translates the different grayshades from the signal into SNES colors as such:

```
   White       -->  Color 0 Light Gray  -->  Color 1 Dark Gray   -->
Color 2 Black       -->  Color 3
```

Note that Game Boy colors 0-3 are assigned to user-selectable grayshades by the gameboys BGP, OBP1, and OBP2 registers. There is thus no fixed relationship between Game Boy colors 0-3 and SNES colors 0-3.

Using Game Boy BGP/OBP Registers A direct translation of GB color 0-3 into SNES color 0-3 may be produced by setting BGP/OBP registers to a value of 0E4h each. However, in case that your program uses black background for example, then you may internally assign background as "White" at the Game Boy side by BGP/OBP registers (which is then interpreted as SNES color 0, which is shared for all SNES palettes). The advantage is that you may define Color 0 as Black at the SNES side, and may assign custom colors for Colors 1-3 of each SNES palette.

System Color Palette Memory

Beside for the actually visible palettes, up to 512 palettes of 4 colors each may be defined in SNES RAM. The palettes are just stored in RAM without any relationship to the displayed picture; however, these pre-defined colors may be transferred to actually visible palettes slightly faster than when transferring palette data by separate command packets.

9.7 SGB Palette Commands

SGB Command 00h - PAL01

Transmit color data for SGB palette 0, color 0-3, and for SGB palette 1, color 1-3 (without separate color 0).

```
   Byte  Content 0     Command*8+Length (fixed length=01h) 1-E   Color
Data for 7 colors of 2 bytes (16bit) each: Bit 0-4   - Red Intensity
(0-31) Bit 5-9   - Green Intensity (0-31) Bit 10-14 - Blue Intensity
(0-31) Bit 15    - Not used (zero) F     Not used (00h)
```

This is the same RGB5 format as Game Boy Color palette entry, though without the LCD correction. The value transferred as color 0 will be applied for all four palettes.

SGB Command 01h - PAL23

Same as above PAL01, but for Palettes 2 and 3 respectively.

SGB Command 02h - PAL03

Same as above PAL01, but for Palettes 0 and 3 respectively.

SGB Command 03h - PAL12

Same as above PAL01, but for Palettes 1 and 2 respectively.

SGB Command 0Ah - PAL_SET

Used to copy pre-defined palette data from SGB system color palettes to actual SNES palettes.

Note: all palette numbers are little-endian.

```
   Byte  Content 0     Command*8+Length (fixed length=1) 1-2   System
Palette number for SGB Color Palette 0 (0-511) 3-4   System Palette
number for SGB Color Palette 1 (0-511) 5-6   System Palette number
for SGB Color Palette 2 (0-511) 7-8   System Palette number for SGB
Color Palette 3 (0-511) 9     Attribute File Bit 0-5 - Attribute File
Number (00h-2Ch) (Used only if Bit7=1) Bit 6   - Cancel Mask           (0=No
change, 1=Yes) Bit 7   - Use Attribute File    (0=No, 1=Apply above
ATF Number) A-F   Not used (zero)
```

Before using this function, System Palette data should be initialized by PAL_TRN command, and (when used) Attribute File data should be initialized by ATTR_TRN.

SGB Command 0Bh - PAL_TRN

Used to initialize SGB system color palettes in SNES RAM. System color palette memory contains 512 pre-defined palettes, these palettes do not directly affect the display, however, the PAL_SET command may be later used to transfer four of these 'logical' palettes to actual visible 'physical' SGB palettes. Also, the OBJ_TRN function will use groups of 4 System Color Palettes (4*4 colors) for SNES OBJ palettes (16 colors).

```
   Byte  Content 0     Command*8+Length (fixed length=1) 1-F   Not
used (zero)
```

The palette data is sent by VRAM-Transfer (4 KBytes).

```
   000-FFF  Data for System Color Palette 0-511
```

Each Palette consists of four 16bit-color definitions (8 bytes). Note: The data is stored at 3000h-3FFFh in SNES memory.

9.8 SGB Color Attribute Commands

SGB Command 04h - ATTR_BLK

Used to specify color attributes for the inside or outside of one or more rectangular screen regions.

Byte Content 0 Command*8+Length (length=1..7) 1 Number of Data Sets (01h..12h) 2-7 Data Set #1 Byte 0 - Control Code (0-7) Bit 0 - Change Colors inside of surrounded area (1=Yes) Bit 1 - Change Colors of surrounding character line (1=Yes) Bit 2 - Change Colors outside of surrounded area (1=Yes) Bit 3-7 - Not used (zero) Exception: When changing only the Inside or Outside, then the Surrounding line becomes automatically changed to same color. Byte 1 - Color Palette Designation Bit 0-1 - Palette Number for inside of surrounded area Bit 2-3 - Palette Number for surrounding character line Bit 4-5 - Palette Number for outside of surrounded area Bit 6-7 - Not used (zero) Data Set Byte 2 - Coordinate X1 (left) Data Set Byte 3 - Coordinate Y1 (upper) Data Set Byte 4 - Coordinate X2 (right) Data Set Byte 5 - Coordinate Y2 (lower) Specifies the coordinates of the surrounding rectangle. 8-D Data Set #2 (if any) E-F Data Set #3 (continued at 0-3 in next packet) (if any)

When sending three or more data sets, data is continued in further packet(s). Unused bytes at the end of the last packet should be set to zero. The format of the separate Data Sets is described below.

SGB Command 05h - ATTR_LIN

Used to specify color attributes of one or more horizontal or vertical character lines.

Byte Content 0 Command*8+Length (length=1..7) 1 Number of Data Sets (01h..6Eh) (one byte each) 2 Data Set #1 Bit 0-4 - Line Number (X- or Y-coordinate, depending on bit 7) Bit 5-6 - Palette Number (0-3) Bit 7 - H/V Mode Bit (0=Vertical line, 1=Horizontal Line) 3 Data Set #2 (if any) 4 Data Set #3 (if any) etc.

When sending 15 or more data sets, data is continued in further packet(s). Unused bytes at the end of the last packet should be set to zero. The format of the separate Data Sets (one byte each) is described below. The length of each line reaches from one end of the screen to the other end. In case that some lines overlap each other, then lines from lastmost data sets will overwrite lines from previous data sets.

SGB Command 06h - ATTR_DIV

Used to split the screen into two halfes, and to assign separate color attributes to each half, and to the division line between them.

Byte Content 0 Command*8+Length (fixed length=1) 1 Color Palette Numbers and H/V Mode Bit Bit 0-1 Palette Number below/right of division line Bit 2-3 Palette Number above/left of division line Bit 4-5 Palette Number for division line Bit 6 H/V Mode Bit (0=split left/right, 1=split above/below) 2 X- or Y-Coordinate (depending on H/V bit) 3-F Not used (zero)

SGB Command 07h - ATTR_CHR

Used to specify color attributes for separate characters.

```
   Byte  Content 0     Command*8+Length (length=1..6) 1     Beginning
X-Coordinate 2     Beginning Y-Coordinate 3-4   Number of Data Sets
(1-360) 5     Writing Style   (0=Left to Right, 1=Top to Bottom) 6
Data Sets 1-4   (Set 1 in MSBs, Set 4 in LSBs) 7     Data Sets 5-8
(if any) 8     Data Sets 9-12  (if any) etc.
```

When sending 41 or more data sets, data is continued in further packet(s). Unused bytes at the end of the last packet should be set to zero. Each data set consists of two bits, indicating the palette number for one character. Depending on the writing style, data sets are written from left to right, or from top to bottom. In either case the function wraps to the next row/column when reaching the end of the screen.

SGB Command 15h - ATTR_TRN

Used to initialize Attribute Files (ATFs) in SNES RAM. Each ATF consists of 20x18 color attributes for the Game Boy screen. This function does not directly affect display attributes. Instead, one of the defined ATFs may be copied to actual display memory at a later time by using ATTR_SET or PAL_SET functions.

```
   Byte  Content 0     Command*8+Length (fixed length=1) 1-F   Not
used (zero)
```

The ATF data is sent by VRAM-Transfer (4 KBytes).

```
   000-FD1  Data for ATF0 through ATF44 (4050 bytes) FD2-FFF  Not used
```

Each ATF consists of 90 bytes, that are 5 bytes (20x2bits) for each of the 18 character lines of the Game Boy window. The two most significant bits of the first byte define the color attribute (0-3) for the first character of the first line, the next two bits the next character, and so on.

SGB Command 16h - ATTR_SET

Used to transfer attributes from Attribute File (ATF) to Game Boy window.

```
   Byte  Content 0     Command*8+Length (fixed length=1) 1     Attribute
File Number (00-2Ch), Bit 6=Cancel Mask 2-F   Not used (zero)
```

When above Bit 6 is set, the Game Boy screen becomes re-enabled after the transfer (in case it has been disabled/frozen by MASK_EN command). Note: The same functions may be (optionally) also included in PAL_SET commands, as described in the chapter about Color Palette Commands.

9.9 SGB Sound Functions

SGB Command 08h - SOUND

Used to start/stop internal sound effect, start/stop sound using internal tone data.

```
   Byte  Content 0     Command*8+Length (fixed length=1) 1     Sound
Effect A (Port 1) Decrescendo 8bit Sound Code 2     Sound Effect B
(Port 2) Sustain     8bit Sound Code 3     Sound Effect Attributes
Bit 0-1 - Sound Effect A Pitch  (0..3=Low..High) Bit 2-3 - Sound Effect
A Volume (0..2=High..Low, 3=Mute on) Bit 4-5 - Sound Effect B Pitch
(0..3=Low..High) Bit 6-7 - Sound Effect B Volume (0..2=High..Low, 3=Not
used) 4     Music Score Code (must be zero if not used) 5-F   Not used
(zero)
```

See Sound Effect Tables below for a list of available pre-defined effects.

Notes:

1. Mute is only active when both bits D2 and D3 are 1.
2. When the volume is set for either Sound Effect A or Sound Effect B, mute is turned off.
3. When Mute on/off has been executed, the sound fades out/fades in.
4. Mute on/off operates on the (BGM) which is reproduced by Sound Effect A, Sound Effect B, and the Super NES APU. A "mute off" flag does not exist by itself. When mute flag is set, volume and pitch of Sound Effect A (port 1) and Sound Effect B (port 2) must be set.

SGB Command 09h - SOU_TRN

Used to transfer sound code or data to SNES Audio Processing Unit memory (APU-RAM).

```
   Byte  Content 0     Command*8+Length (fixed length=1) 1-F   Not
used (zero)
```

The sound code/data is sent by VRAM-Transfer (4 KBytes).

```
   000      One (or two ???) 16bit expression(s ???) indicating the
transfer destination address and transfer length. ...-...  Transfer
Data ...-FFF  Remaining bytes not used
```

Possible destinations in APU-RAM are:

```
   0400h-2AFFh  APU-RAM Program Area (9.75KBytes) 2B00h-4AFFh  APU-RAM
Sound Score Area (8Kbytes) 4DB0h-EEFFh  APU-RAM Sampling Data Area
(40.25 Kbytes)
```

This function may be used to take control of the SNES sound chip, and/or to access the SNES MIDI engine. In either case it requires deeper knowledge of SNES sound programming.

SGB Sound Effect A/B Tables

Below lists the digital sound effects that are pre-defined in the SGB/SNES BIOS, and which can be used with the SGB "SOUND" Command. Effect A and B may be simultaneously reproduced. The P-column indicates the recommended Pitch value, the V-column indicates the numbers of Voices used. Sound Effect A uses voices 6,7. Sound Effect B uses voices 0,1,4,5. Effects that use less voices will use only the upper voices (eg. 4,5 for Effect B with only two voices).

Sound Effect A Flag Table

	Code	Description	P	V	Code	Description	P
V	00	Dummy flag, re-trigger	-	2	18	Fast Jump	3
1	80	Effect A, stop/silent	-	2	19	Jet (rocket) takeoff	0
1	01	Nintendo	3	1	1A	Jet (rocket) landing	0
1	02	Game Over	3	2	1B	Cup breaking	2
2	03	Drop	3	1	1C	Glass breaking	1
2	04	OK ... A	3	2	1D	Level UP	2
2	05	OK ... B	3	2	1E	Insert air	1
1	06	Select...A	3	2	1F	Sword swing	1
1	07	Select...B	3	1	20	Water falling	2
1	08	Select...C	2	2	21	Fire	1
1	09	Mistake...Buzzer	2	1	22	Wall collapsing	1
2	0A	Catch Item	2	2	23	Cancel	1
2	0B	Gate squeaks 1 time	2	2	24	Walking	1
2	0C	Explosion...small	1	2	25	Blocking strike	1
2	0D	Explosion...medium	1	2	26	Picture floats on & off	
3 2	0E	Explosion...large	1	2	27	Fade in	0
2	0F	Attacked...A	3	1	28	Fade out	0
2	10	Attacked...B	3	2	29	Window being opened	1
2	11	Hit (punch)...A	0	2	2A	Window being closed	0
2	12	Hit (punch)...B	0	2	2B	Big Laser	3
2	13	Breath in air	3	2	2C	Stone gate closes/opens	
0 2	14	Rocket Projectile...A	3	2	2D	Teleportation	3
1	15	Rocket Projectile...B	3	2	2E	Lightning	0
2	16	Escaping Bubble	2	1	2F	Earthquake	0
2	17	Jump	3	1	30	Small Laser	2
2							

Sound effect A is used for formanto sounds (percussion sounds).

Sound Effect B Flag Table

	Code	Description	P	V	Code	Description	P
V	00	Dummy flag, re-trigger	-	4	0D	Waterfall	2
2	80	Effect B, stop/silent	-	4	0E	Small character running	
3 1	01	Applause...small group	2	1	0F	Horse running	3
1	02	Applause...medium group	2	2	10	Warning sound	1
1	03	Applause...large group	2	4	11	Approaching car	0
1	04	Wind	1	2	12	Jet flying	1
1	05	Rain	1	1	13	UFO flying	2
1	06	Storm	1	3	14	Electromagnetic waves	0
1	07	Storm with wind/thunder	2	4	15	Score UP	3
1	08	Lightning	0	2	16	Fire	2
1	09	Earthquake	0	2	17	Camera shutter, formanto	
3 4	0A	Avalanche	0	2	18	Write, formanto	0

`1 0B  Wave                    0 1     19  Show up title, formanto`
`0 1 0C  River                   3 2`

Sound effect B is mainly used for looping sounds (sustained sounds).

9.10 SGB System Control Commands

SGB Command 17h - MASK_EN

Used to mask the Game Boy window, among others this can be used to freeze the Game Boy screen before transferring data through VRAM (the SNES then keeps displaying the Game Boy screen, even though VRAM doesn't contain meaningful display information during the transfer).

`Byte  Content 0     Command*8+Length (fixed length=1) 1     Gameboy Screen Mask (0-3) 0  Cancel Mask   (Display activated) 1  Freeze Screen (Keep displaying current picture) 2  Blank Screen  (Black) 3  Blank Screen  (Color 0) 2-F   Not used (zero)`

Freezing works only if the SNES has stored a picture, ie. if necessary wait one or two frames before freezing (rather than freezing directly after having displayed the picture). The Cancel Mask function may be also invoked (optionally) by completion of PAL_SET and ATTR_SET commands.

SGB Command 0Ch - ATRC_EN

Used to enable/disable Attraction mode, which is enabled by default.

Built-in borders other than the Game Boy frame and the plain black border have a "screen saver" activated by pressing R, L, L, L, L, R or by leaving the controller alone for roughly 7 minutes (tested with 144p Test Suite). It is speculated that the animation may have interfered with rarely-used SGB features, such as OBJ_TRN or JUMP, and that Attraction Disable disables this animation.

`Byte  Content 0     Command*8+Length    (fixed length=1) 1     Attraction Disable  (0=Enable, 1=Disable) 2-F   Not used (zero)`

SGB Command 0Dh - TEST_EN

Used to enable/disable test mode for "SGB-CPU variable clock speed function". This function is disabled by default.

This command does nothing on some SGB revisions. (SGBv2 confirmed, unknown on others)

`Byte  Content 0     Command*8+Length    (fixed length=1) 1     Test Mode Enable    (0=Disable, 1=Enable) 2-F   Not used (zero)`

Maybe intended to determine whether SNES operates at 50Hz or 60Hz display refresh rate ??? Possibly result can be read-out from joypad register ???

SGB Command 0Eh - ICON_EN

Used to enable/disable ICON function. Possibly meant to enable/disable SGB/SNES popup menues which might otherwise activated during gameboy game play. By default all functions are enabled (0).

```
   Byte  Content 0     Command*8+Length    (fixed length=1) 1     Disable
Bits Bit 0 - Use of SGB-Built-in Color Palettes    (1=Disable) Bit
1 - Controller Set-up Screen    (0=Enable, 1=Disable) Bit 2 - SGB Register
File Transfer (0=Receive, 1=Disable) Bit 3-6 - Not used (zero) 2-F
Not used (zero)
```

Above Bit 2 will suppress all further packets/commands when set, this might be useful when starting a monochrome game from inside of the SGB-menu of a multi-gamepak which contains a collection of different games.

SGB Command 0Fh - DATA_SND

Used to write one or more bytes directly into SNES Work RAM.

```
   Byte  Content 0     Command*8+Length    (fixed length=1) 1     SNES
Destination Address, low 2     SNES Destination Address, high 3     SNES
Destination Address, bank number 4     Number of bytes to write (01h-0Bh)
5     Data Byte #1 6     Data Byte #2 (if any) 7     Data Byte #3 (if
any) etc.
```

Unused bytes at the end of the packet should be set to zero, this function is restricted to a single packet, so that not more than 11 bytes can be defined at once. Free Addresses in SNES memory are Bank 0 1800h-1FFFh, Bank 7Fh 0000h-FFFFh.

SGB Command 10h - DATA_TRN

Used to transfer binary code or data directly into SNES RAM.

```
   Byte  Content 0     Command*8+Length    (fixed length=1) 1     SNES
Destination Address, low 2     SNES Destination Address, high 3     SNES
Destination Address, bank number 4-F   Not used (zero)
```

The data is sent by VRAM-Transfer (4 KBytes).

```
   000-FFF  Data
```

Free Addresses in SNES memory are Bank 0 1800h-1FFFh, Bank 7Fh 0000h-FFFFh. The transfer length is fixed at 4KBytes ???, so that directly writing to the free 2KBytes at 0:1800h would be a not so good idea ???

SGB Command 12h - JUMP

Used to set the SNES program counter and NMI (vblank interrupt) handler to specific addresses.

```
   Byte  Content 0     Command*8+Length    (fixed length=1) 1     SNES
Program Counter, low 2     SNES Program Counter, high 3     SNES Program
Counter, bank number 4     SNES NMI Handler, low 5     SNES NMI Handler,
high 6     SNES NMI Handler, bank number 7-F   Not used, zero
```

The game *Space Invaders* uses this function when selecting “Arcade mode” to execute SNES program code which has been previously transferred from the SGB to the SNES. The SNES CPU is a Ricoh 5A22, which combines a 65C816 core licensed from WDC with a custom memory controller. For more information, see “fullsnes” by nocash.

Some notes for intrepid Super NES programmers seeking to use a flash cartridge in a Super Game Boy as a storage server:

- JUMP overwrites the NMI handler even if it is $000000.
- The SGB system software does not appear to use NMIs.
- JUMP can return to SGB system software via a 16-bit RTS. To do this, JML to a location in bank $00 containing byte value $60, such as any of the stubbed commands.
- IRQs and COP and BRK instructions are not useful because their handlers still point into SGB ROM. Use SEI WAI.
- If a program called through JUMP does not intend to return to SGB system software, it can overwrite all Super NES RAM except $0000BB through $0000BD, the NMI vector.
- To enter APU boot ROM, write $FE to $2140. Echo will still be on though.

9.11 SGB Multiplayer Command

SGB Command 11h - MLT_REQ

Used to request multiplayer mode (ie. input from more than one joypad). Because this function provides feedback from the SGB/SNES to the Game Boy program, it is also used to detect SGB hardware.

```
   Byte  Content 0     Command*8+Length    (fixed length=1) 1     Multiplayer
Control (0-3) (Bit0=Enable, Bit1=Two/Four Players) 0 = One player 1
= Two players 3 = Four players 2-F   Not used (zero)
```

In one player mode, the second joypad (if any) is used for the SGB system program. In two player mode, both joypads are used for the game. Because SNES have only two joypad sockets, four player mode requires an external “Multiplayer 5” adapter.

Changing the number of active players ANDs the currently selected player minus one with the number of players in that mode minus one. For example if you go from four players to two players and player 4 was active player 2 will then be active because 3 AND 1 is 1. However, sending the MLT_REQ command will increment the counter several times so results may not be exactly as expected. The most frequent case is going from one player to two-or-four player which will always start with player 1 active.

Reading Multiple Controllers (Joypads)

When having enabled multiple controllers by MLT_REQ, data for each joypad can be read out through JOYPAD register (FF00) as follows: First set P14 and P15 both HIGH (deselect both Buttons and Cursor keys), you can now read the lower 4bits of FF00 which indicate the joypad ID for the following joypad input:

`0Fh  Joypad 1 0Eh  Joypad 2 0Dh  Joypad 3 0Ch  Joypad 4`

Next, read joypad state as normally. When completed, set P14 and P15 back HIGH, this automatically increments the joypad number (or restarts counting once reached the lastmost joypad). Repeat the procedure until you have read-out states for all two (or four) joypads.

If for whatever reason you want to increment the joypad number without reading the joypad state you only need to set P15 to LOW before setting it back to HIGH. Adjusting P14 does not affect whether or not the joypad number will advance, However, if you set P15 to LOW then HIGH then LOW again without bringing both P14 and P15 HIGH at any point, it cancels the increment until P15 is lowered again. There are games, such as Pokémon Yellow, which rely on this cancelling when detecting the SGB.

9.12 SGB Border and OBJ Commands

SGB Command 13h - CHR_TRN

Used to transfer tile data (characters) to SNES Tile memory in VRAM. This normally used to define BG tiles for the SGB Border (see PCT_TRN), but might be also used to define moveable SNES foreground sprites (see OBJ_TRN).

`Byte  Content 0     Command*8+Length    (fixed length=1) 1     Tile Transfer Destination Bit 0   - Tile Numbers   (0=Tiles 00h-7Fh, 1=Tiles 80h-FFh) Bit 1   - Tile Type      (0=BG Tiles, 1=OBJ Tiles) Bit 2-7   - Not used (zero) 2-F   Not used (zero)`

The tile data is sent by VRAM-Transfer (4 KBytes).

`000-FFF  Bitmap data for 128 Tiles`

Each tile occupies 32 bytes (8x8 pixels, 16 colors each). When intending to transfer more than 128 tiles, call this function twice (once for tiles 00h-7Fh, and once for tiles 80h-FFh). Note: The BG/OBJ Bit seems to have no effect and writes to the same VRAM addresses for both BG and OBJ ???

TODO: explain tile format

SGB Command 14h - PCT_TRN

Used to transfer tile map data and palette data to SNES BG Map memory in VRAM to be used for the SGB border. The actual tiles must be separately transferred by using the CHR_TRN function.

`Byte  Content 0     Command*8+Length    (fixed length=1) 1-F   Not used (zero)`

The map data is sent by VRAM-Transfer (4 KBytes).

```
   000-6FF  BG Map 32x28 Entries of 16bit each (1792 bytes) 700-7FF
Not used, don't care 800-87F  BG Palette Data (Palettes 4-7, each 16
colors of 16bits each) 880-FFF  Not used, don't care
```

Each BG Map Entry consists of a 16bit value as such: `VH01 PP00 NNNN NNNN`

```
   Bit 0-9   - Character Number (use only 00h-FFh, upper 2 bits zero)
Bit 10-12 - Palette Number   (use only 4-7, officially use only 4-6)
Bit 13    - BG Priority      (use only 0) Bit 14    - X-Flip           (0=Normal,
1=Mirror horizontally) Bit 15    - Y-Flip           (0=Normal, 1=Mirror
vertically)
```

The 32x28 map entries correspond to 256x224 pixels of the Super NES screen. The 20x18 entries in the center of the 32x28 area should be set to a blank (solid color 0) tile as transparent space for the Game Boy window to be displayed inside. Non-transparent border data will cover the Game Boy window (for example, *Mario's Picross* does this, as does *WildSnake* to a lesser extent).

All borders repeat tiles. Assuming that the blank space for the GB screen is a single tile, as is the letterbox in a widescreen border, a border defining all unique tiles would have to define this many tiles:

- (256*224-160*144)/64+1 = 537 tiles in fullscreen border
- (256*176-160*144)/64+2 = 346 tiles in widescreen border

But the CHR RAM allocated by SGB for border holds only 256 tiles. This means a fullscreen border must repeat at least 281 tiles and a widescreen border at least 90.

SGB Command 18h - OBJ_TRN

Used to transfer OBJ attributes to SNES OAM memory. Unlike all other functions with the ending _TRN, this function does not use the usual one-shot 4KBytes VRAM transfer method. Instead, when enabled (below execute bit set), data is permanently (each frame) read out from the lower character line of the Game Boy screen. To suppress garbage on the display, the lower line is masked, and only the upper 20x17 characters of the Game Boy window are used - the masking method is unknwon - frozen, black, or recommended to be covered by the SGB border, or else ??? Also, when the function is enabled, "system attract mode is not performed" - whatever that means ???

This command does nothing on some SGB revisions. (SGBv2, SGB2?)

```
   Byte  Content 0      Command*8+Length (fixed length=1) 1      Control
Bits Bit 0   - SNES OBJ Mode enable (0=Cancel, 1=Enable) Bit 1   -
Change OBJ Color     (0=No, 1=Use definitions below) Bit 2-7 - Not
used (zero) 2-3   System Color Palette Number for OBJ Palette 4 (0-511)
4-5   System Color Palette Number for OBJ Palette 5 (0-511) 6-7   System
Color Palette Number for OBJ Palette 6 (0-511) 8-9   System Color Palette
Number for OBJ Palette 7 (0-511) These color entries are ignored if
```

above Control Bit 1 is zero. Because each OBJ palette consists of 16 colors, four system palette entries (of 4 colors each) are transferred into each OBJ palette. The system palette numbers are not required to be aligned to a multiple of four, and will wrap to palette number 0 when exceeding 511. For example, a value of 511 would copy system palettes 511, 0, 1, 2 to the SNES OBJ palette. A-F   Not used (zero)

The recommended method is to "display" Game Boy BG tiles F9h..FFh from left to right as first 7 characters of the bottom-most character line of the Game Boy screen. As for normal 4KByte VRAM transfers, this area should not be scrolled, should not be overlapped by Game Boy OBJs, and the Game Boy BGP palette register should be set up properly. By following that method, SNES OAM data can be defined in the 70h bytes of the gameboy BG tile memory at following addresses:

8F90-8FEF SNES OAM, 24 Entries of 4 bytes each (96 bytes) 8FF0-8FF5 SNES OAM MSBs, 24 Entries of 2 bits each (6 bytes) 8FF6-8FFF Not used, don't care (10 bytes)

The format of SNES OAM Entries is:

```
Byte 0  OBJ X-Position (0-511, MSB is separately stored, see below)
Byte 1  OBJ Y-Position (0-255)
Byte 2-3  Attributes (16bit)
  Bit 0-8    Tile Number     (use only 00h-FFh, upper bit zero)
  Bit 9-11   Palette Number  (use only 4-7)
  Bit 12-13  OBJ Priority    (use only 3)
  Bit 14     X-Flip          (0=Normal, 1=Mirror horizontally)
  Bit 15     Y-Flip          (0=Normal, 1=Mirror vertically)
```

The format of SNES OAM MSB Entries is:

Actually, the format is unknown ??? However, 2 bits are used per entry: One bit is the most significant bit of the OBJ X-Position. The other bit specifies the OBJ size (8x8 or 16x16 pixels).

9.13 Undocumented SGB commands

The following information has been extracted from disassembling a SGBv2 firmware; it should be verified on other SGB revisions.

The SGB firmware explicitly ignores all commands with ID >= $1E. This leaves undocumented commands $19 to $1D inclusive.

Stubbed commands

Commands $1A to $1F (inclusive)'s handlers are stubs (only contain a `RTS`). This is interesting, since the command-processing function explicitly ignores commands $1E and $1F.

SGB command 19h

The game Donkey Kong '94 appears to send this command, and it appears to set a flag in the SGB's memory. It's not known yet what it does, though. # Registers and Flags

Registers

16bit	Hi	Lo	Name/Function
AF	A	-	Accumulator & Flags
BC	B	C	BC
DE	D	E	DE
HL	H	L	HL
SP	-	-	Stack Pointer
PC	-	-	Program Counter/Pointer

As shown above, most registers can be accessed either as one 16bit register, or as two separate 8bit registers.

The Flag Register (lower 8bit of AF register)

Bit	Name	Set	Clr	Expl.
7	zf	Z	NZ	Zero Flag
6	n	-	-	Add/Sub-Flag (BCD)
5	h	-	-	Half Carry Flag (BCD)
4	cy	C	NC	Carry Flag
3-0	-	-	-	Not used (always zero)

Conatins the result from the recent instruction which has affected flags.

The Zero Flag (Z)

This bit becomes set (1) if the result of an operation has been zero (0). Used for conditional jumps.

The Carry Flag (C, or Cy)

Becomes set when the result of an addition became bigger than FFh (8bit) or FFFFh (16bit). Or when the result of a subtraction or comparision became less than zero (much as for Z80 and 80x86 CPUs, but unlike as for 65XX and ARM CPUs). Also the flag becomes set when a rotate/shift operation has shifted-out a "1"-bit. Used for conditional jumps, and for instructions such like ADC, SBC, RL, RLA, etc.

The BCD Flags (N, H)

These flags are (rarely) used for the DAA instruction only, N Indicates whether the previous instruction has been an addition or subtraction, and H indicates carry for lower 4bits of the result, also for DAA, the C flag must indicate carry for upper 8bits. After adding/subtracting two BCD numbers, DAA is intended to convert the result into BCD format; BCD numbers are ranged from 00h to 99h rather than 00h to FFh. Because C and H flags must contain carry-outs for each digit, DAA cannot be used for 16bit operations (which have 4 digits), or for INC/DEC operations (which do not affect C-flag).

Chapter 10

Instruction Set

Tables below specify the mnemonic, opcode bytes, clock cycles, affected flags (ordered as znhc), and explanatation. The timings assume a CPU clock frequency of 4.194304 MHz (or 8.4 MHz for CGB in double speed mode), as all Game Boy timings are divideable by 4, many people specify timings and clock frequency divided by 4.

GMB 8bit-Loadcommands

ld	r,r	xx	4	----	r=r
ld	r,n	xx nn	8	----	r=n
ld	r,(HL)	xx	8	----	r=(HL)
ld	(HL),r	7x	8	----	(HL)=r
ld	(HL),n	36 nn	12	----	
ld	A,(BC)	0A	8	----	
ld	A,(DE)	1A	8	----	
ld	A,(nn)	FA	16	----	
ld	(BC),A	02	8	----	
ld	(DE),A	12	8	----	
ld	(nn),A	EA	16	----	
ld	A,(FF00+n)	F0 nn	12	----	read from io-port n (memory FF00+n)
ld	(FF00+n),A	E0 nn	12	----	write to io-port n (memory FF00+n)
ld	A,(FF00+C)	F2	8	----	read from io-port C (memory FF00+C)
ld	(FF00+C),A	E2	8	----	write to io-port C (memory FF00+C)
ldi	(HL),A	22	8	----	(HL)=A, HL=HL+1
ldi	A,(HL)	2A	8	----	A=(HL), HL=HL+1
ldd	(HL),A	32	8	----	(HL)=A, HL=HL-1
ldd	A,(HL)	3A	8	----	A=(HL), HL=HL-1

GMB 16bit-Loadcommands

ld	rr,nn	x1 nn nn	12	----	rr=nn (rr may be BC,DE,HL or SP)

```
ld   SP,HL       F9          8 ---- SP=HL
push rr          x5         16 ---- SP=SP-2  (SP)=rr   (rr may be BC,DE,HL,AF)
pop  rr          x1         12 (AF) rr=(SP)  SP=SP+2   (rr may be BC,DE,HL,AF)
```

GMB 8bit-Arithmetic/logical Commands

```
add  A,r         8x          4 z0hc A=A+r
add  A,n         C6 nn       8 z0hc A=A+n
add  A,(HL)      86          8 z0hc A=A+(HL)
adc  A,r         8x          4 z0hc A=A+r+cy
adc  A,n         CE nn       8 z0hc A=A+n+cy
adc  A,(HL)      8E          8 z0hc A=A+(HL)+cy
sub  r           9x          4 z1hc A=A-r
sub  n           D6 nn       8 z1hc A=A-n
sub  (HL)        96          8 z1hc A=A-(HL)
sbc  A,r         9x          4 z1hc A=A-r-cy
sbc  A,n         DE nn       8 z1hc A=A-n-cy
sbc  A,(HL)      9E          8 z1hc A=A-(HL)-cy
and  r           Ax          4 z010 A=A & r
and  n           E6 nn       8 z010 A=A & n
and  (HL)        A6          8 z010 A=A & (HL)
xor  r           Ax          4 z000
xor  n           EE nn       8 z000
xor  (HL)        AE          8 z000
or   r           Bx          4 z000 A=A | r
or   n           F6 nn       8 z000 A=A | n
or   (HL)        B6          8 z000 A=A | (HL)
cp   r           Bx          4 z1hc compare A-r
cp   n           FE nn       8 z1hc compare A-n
cp   (HL)        BE          8 z1hc compare A-(HL)
inc  r           xx          4 z0h- r=r+1
inc  (HL)        34         12 z0h- (HL)=(HL)+1
dec  r           xx          4 z1h- r=r-1
dec  (HL)        35         12 z1h- (HL)=(HL)-1
daa              27          4 z-0x decimal adjust akku
cpl              2F          4 -11- A = A xor FF
```

GMB 16bit-Arithmetic/logical Commands

```
add  HL,rr     x9            8 -0hc HL = HL+rr     ;rr may be BC,DE,HL,SP
inc  rr        x3            8 ---- rr = rr+1      ;rr may be BC,DE,HL,SP
dec  rr        xB            8 ---- rr = rr-1      ;rr may be BC,DE,HL,SP
add  SP,dd     E8           16 00hc SP = SP +/- dd ;dd is 8bit signed number
ld   HL,SP+dd  F8           12 00hc HL = SP +/- dd ;dd is 8bit signed number
```

GMB Rotate- und Shift-Commands

```
rlca        07          4 000c rotate akku left
rla         17          4 000c rotate akku left through carry
rrca        0F          4 000c rotate akku right
rra         1F          4 000c rotate akku right through carry
rlc  r      CB 0x       8 z00c rotate left
rlc  (HL)   CB 06      16 z00c rotate left
rl   r      CB 1x       8 z00c rotate left through carry
rl   (HL)   CB 16      16 z00c rotate left through carry
rrc  r      CB 0x       8 z00c rotate right
rrc  (HL)   CB 0E      16 z00c rotate right
rr   r      CB 1x       8 z00c rotate right through carry
rr   (HL)   CB 1E      16 z00c rotate right through carry
sla  r      CB 2x       8 z00c shift left arithmetic (b0=0)
sla  (HL)   CB 26      16 z00c shift left arithmetic (b0=0)
swap r      CB 3x       8 z000 exchange low/hi-nibble
swap (HL)   CB 36      16 z000 exchange low/hi-nibble
sra  r      CB 2x       8 z00c shift right arithmetic (b7=b7)
sra  (HL)   CB 2E      16 z00c shift right arithmetic (b7=b7)
srl  r      CB 3x       8 z00c shift right logical (b7=0)
srl  (HL)   CB 3E      16 z00c shift right logical (b7=0)
```

GMB Singlebit Operation Commands

```
bit  n,r    CB xx       8 z01- test bit n
bit  n,(HL) CB xx      12 z01- test bit n
set  n,r    CB xx       8 ---- set bit n
set  n,(HL) CB xx      16 ---- set bit n
res  n,r    CB xx       8 ---- reset bit n
res  n,(HL) CB xx      16 ---- reset bit n
```

GMB CPU-Controlcommands

```
ccf         3F          4 -00c cy=cy xor 1
scf         37          4 -001 cy=1
nop         00          4 ---- no operation
halt        76        N*4 ---- halt until interrupt occurs (low power)
stop        10 00       ? ---- low power standby mode (VERY low power)
di          F3          4 ---- disable interrupts, IME=0
ei          FB          4 ---- enable interrupts, IME=1
```

GMB Jumpcommands

```
jp   nn     C3 nn nn   16 ---- jump to nn, PC=nn
jp   HL     E9          4 ---- jump to HL, PC=HL
jp   f,nn   xx nn nn 16;12 ---- conditional jump if nz,z,nc,c
```

```
jr   PC+dd     18 dd       12 ---- relative jump to nn (PC=PC+/-7bit)
jr   f,PC+dd   xx dd     12;8 ---- conditional relative jump if nz,z,nc,c
call nn        CD nn nn    24 ---- call to nn, SP=SP-2, (SP)=PC, PC=nn
call f,nn      xx nn nn 24;12 ---- conditional call if nz,z,nc,c
ret            C9          16 ---- return, PC=(SP), SP=SP+2
ret  f         xx        20;8 ---- conditional return if nz,z,nc,c
reti           D9          16 ---- return and enable interrupts (IME=1)
rst  n         xx          16 ---- call to 00,08,10,18,20,28,30,38
```

Chapter 11

Comparison with Z80

Comparison with 8080

The Game Boy CPU has a bit more in common with an older Intel 8080 CPU than the more powerful Zilog Z80 CPU. It is missing a handful of 8080 instructions but does support JR and almost all CB-prefixed instructions. Also, all known Game Boy assemblers use the more obvious Z80-style syntax, rather than the chaotic 8080-style syntax.

Unlike the 8080 and Z80, the Game Boy has no dedicated I/O bus and no IN/OUT opcodes. Instead, I/O ports are accessed directly by normal LD instructions, or by new LD (FF00+n) opcodes.

The sign and parity/overflow flags have been removed, as have the 12 RET, CALL, and JP instructions conditioned on them. So have EX (SP),HL (XTHL) and EX DE,HL (XCHG).

Comparison with Z80

In addition to the removed 8080 instructions, the other exchange instructions have been removed (including total absence of second register set).

All DD- and FD-prefixed instructions are missing. That means no IX- or IY-registers.

All ED-prefixed instructions are missing. That means 16bit memory accesses are mostly missing, 16bit arithmetic functions are heavily cut-down, and some other missing commands. IN/OUT (C) are replaced with new LD ($FF00+C) opcodes. Block commands are gone, but autoincrementing HL accesses are added.

The Game Boy operates approximately as fast as a 4 MHz Z80 (8 MHz in CGB double speed mode), with execution time of all instructions having been rounded up to a multiple of 4 cycles.

Moved, Removed, and Added Opcodes

Opcode	Z80	GMB
08	EX AF,AF	LD (nn),SP
10	DJNZ PC+dd	STOP
22	LD (nn),HL	LDI (HL),A
2A	LD HL,(nn)	LDI A,(HL)
32	LD (nn),A	LDD (HL),A
3A	LD A,(nn)	LDD A,(HL)
D3	OUT (n),A	-
D9	EXX	RETI
DB	IN A,(n)	-
DD	<IX>	-
E0	RET PO	LD (FF00+n),A
E2	JP PO,nn	LD (FF00+C),A
E3	EX (SP),HL	-
E4	CALL P0,nn	-
E8	RET PE	ADD SP,dd
EA	JP PE,nn	LD (nn),A
EB	EX DE,HL	-
EC	CALL PE,nn	-
ED	<pref>	-
F0	RET P	LD A,(FF00+n)
F2	JP P,nn	LD A,(FF00+C)
F4	CALL P,nn	-
F8	RET M	LD HL,SP+dd
FA	JP M,nn	LD A,(nn)
FC	CALL M,nn	-
FD	<IY>	-
CB3X	SLL r/(HL)	SWAP r/(HL)

Note: The unused (-) opcodes will lock up the Game Boy CPU when used.

Chapter 12

The Cartridge Header

An internal information area is located at 0100-014F in each cartridge. It contains the following values:

0100-0103 - Entry Point

After displaying the Nintendo Logo, the built-in boot procedure jumps to this address (100h), which should then jump to the actual main program in the cartridge. Usually this 4 byte area contains a NOP instruction, followed by a JP 0150h instruction. But not always.

0104-0133 - Nintendo Logo

These bytes define the bitmap of the Nintendo logo that is displayed when the Game Boy gets turned on. The hexdump of this bitmap is:

```
CE ED 66 66 CC 0D 00 0B 03 73 00 83 00 0C 00 0D
00 08 11 1F 88 89 00 0E DC CC 6E E6 DD DD D9 99
BB BB 67 63 6E 0E EC CC DD DC 99 9F BB B9 33 3E
```

The Game Boy's boot procedure verifies the content of this bitmap (after it has displayed it), and LOCKS ITSELF UP if these bytes are incorrect. A CGB verifies only the first 18h bytes of the bitmap, but others (for example a pocket gameboy) verify all 30h bytes.

0134-0143 - Title

Title of the game in UPPER CASE ASCII. If it is less than 16 characters then the remaining bytes are filled with 00's. When inventing the CGB, Nintendo has reduced the length of this area to 15 characters, and some months later they had the fantastic idea to reduce it to 11 characters only. The new meaning of the ex-title bytes is described below.

013F-0142 - Manufacturer Code

In older cartridges this area has been part of the Title (see above), in newer cartridges this area contains an 4 character uppercase manufacturer code. Purpose and Deeper Meaning unknown.

0143 - CGB Flag

In older cartridges this byte has been part of the Title (see above). In CGB cartridges the upper bit is used to enable CGB functions. This is required, otherwise the CGB switches itself into Non-CGB-Mode. Typical values are:

```
80h - Game supports CGB functions, but works on old gameboys also.
C0h - Game works on CGB only (physically the same as 80h).
```

Values with Bit 7 set, and either Bit 2 or 3 set, will switch the gameboy into a special non-CGB-mode with uninitialized palettes. Purpose unknown, eventually this has been supposed to be used to colorize monochrome games that include fixed palette data at a special location in ROM.

0144-0145 - New Licensee Code

Specifies a two character ASCII licensee code, indicating the company or publisher of the game. These two bytes are used in newer games only (games that have been released after the SGB has been invented). Older games are using the header entry at `014B` instead.

Sample licensee codes :

Code	Publisher
00	none
01	Nintendo R&D1
08	Capcom
13	Electronic Arts
18	Hudson Soft
19	b-ai
20	kss
22	pow
24	PCM Complete
25	san-x
28	Kemco Japan
29	seta
30	Viacom
31	Nintendo
32	Bandai
33	Ocean/Acclaim
34	Konami
35	Hector

Code	Publisher
37	Taito
38	Hudson
39	Banpresto
41	Ubi Soft
42	Atlus
44	Malibu
46	angel
47	Bullet-Proof
49	irem
50	Absolute
51	Acclaim
52	Activision
53	American sammy
54	Konami
55	Hi tech entertainment
56	LJN
57	Matchbox
58	Mattel
59	Milton Bradley
60	Titus
61	Virgin
64	LucasArts
67	Ocean
69	Electronic Arts
70	Infogrames
71	Interplay
72	Broderbund
73	sculptured
75	sci
78	THQ
79	Accolade
80	misawa
83	lozc
86	Tokuma Shoten Intermedia
87	Tsukuda Original
91	Chunsoft
92	Video system
93	Ocean/Acclaim
95	Varie
96	Yonezawa/s'pal
97	Kaneko
99	Pack in soft
A4	Konami (Yu-Gi-Oh!)

0146 - SGB Flag

Specifies whether the game supports SGB functions, common values are:

- `00h` : No SGB functions (Normal Gameboy or CGB only game)
- `03h` : Game supports SGB functions

The SGB disables its SGB functions if this byte is set to another value than `03h`.

0147 - Cartridge Type

Specifies which Memory Bank Controller (if any) is used in the cartridge, and if further external hardware exists in the cartridge.

Code	Type
`00h`	ROM ONLY
`01h`	MBC1
`02h`	MBC1+RAM
`03h`	MBC1+RAM+BATTERY
`05h`	MBC2
`06h`	MBC2+BATTERY
`08h`	ROM+RAM
`09h`	ROM+RAM+BATTERY
`0Bh`	MMM01
`0Ch`	MMM01+RAM
`0Dh`	MMM01+RAM+BATTERY
`0Fh`	MBC3+TIMER+BATTERY
`10h`	MBC3+TIMER+RAM+BATTERY
`11h`	MBC3
`12h`	MBC3+RAM
`13h`	MBC3+RAM+BATTERY
`19h`	MBC5
`1Ah`	MBC5+RAM
`1Bh`	MBC5+RAM+BATTERY
`1Ch`	MBC5+RUMBLE
`1Dh`	MBC5+RUMBLE+RAM
`1Eh`	MBC5+RUMBLE+RAM+BATTERY
`20h`	MBC6
`22h`	MBC7+SENSOR+RUMBLE+RAM+BATTERY
`FCh`	POCKET CAMERA
`FDh`	BANDAI TAMA5
`FEh`	HuC3
`FFh`	HuC1+RAM+BATTERY

0148 - ROM Size

Specifies the ROM Size of the cartridge. Typically calculated as “32KB shl N”.

code	Size	Banks
`00h`	32 KByte	no ROM banking
`01h`	64 KByte	4 banks
`02h`	128 KByte	8 banks
`03h`	256 KByte	16 banks
`04h`	512 KByte	32 banks
`05h`	1 MByte	64 banks only 63 banks used by MBC1
`06h`	2 MByte	128 banks only 125 banks used by MBC1
`07h`	4 MByte	256 banks
`08h`	8 MByte	512 banks
`52h`	1.1 MByte	72 banks
`53h`	1.2 MByte	80 banks
`54h`	1.5 MByte	96 banks

0149 - RAM Size

Specifies the size of the external RAM in the cartridge (if any).

```
 00h - None
 01h - 2 KBytes
 02h - 8 Kbytes
 03h - 32 KBytes (4 banks of 8KBytes each)
 04h - 128 KBytes (16 banks of 8KBytes each)
 05h - 64 KBytes (8 banks of 8KBytes each)
```

When using a MBC2 chip 00h must be specified in this entry, even though the MBC2 includes a built-in RAM of 512 x 4 bits.

014A - Destination Code

Specifies if this version of the game is supposed to be sold in Japan, or anywhere else. Only two values are defined.

```
00h - Japanese
01h - Non-Japanese
```

014B - Old Licensee Code

Specifies the games company/publisher code in range 00-FFh. A value of 33h signalizes that the New License Code in header bytes 0144-0145 is used instead. (Super Game Boy functions won’t work if <> $33.) A list of licensee codes can be found here.

014C - Mask ROM Version number

Specifies the version number of the game. That is usually 00h.

014D - Header Checksum

Contains an 8 bit checksum across the cartridge header bytes 0134-014C. The checksum is calculated as follows:

```
x=0:FOR i=0134h TO 014Ch:x=x-MEM[i]-1:NEXT
```

The lower 8 bits of the result must be the same than the value in this entry. The GAME WON'T WORK if this checksum is incorrect.

014E-014F - Global Checksum

Contains a 16 bit checksum (upper byte first) across the whole cartridge ROM. Produced by adding all bytes of the cartridge (except for the two checksum bytes). The Game Boy doesn't verify this checksum.

Chapter 13

Memory Bank Controllers

As the gameboys 16 bit address bus offers only limited space for ROM and RAM addressing, many games are using Memory Bank Controllers (MBCs) to expand the available address space by bank switching. These MBC chips are located in the game cartridge (ie. not in the gameboy itself).

In each cartridge, the required (or preferred) MBC type should be specified in the byte at 0147h of the ROM, as described in the the cartridge header. Several different MBC types are available:

Chapter 14

No MBC

(32KByte ROM only)

Small games of not more than 32KBytes ROM do not require a MBC chip for ROM banking. The ROM is directly mapped to memory at 0000-7FFFh. Optionally up to 8KByte of RAM could be connected at A000-BFFF, even though that could require a tiny MBC-like circuit, but no real MBC chip. # MBC1

(max 2MByte ROM and/or 32KByte RAM)

This is the first MBC chip for the Game Boy. Any newer MBC chips are working similiar, so that is relative easy to upgrade a program from one MBC chip to another - or even to make it compatible to several different types of MBCs.

Note that the memory in range 0000-7FFF is used for both reading from ROM, and for writing to the MBCs Control Registers.

14.0.1 ROM/RAM access

0000-3FFF - ROM Bank 00/20/40/60 (Read Only)

This area normally contains the first 16KBytes (bank 00) of the cartridge ROM. Can contain banks 20/40/60 in mode 1 (see below), or banks 10/20/30 in mode 1 for a 1MB MBC1 multi-cart (see below).

4000-7FFF - ROM Bank 01-7F (Read Only)

This area may contain any of the further 16KByte banks of the ROM. Cannot address any banks where the main ROM banking register would be 00h, which usually means banks 00/20/40/60. Instead, it automatically maps to 1 bank higher (01/21/41/61).

A000-BFFF - RAM Bank 00-03, if any (Read/Write)

This area is used to address external RAM in the cartridge (if any). External RAM is often battery buffered, allowing to store game positions or high score tables, even if the Game Boy is turned off, or if the cartridge is removed from the Game Boy. Available RAM sizes are: 2KByte (at A000-A7FF), 8KByte (at A000-BFFF), and 32KByte (in form of four 8K banks at A000-BFFF).

14.0.2 Control Registers

0000-1FFF - RAM Enable (Write Only)

Before external RAM can be read or written, it must be enabled by writing to this address space. It is recommended to disable external RAM after accessing it, in order to protect its contents from damage during power down of the Game Boy. Usually the following values are used:

```
00h  Disable RAM (default)
0Ah  Enable RAM`
```

Practically any value with 0Ah in the lower 4 bits enables RAM, and any other value disables RAM. RAM is automatically disabled when the gameboy is powered off. It is unknown why Ah is the value used to enable RAM.

2000-3FFF - ROM Bank Number (Write Only)

This 5 bit register (range 01h-1Fh) selects the ROM bank number. Higher bits are discarded - E1h (binary ~~111~~00001) would select bank 01h. If the ROM Bank Number is set to a higher value than the number of banks in the cart, the bank number is masked to the required number of bits. e.g. a 256 kB cart only needs a 4 bit bank number to address all of its 16 banks, so this register is masked to 4 bits. The upper bit would be ignored.

On larger carts which need a >5 bit bank number, the secondary banking register at 4000-5FFF is used to supply an additional 2 bits for the effective bank number: `Selected ROM Bank = (Secondary Bank << 5) + ROM Bank`.

The ROM Bank Number defaults to 01h at power-on. When 00h is written, the MBC translates that to bank 01h also. Not being able to select bank 00h isn't normally a problem, because bank 00h is usually mapped to the 0000-3FFF range. But on large carts (using the secondary banking register to specify the upper ROM Bank bits), the same happens for banks 20h, 40h, and 60h, as this register would need to be 00h for those addresses. Any attempt to address these ROM Banks will select Bank 21h, 41h, and 61h instead. The only way to access banks 20h, 40h or 60h is to enter mode 1, which remaps the 0000-3FFF range. This has its own problems for game developers as that range contains interrupt handlers, so it's mostly only used in multi-game compilation carts (see below).

4000-5FFF - RAM Bank Number - or - Upper Bits of ROM Bank Number (Write Only)

This second 2 bit register can be used to select a RAM Bank in range from 00-03h (32 kB ram carts only), or to specify the upper two bits (Bit 5-6) of the ROM Bank number (1 MB ROM or larger carts only). If neither ROM nor RAM is large enough, setting this register does nothing.

In 1MB MBC1 multi-carts (see below), this 2 bit register is instead applied to bits 4-5 of the ROM bank number, and the top bit of the main 5-bit main ROM banking register is ignored.

6000-7FFF - Banking Mode Select (Write Only)

This 1bit Register selects between the two MBC1 banking modes, controlling the behaviour of the secondary 2 bit banking register (above). If the cart is not large enough to use the 2 bit register (<= 8kB RAM / <= 512 kB ROM) this mode select has no observable effect. The program may freely switch between the two modes at any time.

```
00h = Simple ROM Banking Mode (default)
01h = RAM Banking Mode / Advanced ROM Banking Mode
```

In mode 0, the 2-bit secondary banking register can only affect the 4000-7FFF banked ROM area. If the cart is a "small ROM"/"large RAM" cart (<1 MB ROM, >8kB RAM) then 4000-7FFF is unaffected by this register anyway, so the practical effect is that RAM banking is disabled and A000-BFFF is locked to only be able to access bank 0 of RAM, with the 2-bit secondary banking register entirely ignored. For large ROM carts, the

In mode 1, the behaviour differs depending on whether the current cart is a "large RAM" cart (>8kB RAM) or "large ROM" cart (1 MB or larger). For large RAM carts, switching to mode 1 enables RAM banking and (if RAM is enabled) immediately switches the A000-BFFF RAM area to the bank selected by the 2-bit secondary banking register.

For "large ROM" carts, mode 1 has the 4000-7FFF banked ROM area behave the same as mode 0, but additionally the "unbankable" "bank 0" area 0000-3FFF is now also affected by the 2-bit secondary banking register, meaning it can now be switched between banks 00h, 20h, 40h, and 60h. These banks are inaccessible in mode 0 - they cannot be mapped to the 4000-7FFF banked ROM area.

Note for 1 MB Multi-Game Compilation Carts

Also known as MBC1m, these carts have an alternative wiring, that ignores the top bit of the main ROM banking register (making it a 4 bit register) and applies the 2 bit register to bits 4-5 of the bank number (instead of the usual bits 5-6). This means that in mode 1 the 2 bit register selects banks 00h, 10h, 20h, or 30h, rather than the usual 00h, 20h, 40h or 60h.

These carts make use of the fact that mode 1 remaps the 0000-3FFF area to switch games. The 2 bit register is used to select the game - switching the zero bank and the region of banks that the 4000-7FFF rom area can access to those for the selected game, and then the game only changes the main ROM banking register. As far as the selected game knows, it's running from a 256 kB cart!

These carts can normally be identified by having a Nintendo copyright header in bank 0x10. A badly dumped multi-cart ROM can be identified by having duplicate content in banks 10-1Fh (dupe of 00-0Fh) and banks 30-3Fh (dupe of 20-2Fh). There is a known bad dump of the Mortal Kombat I & II collection around.

An "MBC1M" compilation cart ROM can be converted into a regular MBC1 ROM by increasing the ROM size to 2MB, and duplicating each sub-rom - 00-0Fh duplicated into 10-1Fh, the original 10-1Fh placed in 20-2Fh and duplicated into 30-3Fh, and so on.

Chapter 15

MBC2

(max 256KByte ROM and 512x4 bits RAM)

0000-3FFF - ROM Bank 00 (Read Only)

Same as for MBC1.

4000-7FFF - ROM Bank 01-0F (Read Only)

Same as for MBC1, but only a total of 16 ROM banks is supported.

A000-A1FF - 512x4bits RAM, built-in into the MBC2 chip (Read/Write)

The MBC2 doesn't support external RAM, instead it includes 512x4 bits of built-in RAM (in the MBC2 chip itself). It still requires an external battery to save data during power-off though. As the data consists of 4bit values, only the lower 4 bits of the "bytes" in this memory area are used.

0000-1FFF - RAM Enable (Write Only)

The least significant bit of the upper address byte must be zero to enable/disable cart RAM. For example the following addresses can be used to enable/disable cart RAM: 0000-00FF, 0200-02FF, 0400-04FF, ..., 1E00-1EFF. The suggested address range to use for MBC2 ram enable/disable is 0000-00FF.

2000-3FFF - ROM Bank Number (Write Only)

Writing a value (XXXXBBBB - X = Don't cares, B = bank select bits) into 2000-3FFF area will select an appropriate ROM bank at 4000-7FFF.

The least significant bit of the upper address byte must be one to select a ROM bank. For example the following addresses can be used to select a ROM bank: 2100-21FF, 2300-23FF, 2500-25FF, ..., 3F00-3FFF. The suggested address range to use for MBC2 rom bank selection is 2100-21FF.

Chapter 16

MBC3

(max 2MByte ROM and/or 32KByte RAM and Timer)

Beside for the ability to access up to 2MB ROM (128 banks), and 32KB RAM (4 banks), the MBC3 also includes a built-in Real Time Clock (RTC). The RTC requires an external 32.768 kHz Quartz Oscillator, and an external battery (if it should continue to tick when the Game Boy is turned off).

0000-3FFF - ROM Bank 00 (Read Only)

Same as for MBC1.

4000-7FFF - ROM Bank 01-7F (Read Only)

Same as for MBC1, except that accessing banks 20h, 40h, and 60h is supported now.

A000-BFFF - RAM Bank 00-03, if any (Read/Write)

A000-BFFF - RTC Register 08-0C (Read/Write)

Depending on the current Bank Number/RTC Register selection (see below), this memory space is used to access an 8KByte external RAM Bank, or a single RTC Register.

0000-1FFF - RAM and Timer Enable (Write Only)

Mostly the same as for MBC1, a value of 0Ah will enable reading and writing to external RAM - and to the RTC Registers! A value of 00h will disable either.

2000-3FFF - ROM Bank Number (Write Only)

Same as for MBC1, except that the whole 7 bits of the RAM Bank Number are written directly to this address. As for the MBC1, writing a value of 00h, will select Bank 01h instead. All other values 01-7Fh select the corresponding ROM Banks.

4000-5FFF - RAM Bank Number - or - RTC Register Select (Write Only)

As for the MBC1s RAM Banking Mode, writing a value in range for 00h-03h maps the corresponding external RAM Bank (if any) into memory at A000-BFFF. When writing a value of 08h-0Ch, this will map the corresponding RTC register into memory at A000-BFFF. That register could then be read/written by accessing any address in that area, typically that is done by using address A000.

6000-7FFF - Latch Clock Data (Write Only)

When writing 00h, and then 01h to this register, the current time becomes latched into the RTC registers. The latched data will not change until it becomes latched again, by repeating the write 00h->01h procedure. This is supposed for **reading** from the RTC registers. This can be proven by reading the latched (frozen) time from the RTC registers, and then unlatch the registers to show the clock itself continues to tick in background.

The Clock Counter Registers

```
08h  RTC S   Seconds   0-59 (0-3Bh)
09h  RTC M   Minutes   0-59 (0-3Bh)
0Ah  RTC H   Hours     0-23 (0-17h)
0Bh  RTC DL  Lower 8 bits of Day Counter (0-FFh)
0Ch  RTC DH  Upper 1 bit of Day Counter, Carry Bit, Halt Flag
      Bit 0  Most significant bit of Day Counter (Bit 8)
      Bit 6  Halt (0=Active, 1=Stop Timer)
      Bit 7  Day Counter Carry Bit (1=Counter Overflow)
```

The Halt Flag is supposed to be set before **writing** to the RTC Registers.

The Day Counter

The total 9 bits of the Day Counter allow to count days in range from 0-511 (0-1FFh). The Day Counter Carry Bit becomes set when this value overflows. In that case the Carry Bit remains set until the program does reset it. Note that you can store an offset to the Day Counter in battery RAM. For example, every time you read a non-zero Day Counter, add this Counter to the offset in RAM, and reset the Counter to zero. This method allows to count any number of days, making your program Year-10000-Proof, provided that the cartridge gets used at least every 511 days.

Delays

When accessing the RTC Registers it is recommended to execute a 4ms delay (4 Cycles in Normal Speed Mode) between the separate accesses.

Chapter 17

MBC5

It can map up to 64 Mbits (8 MBytes) of ROM.

MBC5 (Memory Bank Controller 5) is the 4th generation MBC. There apparently was no MBC4, presumably because of the superstition about the number 4 in Japanese culture. It is the first MBC that is guranteed to work properly with GBC double speed mode.

0000-3FFF - ROM Bank 00 (Read Only)

Same as for MBC1.

4000-7FFF - ROM Bank 00-1FF (Read Only)

Same as for MBC1, except that accessing up to bank 1E0h is supported now. Also, bank 0 is actually bank 0.

A000-BFFF - RAM Bank 00-0F, if any (Read/Write)

Same as for MBC1, except RAM sizes are 64kbit, 256kbit and 1mbit.

0000-1FFF - RAM Enable (Write Only)

Mostly the same as for MBC1, a value of 0Ah will enable reading and writing to external RAM. A value of 00h will disable it.

2000-2FFF - Low 8 bits of ROM Bank Number (Write Only)

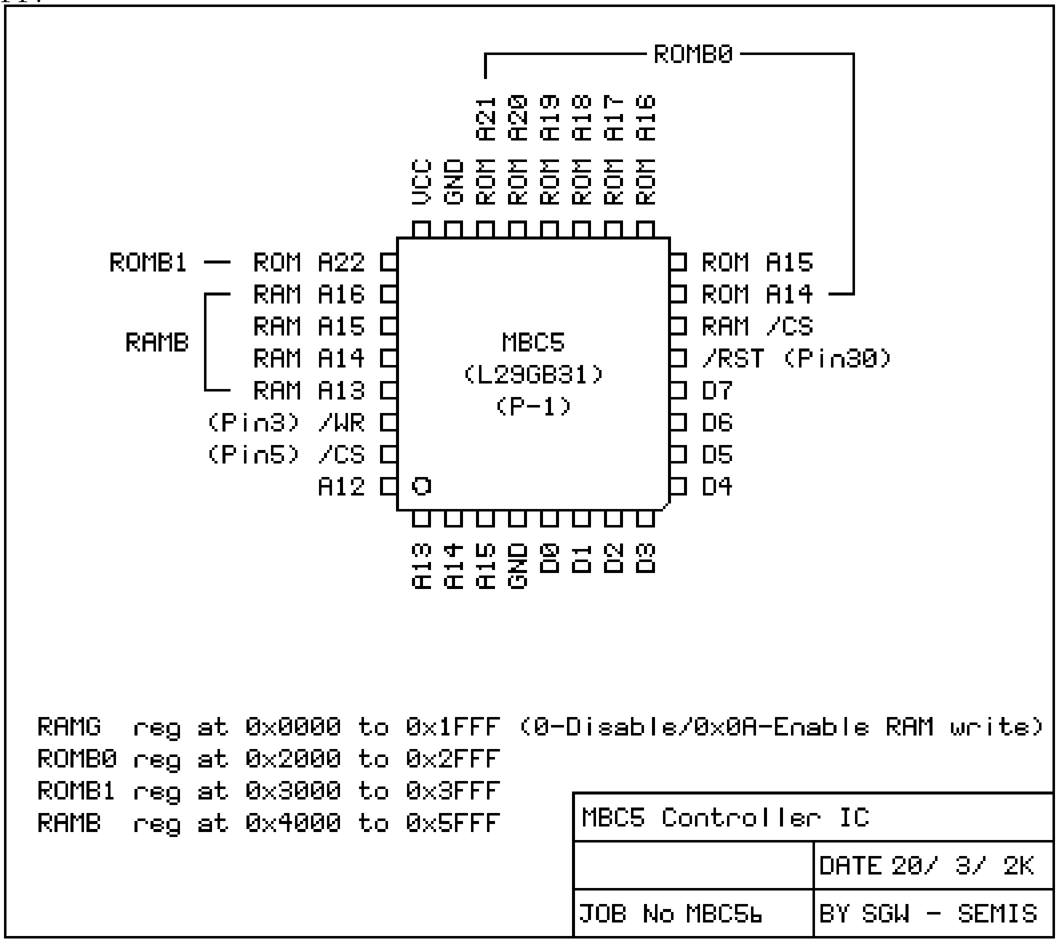
The lower 8 bits of the ROM bank number goes here. Writing 0 will indeed give bank 0 on MBC5, unlike other MBCs.

3000-3FFF - High bit of ROM Bank Number (Write Only)

The 9th bit of the ROM bank number goes here.

4000-5FFF - RAM Bank Number (Write Only)

As for the MBC1s RAM Banking Mode, writing a value in range for 00h-0Fh maps the corresponding external RAM Bank (if any) into memory at A000-BFFF.



# MBC6

17.0.1 Overview

MBC6 (Memory Bank Controller 6) is an unusual MBC that contains two separately switchable ROM banks ($4000 and 6000)*andRAMbanks*(A000 and $B000), SRAM and an 8 Mbit Macronix MX29F008TC-14 flash memory chip. It is only used in one game, Net de Get: Minigame @ 100, which uses the Mobile Adapter to connect to the web to download minigames onto the local flash. Both ROM banks and both RAM banks are views into the same ROM and RAM, but with separately adjustable offsets. Since the banked regions are smaller the effective number of banks is twice what it usually would be; 8 kB

ROM banks instead of 16 kB and 4 kB RAM banks instead of 8 kB.

0000-3FFF - ROM Bank 00 (Read Only)

Read-only access to ROM bank 0.

4000-5FFF - ROM/Flash Bank A 00-7F (Read/Write for flash, Read Only for ROM)

Read-only access to ROM and flash banks 00-7F, switchable independently from ROM/Flash Bank B.

6000-7FFF - ROM/Flash Bank B 00-7F (Read/Write for flash, Read Only for ROM)

Read-only access to ROM and flash banks 00-7F, switchable independently from ROM/Flash Bank A.

A000-AFFF - RAM Bank A 00-07 (Read/Write)

Read/write access to RAM banks 00-07, switchable independently from RAM Bank B.

B000-BFFF - RAM Bank B 00-07 (Read/Write)

Read/write access to RAM banks 00-07, switchable independently from RAM Bank A.

0000-03FF - RAM Enable (Write Only)

Mostly the same as for MBC1, a value of 0Ah will enable reading and writing to external RAM. A value of 00h will disable it.

0400-07FF - RAM Bank A Number (Write Only)

Select the active RAM Bank A (A000-AFFF)

0800-0BFF - RAM Bank B Number (Write Only)

Select the active RAM Bank B (B000-BFFF)

0C00-0FFF - Flash Enable (Write Only)

Enable or disable access to the flash chip. Only the lowest bit (0 for disable, 1 for enable) is used. Flash Write Enable must be active to change this.

1000 - Flash Write Enable (Write Only)

Enable or disable write mode for the flash chip. Only the lowest bit (0 for disable, 1 for enable) is used. Note that this maps to the /WE pin on the flash chip, not whether or not writing to the bus is enabled; some flash commands (e.g. JEDEC ID query) still work with this off so long as Flash Enable is on.

2000-27FF - ROM/Flash Bank A Number (Write Only)

The number for the active bank in ROM/Flash Bank A.

2800-2FFF - ROM/Flash Bank A Select (Write Only)

Selects whether the ROM or the Flash is mapped into ROM/Flash Bank A. A value of 00 selects the ROM and 08 selects the flash.

3000-37FF - ROM/Flash Bank B Number (Write Only)

The number for the active bank in ROM/Flash Bank B.

3800-3FFF - ROM/Flash Bank B Select (Write Only)

Selects whether the ROM or the Flash is mapped into ROM/Flash Bank B. A value of 00 selects the ROM and 08 selects the flash.

Flash Commands

The flash chip is mapped directly into the A or B address space, which means standard flash access commands are used. To issue a command, you must write the value $AA to $5555 then $55 to $2AAA and, which are mapped as 2:5555/1:4AAA for bank A or 2:7555/1:6AAA for bank B followed by the command at either 2:5555/2:7555, or a relevant address, depending on the command.

The commands and access sequences are as follows, were X refers to either 4 or 6 and Y to 5 or 7, depending on the bank region:

```
------------- ------------- ------------- ------------- ------------- ------------- ------
2:Y555=$AA    1:XAAA=$55    2:Y555=$80    2:Y555=$AA    1:XAAA=$55    ?:X000=$30    Erase secto
2:Y555=$AA    1:XAAA=$55    2:Y555=$80    2:Y555=$AA    1:XAAA=$55    ?:Y555=$10    Erase chip\
2:Y555=$AA    1:XAAA=$55    2:Y555=$90                                                 ID mode
2:Y555=$AA    1:XAAA=$55    2:Y555=$A0                                                 Program
2:Y555=$AA    1:XAAA=$55    2:Y555=$F0                                                 Exit ID/
2:Y555=$AA    1:XAAA=$55    ?:X000=$F0                                                 Exit era
?:????=$F0                                                                               Exit p
------------- ------------- ------------- ------------- ------------- ------------- ------
```

Commands marked with * require the Write Enable bit to be 1. These will make the flash read out status bytes instead of values. A status of $80 means the operation has finished and you should exit the mode using the appropriate command. A status of $10 indicates a timeout.

Programming must be done by first erasing a sector, activating write mode, writing out 128 bytes (aligned), then writing a 0 to the final address to commit the write, waiting for the status to indicate completion, and writing $F0 to the final address again to exit program mode. If a sector is not erased first programming will not work properly. In some cases it will only allow the stored bytes to be anded instead of replaced; in others it just won't work at all. The only way to set the bits back to 1 is to erase the sector entirely. It is recommended to check the flash to make sure all bytes were written properly and re-write (without erasing) the 128 byte block if some bits didn't get set to 0 properly. After writing all blocks in a sector Flash Write Enable should be set to 0.

Source: 1

Chapter 18

MBC7

18.0.1 Overview

MBC7 (Memory Bank Controller 7) is an MBC containing a 2-axis accelerometer (ADXL202E) and a 256 byte EEPROM (93LC56). A000-BFFF does not directly address the EEPROM, as most MBCs do, but rather contains several registers that can be read or written one at a time. This makes EEPROM access very slow due to needing multiple writes per address.

0000-3FFF - ROM Bank 00 (Read Only)

Same as for MBC5.

4000-7FFF - ROM Bank 00-7F (Read Only)

Same as for MBC5. (Bank 0 mapping needs confirmation)

A000-AFFF - RAM Registers (Read/Write)

Must be enabled via 0000 and 4000 region writes (see respective sections), otherwise reads read FFh and writes do nothing. Registers are addressed through bits 4-7 of the address. Bits 0-3 and 8-11 are ignored.

Accelerometer data must be latched before reading. Data is 16-bit and centered at the value 81D0. Earth's gravity affects the value by roughly 70h, with larger acceleration providing a larger range. Maximum range is unknown.

Ax0x/Ax1x - Latch Accelerometer (Write Only)

Write 55h to Ax0x to erase the latched data (reset back to 8000) then AAh to Ax1x to latch the accelerometer and update the addressable registers. Reads return FFh. Other writes do not appear to do anything (Partially unconfirmed). Note that you cannot re-latch the accelerometer value without first erasing it; attempts to do so yield no change.

Ax2x/Ax3x - Accelerometer X value (Read Only)

Ax2x contains the low byte of the X value (lower values are towards the right and higher values are towards the left), and Ax3x contains the high byte. Reads 8000 before first latching.

Ax4x/Ax5x - Accelerometer Y value (Read Only)

Ax4x contains the low byte of the Y value (lower values are towards the bottom and higher values are towards the top), and Ax5x contains the high byte. Reads 8000 before first latching.

Ax6x/Ax7x - Unknown

Ax6x always reads 00h and Ax7x always reads FFh. Possibly reserved for Z axis, which does not exist on this accelerometer.

Ax8x - EEPROM (Read/Write)

Values in this register correspond to 4 pins on the EEPROM:

- Bit 0: Data Out (DO)
- Bit 1: Data In (DI)
- Bit 6: Clock (CLK or SK in existing code)
- Bit 7: Chip Select (CS)

The other pins (notably ORG, which controls 8-bit vs 16-bit addressing) do not appear to be connected to this register.

Commands are sent to the EEPROM by shifting in a bitstream to DI while manually clocking CLK. All commands must be preceded by a 1 bit, and existing games precede the 1 bit with a 0 bit (though this is not necessary):

- Write 00h (lower CS)
- Write 80h (raise CS)
- Write C0h (shift in 0 bit)
- Write 82h (lower CS, raise DI)
- Write C2h (shift in 1 bit)
- Write command

The following commands exist, each 10 bits (excluding data shifted in or out). ”x” means the value of this bit is ignored. ”A” means the relevant bit of the address. All data is shifted in or out MSB first. Note that data is addressed 16 bits at a time, so address 1 corresponds to bits 16-31, thus bytes 2-3.

- READ: 10xAAAAAAAb (then shift out 16 bits)
- EWEN (Erase/Write enable): 0011xxxxxxb
- EWDS (Erase/Write disable): 0000xxxxxxb
- WRITE: 01xAAAAAAAb (then shift in 16 bits)

- ERASE (fill address with FFFF): 11xAAAAAAAb
- ERAL (fill EEPROM with FFFF): 0010xxxxxxb
- WRAL (fill EEPROM with value): 0001xxxxxxb (then shift in 16 bits)

All programming operations (WRITE/ERASE/WRAL/ERAL) must be preceded with EWEN.

According to the datasheet, programming operations take time to settle. Continue clocking and check the value of DO to verify if command is still running. Data sheet says that the signal to DO is RDY, thus it reads a 1 when the command finishes.

Datasheet: 1

Ax9x-AxFx - Unused

Reads out FFh.

B000-BFFF - Unknown

Only seems to read out FFh.

0000-1FFF - RAM Enable 1 (Write Only)

Mostly the same as for MBC1, a value of 0Ah will enable reading and writing to RAM registers. A value of 00h will disable it. Please note that the RAM must second be enabled in the second RAM enable section as well (4000-5FFF)

2000-3FFF - ROM Bank Number (Write Only)

The ROM bank number goes here.

4000-5FFF - RAM Enable 2 (Write Only)

Writing 40h to this region enables access to the RAM registers. Writing any other value appears to disable access to RAM, but this is not fully tested. Please note that the RAM must first be enabled in the first RAM enable section as well (0000-1FFF)

Source: 2

Chapter 19

HuC1

19.0.1 HuC1

HuC1 is an MBC used by some Game Boy games which besides doing the usual MBC stuff, also provides IR comms. A lot of sources on the internet said that HuC1 was ”similar to MBC1”, but they didn’t provide any detail. I took a look, and it turns out that HuC1 differs from MBC1 quite a lot.

19.0.2 Memory Map

```
0000-1FFF   IR select 2000-3FFF   ROM bank select 4000-5FFF   RAM bank
select 6000-7FFF   Nothing? -- A000-BFFF   Cart RAM or IR register
```

19.0.3 0000-1FFF IR Select (Write Only)

Most MBCs let you disable the cartridge RAM to prevent accidental writes. HuC1 doesn’t do this, instead you use this register to switch the A000-BFFF region between ”RAM mode” and ”IR mode” (described below). Write 0x0E to switch to IR mode, and anything else to switch to RAM mode. Nevertheless some HuC1 games attempt to write 0x0A and 0x00 to this region as if it would enable/disable cart RAM.

19.0.4 2000-3FFF ROM Bank Number (Write Only)

HuC1 can accept a bank number of at least 6 bits here.

19.0.5 4000-5FFF RAM Bank Select (Write Only)

HuC1 can accept a bank number of at least 2 bits here.

19.0.6 6000-7FFF Nothing? (Write Only)

Writes to this region seem to have no effect. Even so, some games do write to this region, as if it had the same effect as on MBC1. You may safely ignore these writes.

19.0.7 A000-BFFF Cart RAM or IR register (Read/Write)

When in ”IR mode” (wrote 0x0E to 0x0000), the IR register is visible here. Write to this region to control the IR transmitter. 0x01 turns it on, 0x00 turns it off. Read from this region to see either 0xC1 (saw light) or 0xC0 (did not see light). When in ”RAM mode” (wrote something other than 0x0E to 0x000) this region behaves like normal cart RAM.

19.0.8 External links

- Source on jrra.zone

Chapter 20

Other MBCs

20.0.1 Multicart MBCs

MBC1M uses the MBC1 IC, but the board does not connect the MBC1's A18 address output to the ROM. This allows including multiple 2 Mbit (16 bank) games, with SRAM bank select ($4000) to select which of up to four games is switched in. In theory, a MBC1M board could be made for 1 Mbit or 512 kbit games by additionally not connecting A17 and A16 outputs, but this appears not to have been done in licensed games.

MMM01 is a more complex that allows for games of different sizes Docs on Tauwasser.eu

Bung and **EMS** MBCs are reported to exist.

EMS

PinoBatch learned the game selection protocol for EMS flash carts from beware, who in turn learned it from nitro2k01. Take this with a grain of salt, as it hasn't been verified on the authentic EMS hardware.

A header matching any of the following is detected as EMS mapper:

- Header name is "EMSMENU", NUL-padded
- Header name is "GB16M", NUL-padded
- Cartridge type ($0147) = $1B and region ($014A) = $E1

Registers:

$2000 write: Normal behavior, plus save written value in $2000 latch $1000 write: $A5 enables configure mode, $98 disables, and other values have no known effect $7000 write while configure mode is on: Copy $2000 latch to OR mask

After the OR mask has been set, all reads from ROM will OR A21-A14 (the bank number) with the OR mask. This chooses which game is visible to the CPU. If the OR mask is not aligned to the game size, the results may be nonsensical.

The mapper does not support an outer bank for battery SRAM.

To start a game, do the following in code run from RAM: Write $A5 to $1000, write game starting bank number to $2000, write any value to $7000, write $98 to $1000, write $01 to $2000 (so that 32K games work), jump to $0100.

Wisdom Tree

The Wisdom Tree mapper is a simple, cost-optimized one chip design consisting of a 74LS377 octal latch, aside from the ROM chip. Because the mapper consists of a single standard 74 series logic chip, it has two unusual properties:

First, unlike a usual MBC, it switches the whole 32 kiB ROM area instead of just the $4000-$7FFF area. If you want to use the interrupt vectors with this cart, you should duplicate them across all banks. Additionally, since the initial state of the '377 can't be guaranteed, the ROM header and some code for switching to a known bank should also be included in every bank. This also means that the Wisdom Tree mapper could be used as a multicart mapper for 32 kiB ROMs, assuming there was enough ROM space in each bank for some small initialization code, and none of the ROMs wrote to the $0000-$7FFF area. For example, if the last 5 bytes of all banks are unused, games can be patched as follows:

```
; At $0100 in all banks but the first
  nop
  jp $7FFB

; At $7FFB in all banks
  ld hl, $0100
  ld [hl], a
  jp hl
```

Second, because the '377 latches data on the *positive* edge, and the value on the Game Boy data bus is no longer valid when the positive edge of the write pulse arrives, the designer of this mapper chose to use the A7-A0 address lines for selecting a bank instead of the data lines. Thus, the value you write is ignored, and the lower 8 bits of the address is used. For example, to select bank $XX, you would write any value to address $YYXX, where $YY is in the range $00-$7F.

An emulator can detect a ROM designed for Wisdom Tree mapper in one of two ways:

- ROM contains "WISDOM TREE" or "WISDOM\x00TREE" (the space can be $20 or $00), $0147 = $00, $0148 = $00, size > 32k. This method works for the games released by Wisdom Tree, Inc.
- $0147 = $C0, $014A = $D1. These are the values recommended by beware for 3rd party developers to indicate that the ROM is targeting Wisdom Tree mapper hardware. (See below.)

Magic values for detection of multicarts in emulators

Sometimes it may be useful to allow a ROM to be detected as a multicart in emulator, for example for development of a menu for physical multicart hardware. These are values suggested by beware, and supported in BGB, for signaling that your ROM is supposed to interface a multicart mapper. Emulator authors who are interested in supporting multicart mappers are encouraged to support detection of these values in addition to the values described in each section, which are heuristics based on ROMs in the wild, which may not always be suitable for newly produced software. The values are deliberately chosen to be high entropy ("random") such that the risk of an accidental false positive is unlikely.

- $0147 = $c0, $014a = $d1 -> Detect as Wisdom Tree
- $0147 = $1b, $014a = $e1 -> Detect as EMS multicart
- $0147 = $be -> Detect as Bung multicart

20.0.2 MBC Timing Issues

Among Nintendo MBCs, only the MBC5 is guaranteed by Nintendo to support the tighter timing of CGB Double Speed Mode. There have been rumours that older MBCs (like MBC1-3) wouldn't be fast enough in that mode. If so, it might be nevertheless possible to use Double Speed during periods which use only code and data which is located in internal RAM. However, despite of the above, a self-made MBC1-EPROM card appears to work stable and fine even in Double Speed Mode though.

Chapter 21

Game Boy Printer

The Gameboy Printer is a portable thermal printer made by SII for Nintendo, which a few games used to print out bonus artwork, certificates, pictures (Gameboy Camera)...

It can use standard 38mm paper and interfaces with the Gameboy through the Link port.

It is operated by an embedded 8bit microcontroller which has its own 8KiB of RAM to buffer incoming graphics data. Those 8KiB allow a maximum bitmap area of 160*200 (8192/160*4) pixels between prints.

21.0.1 Communication

The Gameboy Printer doesn't use the full-duplex capability of the Link port. It accepts variable length data packets and then answers back its status after two $00 writes.

The packets all follow this format:

	Size (bytes)	GB -> Printer	Printer -> GB
Magic bytes	2	$88	0x00
Command	1	$33	0x00
Compression flag	1	See below	0x00
Length of data	2	0/1	0x00
Command-specific data	Variable	LSB	0x00
Checksum	2	MSB	0x00
Alive indicator	1	See below	0x00
Status	1	LSB	0x00

The checksum is simply a sum of every byte sent except the magic bytes and obviously, the checksum itself.

21.0.2 Detection

Send these 9 bytes: $88,$33,$0F,$00,$00,$00,$0F,$00 (Command $0F, no data).

Send $00 and read input, if the byte is $81, then the printer is there. Send a last $00, just for good measure. Input can be ignored.

21.0.3 Command 1: Initialize

This clears the printer's buffer RAM.

No data required. The normal status replied should be $00.

21.0.4 Command 2: Start printing

Data length: 4 bytes

- Byte 1: Number of sheets to print (0-255). 0 means line feed only.
- Byte 2: Margins, high nibble is the feed before printing, low nibble is after printing. GB Camera sends $13 by default.
- Byte 3: Palette, typically $E4 (0b11100100)
- Byte 4: 7 bits exposure value, sets the burning time for the print head. GB Camera sends $40 by default. Official manual mentions -25% darkness for $00 and +25% for $7F.

21.0.5 Command 4: Fill buffer

Data length: max. $280 (160*16 pixels in 2BPP) To transfer more than $280 bytes, multiple "command 4 packets" have to be sent.

The graphics are organized in the normal tile format (16 bytes per tile), and the tiles are sent in the same order they occur on your tilemap (do keep in mind though that the printer does *not* have 32x32 tiles space for a map, but only 20x18).

An empty data packet must be sent before sending command 2 to print the data, otherwise the print command will be ignored.

21.0.6 Command $F: Read status

No data required, this is a "nop" command used only to read the Status byte.

21.0.7 Status byte

A nonzero value for the higher nibble indicates something went wrong.

Bit 7	Low Battery	Set when the voltage is below threshold
Bit 6	Other error	
Bit 5	Paper jam	Set when the encoder gives no pulses when the motor is powere
Bit 4	Packet error	

Bit 3	Unprocessed data	Set when there's unprocessed data in memory - AKA ready to pr
Bit 2	Image data full	
Bit 1	Currently printing	Set as long as the printer's burnin' paper
Bit 0	Checksum error	Set when the calculated checksum doesn't match the received c

21.0.8 Example

- Send command 1, the answer should be $81, $00
- Send command 4 with $280 of your graphics, the answer should still be $81, $00
- Ask for status with command $F, the answer should now be $81, $08 (ready to print)
- Send an empty command 4 packet, the answer should still be $81, $08
- Send command 2 with your arguments (margins, palette, exposure), the answer should still be $81, $08
- Ask for status with command $F until it changes to $81, $06 (printing !)
- Ask for status with command $F until it changes to $81, $04 (printing done)
- Optionally send 16 zero bytes to clear the printer's receive buffer (GB Camera does it)

21.0.9 Tips

- **The printer has a timeout of 100ms for packets. If no packet is received within that time, the printer will return to an initialized state (meaning the link and graphics buffers are reset).**
- **There appears to be an undocumented timeout for the bytes of a packet. It's best to send a packet completely or with very little delay between the individual bytes, otherwise the packet may not be accepted.**
- To print things larger than 20x18 (like GB Camera images with big borders), multiple data packets with a following print command need to be sent. The print command should be set to no linefeed (neither before nor after) to allow for continuous printing.

21.0.10 Compression

Some sort of RLE ? The GB Camera doesn't use it.

(Details and pictures, need to be copied here)

Chapter 22

Power Up Sequence

When the Game Boy is powered up, a 256 byte program starting at memory location 0 is executed. This program is located in a ROM inside the GameBoy. The first thing the program does is read the cartridge locations from $104 to $133 and place this graphic of a Nintendo logo on the screen at the top.

This image is then scrolled until it is in the middle of the screen. Two musical notes are then played on the internal speaker. Again, the cartridge locations $104 to $133 are read but this time they are compared with a table in the internal rom.

If any byte fails to compare, then the Game Boy stops comparing bytes and simply halts all operations.

If all locations compare the same, then the GameBoy starts adding all of the bytes in the cartridge from $134 to $14d. A value of 25 decimal is added to this total.

If the least significant byte of the result is a not a zero, then the Game Boy will stop doing anything.

If it is a zero, then the internal ROM is disabled and cartridge program execution begins at location $100 with the following register values:

```
AF=$01B0
BC=$0013
DE=$00D8
HL=$014D
Stack Pointer=$FFFE
[$FF05] = $00   ; TIMA
[$FF06] = $00   ; TMA
[$FF07] = $00   ; TAC
[$FF10] = $80   ; NR10
[$FF11] = $BF   ; NR11
[$FF12] = $F3   ; NR12
[$FF14] = $BF   ; NR14
[$FF16] = $3F   ; NR21
```

```
[$FF17] = $00   ; NR22
[$FF19] = $BF   ; NR24
[$FF1A] = $7F   ; NR30
[$FF1B] = $FF   ; NR31
[$FF1C] = $9F   ; NR32
[$FF1E] = $BF   ; NR33
[$FF20] = $FF   ; NR41
[$FF21] = $00   ; NR42
[$FF22] = $00   ; NR43
[$FF23] = $BF   ; NR44
[$FF24] = $77   ; NR50
[$FF25] = $F3   ; NR51
[$FF26] = $F1-GB, $F0-SGB ; NR52
[$FF40] = $91   ; LCDC
[$FF42] = $00   ; SCY
[$FF43] = $00   ; SCX
[$FF45] = $00   ; LYC
[$FF47] = $FC   ; BGP
[$FF48] = $FF   ; OBP0
[$FF49] = $FF   ; OBP1
[$FF4A] = $00   ; WY
[$FF4B] = $00   ; WX
[$FFFF] = $00   ; IE
```

It is not a good idea to assume the above values will always exist. A later version Game Boy could contain different values than these at reset. Always set these registers on reset rather than assume they are as above.

Please note that Game Boy internal RAM on power up contains random data.

All of the Game Boy emulators tend to set all RAM to value $00 on entry.

Cart RAM the first time it is accessed on a real Game Boy contains random data. It will only contain known data if the Game Boy code initializes it to some value.

Chapter 23

Reducing Power Consumption

The following programming techniques can be used to reduce the power consumption of the Game Boy hardware and extend the life of the batteries.

23.1 Using the HALT Instruction

The HALT instruction should be used whenever possible to reduce power consumption & extend the life of the batteries. This command stops the system clock, reducing the power consumption of both the CPU and ROM.

The CPU will remain stopped until an interrupt occurs at which point the interrupt is serviced and then the instruction immediately following the HALT is executed.

Depending on how much CPU time is required by a game, the HALT instruction can extend battery life anywhere from 5 to 50% or possibly more.

When waiting for a vblank event, this would be a BAD example:

```
@@wait:
 ld   a,(0FF44h)      ;LY
 cp   a,144
 jr   nz,@@wait
```

A better example would be a procedure as shown below. In this case the vblank interrupt must be enabled, and your vblank interrupt procedure must set vblank_flag to a non-zero value.

```
 ld   hl,vblank_flag  ;hl=pointer to vblank_flag
 xor  a               ;a=0
@@wait:               ;wait...
 halt                 ;suspend CPU - wait for ANY interrupt
 cp   a,(hl)          ;vblank flag still zero?
```

```
  jr   z,@@wait         ;wait more if zero
  ld   (hl),a           ;set vblank_flag back to zero
```

The vblank_flag is used to determine whether the HALT period has been terminated by a vblank interrupt, or by another interrupt. In case your program has all other interrupts disabled, then it would be okay to replace the above procedure by a single HALT instruction.

Another possibility is, if your game uses no other interrupt than VBlank (or uses no interrupt), to only enable VBlank interrupts and simply use a halt instruction, which will only resume main code execution when a VBlank occurs.

Remember when using HALT to wait between VBlanks, your interrupt routines MUST enable interrupts (ie with ei during the execution, or better, using the RETI instruction)

23.2 Using the STOP Instruction

The STOP instruction is intended to switch the Game Boy into VERY low power standby mode. For example, a program may use this feature when it hasn't sensed keyboard input for a longer period (for example, when somebody forgot to turn off the gameboy).

Before invoking STOP, it might be required to disable Sound and Video manually (as well as IR-link port in CGB). Much like HALT, the STOP state is terminated by interrupt events. STOP is commonly terminated with a joypad interrupt.

During STOP mode, the display will turn white, so avoid using it in your game's main loop.

23.3 Disabling the Sound Controller

If your program doesn't use sound at all (or during some periods) then write 00h to register FF26 to save 16% or more on GB power consumption. Sound can be turned back on by writing 80h to the same register, all sound registers must be then re-initialized. When the Game Boy is turned on, sound is enabled by default, and must be turned off manually when not used.

23.4 Not using CGB Double Speed Mode

Because CGB Double Speed mode consumes more power, it's recommended to use normal speed when possible. There's limited ability to switch between both speeds, for example, a game might use normal speed in the title screen, and double speed in the game, or vice versa. However, during speed switch, the display collapses for a short moment, so it's not a good idea to alter speeds within active game or title screen periods.

23.5 Using the Skills

Most of the above power saving methods will produce best results when using efficient and tight assembler code which requires as little CPU power as possible. Using a high level language will require more CPU power and these techniques will not have as big as an effect.

To optimize you code, it might be a good idea to look at this page, although it applies to the original Z80 CPU, so one must adapt the optimizations to the GBZ80.

Chapter 24

Sprite RAM Bug

There is a flaw in the Game Boy hardware that causes trash to be written to OAM RAM if the following commands are used while their 16-bit content is in the range of \$FE00 to \$FEFF while the PPU is in mode 2:

```
inc rr        dec rr          ;rr = bc,de, or hl
ldi a,(hl)    ldd a,(hl)
ldi (hl),a    ldd (hl),a
```

Sprites 1 & 2 (\$FE00 & \$FE04) are not affected by this bug.
Game Boy Color and Advance are not affected by this bug.

24.1 Accurate Description

The Sprite RAM Bug (or OAM Bug) actually consists of two different bugs:

- Attempting to read or write from OAM (Including the $FFA0-$FEFF region) while the PPU is in mode 2 (OAM mode) will corrupt it.
- Performing an increase or decrease operation on any 16-bit register (BC, DE, HL, SP or PC) while that register is in the OAM range (\$FE00 - \$FEFF) will trigger a memory write to OAM, causing a corruption.

24.2 Affected Operations

The following operations are affected by this bug:

- Any memory access instruction, if it accesses OAM
- `inc rr`, `dec rr` - if `rr` is a 16-bit register pointing to OAM, it will trigger a write and corrupt OAM
- `ldi [hl], a`, `ldd [hl], a`, `ldi a, [hl]`, `ldd a, [hl]`- these will trigger a corruption twice if `hl` points to OAM; once for the usual memory access, and once for the extra write trigger by the inc/dec

- `pop rr`, the `ret` family - For some reason, pop will trigger the bug only 3 times (instead of the expected 4 times); one read, one glitched write, and another read without a glitched write. This also applies to the ret instructions.
- `push rr`, the `call` family, `rst xx` and interrupt handling - Pushing to the stack will trigger the bug 4 times; two usual writes and two glitched write caused by the decrease. However, since one glitched write occur in the same cycle as a actual write, this will effectively behave like 3 writes.
- Executing code from OAM - If PC is inside OAM (executing FF, i.e.`rst $38`) the bug will trigger twice, once for increasing PC inside OAM (triggering a write), and once for reading from OAM. If a multi-byte opcode is executed from $FDFF or $FDFE, and bug will similarly trigger twice for every read from OAM.

24.3 Corruption Patterns

The OAM is split into 20 rows of 8 bytes each, and during mode 2 the PPU reads those rows consecutively; one every 1 M-cycle. The operations patterns rely on type of operation (read/write/both) used on OAM during that M-cycle, as well as the row currently accessed by the PPU. The actual read/write address used, or the written value have no effect. Additionally, keep in mind that OAM uses a 16-bit data bus, so all operations are on 16-bit words.

24.3.1 Write Corruption

A write corruption corrupts the currently access row in the following manner, as long as it's not the first row (containing the first two sprites):

- The first word in the row is replaced with this bitwise expression: `((a ^ c) & (b ^ c)) ^ c`, where `a` is the original value of that word, `b` is the first word in the preceding row, and `c` is the third word in the preceding row.
- The last three words are copied from the last three words in the preceding row.

24.3.2 Read Corruption

A read corruption works similarly to a write corruption, except the bitwise expression is `b | (a & c)`.

24.3.3 Write During Increase/Decrease

If a register is increased or decreased in the same M cycle of a write, this will effectively trigger two writes in a single M-cycle. However, this case behaves just like a single write.

24.3.4 Read During Increase/Decrease

If a register is increased or decreased in the same M cycle of a write, this will effectively trigger both a read **and** a write in a single M-cycle, resulting in a more complex corruption pattern:

- This corruption will not happen if the accessed row is one of the first four, as well as if it's the last row:
  - The first word in the row preceding the currently accessed row is replaced with the following bitwise expression: `(b & (a | c | d)) | (a & c & d)` where `a` is the first word two rows before the currently accessed row, `b` is the first word in the preceding row (the word being corrupted), `c` is the first word in the currently accessed row, and `d` is the third word in the preceding row.
  - The contents of the preceding row is copied (after the corruption of the first word in it) both to the currently accessed row and to two rows before the currently accessed row
- Regardless of wether the previous corruption occurred or not, a normal read corruption is then applied.

Chapter 25

External Connectors

25.0.1 Cartridge Slot

Pin	Name	Expl.
1	VDD	Power Supply +5V DC
2	PHI	System Clock
3	/WR	Write
4	/RD	Read
5	/CS	Chip Select
6-21	A0-A15	Address Lines
22-29	D0-D7	Data Lines
30	/RES	Reset signal
31	VIN	External Sound Input
32	GND	Ground

25.0.2 Link Port

Pin numbers are arranged as 2,4,6 in upper row, 1,3,5 un lower row; outside view of Game Boy socket; flat side of socket upside. Colors as used in most or all standard link cables, because SIN and SOUT are crossed, colors Red and Orange are exchanged at one cable end.

Pin	Name	Color	Expl.
1	VCC	-	+5V DC
2	SOUT	red	Data Out
3	SIN	orange	Data In
4	P14	-	Not used
5	SCK	green	Shift Clock
6	GND	blue	Ground

Note: The original Game Boy used larger plugs (unlike pocket gameboys and newer), linking between older/newer gameboys is possible by using cables with one large and one small plug though.

25.0.3 Stereo Sound Connector (3.5mm, female)

Pin	Expl.
Tip	Sound Left
Middle	Sound Right
Base	Ground

25.0.4 External Power Supply

…

25.0.5 References

- Antonio Niño Díaz - The Cycle-Accurate Game Boy Docs
- Antonio Niño Díaz - Game Boy Camera RE
- Costis Sideris. The quest for dumping GameBoy Boot ROMs!
- Tauwasser. MBC1 - Tauwasser's Wiki
- Tauwasser. MBC2 - Tauwasser's Wiki
- Gekkio. Game Boy: Complete Technical Reference
- Game Boy CPU (SM83) instruction set
- Gekkio. Dumping the Super Game Boy 2 boot ROM
- exezin. OAM DMA tutorial
- MBC5 schematic
- Furrtek - Reverse-engineered schematics for DMG-CPU-B
- Furrtek - Game Boy Printer
- Pan of ATX, Marat Fayzullin, Felber Pascal, Robson Paul, and Korth Martin - Pan Docs (previous versions and revisions)
- Jeff Frohwein - DMG, SGB, MBC schematics
- Pat Fagan - z80gboy.txt