BameGoy Design Document

Nicolas Alarcon (na2946) , Donovan Sproule (das2313), Claire Cizdziel (ctc2156)

Spring 2024

Hardware:

Cartridge Data

- Cartridge headers located **$0100 – $014F** (detailed below)
- We will be using test ROM cartridges, specifically **Tetris** [7]

1. Resource Budgets

   Memory takes up 64 KiB

ROM Bank	16KiB
Switchable ROM	16KiB
VRAM	8KiB
External RAM	8KiB
WRAM	4KiB
OAM	160 bytes
High RAM	127 bytes
IE	1 bytes

2. The Hardware/Software Interface

   Byte usage

$0000 – $7FFF : Cartridge ROM	Includes game data and internal work RAM, VRAM	32KiB

IE: 5 bits

Joypad: 4 bits

SB: 8 bits

SC: 8 bits

TAC: 3 bits

IF: 5 bits

LCDC: 8 bits

STAT: 8 bits

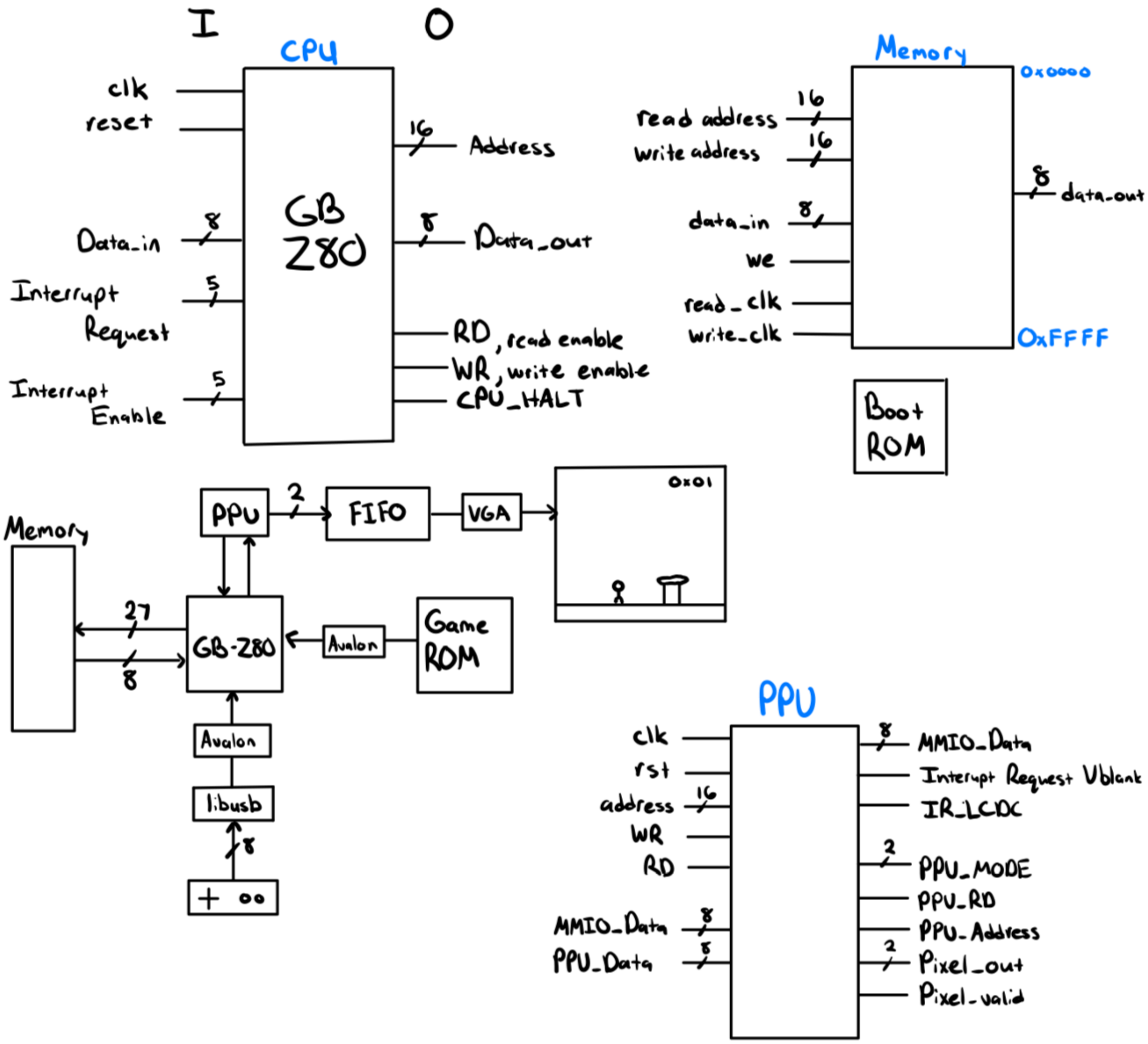
SCX: 8 bits

SCY: 8 bits

BGP: 8 bits

OAM: 8 bits

System Block Diagram and Overview



The BameGoy system will make use of the Z80 core to write and read from the 16-bit address space. The Game ROM will be loaded into memory via software. The PPU will send addresses and data to the Z80 to write and read from RAM to access VRAM and OAM, as well as the game information for x and y position of sprites and tiles. The PPU will then output 2 bits per pixel, which will be stored in a framebuffer and sent over VGA to the monitor. The controller interrupts and data will be read via libusb and sent over the Avalon bus via ioctl to be written to $FF00. We will be testing the system with test ROMs [7] and tetris, which does not require a Memory Bank Controller to operate.

We will not be implementing audio support or serial connection.

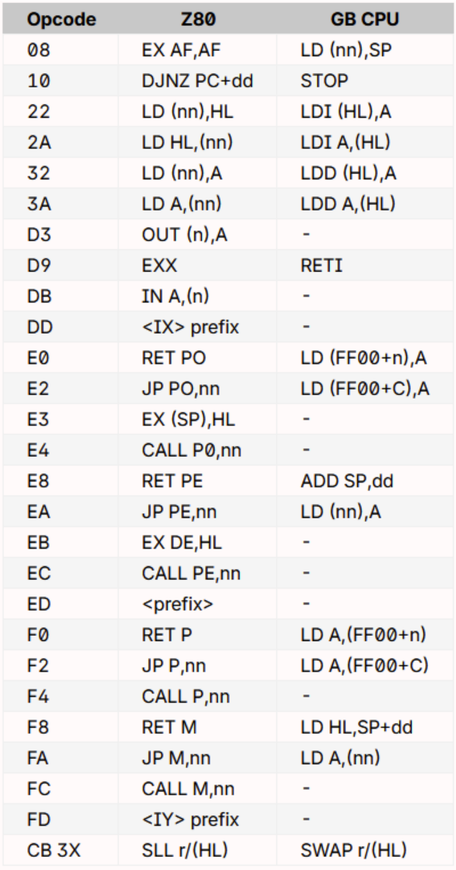
Hardware

BameGoy CPU

The original GameBoy CPU was a Sharp SM83 CPU, but we will be making use of a modified open-source Z80 that is compatible with the GameBoy instruction set. The sign, parity/overflow flags have been removed, as have the instructions dependent on them. All DD-, ED-, and FD- prefixed instructions, as well as the IX- and IY- registers have been removed. Block instructions are gone, as well as most 16 bit operations, but auto incrementing HL accesses have been added.

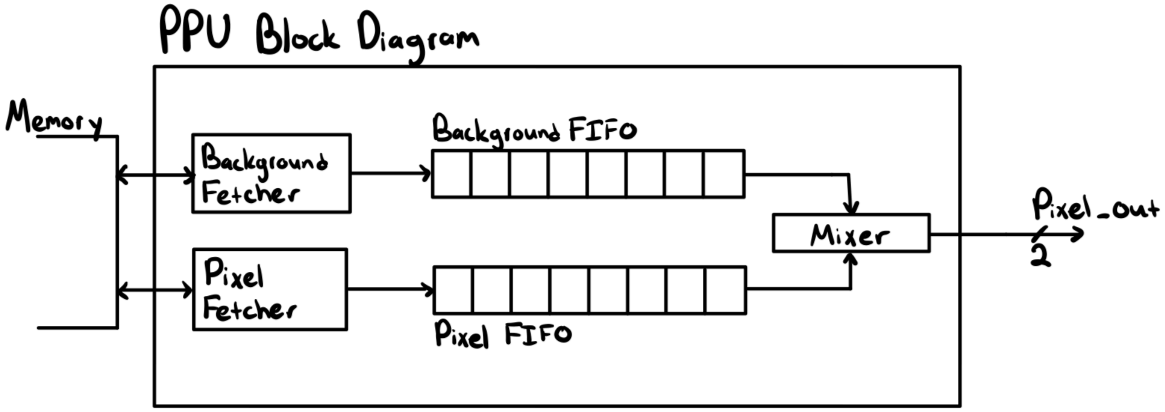
The Game Boy operates about as fast as a 4MHz Z80, and all instructions take multiples of 4 cycles to complete. For a list of the GB-Z80 ISA refer to [6].

Moved, Removed, and Added Opcodes [4]



Opcode	Z80	GB CPU
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> prefix	-
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	<prefix>	-
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> prefix	-
CB 3X	SLL r/(HL)	SWAP r/(HL)

BameGoy PPU



The gameboy's screen output data is handled almost exclusively by the Pixel Processing Unit (PPU). It uses a 32x32 tile system featuring two overlays, the screen and the window. The window is 32x32 tiles but the screen which is what actually gets displayed at a given time is 20x18.

Tiles

Each tile is 8x8 pixels making the total resolution of the original gameboy 160x144. The tiles are stored in memory in registers ($8000-$97FF) as a series of rows, each row being 2 bytes long. The two bytes are merged together and the binary number value of the corresponding index encodes the color, allowing for the ability to use 4 separate colors.

Operation

At a very high level, the PPU operates using scanlines where it draws a row at a time starting at the top left corner until it fills up the screen; the current scanline is stored in the register LY. It takes these pixels from 2 shift registers, the Background and Pixel FIFO. Each register can only hold data about 8 pixels at a time, and are populated using their respective fetching routines. The process runs in the background during the 3 modes of operation of the PPU.

Modes

(Mode 2) The first mode the PPU does is the OAM scan. Games hold sprite graphics in tiles, as a sprite becomes relevant to the scene it is placed in the Object Attribute Memory (OAM) ($FE00-$FE9F). These registers can only hold up to 40 sprites, only 10 of which can be drawn at a time. Various information about location, the tile number for graphics and other flags are held here (see OAM Sprite Memory) . During the OAM scan the PPU will traverse a scanline looking for collisions with a sprite in the OAM. If it finds a sprite it marks it and holds it in another special register.

(Mode 3) The next mode is for drawing. This is where the PPU runs the fetching steps, pulls from the shift registers and draws output to the screen.

(Mode 0) This stage is called the h-blank. It pads the end of lines to use up extra clock cycles and the PPU effectively pauses during blank modes.

(Mode 1) Similar to the stage above but this pads the PPU in the vertical direction for 10 pseudo-scanlines..

Fetching

As stated before, each FIFO can hold up to 8 pixels at a time, storing information about color, palette, priority and background priority. These are populated as a queue, per scan line. One FIFO is used for information on the sprites and another on the background. The basic operation for populating each FIFO is the fetch: The PPU will find the corresponding tile number for the graphic from the map data or sprite data, grab the tile data and put it into the FIFO. In order to set which tiles should be displayed in the Background / Window grids, background maps are used. The VRAM sections $9800-$9BFF and $9C00-$9FFF each contain one of these background maps. A background map consists of 32x32 bytes representing tile numbers organized row by row.

This process consists of 4 steps each consisting of 2 cycles: fetching tile number, fetch first byte of tile data, fetch second byte of tile data, and finally push to FIFO. The background fifo can only be written to if it is empty. The tile number is fetched from VRAM or OAM, depending on whether it is background or a sprite, the address of which is taken from the current position on the screen.

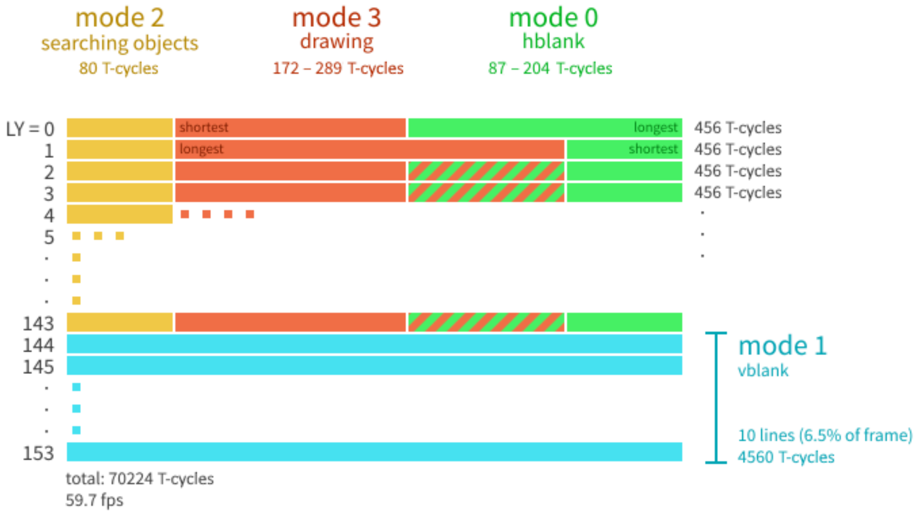
Pixel Mixing

Every cycle, the PPU attempts to push to the LCD. If there is a pixel in the background FIFO, it will do so. If not, it will wait.

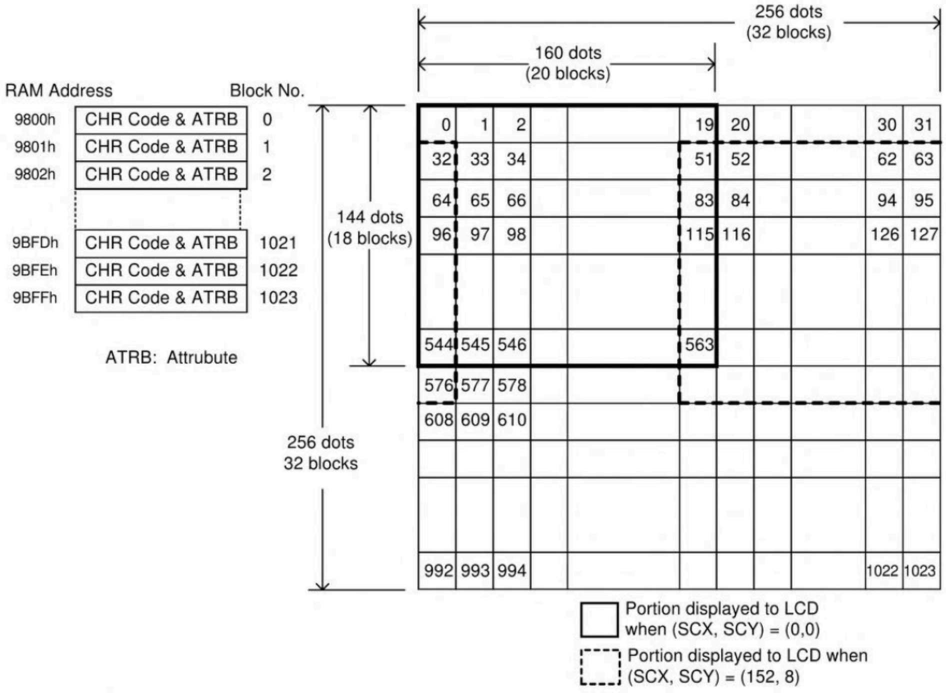
If both FIFOs have pixels in them when rendering data, both will be shifted out. The background pixel will be output to the screen if the sprite color is 0 or if the bg-to-obj priority is high and the background pixel color is not equal to 0. Otherwise, the sprite pixel is pushed.

The SCX register is defined for background scrolling, the first SCX mod 8 pixels at the start of a scanline will be discarded, extending PPU mode 3 by SCX mod 8 cycles.

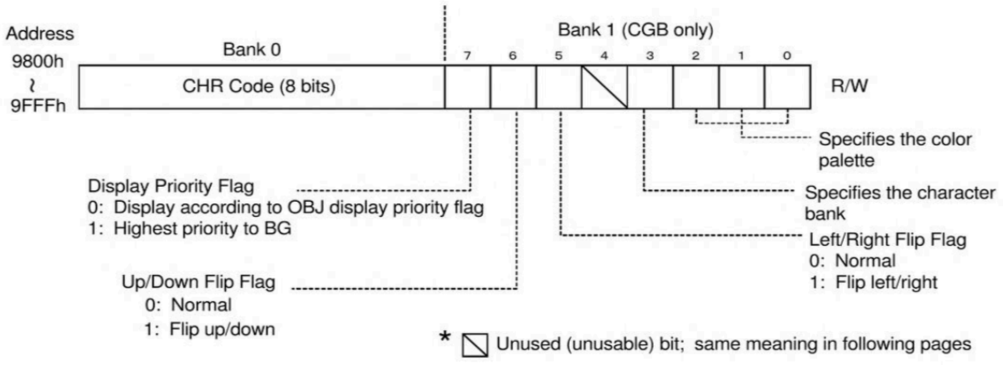
Simplified Timing Diagram for the PPU



Video Output



[1]



[2]

Data for 32 x 32 character codes (256 x 256 dots) can be specified from 9800h or 9C00h as BG display data. Of these, data for 20 x 18 character codes (160 x 144 pixels) are displayed to the original GameBoy LCD screen. As VGA has a resolution of 640x480 pixels, we will be sending this data to a FIFO framebuffer on the VGA module 160 pixels long to be displayed at a 3:1 ratio to the original, for an LCD viewport of 480x432 pixels centered on the monitor. For this, we will be modifying the VGA counters module used in lab 3. For further technical data, see page 10.

Software

To simulate the memory bank swapping we are going to map the BameGoy memory in the virtual address space of our C program. Here we will load in the game file and post chunks at a time to ROM simulate the swapping mechanism in the hardware of the original gameboy.

We will also use the software to read in controller input from our USB NES controller. On an interrupt, the controllers send a singular byte mapping each of its 8 possible inputs to a bit. When we receive this, we will send an interrupt flag to the BameGoy's cpu and update the joypad register ($FF00) with the appropriate values.

These operations will be happening in multiple threads so they can all occur simultaneously.

Technical Data

CPU: GB-Z80

Registers

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

The CPU registers can be accessed as a 16 bit register or as two 8 bit registers.

Flags Register

Bit	Name	Description
7	z	Zero flag
6	n	Subtraction flag (BCD)
5	h	Half Carry flag (BCD)
4	c	Carry flag

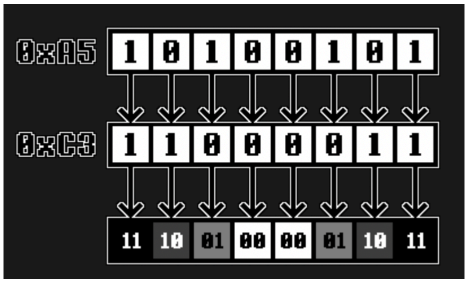
RAM: 8kiB (DMG), 32 kiB (CGB)

Video:

- 160x144 pixels (20 x 18 tiles) → **480x432 pixels in a window of 640x480 (VGA)**
- 2 backgrounds: main and window (for scrolling)
- Max sprites: 40 → max 10 per scanline
- 8×8 pixel tiles
- 456 cycles per line
- 70224 cycles per frame
- VDraw: 144 lines, 65664 cycles
- VBlank: 10 lines, 4560 cycles
- Clock Speed: 4.194304 MHz (2^22 Hz)
- Horizontal Sync: 9198 KHz
- Vertical Sync: 59.73 Hz

Pixel Data:

- 2BPP format - 2 bits per pixel (2 bits to store color data)
- 2 bytes → 8 pixels
- (Each bit of first byte) + (bit in same position on second byte) => calculates color number



[3]

Tile Data:

- Graphics encoded in 2BPP format, stored in VRAM $8000-$97FF
- “Tile Numbers” 16-byte-block in section of VRAM
- Addressing methods: 8000 and 8800
- 8000 Method:
    - $8000: base pointer
    - TILE_NUMBER: unsigned 8-bit integer

    - Add $8000 to (TILE_NUMBER * 16)
- 8800 Method:
    - $9000: base pointer
    - SIGNED_TILE_NUMBER: signed 8-bit integer
    - Add $9000 to (TILE_NUMBER * 16)
- We will use 8000 Method to directly map the tile number to memory address.

Cartridge Header

Cartridge headers located at **$0100 – $014F:**

Entry point	**$0100 – $0103**
Nintendo logo	**$0104 – $0133**
Title	**$0134 – $0143**
Manufacturer code	**$013F – $0142**
CGB - Enable color mode or monochrome compatibility mode - $80: game supports CGB enhancements, backwards compatible with monochrome	**$0143**
New licensee code - Only meaningful if $33 (which our game will be)	**$0144 – $0145**
SGB flag - Ignored if byte set to value other than $03	**$0146**
Cartridge type - Byte indicates the kind of hardware present on the cartridge - esp. Its mapper	**$0147**
ROM size - Typically given by 32 KiB × (1 << <value>)	**$0148**
RAM size - If cartridge type doesn't include “RAM” in name, set to 0	**$0149**
Destination code - $00: Japan - $01: Overseas only	**$014A**
Old licensee code - Not consider – our new licensee code $33 will override	**$014B**
Mask ROM version num	**$014C**
Header checksum - If byte at $014D doesn’t match the lower 8 bits of checksum, boot ROM will lock and the program in the cartridge won’t run	**$014D**
Global checksum	**$014E – $014F**

Memory Map (64 KiB resource budget)

IE register, interrupt enabler	**$FFFF**
External bus (2x 16 KiB ROM regions)	**$0000 – $7FFF**
VRAM (8 KiB)	**$8000 – $9FFF**
External bus (8 KiB RAM region)	**$A000 – $BFFF**
WRAM (4 KiB Work RAM)	**$C000 – $DFFF**
ECHO (WRAM secondary mapping, **not emulated**)	**$E000 – $FDFF**
Object Attribute Memory (OAM), Sprite use	**$FE00 – $FE9F**
High RAM (HRAM)	**$FF80 – $FFFE**
Memory mapped I/O	**$FF00 – $FF7F**
Joypad input	**$FF00**
SB: Serial Byte	**$FF01**
SC: Serial Control	**$FF02**
DIV: Clock Divider	**$FF04**
TIMA: Timer Value	**$FF05**
TMA: Timer Reload	**$FF06**
TAC: Timer Control	**$FF07**
IF: Interrupts asserted	**$FF0F**
Audio channels - unused for this project	**$FF10 – $FF26**
LCDC: LCD control	**$FF40**
STAT: LCD status	**$FF41**
SCY: Background vert. scroll	**$FF42**
SCX: Background horiz. scroll	**$FF43**
LY: LCD Y coordinate	**$FF44**
LYC: LCD Y compare	**$FF45**
DMA: OAM DMA source address	**$FF46**
BGP: Background palette	**$FF47**
OBP0: OBJ palette 0	**$FF48**
OBP1: OBJ palette 1	**$FF49**
WY: Window Y coord	**$FF4A**

WX: Window X coord	**$FF4B**
Boot ROM control	**$FF50**

Mapping Key:

- 1: Unmapped (hi-Z), always reads 1
- I: Input only (readable by CPU)
- O: Output only (writable by CPU)
- B: Bidirectional (read/write by CPU)

Memory Map - Extended Descriptions

IE: Interrupt Enabler **$FFFF**

0: disable
1: enable

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Transition from high to low of Pin #P10-P13	Serial I/O transfer complete	Timer overflow	LCDC	V-Blank

Joypad input **$FF00**

Keys pulled low (to 0) when press and high (to 1) when unpressed

- If bits 4 and 5 both low, then combination of D-pad and face buttons ANDed
- If neither bit 4 or 5 is selected, then all keys read 1

Bit	**4 low**	**5 low**
0	Right	A
1	Left	B
2	Up	Select
3	Down	Select

SB: Serial Byte **$FF01**

Serial transfer data (R/W)

8 Bits of data to be read/written

SC: Serial Control **$FF02**

SIO Control (RW)

Bit 7 - Transfer Start Flag

0: Non transfer

1: Start transfer

Bit 0 - Shift Clock

0: External Clock (500KHz Max.)

1: Internal Clock (8192Hz)

TAC: Timer Control **$FF07**

Timer control (R/W)

Bit 2 - Timer Stop

0: Stop Timer

1: Start Timer

Bits 1+0 - Input Clock Select

0: 4.096 KHz (~4.194 KHz SGB)

1: 262.144 KHz (~268.4 KHz SGB)

2: 65.536 KHz (~67.11 KHz SGB)

3: 16.384 KHz (~16.78 KHz SGB)

IF: Interrupts asserted **$FF0F**

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Transition from high to low of Pin #P10-P13	Serial I/O transfer complete	Timer overflow	LCDC	V-Blank

LCDC: LCD Control Register **$FF40**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LCD Display Enable	Window Tile Map Select	Window Display Enable	Tile Data Select	BG Tile Map Select	Sprite Size	Sprite Enable	BG/Window Enable

Bit 7 : Enables screens - if set to 0, disables PPU completely

0: Stop completely (no picture on screen)

1: Operation

Bit 6: Uses $9800-$9BFF - if set to 1, sets Window to use background map at $9C00-$9FFF

0: $9800-$9BFF

1: $9C00-$9FFF

Bit 5: Enables display - if set to 0, hides window layer completely

0: off

1: on

Bit 4: Uses 8800 method for tile select - if set to 1, fetches Tile Data using 8000 method

0: $8800-$97FF

1: $8000-$8FFF <- Same area as OBJ

Bit 3: Uses $9800-$9BFF - if set to 1, Background will use background map at $9C00-$9FFF

0: $9800-$9BFF

1: $9C00-$9FFF

Bit 2: 1x1 Tile Sprite size - if set to 1, sprites displayed as 1x2 Tiles (8x16 pixel)

0: 8*8

1: 8*16 (width*height)

Bit 1: Sprites only drawn if set to 1

0: off

1: on

Bit 0: Sprites unaffected - if set to 0, neither Background nor Window tiles are drawn

0: off

1: on

STAT: LCD Status Register $FF41

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Unused (Always 1)	LYC=LY STAT Interrupt Enable	Mode 2 STAT Interrupt Enable	Mode 1 STAT Interrupt Enable	Mode 0 STAT Interrupt Enable	Coincidence Flag	PPU Mode	PPU Mode

Bit 7 : Unused

Bit 6: If set to 1, enable “LYC=LY condition” to trigger STAT interrupt

Bit 5-3: “Condition enablers” for STAT interrupt - trigger when PPU enters specific mode

Bit	PPU Mode
STAT.5	Mode 2 (OAM Scan)
STAT.4	Mode 1 (VBlank)
STAT.3	Mode 0 (HBlank)

Bit 2: Set by PPU if value of LY register = LYC register

0: LYC ≠ LCDC LY

1: LYC = LCDC LY

Bit 1-0: Set by PPU depending on mode

0 : H-Blank

1 : V-Blank

2 : OAM Scan

3 : Drawing, Transferring Data to LCD Driver

SCX: Background horiz. scroll $FF43

8 bit value $00-$FF to scroll BG X screen position

SCY: Background vert. scroll $FF42

8 bit value $00-$FF to scroll BG Y screen position

BGP: Background palette **$FF47**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data for Dot Data 11 (~ darkest color)		Data for Dot Data 10		Data for Dot data 01		Data for Dot Data 00 (~ lightest color)	

OAM Sprite Memory **$FE00 – $FE9F**

Byte 0: Y-Position

Actual Y position on screen = Byte 0 - 16

Byte 1 - X-Position:

Actual X position on screen = Byte 1 - 8

Byte 2 - Tile Number:

Unsigned 8 bit integer used for Sprite fetching.

Byte 3 - Sprite Flags:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OBJ-to-BG Priority	Y-Flip	X-Flip	Palette Number	Mode 0 STAT Interrupt Enable	Coincidence Flag	PPU Mode	PPU Mode

Bit 7: OBJ-to-BG Priority

0 = Sprite is always rendered above background

1 = Background colors 1-3 overlay sprite, sprite is still rendered above color 0

Bit 6: Y-Flip
If set to 1 the sprite is flipped vertically, otherwise rendered as normal

Bit 5: X-Flip
If set to 1 the sprite is flipped horizontally, otherwise rendered as normal

Bit 4: Palette Number
If set to 0, the OBP0 register is used as the palette, otherwise OBP1

Bit 3-0: CGB-Only flags

Resources:

[1] Video output diagram - screen

https://archive.org/details/GameBoyProgManVer1.1/page/n53/mode/2up (p.57)

[2] Video output diagram - banks

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=4#contents

[3] PPU

https://hacktix.github.io/GBEDG/ppu/#an-introduction

[4] Moved, Removed, and Added Opcodes

https://gbdev.io/pandocs/CPU_Comparison_with_Z80.html

[5] Z80 manual

https://www.zilog.com/docs/z80/um0080.pdf

[6] GB- Z80 instruction set

https://rgbds.gbdev.io/docs/v0.7.0/gbz80.7

[7] Test ROMS

https://gbdev.gg8.se/files/roms/blargg-gb-tests/