Emulation

Takanen Edoardo

March 25, 2025

Abstract

Contents

1 Introduction

As a kid, I used to play with some Nintendo (copyright symbol) consoles like the Wii or the DS and I've always been keen about the games they make. This passion for videogames grew on me so that I got interested in the making process of them, leading to game development. I never asked myself one question, though, until this year, which is how are these games able to run onto this consoles? and how can people make emulators so that I could play on my personal computer? Thus, I decided to embrace the unknown world of emulation, because I have always been fascinated by it and always took it for granted. Emulation is not well explained on the Internet. Mainly, the results you will find if you search for it are "the program pretends to be the console" or "you will be able to play old titles". I was not satisfied with these responses, I wanted to know more. Thus, my emulation journey started with looking for a full definition of this process. I will try to give my own definition of emulation, so that I can lay a starting point to a general knowledge that will be then deepened during the paper. With this being said, to "make an emulator" means to develop the software that will do exactly what the hardware of the console does, so that when plugging the game data, the program will know how to read and handle it.
Emulation can only happen when the machine in which we run the software is more powerful than the hardware we want to emulate. For example, if our console has 2Kb of memory, for sure we are not able to emulate it on a computer that has 2Kb or less, since we also have to consider that the host computer will have an operating system running (which uses some of the RAM). I chose to make a Game Boy emulator, because while looking for the retro consoles, it seemed the least difficult when talking about the hardware structure complexity, meaning a good way to start tackling this subject.

2 Premises

This emulator project is purely for understanding the concepts and the theory about how a machine like a console (or similarly a computer) is made (and also for fun). There are many better-developed Game Boy emulators online, and making one that could compete with the other popular ones is nowhere near my goals. Therefore, I could not achieve the level of knowledge I want to reach if I just looked at other people's codes, I want to **fully** understand the subject. Obviously though I have to start somewhere, I do not have the skills to reverse-engineer a real Game Boy (although it would be an extremely interesting challenge), for this reason I will only consult theory guides made by many passionate developers and hackers that already did the work of studying the Game Boy from scratch for us. For a better understanding, I used two sources for this project, in order to have a dual perspective on the study.
For anyone who would like to dig into this challenge, the guides are GBDev and GBEDG.

What this paper is not.
This paper is not and was not intended as a guide, I previously attached some real references. This document is a report of my journey throught the development of the emulator, made for understanding the fundamentals of what is around us, from personal computers to smartphones. It could also be a way for readers to get passionate about this topic and an inspiration for them to make their own emulators (or even better, their own consoles!).

3 General Structure

The first thing I want to cover is in what way we want to structure our emulator.

```
int main() {
    // Hardware components definition
    Memory mem;
    CPU cpu;
    // ...

    // Components initialization
    mem.init();
    cpu.init();
    cpu.load_boot();
    // ...

    while (true) {
        cpu.exec_op(mem); // Executing an operation
        // emulate all the other components
    }

    return 0;
}
```

Actually, when we look at the circuit inside the Game Boy, all the components are, on one side, all on their own, they all execute at the same time. The CPU could be executing a simple addition, while the PPU could be rendering graphics onto the screen, all of these things happen simultaneously. This **can** be done with software, but would mean more complexity. Hence we will pick a less complicated path, and decide to execute the components one at a time.

```
while (true) {
    // Returns how many clock cycles the instruction took
    int cycles = cpu.exec_op();

    // Updating all the other components
    timers.update(cycles);
    ppu.update(cycles);
    // ...
}
```

The real hardware is driven by the clock, while my implementation will be driven by how many clock cycles an instruction took.

4 Memory

Before looking at the main components that shape the Game Boy hardware, I would like to focus on how memory is subdivided inside the console.

4.1 Memory mapping

The address bus had 16 bits, meaning there could be 65'536 unique addresses (64Kb).
Since the Game Boy did not have a flash memory, those 64Kb were all the console could access (this includes all the different RAMs, the cartridge data, and the registers made to control various components). Internally inside the Game Boy, there is some logic that specifies what component will be activated based on the requesting address.
These are the regions into which the memory is split, along with a brief description of their use. Notice that some areas are marked as "Prohibited", though Nintendo has not provided an explanation for this.

Start	End	Description
0000	3FFF	16Kb cartridge ROM
4000	7FFF	16Kb cartridge ROM*
8000	9FFF	8Kb VRAM
A000	BFFF	8Kb External RAM
C000	CFFF	4Kb Work RAM
D000	DFFF	4Kb Work RAM
E000	FDFF	Prohibited area
FE00	FE9F	OAM
FEA0	FEFF	Prohibited area
FF00	FF7F	I/O Registers
FF80	FFFE	High RAM
FFFF	FFFF	IE register

*switchable

Table 1: Game Boy's memory mapping

Another interesting fact I found out when making this emulator is what is inside a cartridge. It is fascinating to know that some cartridges would not only include the ROM banks with the game code, but they could also supply their own additional SRAM (Static Random-Access Memory), as well as a battery to preserve the game saves. Due to limiting memory sizes, games could also have a Memory Bank Controller (MBC) to change what ROM should be pointed for memory region $4000 - 7FFF$.
Notice that all these additional components are **not** supported on my emulator, since the original Tetris DMG cartridge just had a 32 Kb ROM.

4.2 Choices for this project

For simplicity, I decided not to break down all these areas into different regions of memory in the emulator, but I opted for an easier solution, which is to create an array of 65'536 bytes, since the data bus was 8 bits-long, so every address will have exacly one byte of data.
There is a trade-off in choosing this approach tough. On one hand, it makes things simpler to manage, we can have all the memory in one place and it really helps when debugging, but on the other hand, it is not entirely correct. Each region of memory has different restrictions, cartridge memory should be read-only, some areas might not be fully readable and writable sometimes (we will see an example when implementing the PPU).
By choosing this option, we are making all the memory readable and writable for everyone, so tecnically, the game could edit its own code (and this actually happened when I was emulating Tetris!).

4.3 Implementation

```
struct Memory {
private:
    static constexpr u32 MAX_MEM = 64 * 1024;
    // Array of bytes to emulate all the Gameboy's addresses
    Byte Data[MAX_MEM] = {};
public:
    void init();

    /**
     * Functions used for setting and accessing memory as
     * mem[addr] = value         to set
     * Byte value = mem[addr]    to access
     * */
    Byte operator[](u32 addr) const;
    Byte& operator[](u32 addr);

    /**
     * Dumps all the memory in a file,
     * used for debugging purposes
     * */
    void dump(const char* filename);
};
```

This is the entire memory structure. Every component we create will have access to this structure in order to read from and write to memory using the two operator[] functions. I have also added a **dump** function that writes all the bytes to a binary file at the moment the function is called, for easier debugging.

5 CPU

The first component we likely want to implement is the Central Processing Unit (CPU). This component is the most important in our circuit, and is the one that coordinates the other components, executes the program we give to it etc. Thus, the first thing I had to implement were all the instructions that the processor could run.

5.1 Architecture and considerations

The Game Boy's CPU is a custom-made by Sharp Corporation (which had a close relationship with Nintendo at that time), it is often referred to as **DMG-CPU** or **Sharp SM83** and runs at around 4.19 MHz. When making the processor a lot of inspiration was taken from the Zilog 80 and the Intel 8080. Personally, I recently had the possibility to work with a real Z80, and over the past few months, I have gained hands-on experience with its architecture. Specifically, when studying the Zilog, I noticed some differences and similarities with the Game Boy's processor.
For example, the Nintendo processor lacks the IX and IY registers, which in the Zilog were used to set a base address that could be offset with the $LD(IX+d), r$ and $LD(IY+d), r$ to save instruction bytes. Instead, the DMG-CPU introduced a brand new load instruction, LDH (load from high memory), which always offsets from address **FF00**, pointing to **High-RAM** and the **I/O registers**.
I think custom-making their own CPU was the perfect choice for Nintendo, as it allowed them to implement changes like these to better suit their needs, save instruction bytes, and increase performace.

5.2 The op-tables

The list of all instructions is usually arranged in tables, called opcode-tables.
The Game Boy has 2 op-tables.

Game Boy CPU Instructions

8-bit opcodes

Game Boy CPU instructions, organized by opcode

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 1 - - - -	LD BC, d16 3 3 - - - -	LD (BC), A 1 2 - - - -	INC BC 1 2 - - - -	INC B 1 1 Z 0 H -	DEC B 1 1 Z 1 H -	LD B, d8 2 2 - - - -	RLCA 1 1 0 0 0 A7	LD (a16), SP 3 5 - - - -	ADD HL, BC 1 2 - 0 H CY	LD A, (BC) 1 2 - - - -	DEC BC 1 2 - - - -	INC C 1 1 Z 0 H -	DEC C 1 1 Z 1 H -	LD C, d8 2 2 - - - -	RRCA 1 1 0 0 0 A0
1x	STOP 2 1 - - - -	LD DE, d16 3 3 - - - -	LD (DE), A 1 2 - - - -	INC DE 1 2 - - - -	INC D 1 1 Z 0 H -	DEC D 1 1 Z 1 H -	LD D, d8 2 2 - - - -	RLA 1 1 0 0 0 A7	JR s8 2 3 - - - -	ADD HL, DE 1 2 - 0 H CY	LD A, (DE) 1 2 - - - -	DEC DE 1 2 - - - -	INC E 1 1 Z 0 H -	DEC E 1 1 Z 1 H -	LD E, d8 2 2 - - - -	RRA 1 1 0 0 0 A0
2x	JR NZ, s8 2 3/2 - - - -	LD HL, d16 3 3 - - - -	LD (HL+), A 1 2 - - - -	INC HL 1 2 - - - -	INC H 1 1 Z 0 H -	DEC H 1 1 Z 1 H -	LD H, d8 2 2 - - - -	DAA 1 1 Z - 0 CY	JR Z, s8 2 3/2 - - - -	ADD HL, HL 1 2 - 0 H CY	LD A, (HL+) 1 2 - - - -	DEC HL 1 2 - - - -	INC L 1 1 Z 0 H -	DEC L 1 1 Z 1 H -	LD L, d8 2 2 - - - -	CPL 1 1 - 1 1 -
3x	JR NC, s8 2 3/2 - - - -	LD SP, d16 3 3 - - - -	LD (HL-), A 1 2 - - - -	INC SP 1 2 - - - -	INC (HL) 1 3 Z 0 H -	DEC (HL) 1 3 Z 1 H -	LD (HL), d8 2 3 - - - -	SCF 1 1 - 0 0 1	JR C, s8 2 3/2 - - - -	ADD HL, SP 1 2 - 0 H CY	LD A, (HL-) 1 2 - - - -	DEC SP 1 2 - - - -	INC A 1 1 Z 0 H -	DEC A 1 1 Z 1 H -	LD A, d8 2 2 - - - -	CCF 1 1 - 0 0 !CY
4x	LD B, B 1 1 - - - -	LD B, C 1 1 - - - -	LD B, D 1 1 - - - -	LD B, E 1 1 - - - -	LD B, H 1 1 - - - -	LD B, L 1 1 - - - -	LD B, (HL) 1 2 - - - -	LD B, A 1 1 - - - -	LD C, B 1 1 - - - -	LD C, C 1 1 - - - -	LD C, D 1 1 - - - -	LD C, E 1 1 - - - -	LD C, H 1 1 - - - -	LD C, L 1 1 - - - -	LD C, (HL) 1 2 - - - -	LD C, A 1 1 - - - -
5x	LD D, B 1 1 - - - -	LD D, C 1 1 - - - -	LD D, D 1 1 - - - -	LD D, E 1 1 - - - -	LD D, H 1 1 - - - -	LD D, L 1 1 - - - -	LD D, (HL) 1 2 - - - -	LD D, A 1 1 - - - -	LD E, B 1 1 - - - -	LD E, C 1 1 - - - -	LD E, D 1 1 - - - -	LD E, E 1 1 - - - -	LD E, H 1 1 - - - -	LD E, L 1 1 - - - -	LD E, (HL) 1 2 - - - -	LD E, A 1 1 - - - -
6x	LD H, B 1 1 - - - -	LD H, C 1 1 - - - -	LD H, D 1 1 - - - -	LD H, E 1 1 - - - -	LD H, H 1 1 - - - -	LD H, L 1 1 - - - -	LD H, (HL) 1 2 - - - -	LD H, A 1 1 - - - -	LD L, B 1 1 - - - -	LD L, C 1 1 - - - -	LD L, D 1 1 - - - -	LD L, E 1 1 - - - -	LD L, H 1 1 - - - -	LD L, L 1 1 - - - -	LD L, (HL) 1 2 - - - -	LD L, A 1 1 - - - -
7x	LD (HL), B 1 2 - - - -	LD (HL), C 1 2 - - - -	LD (HL), D 1 2 - - - -	LD (HL), E 1 2 - - - -	LD (HL), H 1 2 - - - -	LD (HL), L 1 2 - - - -	HALT 1 1 - - - -	LD (HL), A 1 2 - - - -	LD A, B 1 1 - - - -	LD A, C 1 1 - - - -	LD A, D 1 1 - - - -	LD A, E 1 1 - - - -	LD A, H 1 1 - - - -	LD A, L 1 1 - - - -	LD A, (HL) 1 2 - - - -	LD A, A 1 1 - - - -
8x	ADD A, B 1 1 Z 0 H CY	ADD A, C 1 1 Z 0 H CY	ADD A, D 1 1 Z 0 H CY	ADD A, E 1 1 Z 0 H CY	ADD A, H 1 1 Z 0 H CY	ADD A, L 1 1 Z 0 H CY	ADD A, (HL) 1 2 Z 0 H CY	ADD A, A 1 1 Z 0 H CY	ADC A, B 1 1 Z 0 H CY	ADC A, C 1 1 Z 0 H CY	ADC A, D 1 1 Z 0 H CY	ADC A, E 1 1 Z 0 H CY	ADC A, H 1 1 Z 0 H CY	ADC A, L 1 1 Z 0 H CY	ADC A, (HL) 1 2 Z 0 H CY	ADC A, A 1 1 Z 0 H CY
9x	SUB B 1 1 Z 1 H CY	SUB C 1 1 Z 1 H CY	SUB D 1 1 Z 1 H CY	SUB E 1 1 Z 1 H CY	SUB H 1 1 Z 1 H CY	SUB L 1 1 Z 1 H CY	SUB (HL) 1 2 Z 1 H CY	SUB A 1 1 Z 1 H CY	SBC A, B 1 1 Z 1 H CY	SBC A, C 1 1 Z 1 H CY	SBC A, D 1 1 Z 1 H CY	SBC A, E 1 1 Z 1 H CY	SBC A, H 1 1 Z 1 H CY	SBC A, L 1 1 Z 1 H CY	SBC A, (HL) 1 2 Z 1 H CY	SBC A, A 1 1 Z 1 H CY
Ax	AND B 1 1 Z 0 1 0	AND C 1 1 Z 0 1 0	AND D 1 1 Z 0 1 0	AND E 1 1 Z 0 1 0	AND H 1 1 Z 0 1 0	AND L 1 1 Z 0 1 0	AND (HL) 1 2 Z 0 1 0	AND A 1 1 Z 0 1 0	XOR B 1 1 Z 0 0 0	XOR C 1 1 Z 0 0 0	XOR D 1 1 Z 0 0 0	XOR E 1 1 Z 0 0 0	XOR H 1 1 Z 0 0 0	XOR L 1 1 Z 0 0 0	XOR (HL) 1 2 Z 0 0 0	XOR A 1 1 Z 0 0 0
Bx	OR B 1 1 Z 0 0 0	OR C 1 1 Z 0 0 0	OR D 1 1 Z 0 0 0	OR E 1 1 Z 0 0 0	OR H 1 1 Z 0 0 0	OR L 1 1 Z 0 0 0	OR (HL) 1 2 Z 0 0 0	OR A 1 1 Z 0 0 0	CP B 1 1 Z 1 H CY	CP C 1 1 Z 1 H CY	CP D 1 1 Z 1 H CY	CP E 1 1 Z 1 H CY	CP H 1 1 Z 1 H CY	CP L 1 1 Z 1 H CY	CP (HL) 1 2 Z 1 H CY	CP A 1 1 Z 1 H CY
Cx	RET NZ 1 5/2 - - - -	POP BC 1 3 - - - -	JP NZ, a16 3 4/3 - - - -	JP a16 3 4 - - - -	CALL NZ, a16 3 6/3 - - - -	PUSH BC 1 4 - - - -	ADD A, d8 2 2 Z 0 H CY	RST 0 1 4 - - - -	RET Z 1 5/2 - - - -	RET 1 4 - - - -	JP Z, a16 3 4/3 - - - -		CALL Z, a16 3 6/3 - - - -	CALL a16 3 6 - - - -	ADC A, d8 2 2 Z 0 H CY	RST 1 1 4 - - - -
Dx	RET NC 1 5/2 - - - -	POP DE 1 3 - - - -	JP NC, a16 3 4/3 - - - -		CALL NC, a16 3 6/3 - - - -	PUSH DE 1 4 - - - -	SUB d8 2 2 Z 1 H CY	RST 2 1 4 - - - -	RET C 1 5/2 - - - -	RETI 1 4 - - - -	JP C, a16 3 4/3 - - - -		CALL C, a16 3 6/3 - - - -		SBC A, d8 2 2 Z 1 H CY	RST 3 1 4 - - - -
Ex	LD (a8), A 2 3 - - - -	POP HL 1 3 - - - -	LD (C), A 1 2 - - - -			PUSH HL 1 4 - - - -	AND d8 2 2 Z 0 1 0	RST 4 1 4 - - - -	ADD SP, s8 2 4 0 0 H CY	JP HL 1 1 - - - -	LD (a16), A 3 4 - - - -				XOR d8 2 2 Z 0 0 0	RST 5 1 4 - - - -
Fx	LD A, (a8) 2 3 - - - -	POP AF 1 3 - - - -	LD A, (C) 1 2 - - - -	DI 1 1 - - - -		PUSH AF 1 4 - - - -	OR d8 2 2 Z 0 0 0	RST 6 1 4 - - - -	LD HL, SP+s8 2 3 0 0 H CY	LD SP, HL 1 2 - - - -	LD A, (a16) 3 4 - - - -	EI 1 1 - - - -			CP d8 2 2 Z 1 H CY	RST 7 1 4 - - - -

16-bit opcodes, where the first 8 bits are 0xCB

Game Boy CPU instructions for opcodes prefixed by "CB"

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 2 Z 0 0 B7	RLC C 2 2 Z 0 0 C7	RLC D 2 2 Z 0 0 D7	RLC E 2 2 Z 0 0 E7	RLC H 2 2 Z 0 0 H7	RLC L 2 2 Z 0 0 L7	RLC (HL) 2 4 Z 0 0 (HL)7	RLC A 2 2 Z 0 0 A7	RRC B 2 2 Z 0 0 B0	RRC C 2 2 Z 0 0 C0	RRC D 2 2 Z 0 0 D0	RRC E 2 2 Z 0 0 E0	RRC H 2 2 Z 0 0 H0	RRC L 2 2 Z 0 0 L0	RRC (HL) 2 4 Z 0 0 (HL)0	RRC A 2 2 Z 0 0 A0
1x	RL B 2 2 Z 0 0 B7	RL C 2 2 Z 0 0 C7	RL D 2 2 Z 0 0 D7	RL E 2 2 Z 0 0 E7	RL H 2 2 Z 0 0 H7	RL L 2 2 Z 0 0 L7	RL (HL) 2 4 Z 0 0 (HL)7	RL A 2 2 Z 0 0 A7	RR B 2 2 Z 0 0 B0	RR C 2 2 Z 0 0 C0	RR D 2 2 Z 0 0 D0	RR E 2 2 Z 0 0 E0	RR H 2 2 Z 0 0 H0	RR L 2 2 Z 0 0 L0	RR (HL) 2 4 Z 0 0 (HL)0	RR A 2 2 Z 0 0 A0
2x	SLA B 2 2 Z 0 0 B7	SLA C 2 2 Z 0 0 C7	SLA D 2 2 Z 0 0 D7	SLA E 2 2 Z 0 0 E7	SLA H 2 2 Z 0 0 H7	SLA L 2 2 Z 0 0 L7	SLA (HL) 2 4 Z 0 0 (HL)7	SLA A 2 2 Z 0 0 A7	SRA B 2 2 Z 0 0 B0	SRA C 2 2 Z 0 0 C0	SRA D 2 2 Z 0 0 D0	SRA E 2 2 Z 0 0 E0	SRA H 2 2 Z 0 0 H0	SRA L 2 2 Z 0 0 L0	SRA (HL) 2 4 Z 0 0 (HL)0	SRA A 2 2 Z 0 0 A0
3x	SWAP B 2 2 Z 0 0 0	SWAP C 2 2 Z 0 0 0	SWAP D 2 2 Z 0 0 0	SWAP E 2 2 Z 0 0 0	SWAP H 2 2 Z 0 0 0	SWAP L 2 2 Z 0 0 0	SWAP (HL) 2 4 Z 0 0 0	SWAP A 2 2 Z 0 0 0	SRL B 2 2 Z 0 0 B0	SRL C 2 2 Z 0 0 C0	SRL D 2 2 Z 0 0 D0	SRL E 2 2 Z 0 0 E0	SRL H 2 2 Z 0 0 H0	SRL L 2 2 Z 0 0 L0	SRL (HL) 2 4 Z 0 0 (HL)0	SRL A 2 2 Z 0 0 A0
4x	BIT 0, B 2 2 !r0 0 1 -	BIT 0, C 2 2 !r0 0 1 -	BIT 0, D 2 2 !r0 0 1 -	BIT 0, E 2 2 !r0 0 1 -	BIT 0, H 2 2 !r0 0 1 -	BIT 0, L 2 2 !r0 0 1 -	BIT 0, (HL) 2 3 !(HL)0 0 1 -	BIT 0, A 2 2 !r0 0 1 -	BIT 1, B 2 2 !r1 0 1 -	BIT 1, C 2 2 !r1 0 1 -	BIT 1, D 2 2 !r1 0 1 -	BIT 1, E 2 2 !r1 0 1 -	BIT 1, H 2 2 !r1 0 1 -	BIT 1, L 2 2 !r1 0 1 -	BIT 1, (HL) 2 3 !(HL)1 0 1 -	BIT 1, A 2 2 !r1 0 1 -
5x	BIT 2, B 2 2 !r2 0 1 -	BIT 2, C 2 2 !r2 0 1 -	BIT 2, D 2 2 !r2 0 1 -	BIT 2, E 2 2 !r2 0 1 -	BIT 2, H 2 2 !r2 0 1 -	BIT 2, L 2 2 !r2 0 1 -	BIT 2, (HL) 2 3 !(HL)2 0 1 -	BIT 2, A 2 2 !r2 0 1 -	BIT 3, B 2 2 !r3 0 1 -	BIT 3, C 2 2 !r3 0 1 -	BIT 3, D 2 2 !r3 0 1 -	BIT 3, E 2 2 !r3 0 1 -	BIT 3, H 2 2 !r3 0 1 -	BIT 3, L 2 2 !r3 0 1 -	BIT 3, (HL) 2 3 !(HL)3 0 1 -	BIT 3, A 2 2 !r3 0 1 -
6x	BIT 4, B 2 2 !r4 0 1 -	BIT 4, C 2 2 !r4 0 1 -	BIT 4, D 2 2 !r4 0 1 -	BIT 4, E 2 2 !r4 0 1 -	BIT 4, H 2 2 !r4 0 1 -	BIT 4, L 2 2 !r4 0 1 -	BIT 4, (HL) 2 3 !(HL)4 0 1 -	BIT 4, A 2 2 !r4 0 1 -	BIT 5, B 2 2 !r5 0 1 -	BIT 5, C 2 2 !r5 0 1 -	BIT 5, D 2 2 !r5 0 1 -	BIT 5, E 2 2 !r5 0 1 -	BIT 5, H 2 2 !r5 0 1 -	BIT 5, L 2 2 !r5 0 1 -	BIT 5, (HL) 2 3 !(HL)5 0 1 -	BIT 5, A 2 2 !r5 0 1 -
7x	BIT 6, B 2 2 !r6 0 1 -	BIT 6, C 2 2 !r6 0 1 -	BIT 6, D 2 2 !r6 0 1 -	BIT 6, E 2 2 !r6 0 1 -	BIT 6, H 2 2 !r6 0 1 -	BIT 6, L 2 2 !r6 0 1 -	BIT 6, (HL) 2 3 !(HL)6 0 1 -	BIT 6, A 2 2 !r6 0 1 -	BIT 7, B 2 2 !r7 0 1 -	BIT 7, C 2 2 !r7 0 1 -	BIT 7, D 2 2 !r7 0 1 -	BIT 7, E 2 2 !r7 0 1 -	BIT 7, H 2 2 !r7 0 1 -	BIT 7, L 2 2 !r7 0 1 -	BIT 7, (HL) 2 3 !(HL)7 0 1 -	BIT 7, A 2 2 !r7 0 1 -
8x	RES 0, B 2 2 - - - -	RES 0, C 2 2 - - - -	RES 0, D 2 2 - - - -	RES 0, E 2 2 - - - -	RES 0, H 2 2 - - - -	RES 0, L 2 2 - - - -	RES 0, (HL) 2 4 - - - -	RES 0, A 2 2 - - - -	RES 1, B 2 2 - - - -	RES 1, C 2 2 - - - -	RES 1, D 2 2 - - - -	RES 1, E 2 2 - - - -	RES 1, H 2 2 - - - -	RES 1, L 2 2 - - - -	RES 1, (HL) 2 4 - - - -	RES 1, A 2 2 - - - -
9x	RES 2, B 2 2 - - - -	RES 2, C 2 2 - - - -	RES 2, D 2 2 - - - -	RES 2, E 2 2 - - - -	RES 2, H 2 2 - - - -	RES 2, L 2 2 - - - -	RES 2, (HL) 2 4 - - - -	RES 2, A 2 2 - - - -	RES 3, B 2 2 - - - -	RES 3, C 2 2 - - - -	RES 3, D 2 2 - - - -	RES 3, E 2 2 - - - -	RES 3, H 2 2 - - - -	RES 3, L 2 2 - - - -	RES 3, (HL) 2 4 - - - -	RES 3, A 2 2 - - - -
Ax	RES 4, B 2 2 - - - -	RES 4, C 2 2 - - - -	RES 4, D 2 2 - - - -	RES 4, E 2 2 - - - -	RES 4, H 2 2 - - - -	RES 4, L 2 2 - - - -	RES 4, (HL) 2 4 - - - -	RES 4, A 2 2 - - - -	RES 5, B 2 2 - - - -	RES 5, C 2 2 - - - -	RES 5, D 2 2 - - - -	RES 5, E 2 2 - - - -	RES 5, H 2 2 - - - -	RES 5, L 2 2 - - - -	RES 5, (HL) 2 4 - - - -	RES 5, A 2 2 - - - -
Bx	RES 6, B 2 2 - - - -	RES 6, C 2 2 - - - -	RES 6, D 2 2 - - - -	RES 6, E 2 2 - - - -	RES 6, H 2 2 - - - -	RES 6, L 2 2 - - - -	RES 6, (HL) 2 4 - - - -	RES 6, A 2 2 - - - -	RES 7, B 2 2 - - - -	RES 7, C 2 2 - - - -	RES 7, D 2 2 - - - -	RES 7, E 2 2 - - - -	RES 7, H 2 2 - - - -	RES 7, L 2 2 - - - -	RES 7, (HL) 2 4 - - - -	RES 7, A 2 2 - - - -
Cx	SET 0, B 2 2 - - - -	SET 0, C 2 2 - - - -	SET 0, D 2 2 - - - -	SET 0, E 2 2 - - - -	SET 0, H 2 2 - - - -	SET 0, L 2 2 - - - -	SET 0, (HL) 2 4 - - - -	SET 0, A 2 2 - - - -	SET 1, B 2 2 - - - -	SET 1, C 2 2 - - - -	SET 1, D 2 2 - - - -	SET 1, E 2 2 - - - -	SET 1, H 2 2 - - - -	SET 1, L 2 2 - - - -	SET 1, (HL) 2 4 - - - -	SET 1, A 2 2 - - - -
Dx	SET 2, B 2 2 - - - -	SET 2, C 2 2 - - - -	SET 2, D 2 2 - - - -	SET 2, E 2 2 - - - -	SET 2, H 2 2 - - - -	SET 2, L 2 2 - - - -	SET 2, (HL) 2 4 - - - -	SET 2, A 2 2 - - - -	SET 3, B 2 2 - - - -	SET 3, C 2 2 - - - -	SET 3, D 2 2 - - - -	SET 3, E 2 2 - - - -	SET 3, H 2 2 - - - -	SET 3, L 2 2 - - - -	SET 3, (HL) 2 4 - - - -	SET 3, A 2 2 - - - -
Ex	SET 4, B 2 2 - - - -	SET 4, C 2 2 - - - -	SET 4, D 2 2 - - - -	SET 4, E 2 2 - - - -	SET 4, H 2 2 - - - -	SET 4, L 2 2 - - - -	SET 4, (HL) 2 4 - - - -	SET 4, A 2 2 - - - -	SET 5, B 2 2 - - - -	SET 5, C 2 2 - - - -	SET 5, D 2 2 - - - -	SET 5, E 2 2 - - - -	SET 5, H 2 2 - - - -	SET 5, L 2 2 - - - -	SET 5, (HL) 2 4 - - - -	SET 5, A 2 2 - - - -
Fx	SET 6, B 2 2 - - - -	SET 6, C 2 2 - - - -	SET 6, D 2 2 - - - -	SET 6, E 2 2 - - - -	SET 6, H 2 2 - - - -	SET 6, L 2 2 - - - -	SET 6, (HL) 2 4 - - - -	SET 6, A 2 2 - - - -	SET 7, B 2 2 - - - -	SET 7, C 2 2 - - - -	SET 7, D 2 2 - - - -	SET 7, E 2 2 - - - -	SET 7, H 2 2 - - - -	SET 7, L 2 2 - - - -	SET 7, (HL) 2 4 - - - -	SET 7, A 2 2 - - - -

Figure 1: Game Boy's opcode-tables