

# Emulation

Takanen Edoardo

March 24, 2025

## Abstract

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.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Premises</b>	<b>4</b>
<b>3</b>	<b>General Structure</b>	<b>5</b>

# 1 Introduction

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 Gameboy 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 Gameboy 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 Gameboy (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 Gameboy 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 through 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.

```
1 int main() {
2     // Hardware components definition
3     Memory mem;
4     CPU cpu;
5     // ...
6
7     // Components initialization
8     mem.init();
9     cpu.init();
10    cpu.load_boot();
11    // ...
12
13    while (true) {
14        cpu.exec_op(mem); // Executing an operation
15        // emulate all the other components
16    }
17
18    return 0;
19 }
```

Actually, when we look at the circuit inside the Gameboy, 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.

```
1 while (true) {
2     // Returns how many clock cycles the instruction took
3     int cycles = cpu.exec_op();
4
5     // Updating all the other components
6     timers.update(cycles);
7     ppu.update(cycles);
8     // ...
9 }
```

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

Before looking at the main components that shape the Gameboy hardware, I would like to focus on how memory is subdivided inside the console. The address bus had 16 bits, meaning there could be 65'536 unique addresses (64Kb). Since the Gameboy 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).

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	FDFD	Prohibited area
FE00	FE9F	OAM
FEA0	FEFF	Prohibited area
FF00	FF7F	I/O Registers
FF80	FFFE	High RAM
FFFF	FFFF	IE register

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 exactly one byte of data.

```

1 struct Memory {
2 private:
3     static constexpr u32 MAX_MEM = 64 * 1024;
4     // Array of bytes to emulate all the Gameboy's addresses
5     Byte Data[MAX_MEM] = {};
6 public:
7     void init();
8
9     /**
10    * Functions used for setting and accessing memory as
11    * mem[addr] = value          to set
12    * Byte value = mem[addr]     to access
13    * */
14    Byte operator[](u32 addr) const;
15    Byte& operator[](u32 addr);
16
17    /**
18    * Dumps all the memory in a file,
19    * used for debugging purposes
20    * */
21    void dump(const char* filename);
22 };

```

This is the whole Memory structure. Every component we are going to make will have access to this structure in order to read and set memory.

## Game Boy CPU Instructions

[illegible]

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

[illegible]

Figure 1: Gameboy's opcode-tables