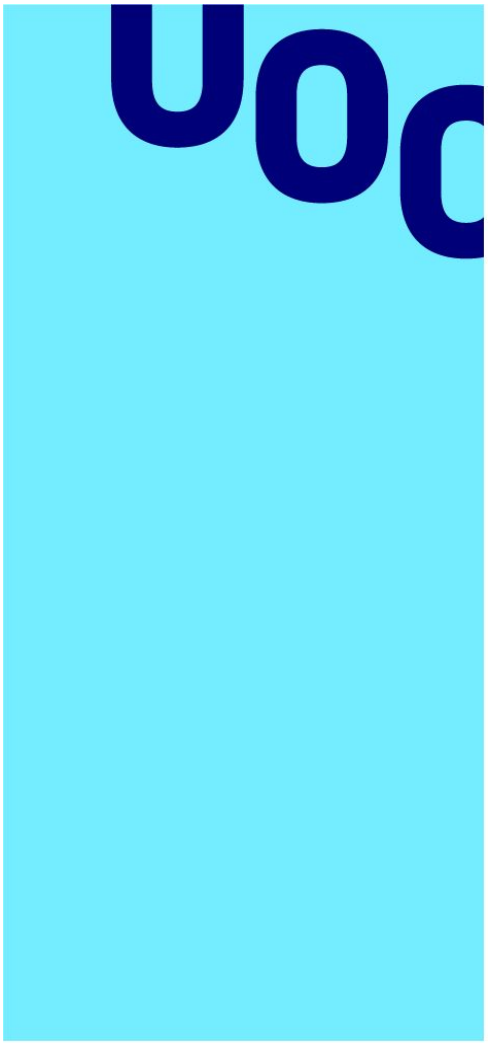
Treball Final de Grau PAC2

David Soler Bartomeu



Universitat Oberta
de Catalunya

Index

Introduction

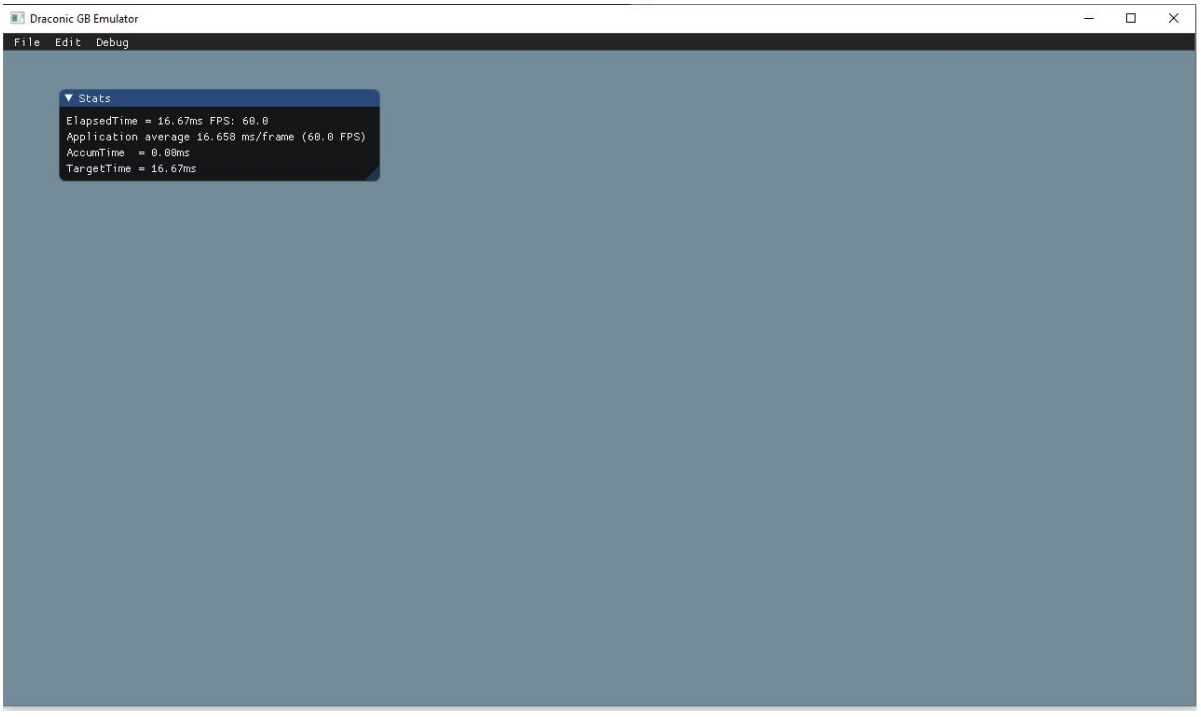
In this PAC the executable file shows the creation of a window, de display of debug information and the processing of input by the user.

An emulator is incredibly different from a video game so in this case there is no gameplay showcased in this deliverable. The main focus of the PAC2 has been the research and gathering of information regarding Game Boy emulator development.

The development will be performed using Visual Studio Community 2019, so downloading the repo and compiling using it will be able to generate a valid executable.

Execution

The emulator can be run by running the ‘DraconicGB.exe’ program found in the folder ‘Executable’. Running the program should open up a window. The window is created using SDL and the menus displayed have been created using ocornut imGUI.



References

The following references have been used in order to gather information about the emulator as well as on how to correctly create a windows program with a use interface.

https://gbdev.io/pandocs/

Memory Map

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

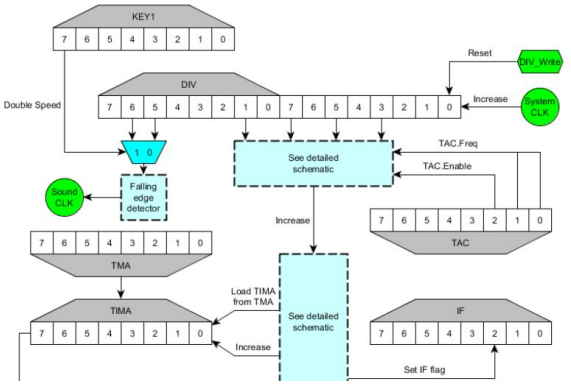
General Memory Map

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)
8000	9FFF	8KB Video RAM (VRAM)	Only bank 0 in Non-CGB mode Switchable bank 0/1 in CGB mode
A000	BFFF	8KB External RAM	In cartridge, switchable 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 Switchable 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)	

https://github.com/AntonioND/giibiiadvance/blob/master/docs/TCAGBD.pdf

5. Timer

The Game Boy timer subsystem is composed by the three timer registers (TIMA, TMA and TAC), and the DIV register. This is a simplified schematic:



https://gekkio.fi/files/gb-docs/gbctr.pdf

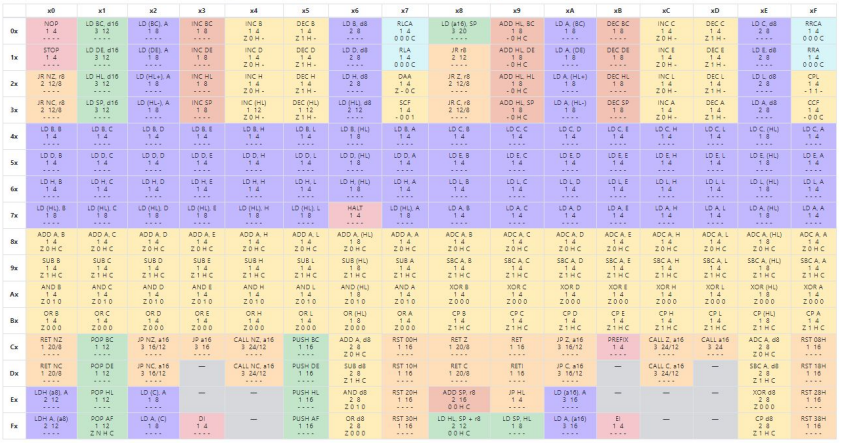
GAME BOY

PROGRAMMING MANUAL

Version 1.1

https://gbdev.io/gb-opcodes/optables/

Game Boy CPU (SM83) instruction set (JSON)



https://rednex.github.io/rgbds/gbz80.7.html

DEC [HL]

Decrement the byte pointed to by **HL** by 1.

Cycles: 3

Bytes: 1

Flags: See DEC r8

DEC r16

Decrement value in register *r16* by 1.

Cycles: 2

Bytes: 1

Flags: None affected.

DEC SP

Decrement value in register **SP** by 1.

Cycles: 2

https://cturt.github.io/cinoop.html

Writing a Game Boy emulator, Cinoop

I've always wanted to write an emulator from scratch, but I've held off for a long time because it's probably the most adv project I've ever wanted to do.

Picking a system to emulate isn't an easy choice; the standard first emulator project seems to be a CHIP-8 emulator. R definitely helped me to understand a lot of emulation concepts, but it seemed a bit too basic. I felt that I got enough out through other people's emulators, and that writing my own would be a pointless exercise.

On the other hand, there's the NES and Game Boy; both of which seemed far too advanced for me!

Eventually, I decided to write a minimalist Game Boy interpreting emulator, without support for custom mappers or sour many inaccuracies). I called the project Cinoop.

Cinoop is written in C and is open source. It can be run on Windows, DS, GameCube, 3DS, Linux based OSes, PSP, a

https://github.com/ocornut/imgui

dear imgui

build passing

(This library is available under a free and permissive license, but needs financial support to sustain its continued improvements. In addition to maintenance and stability the are many desirable features yet to be added. If your company is using dear imgui, please consider reaching out.)

Businesses: support continued development via invoiced technical support, maintenance, sponsoring contracts:
E-mail: contact @ dearimgui dot org

Individuals: support continued maintenance and development with PayPal.

Dear ImGui is a **bloat-free graphical user interface library for C++.** It outputs optimized vertex buffers that you can render anytime in your 3D-pipeline enabled application. It is fast, portable, renderer agnostic and self-contained (no external dependencies).

https://lazyfoo.net/tutorials/SDL/

Table of Contents	
Lesson 01 Hello SDL	In this tutorial we will be setting up the SDL library and creating our first window.
Lesson 02 Getting an Image on the Screen	Now that we can get a window to appear, lets blit an image onto it.
Lesson 03 Event Driven Programming	Here we'll start handling user input by allowing the user to X out the window.
Lesson 04 Key Presses	Here we'll learn to handle keyboard input.
Lesson 05 Optimized Surface Loading and Soft Stretching	Now that we know how to load and blit surfaces, it's time to make our blits faster. We'll also take a smaller image and stretch it
Lesson 06 Extension Libraries and Loading Other Image Formats	Here we'll be using the SDL_image extension library to load png images.
Lesson 07 Texture Loading and Rendering	A big new feature in SDL 2.0 is hardware accelerated texture based 2D rendering. Here we'll be loading an image to render it u
Lesson 08 Geometry Rendering	Another new feature in SDL 2.0 is hardware accelerated primitive rendering. Here we'll be using it to render some common
Lesson 09 The Viewport	SDL 2.0 also lets you control where you render on the screen using the viewport. We'll be using the viewport to create su
Lesson 10 Color Keying	Here we'll use color keying to give textures transparent backgrounds.
Lesson 11 Clip Rendering and Sprite Sheets	Using clip rendering, you can keep multiple images on one texture and render the part you need. We'll be using this to render individual s
Lesson 12	We'll be altering the color of rendered textures using color modulation.