BRING GAME BOY ALIVE IN THE WEB WITH RUST AND WEBASSEMBLY

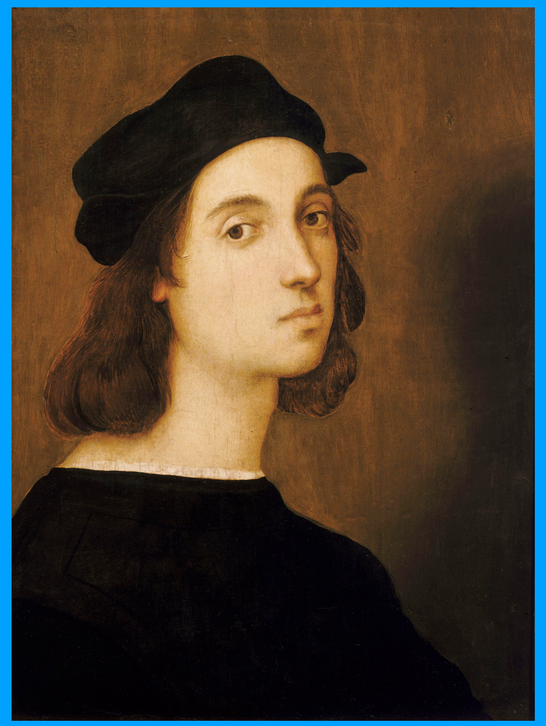
Raphael Amorim

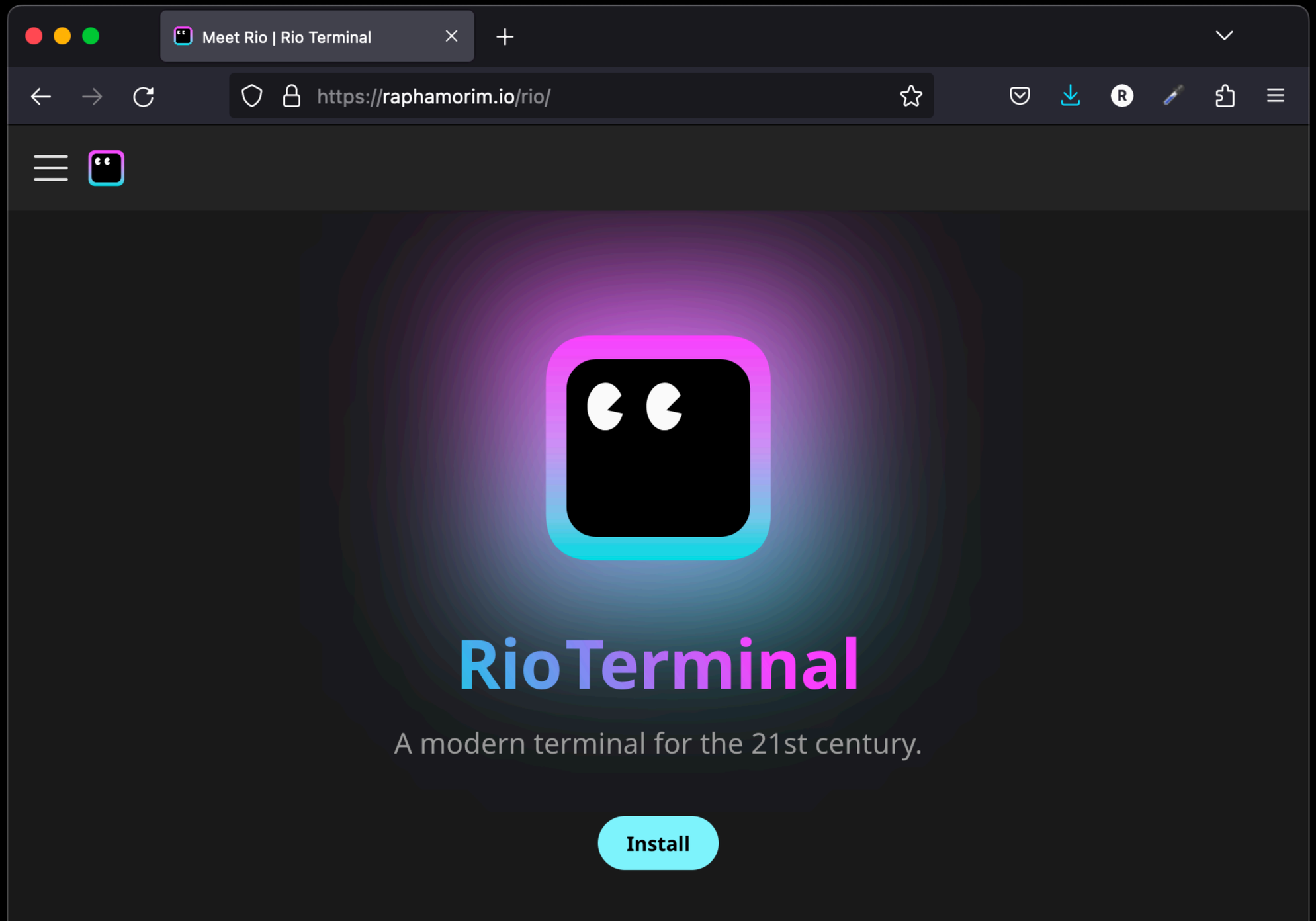
github.com/raphamorim

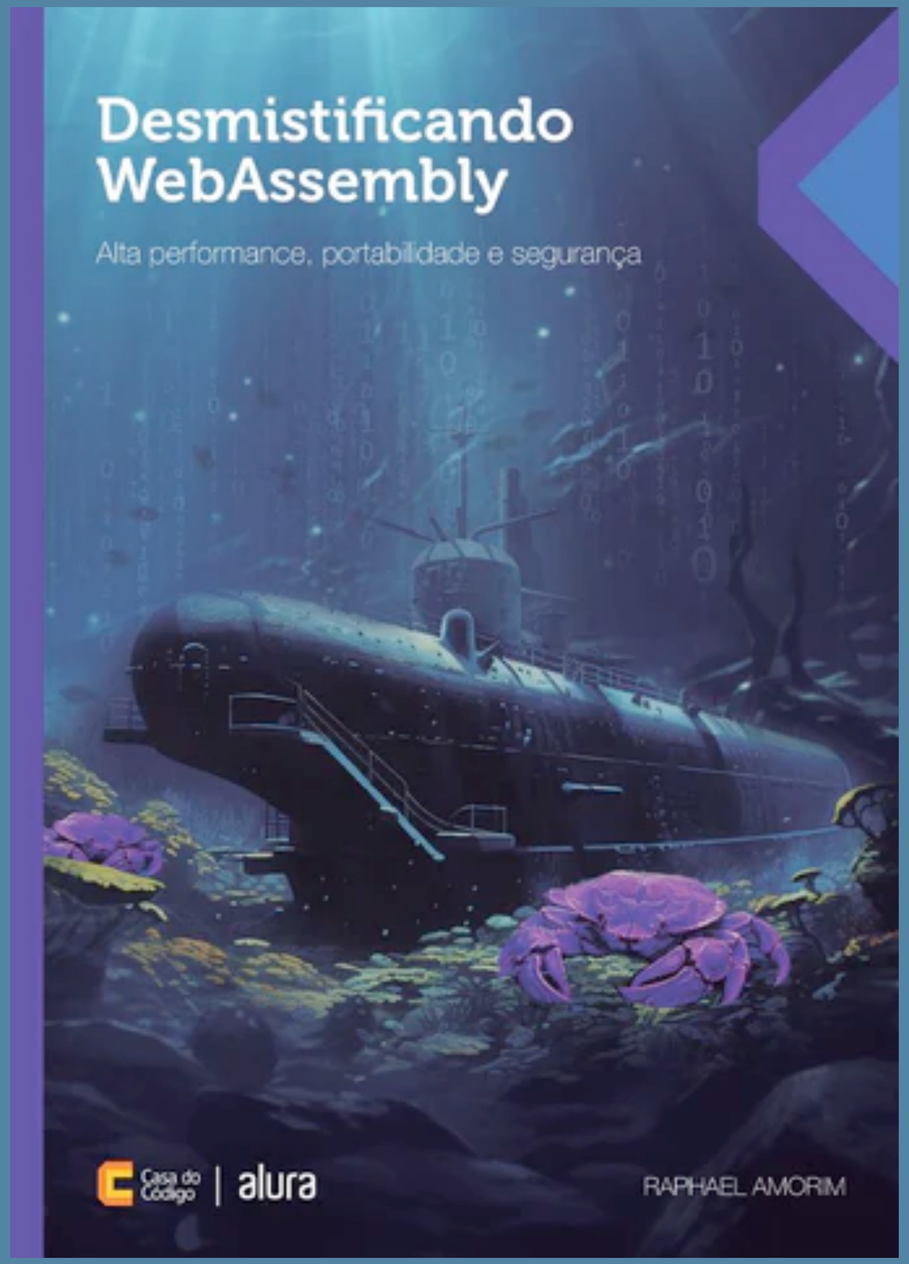
twitter.com/raphamorims

mas.to/@mustache

mustache.bsky.social




Meet Rio | Rio Terminal
https://raphamorim.io/rio/
Rio Terminal
A modern terminal for the 21st century.
Install



If by any chance you speak Portuguese, I wrote book about Rust and WebAssembly

If you don’t, sorry for the additional spam.

DISCLAIMER #1

Nothing of this talk is written in stone.

For example: You should be able to write your Game Boy with Rust, Go, JavaScript, Java, or any programming language that you want to.

There’s no advocation of tech gatekeeping in this talk.

DISCLAIMER #2

This talk does not endorse or promote any type of piracy activity.

The act of build or install an emulator is not illegal.

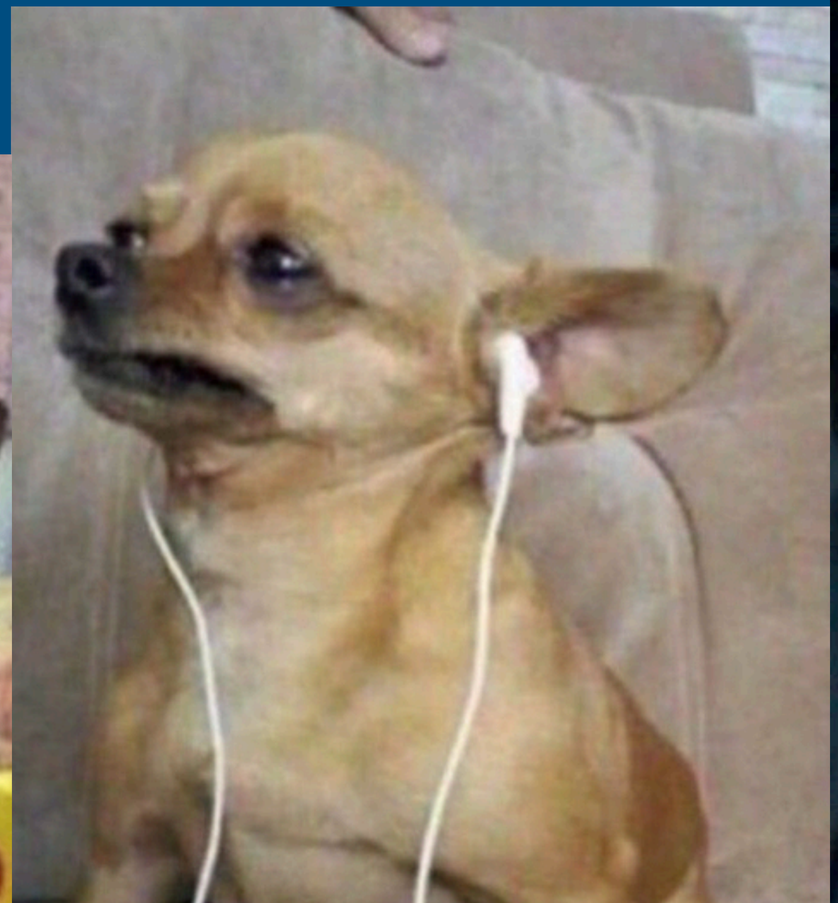
As many others emulators that have been created over past decades, this project is just a study case.

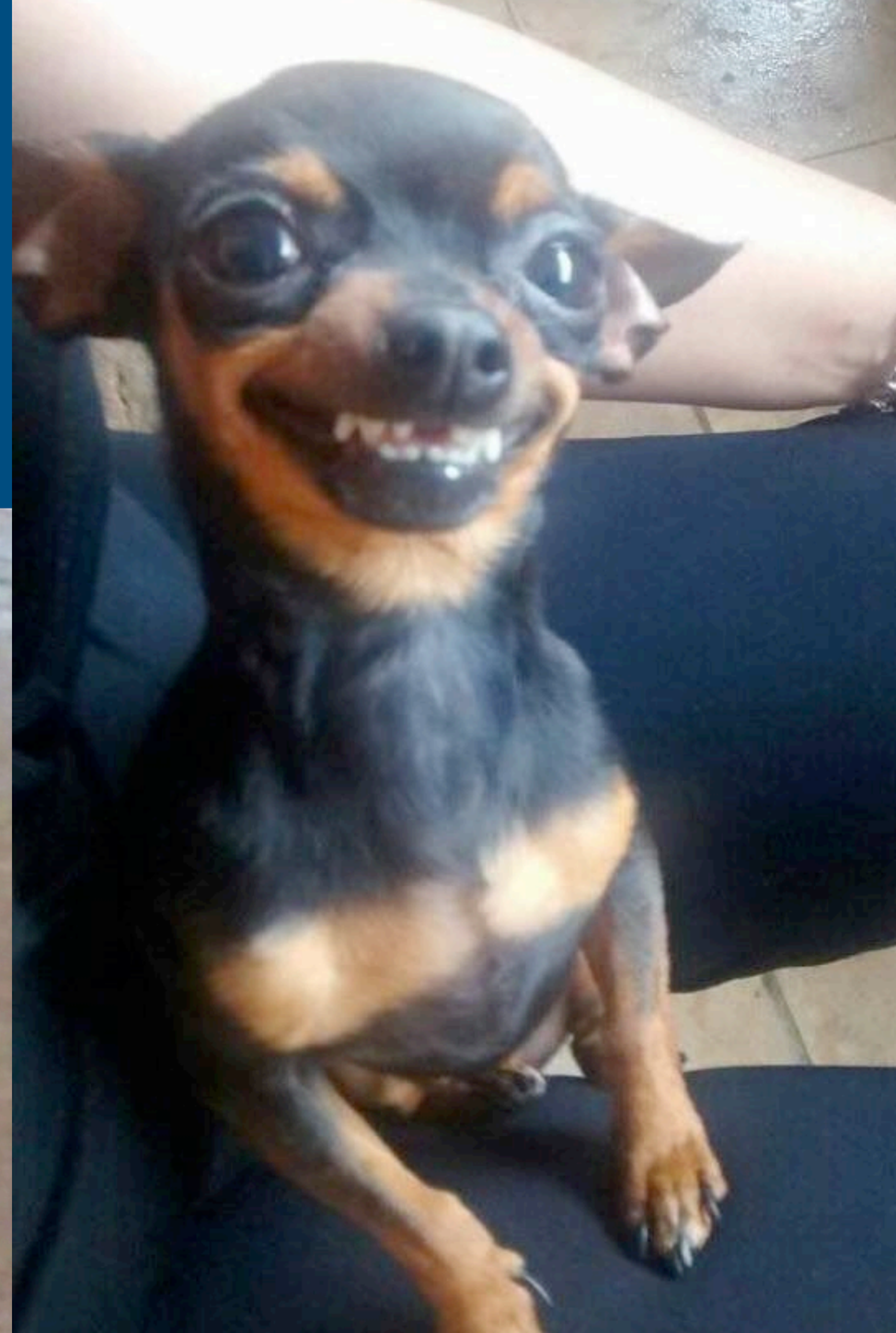
DISCLAIMER #3

There’s a lot of concepts required by a Game Boy emulator to properly work that we will briefly or not address.

However, I'll leave links at the end of the slides so you can deepen your Game Boy understanding.

COOL! LET'S START OUR GAME BOY JOURNEY

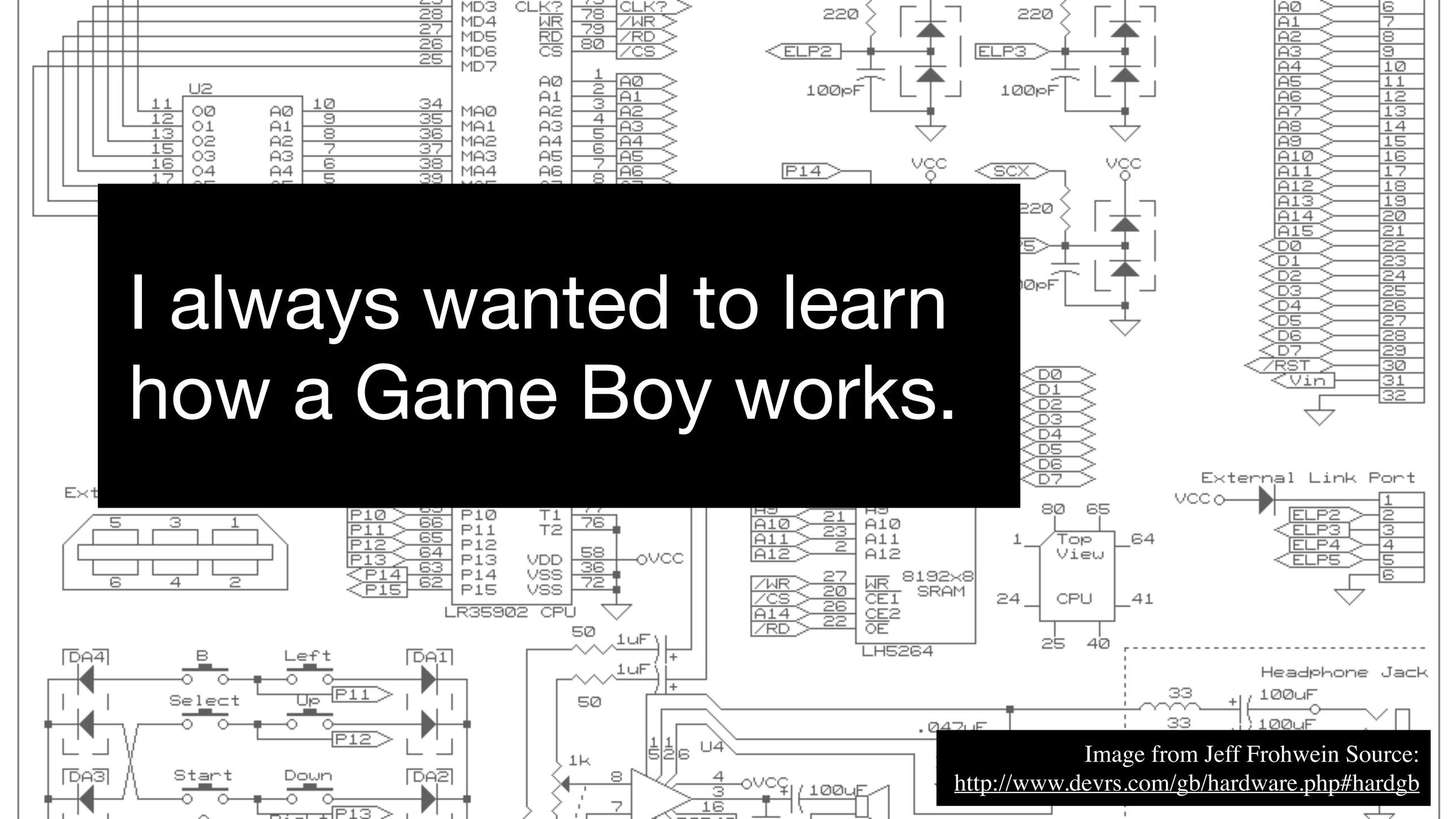




A long time ago someone told me that if you want to learn how a specific computer works.

A long time ago someone told me that if you want to learn how a specific computer works.

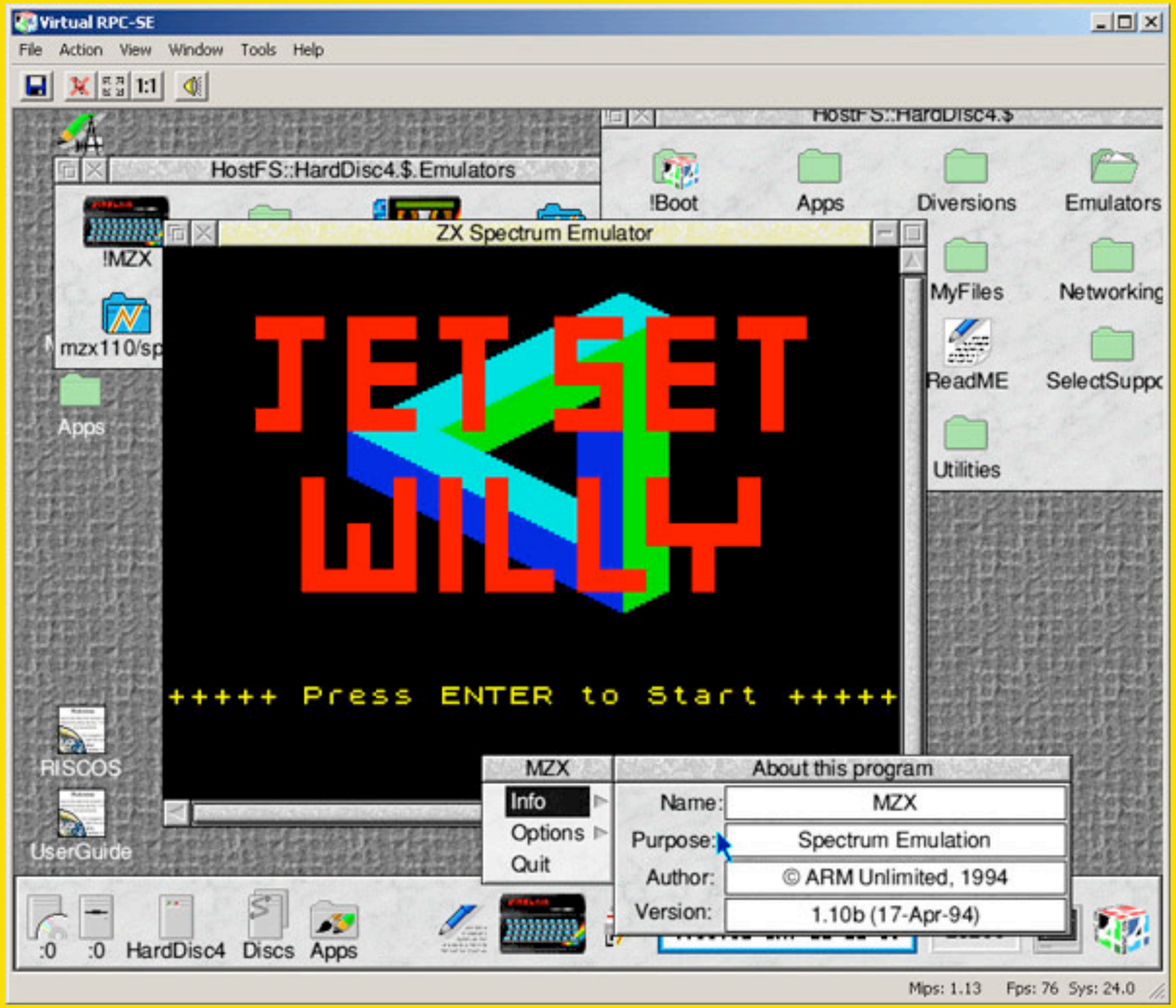
There’s no better way to learn than by emulating that computer.


I always wanted to learn how a Game Boy works.
LR35902 CPU
LH5264
8192x8 SRAM
Top View
CPU
External Link Port
Headphone Jack
Image from Jeff Frohwein Source:
http://www.devrs.com/gb/hardware.php#hardgb

Emulators

An emulator is hardware or software that enables one computer system (called the host) to behave like another computer system (called the guest).

You can see emulation everywhere.



Windows XP running an Archimedes emulator through ZX Spectrum emulator.

Windows XP (host)
ZX Spectrum emulator (guest)

You can see emulation everywhere.

Terminal emulation

The image in the left is a VT100 often emulated by popular terminals emulators nowadays.

Although many terminals emulators are compatible with VT100 is quite common see extension of the original functionalities.

You can see emulation everywhere.

"Harvest Moon: Back to Nature" was originally released for PsOne but also available in other video game consoles like PS5 through emulation.

There are other types of emulators like hardware emulators, network emulators, in-circuit emulators (ICE), server emulators and etcetera.

- http://bgb.bircd.org/pandocs.htm
- https://github.com/mvdnes/rboy
- https://github.com/alexcrichton/jba/tree/rust
- https://github.com/gbdev/pandocs
- http://imrannazar.com/GameBoy-Emulation-in-JavaScript:-The-CPU
- https://multigesture.net/articles/how-to-write-an-emulator-chip-8-interpreter/
- http://emubook.emulation64.com/

gameboy-emulator

Here are 337 public repositories matching this topic...

Language: All ▾ Sort: Most stars ▾

There’s tons of emulators for Game Boy

- https://medium.com/@andrewimm/writing-a-game-boy-emulator-in-wasm-part-1-1ba023eb2c7c
- https://github.com/yodalee/ruGameboy
- https://www.youtube.com/watch?v=LqcEg3IVziQ
- https://realboyemulator.wordpress.com/2013/01/01/the-nintendo-game-boy-1/
- https://gbdev.gg8.se/wiki/articles/DMG_Schematics
- https://chipmusic.org/forums/topic/13608/dmg-main-board-schematic-circuit-arduinoboy/
- https://github.com/torch2424/wasmboy/
- https://rylev.github.io/DMG-01/public/book/introduction.html
- https://github.com/gbdev/awesome-gbdev
- http://marc.rawer.de/Gameboy/Docs/GBProject.pdf
- https://shonumi.github.io/dandocs.html
- https://github.com/Baekalfen/PyBoy/blob/master/PyBoy.pdf
- https://media.ccc.de/v/rustfest-rome-3-gameboy-emulator
- https://github.com/rylev/DMG-01
- https://gbdev.gg8.se/wiki/articles/Gameboy_Bootstrap_ROM

gbdev / awesome-gbdev

Code Issues 22 Pull requests

A curated list of Game Boy development resources such as tools, docs, emulators, related projects and open-source ROMs.

- http://imrannazar.com/GameBoy-Emulation-in-JavaScript:-The-CPU
- http://nocash.emubase.de/pandocs.htm (Available at http://bgb.bircd.org/pandocs.htm)

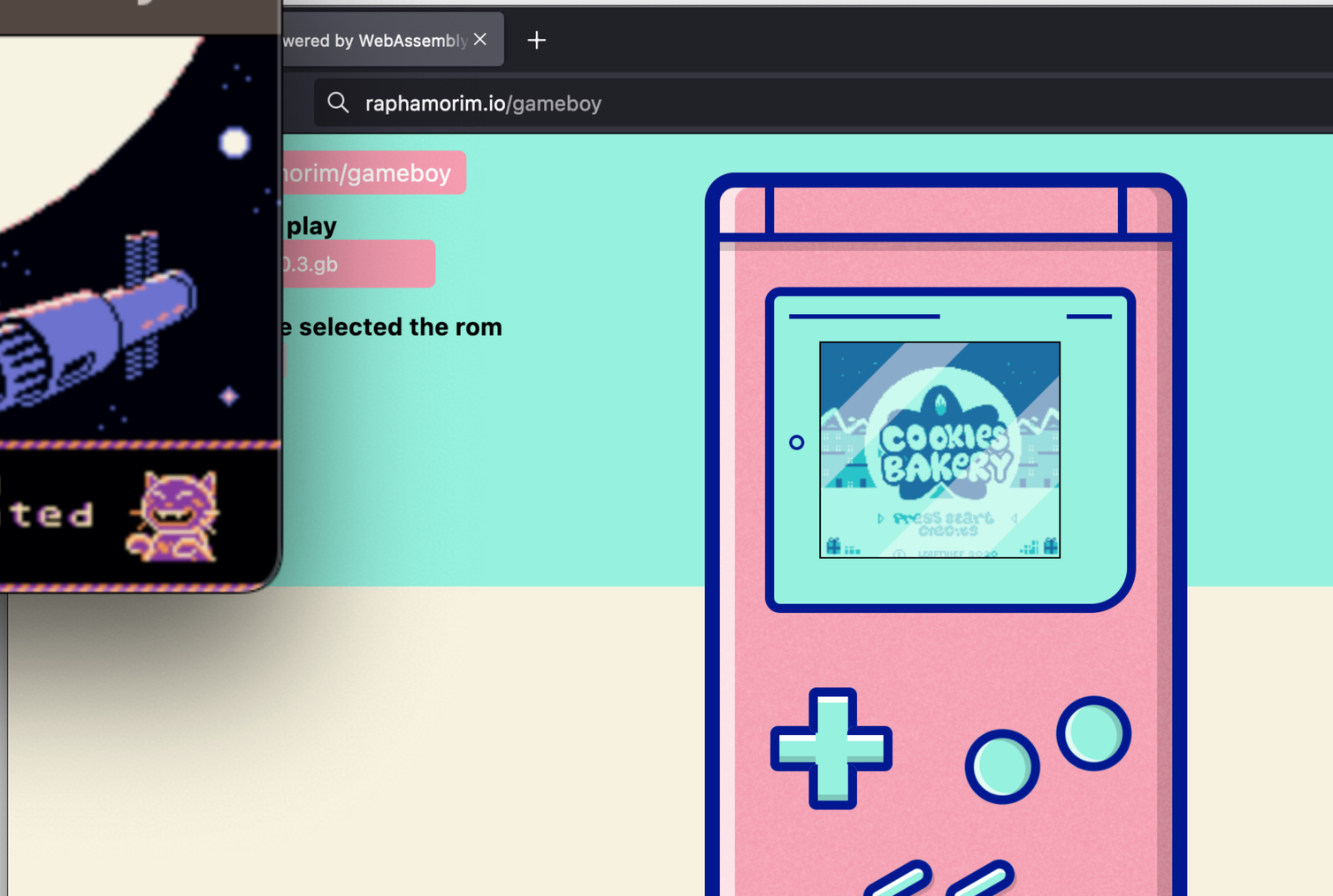
However majority of the emulators that I found were targeting either desktop or web.

What if we could create a Game Boy emulator
that runs as a (1) desktop application
or as a (2) browser application
(or even a (3) terminal application) ?

Terminal

Desktop

Web



cargo init

The code of this talk is available in:
github.com/raphamorim/mini-gameboy-emulator-rustlab

There’s multiple tools/programming languages that can you can use but I’ve decided to use Rust.

Rust

Low-Level control with high-level ergonomics

> Control over indirection (dereferencing), monomorphization, memory layout.

The language doesn’t have a runtime, allowing to create small .wasm sizes since there is no extra stuff being added like a garbage collector.

Now we decided the language let’s **establish our emulation goal**:

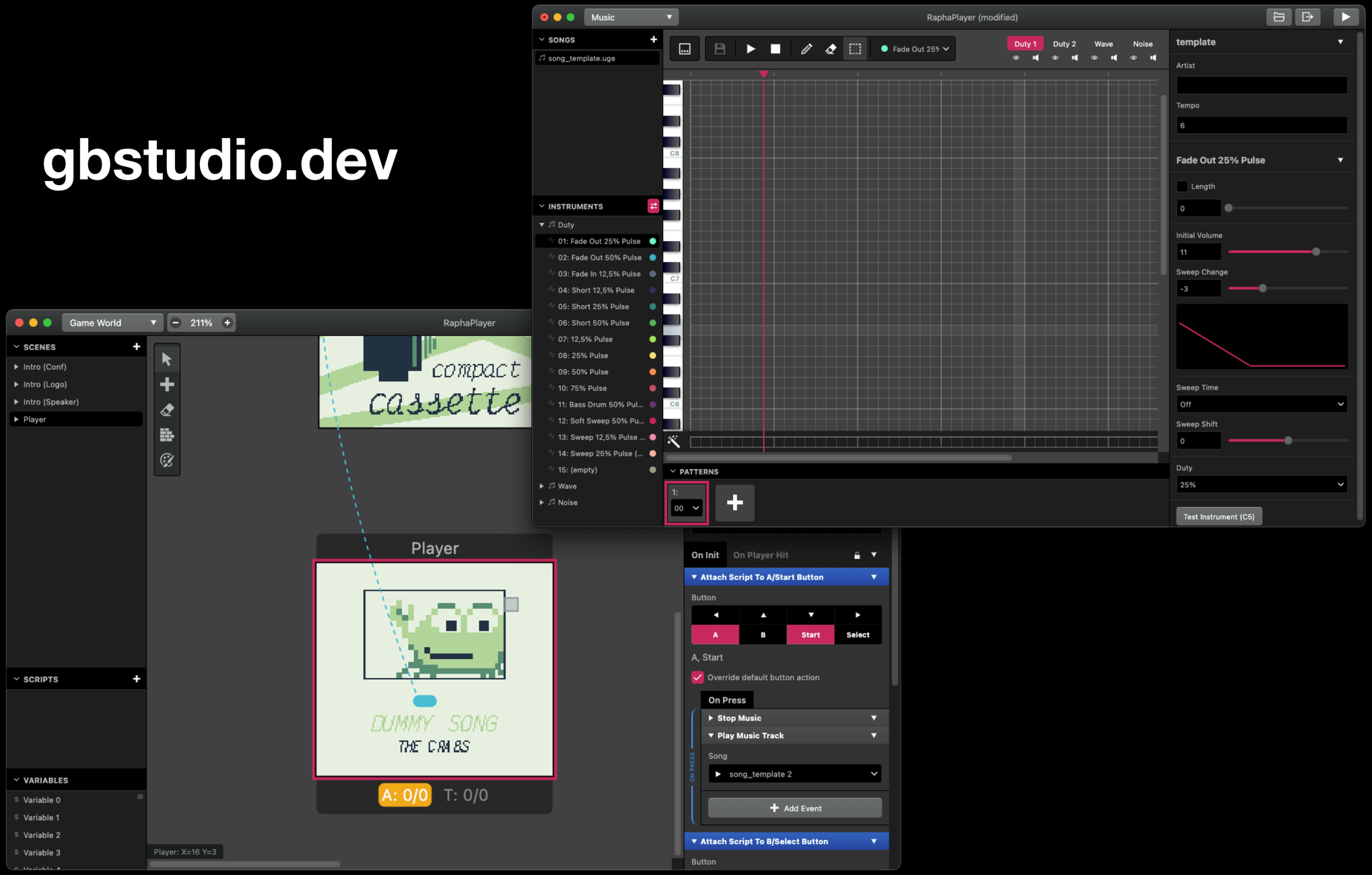
The emulator will run a simple game that was created solely for this conference talk.

Mostly by the fact that:

Emulators are free and legal, ROMs are not *

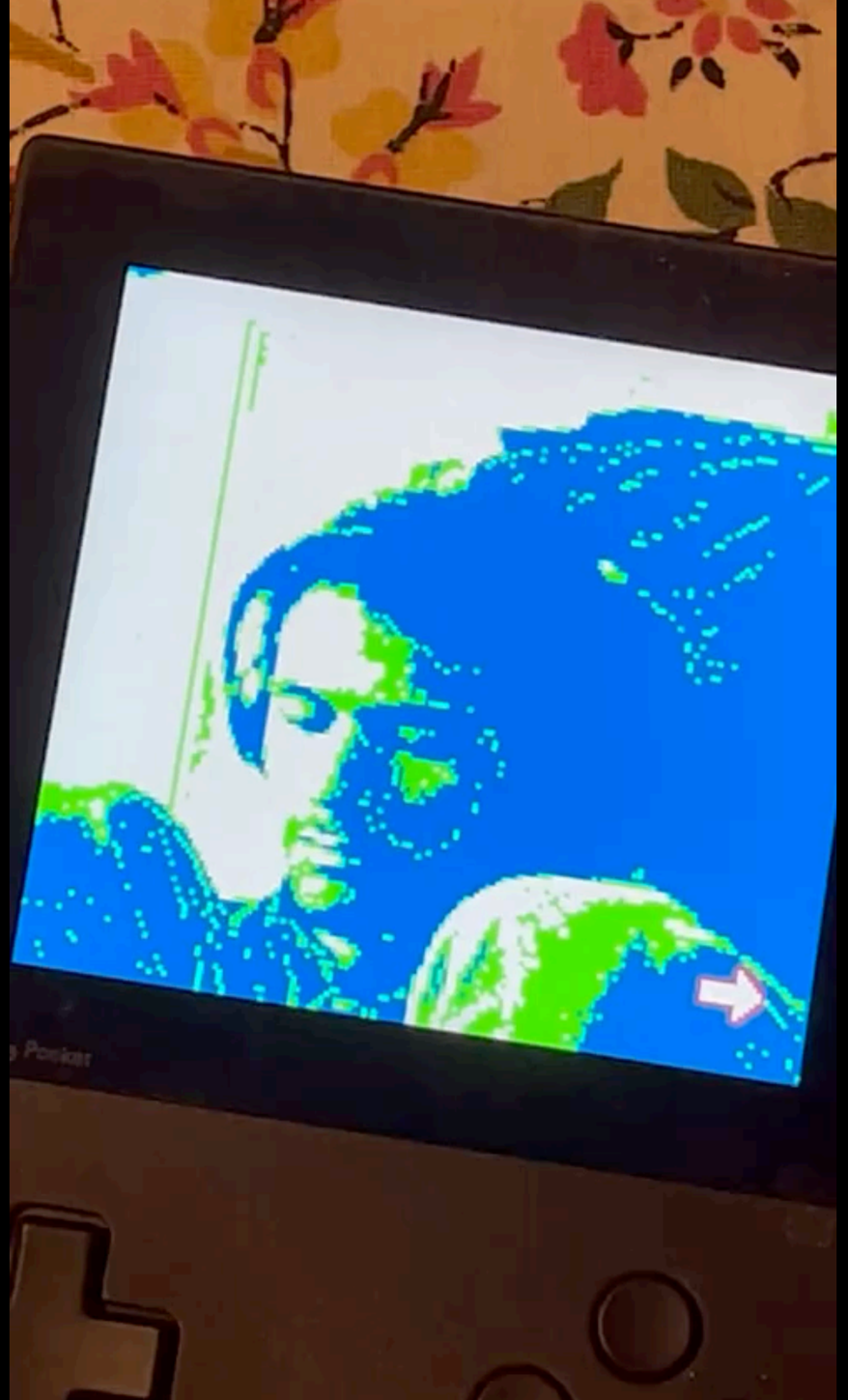
* This statement assumes the ROM was obtained without any permission by the copyright holders.

gbstudio.dev



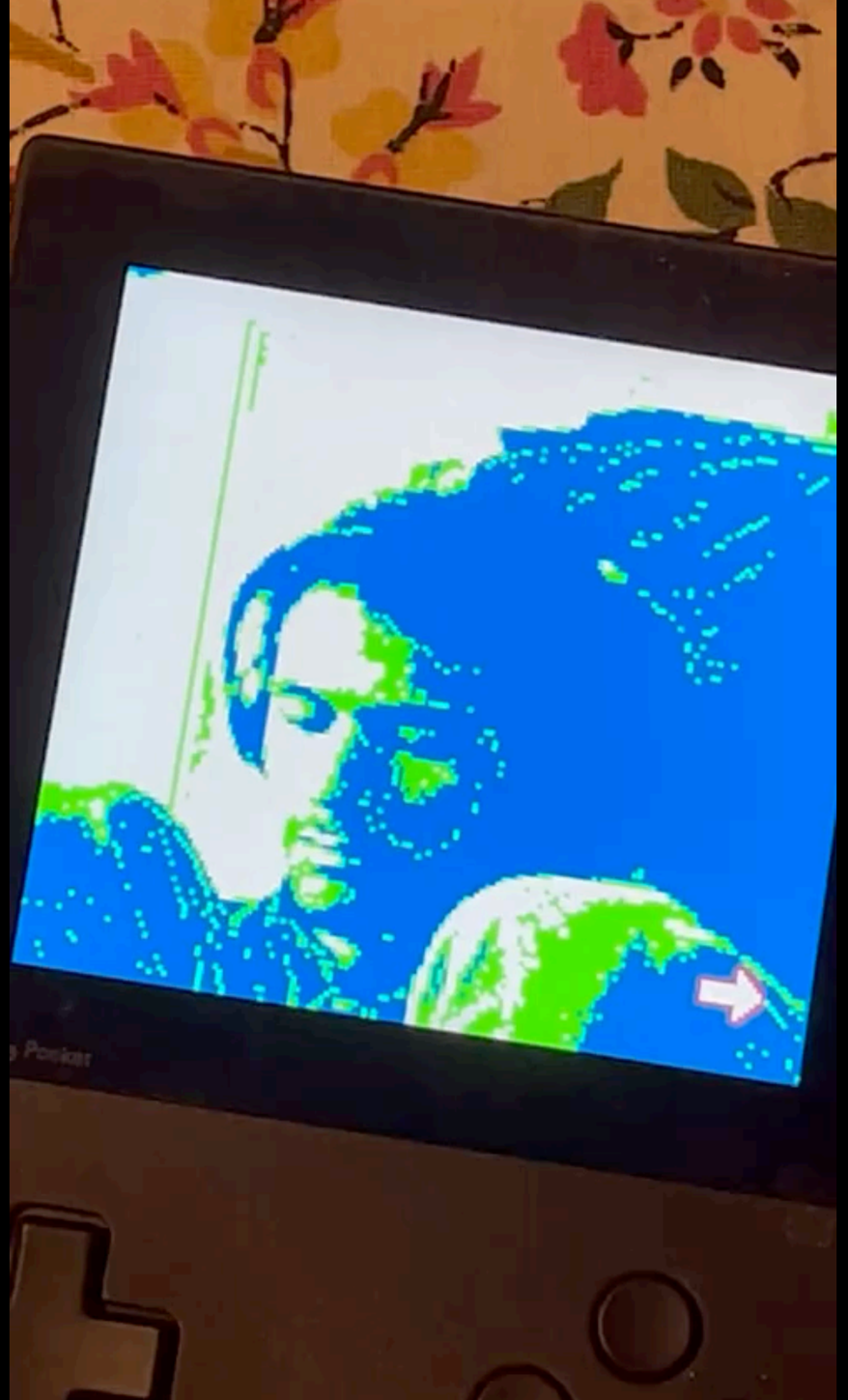
Testing our game using Analog Pocket

analogue.co/pocket

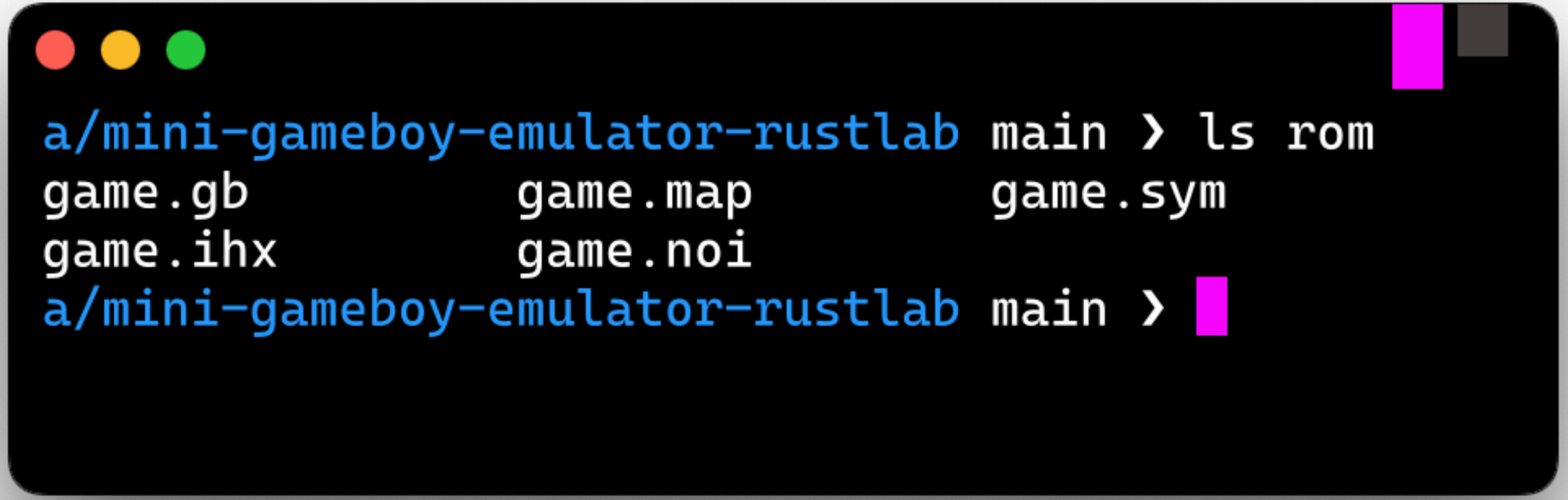


Testing our game using Analog Pocket

analogue.co/pocket



Created a **folder rom** that contains the game file, intel hexadecimal object, debugging maps and other stuff. For this talk we mostly care about the .gb format



For this talk I didn’t want to have an unified render for multiple platforms.

Mostly because we want to **primary learn how a Game Boy works**, so if you have any expertise either with HTML5 Canvas (CanvasRenderingContext2D) or OpenGL is quite easy to contextualise what we are doing through the talk.

```
[package]
name = “gameboy"
version = "0.1.0"
edition = "2021"

[[bin]]
name = "desktop"
path = "src/desktop.rs"

[lib]
name = "wasm"
path = "src/wasm.rs"
crate-type = ["cdylib"]
```

index.html

src/

desktop.rs

wasm.rs

gameboy/

mod.rs

```
[package]
name = “gameboy"
version = "0.1.0"
edition = "2021"

[[bin]]
name = "desktop"
path = "src/desktop.rs"

[lib]
name = "wasm"
path = "src/wasm.rs"
crate-type = ["cdylib"]
```

In this talk we are assuming that:

- [lib] will always be a *cdylib* that targets wasm32
- [lib] contains a completely different code than [[bin]]
- both [lib] and [[bin]] consumes a *gameboy* mod.

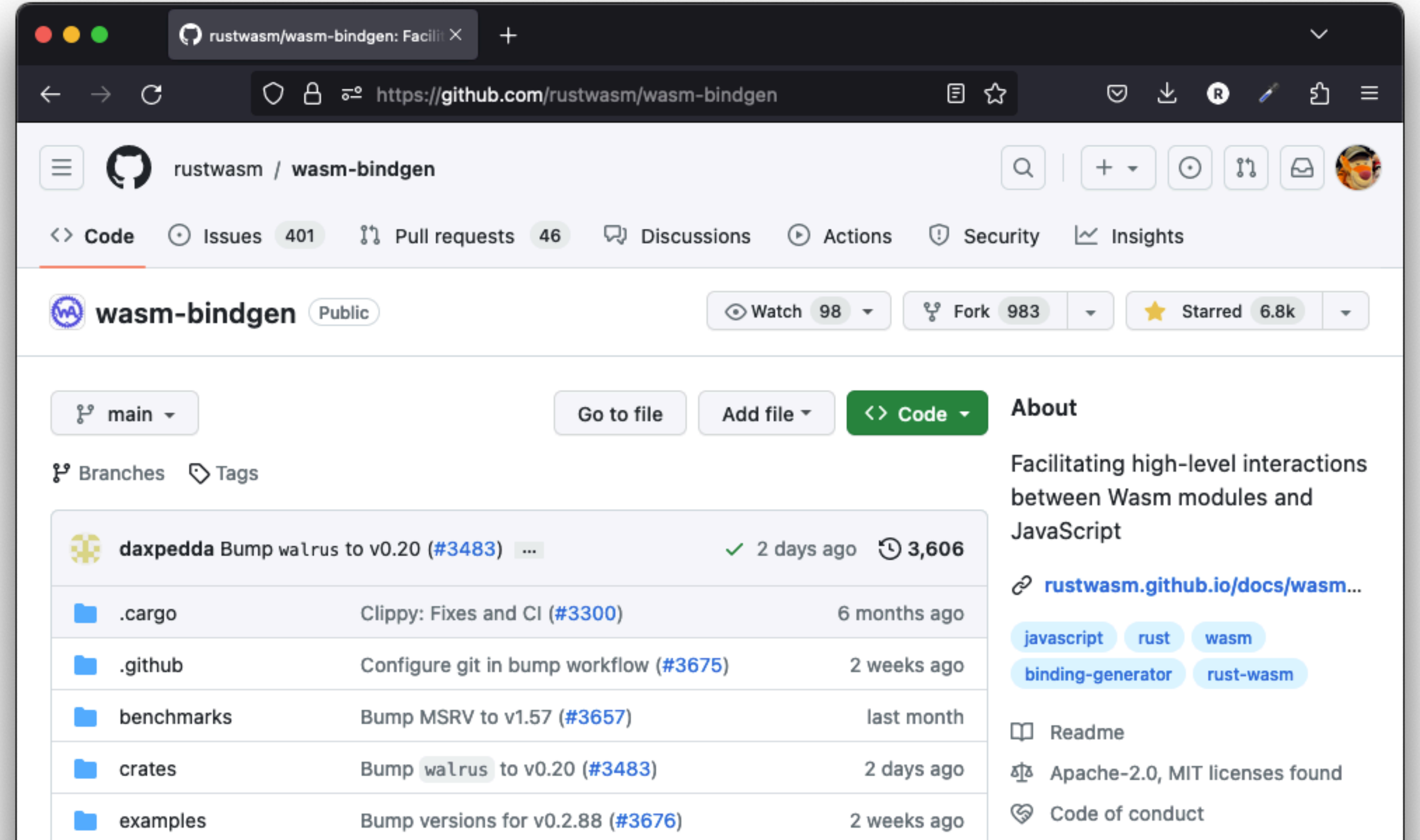
This configuration can be quite chaotic for many cases. Majority of the time you want to reuse code between target archs or better organise it (e.g having distributed crates with cargo workspaces).

src/gameboy/**mod.rs**

```rust
pub enum Button {}

pub struct GameBoy {
    width: u32,
    height: u32,
}
```

```rust
impl GameBoy {
    pub fn new(rom: Vec<u8>) -> Self {
        Self {
            width: 160,
            height: 144,
        }
    }

    pub fn width(&self) -> u32 { self.width }
    pub fn height(&self) -> u32 { self.height }
    pub fn data(&self) -> &[u8] { &[] }
    pub fn frame(&self) {}
    pub fn keydown(&self, button: Button) {}
    pub fn keyup(&self, button: Button) {}
}
```

WebAssembly

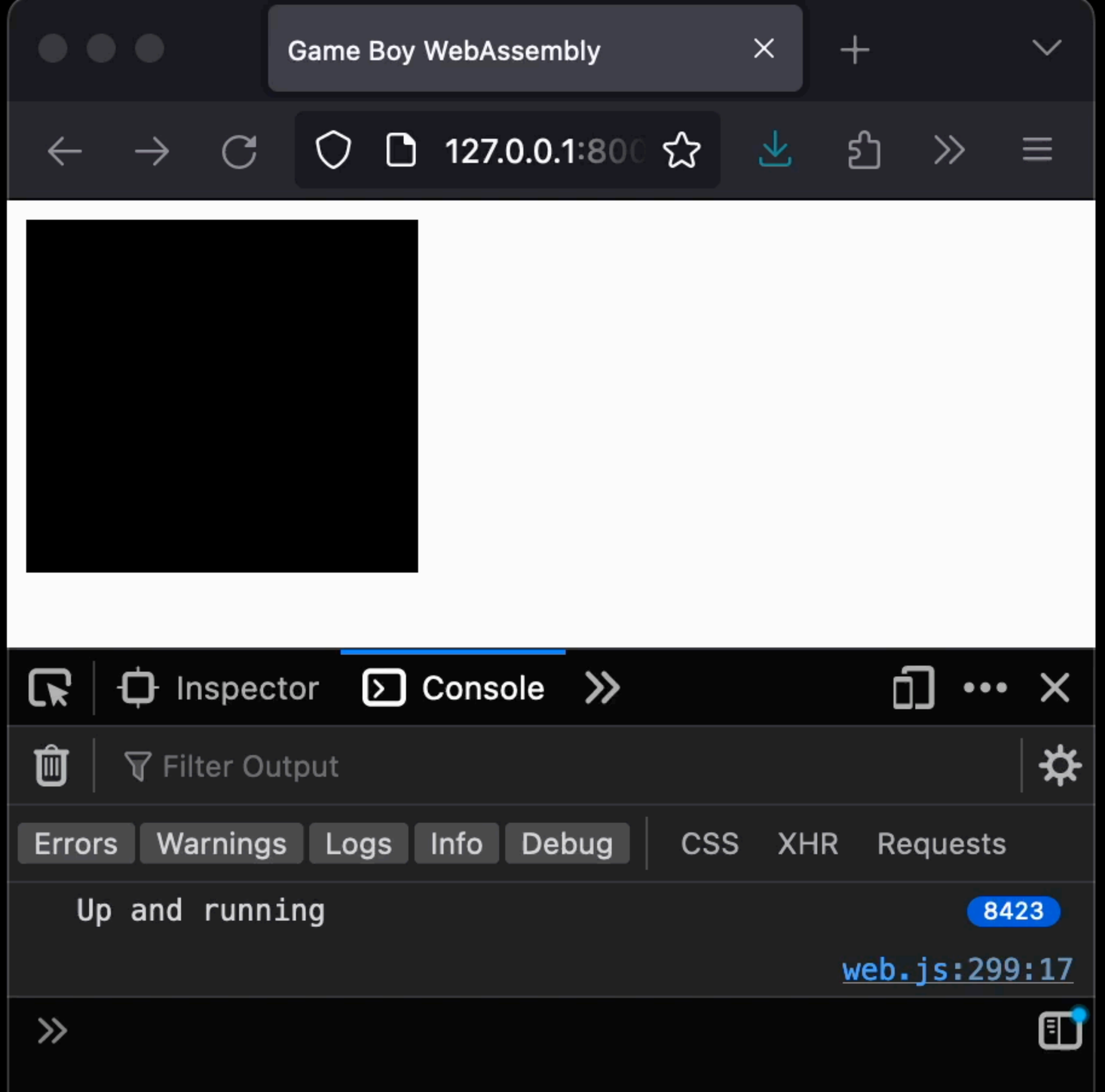

rustwasm/wasm-bindgen: Facili
https://github.com/rustwasm/wasm-bindgen
rustwasm / wasm-bindgen
Code
Issues 401
Pull requests 46
Discussions
Actions
Security
Insights
wasm-bindgen
Public
Watch 98
Fork 983
Starred 6.8k
main
Go to file
Add file
Code
Branches
Tags
daxpedda Bump walrus to v0.20 (#3483) ...
2 days ago
3,606
.cargo
Clippy: Fixes and CI (#3300)
6 months ago
.github
Configure git in bump workflow (#3675)
2 weeks ago
benchmarks
Bump MSRV to v1.57 (#3657)
last month
crates
Bump walrus to v0.20 (#3483)
2 days ago
examples
Bump versions for v0.2.88 (#3676)
2 weeks ago
About
Facilitating high-level interactions between Wasm modules and JavaScript
rustwasm.github.io/docs/wasm...
javascript
rust
wasm
binding-generator
rust-wasm
Readme
Apache-2.0, MIT licenses found
Code of conduct

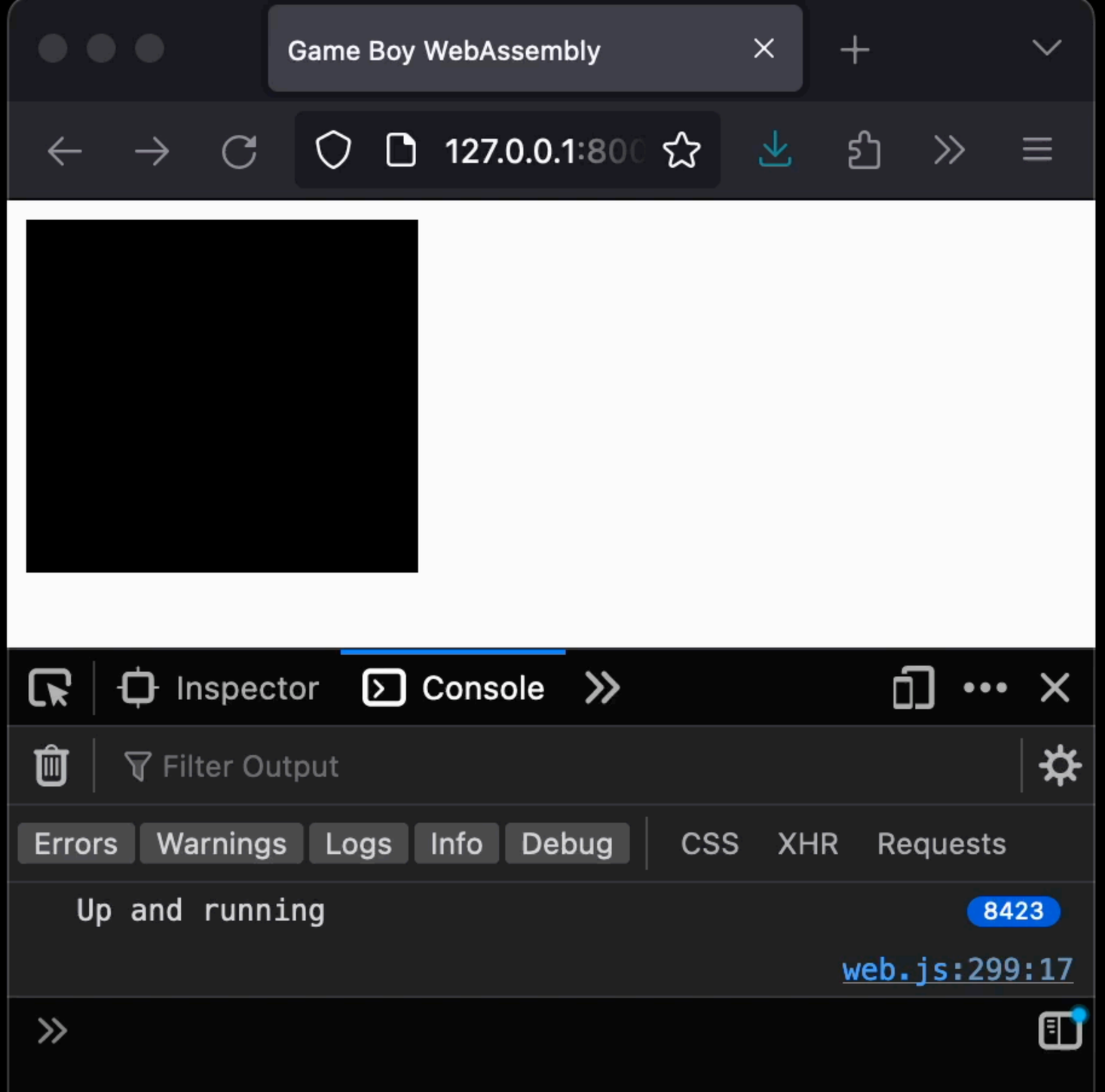
src/web.rs

```rust
#[wasm_bindgen]
pub async fn render(rom: Vec<u8>) -> Result<(), wasm_bindgen::JsValue> {
  let mut gb = GameBoy::new(rom);

  let document = window().document().unwrap();
  let game = document.get_element_by_id("game");
  let canvas = document.create_element("canvas")?
      .dyn_into::<web_sys::HtmlCanvasElement>()?;
  game.unwrap().append_child(&canvas)?;
  canvas.set_width(gb.width());
  canvas.set_height(gb.height());
  let context = canvas.get_context("2d")?.unwrap()
      .dyn_into::<CanvasRenderingContext2d>().unwrap();
```

src/web.rs

```
let f_main = Rc::new(RefCell::new(None));
let f_frame = f_main.clone();
*f_frame.borrow_mut() = Some(Closure::wrap(Box::new(move || {
        gb.frame();
        log("Up and running");
        if let Ok(image_data) = ImageData::new_with_u8_clamped_array_and_sh(
            wasm_bindgen::Clamped(gb.data()),
            gb.width(),
            gb.height(),
        ) {
            context.put_image_data(&image_data, 0.0, 0.0).ok();
        }

        request_animation_frame(f_main.borrow().as_ref().unwrap());
}) as Box<dyn FnMut()>));

request_animation_frame(f_frame.borrow().as_ref().unwrap());
```


Game Boy WebAssembly
127.0.0.1:800
Inspector
Console
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
Up and running
8423
web.js:299:17


Game Boy WebAssembly
127.0.0.1:800
Inspector
Console
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
Up and running
8423
web.js:299:17

OpenGL

src/desktop.rs

```
const VERTEX: &str = r"#version 150 core
in vec2 pos;
in vec3 color;
in vec2 tcoord;
out vec3 Color;
out vec2 coord;
void main() {
   Color = color;
   coord = tcoord;
   gl_Position = vec4(pos, 0.0, 1.0);
}";
```

```
const FRAGMENT: &str = r"#version 150
core
in vec3 Color;
in vec2 coord;
out vec4 outColor;
uniform sampler2D sampler;
void main() {
   outColor = texture(sampler, coord);
}";
```

src/desktop.rs

```
pub fn draw(&self, gb: &GameBoy) { unsafe {
    gl::ClearColor(0.0, 0.0, 1.0, 1.0);
    gl::Clear(gl::COLOR_BUFFER_BIT);

    gl::TexImage2D(
        gl::TEXTURE_2D, 0, gl::RGB as i32,
        gb.width() as i32, gb.height() as i32,
        0, gl::RGBA, gl::UNSIGNED_BYTE,
        gb.data().as_ptr() as *const _,
    );
    assert_eq!(gl::GetError(), 0);

    gl::DrawElements(
        gl::TRIANGLES, 6,
        gl::UNSIGNED_INT, std::ptr::null());
```

src/desktop.rs

```rust
#[inline]
pub fn load_our_game_rom() -> Result<Vec<u8>, Error> {
    use std::{fs::File, io::Read};
    let mut rom = Vec::new();
    let file = File::open("./rom/game.gb");
    file.and_then(|mut f| f.read_to_end(&mut rom))?;
    Ok(rom)
}

fn main() -> Result<(), Error> {
    let rom_data = load_our_game_rom()?;
    let gb = GameBoy::new(rom_data);
```

src/desktop.rs

```
glutin::event::Event::MainEventsCleared ⇒ window.request_redraw(),
glutin::event::Event::RedrawRequested(_) ⇒ {
    gb.frame();
    cx.draw(&gb);
    gl_window.swap_buffers().unwrap();
}
```

src/desktop.rs

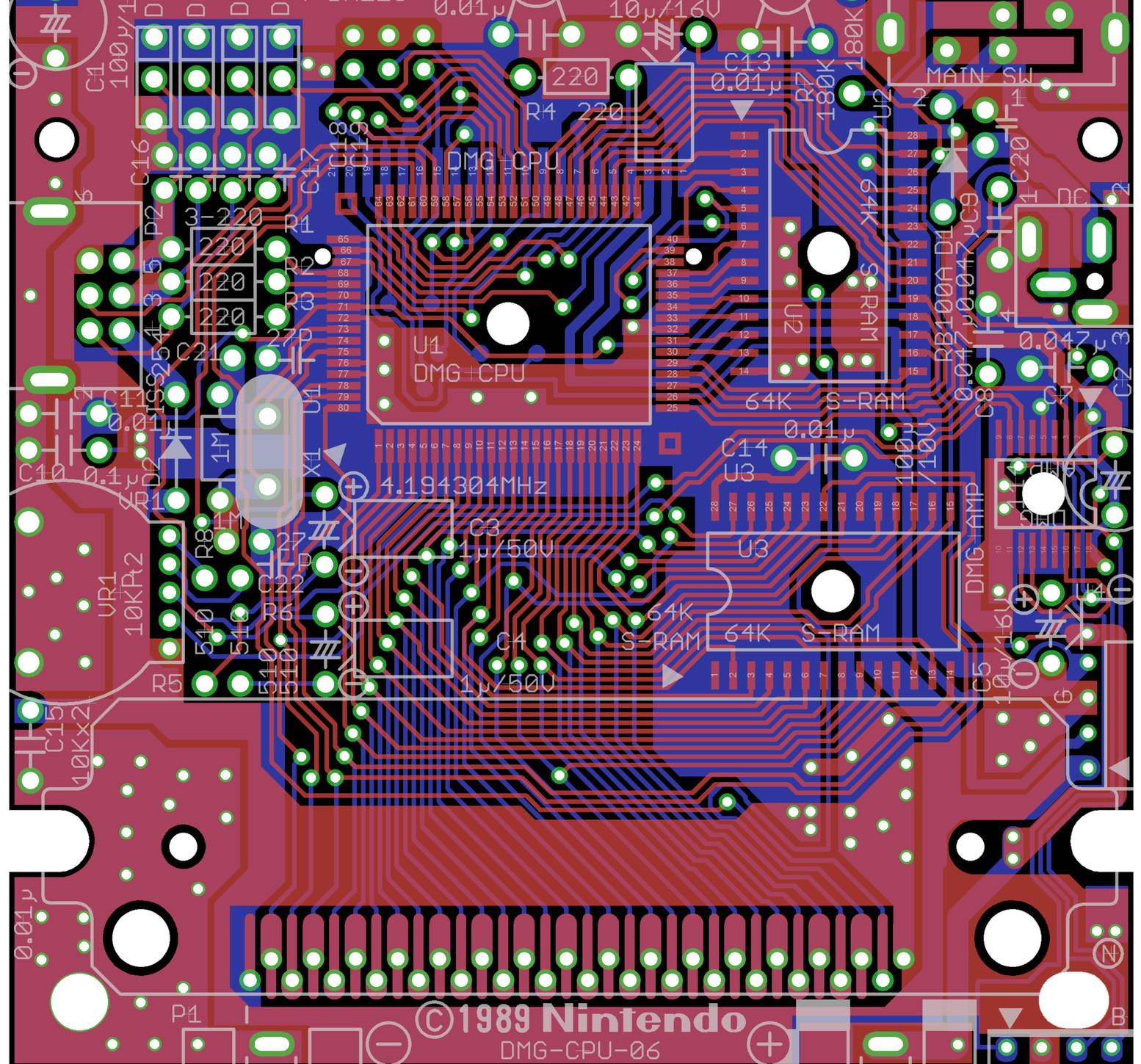
```
let event_loop: glutin::event_loop::EventLoop<()> =
    glutin::event_loop::EventLoop::with_user_event();
let window_builder = glutin::window::WindowBuilder::new()
    .with_title("GameBoy")
    .with_inner_size(glutin::dpi::LogicalSize {
        width: gb.width(),
        height: gb.height(),
    });

let gl_window = glutin::ContextBuilder::new()
    .build_windowed(window_builder, &event_loop)
    .unwrap();
let gl_window = unsafe { gl_window.make_current().unwrap() };
```


GameBoy

The Game Boy

A scan of the main logic board for the DMG* 01 (Game Boy): chipmusic.org/forums/topic/13608/dmg-main-board-schematic-circuit-arduinoboy/

DMG stands for “Dot Matrix Game”



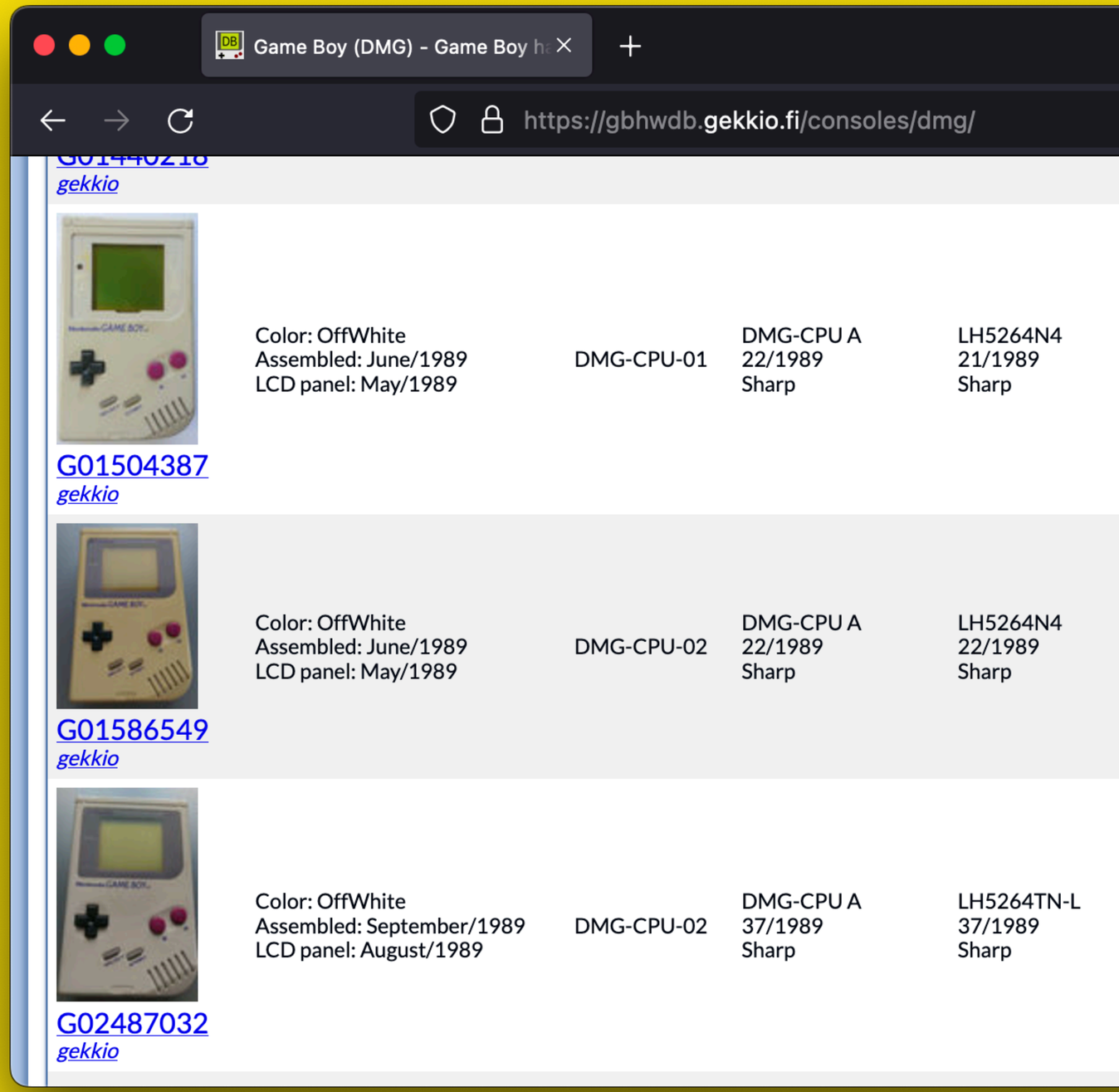
Game Boy technical data

CPU	- 8-bit (Similar to the Z80 processor)
Clock Speed	- 4.194304MHz (4.295454MHz for SGB, max. 8.4MHz for CGB)
Work RAM	- 8K Byte (32K Byte for CGB)
Video RAM	- 8K Byte (16K Byte for CGB)
Screen Size	- 2.6"
Resolution	- 160x144 (20x18 tiles)
Max sprites	- Max 40 per screen, 10 per line
Sprite sizes	- 8x8 or 8x16
Palettes	- 1x4 BG, 2x3 OBJ (for CGB: 8x4 BG, 8x3 OBJ)
Colors	- 4 grayshades (32768 colors for CGB)
Horiz Sync	- 9198 KHz (9420 KHz for SGB)
Vert Sync	- 59.73 Hz (61.17 Hz for SGB)
Sound	- 4 channels with stereo sound
Power	- DC6V 0.7W (DC3V 0.7W for GB Pocket, DC3V 0.6W for CGB)

Retired from http://bgb.bircd.org/pandocs.htm

Game Boys came in different mainboard models, for example, the DMG-CPU-03, DMG-CPU-05 and DMG-CPU-06.

Their differences are very subtle for the programmers.



Simplified GB hardware overview

`Input → Process → Output`

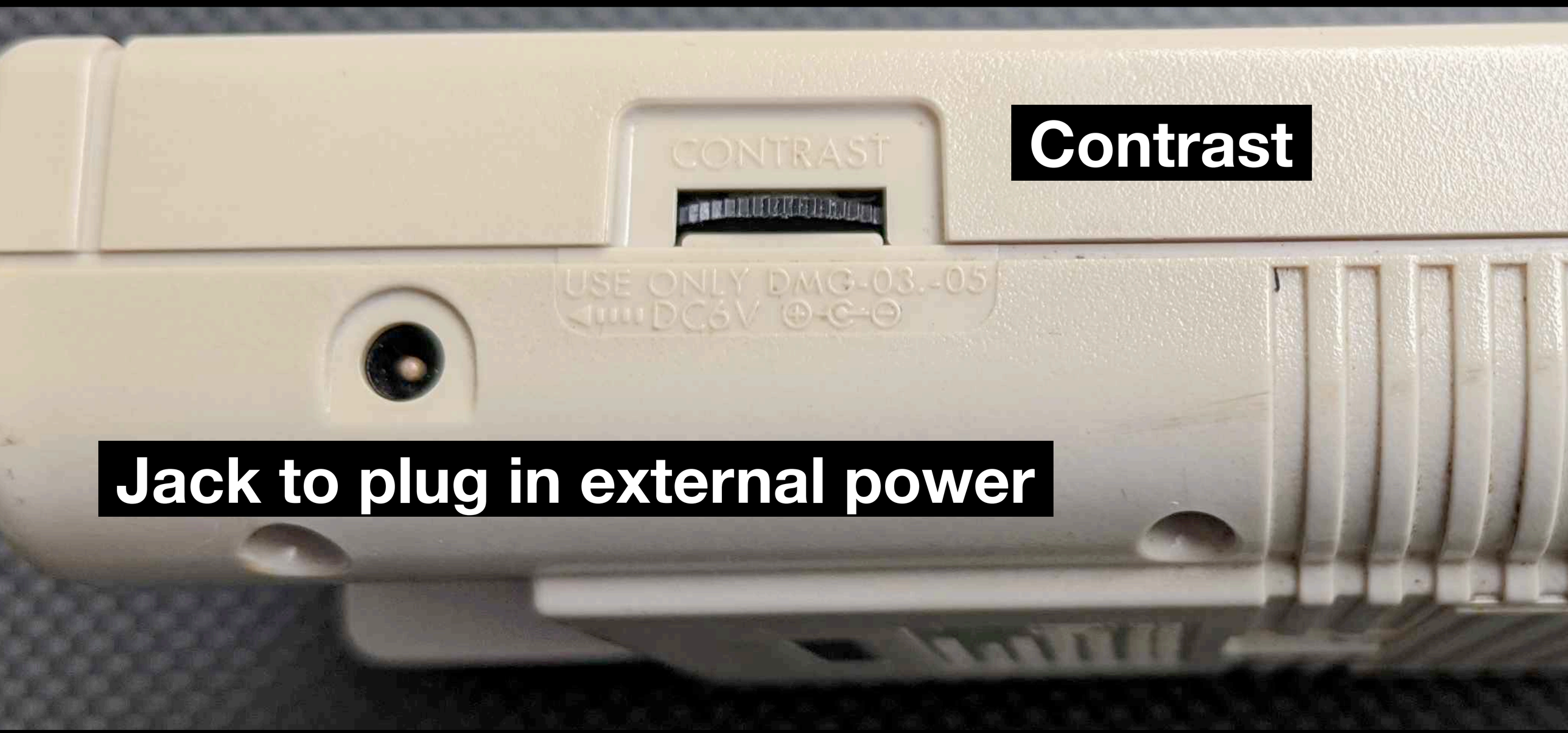
Photograph:

1. https://en.wikipedia.org/wiki/File:Game-Boy-Original.jpg
2. https://www.youtube.com/watch?v=lW9uKZE4yJ0

Input

Photograph:

1. https://b13rg.github.io/Gameboy_DMG/

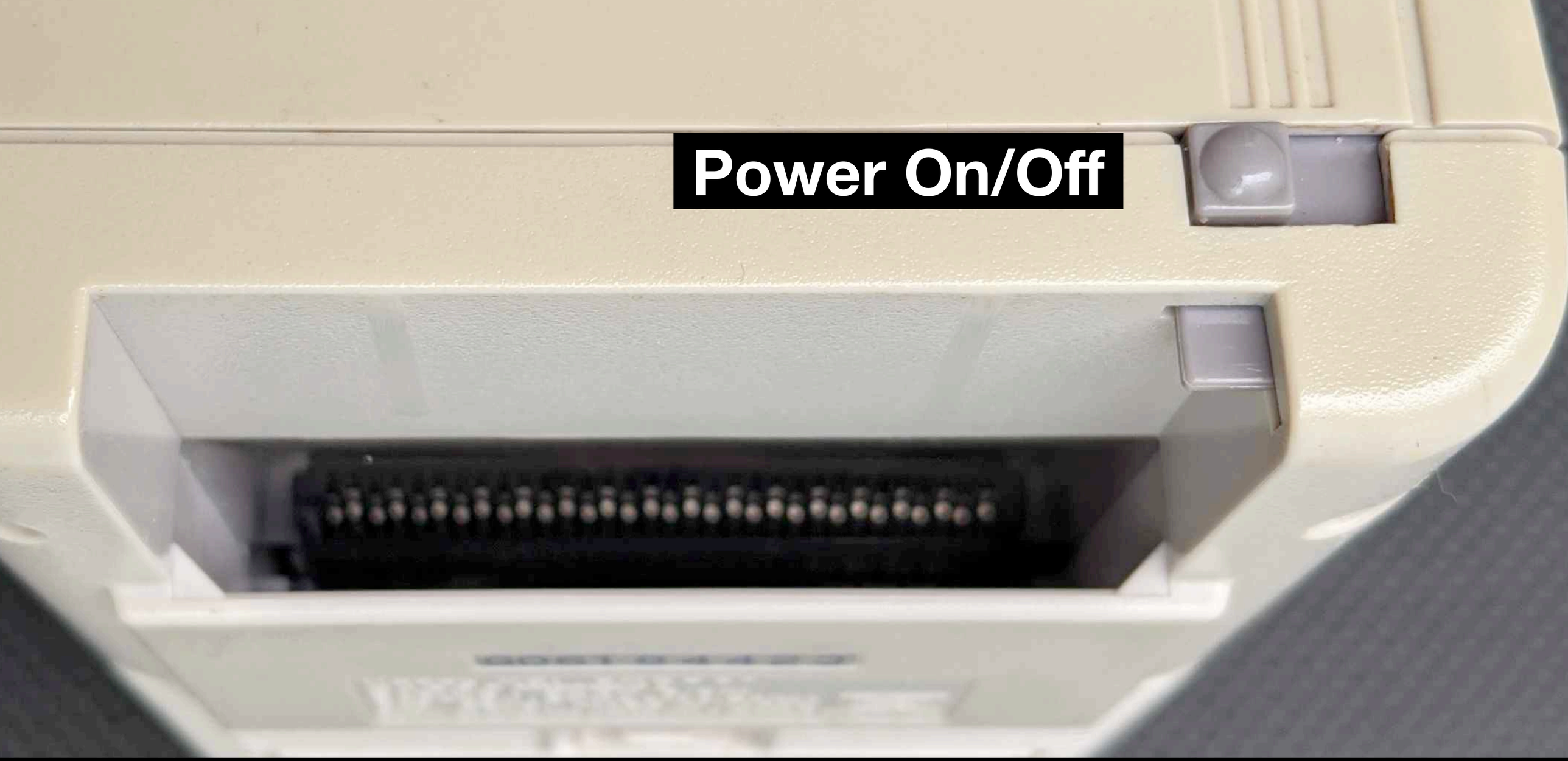


Contrast

Jack to plug in external power

Photograph:

1. https://b13rg.github.io/Gameboy_DMG/

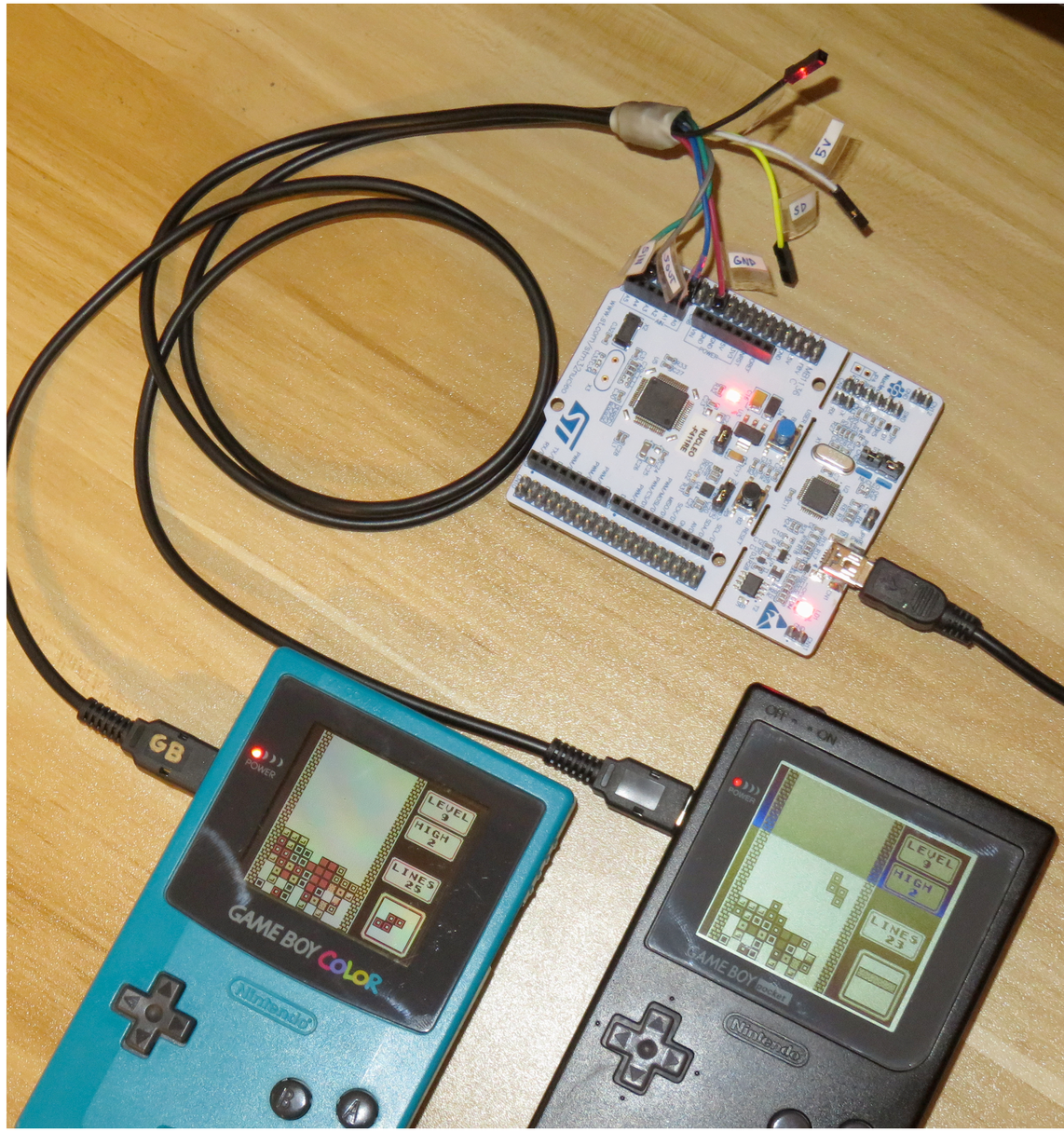


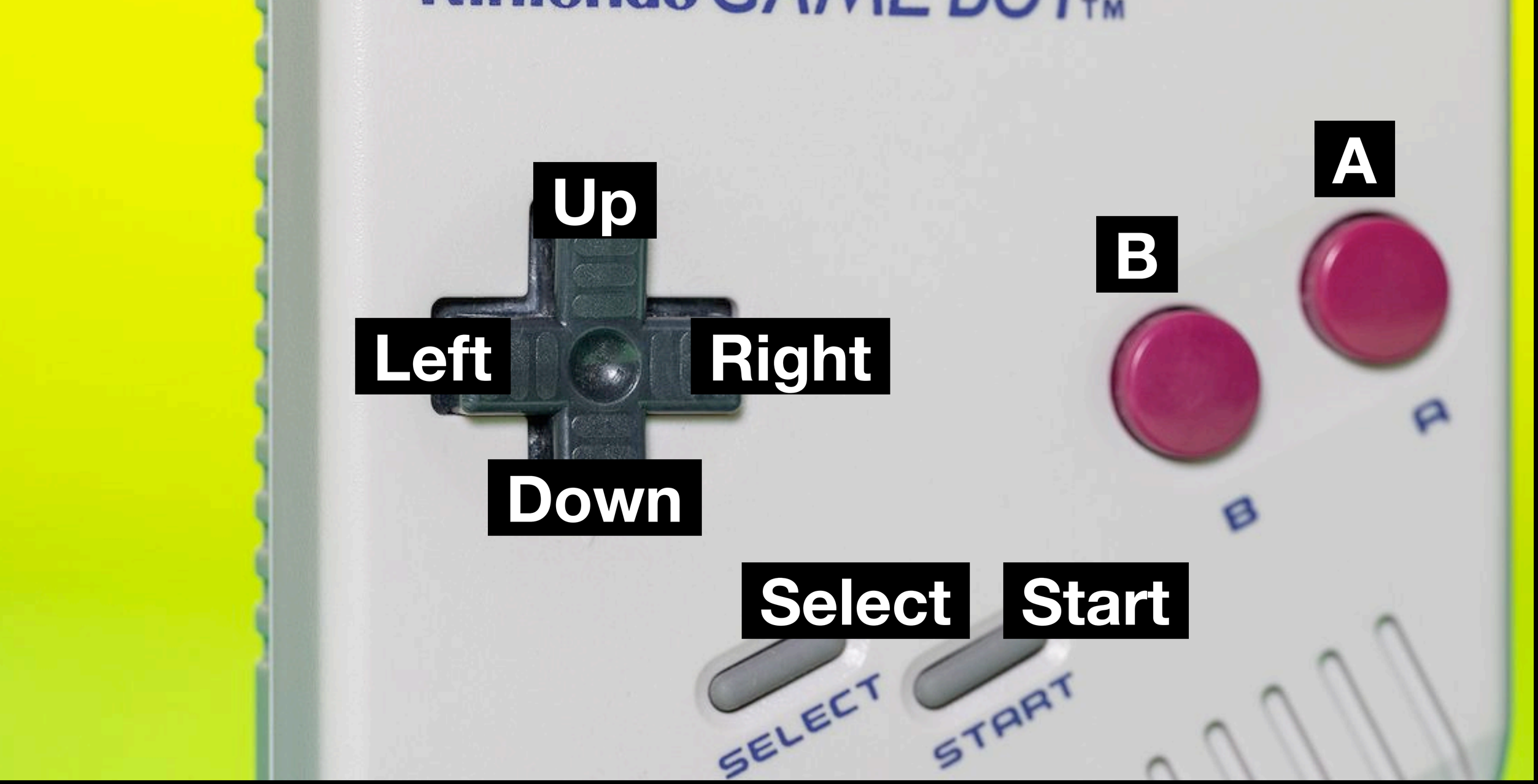
Photograph:

1. https://b13rg.github.io/Gameboy_DMG/

An interface for serial communication and a cartridge slot for game cartridges

Image retired from: https://dhole.github.io/post/gameboy_serial_1/





Photograph:

1. https://www.polygon.com/2019/4/19/18295061/game-boy-history-timeline-tetris-pokemon-nintendo

src/gameboy/**mod.rs**

```
pub enum Button {
    A,
    B,
    Left,
    Right,
    Up,
    Down,
    Start,
    Select,
}
```

src/**desktop.rs**

```
if let Some(virt_keycode) = input.virtual_keycode {
    let button = match virt_keycode {
        VirtualKeyCode::A => Button::A,
        VirtualKeyCode::B => Button::B,
        VirtualKeyCode::Z => Button::Select,
        VirtualKeyCode::X => Button::Start,
        VirtualKeyCode::Left => Button::Left,
        VirtualKeyCode::Right => Button::Right,
        VirtualKeyCode::Down => Button::Down,
        VirtualKeyCode::Up => Button::Up,
        _ => {
            *control_flow = glutin::event_loop::ControlFlow::Poll;
            return;
        }
    };
    match input.state {
        ElementState::Pressed => gb.keydown(button),
        ElementState::Released => gb.keyup(button),
    }
}
```

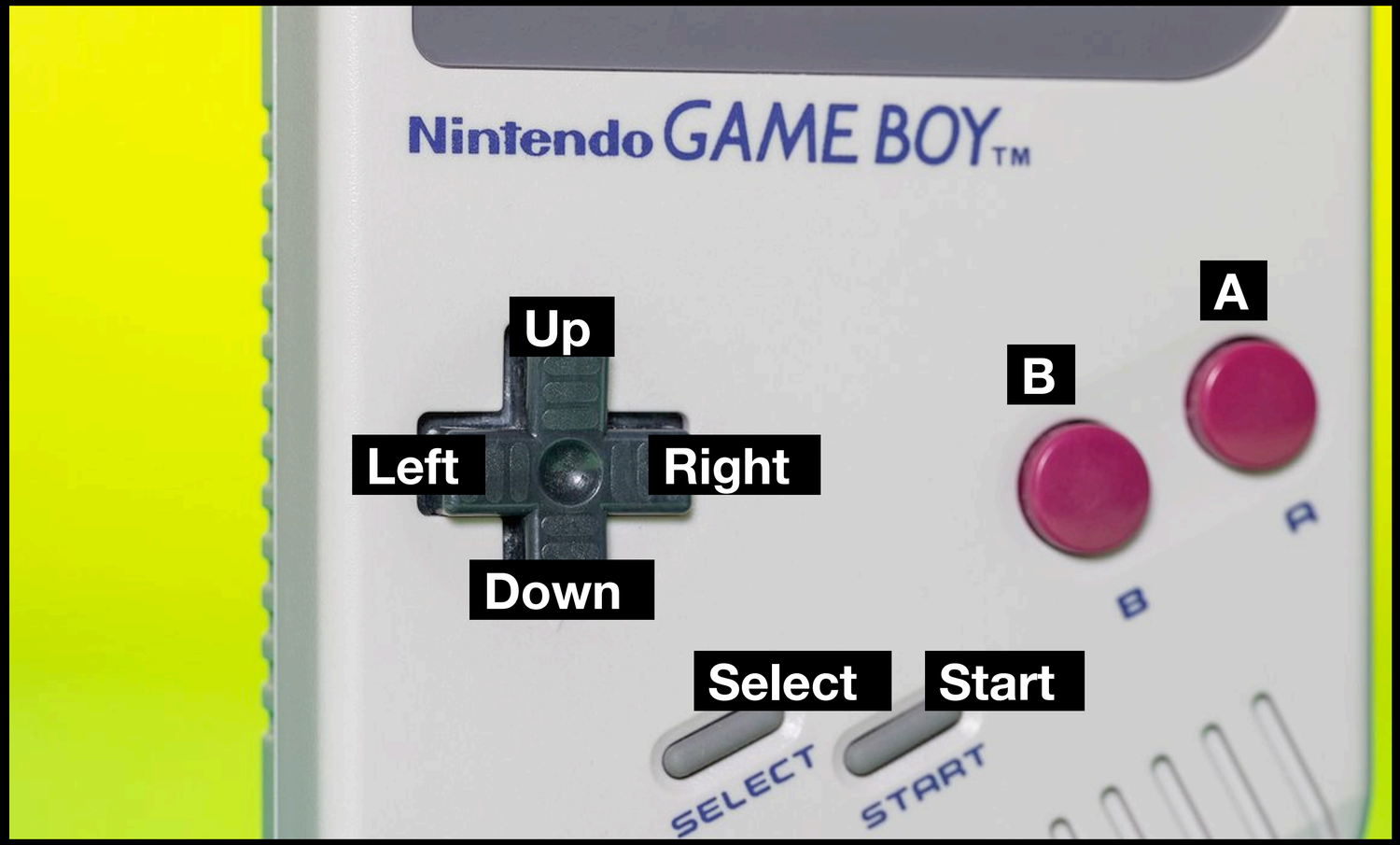
src/**web.rs**

```
let current_key_code: Rc<RefCell<i32>> = Rc::new(RefCell::new(0));
{
    let key_code = current_key_code.clone();
    let closure =
        Closure::<dyn FnMut(_)>::new(move |event: KeyboardEvent| {
            *key_code.borrow_mut() = event.key_code() as i32;
        });
    add_event_listener("keydown", closure.as_ref().unchecked_ref());
    closure.forget();

    let key_code = current_key_code.clone();
    let closure =
        Closure::<dyn FnMut(_)>::new(move |event: KeyboardEvent| {
            *key_code.borrow_mut() = (event.key_code() as i32) * -1;
        });
    add_event_listener("keyup", closure.as_ref().unchecked_ref());
    closure.forget();
}
```

src/**web.rs**

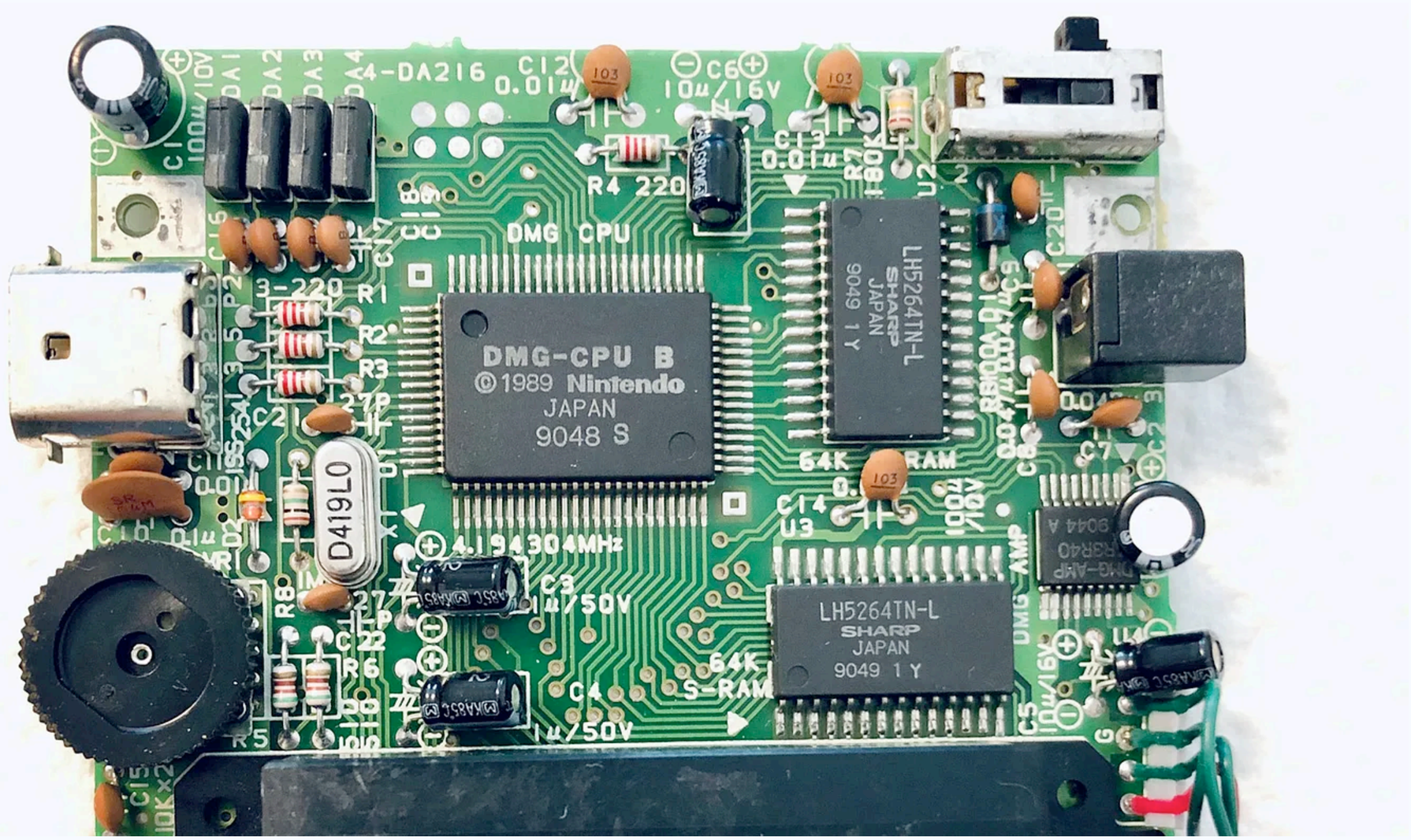
```
let key: RefMut<_> = key_code.borrow_mut();
match *key {
    // A
    65 ⇒ gb.keydown(Button::A), -65 ⇒ gb.keyup(Button::A),
    // S
    83 ⇒ gb.keydown(Button::B), -83 ⇒ gb.keyup(Button::B),
    // Z
    90 ⇒ gb.keydown(Button::Select), -90 ⇒ gb.keyup(Button::Select),
    // X
    88 ⇒ gb.keydown(Button::Start), -88 ⇒ gb.keyup(Button::Start),
    // Left
    37 ⇒ gb.keydown(Button::Left), -37 ⇒ gb.keyup(Button::Left),
    // Right
    39 ⇒ gb.keydown(Button::Right), -39 ⇒ gb.keyup(Button::Right),
    // Up
    38 ⇒ gb.keydown(Button::Up), -38 ⇒ gb.keyup(Button::Up),
    // Down
    40 ⇒ gb.keydown(Button::Down), -40 ⇒ gb.keyup(Button::Down),
    _ ⇒ (),
}
gb.frame();
```

FF00	P1	Joypad (R/W)
5	P15	Select Button Keys
4	P14	Select Direction Keys
3	P13	Input Down / Start
2	P12	Input Up / Select
1	P11	Input Left / Button B
0	P10	Input Right / Button A

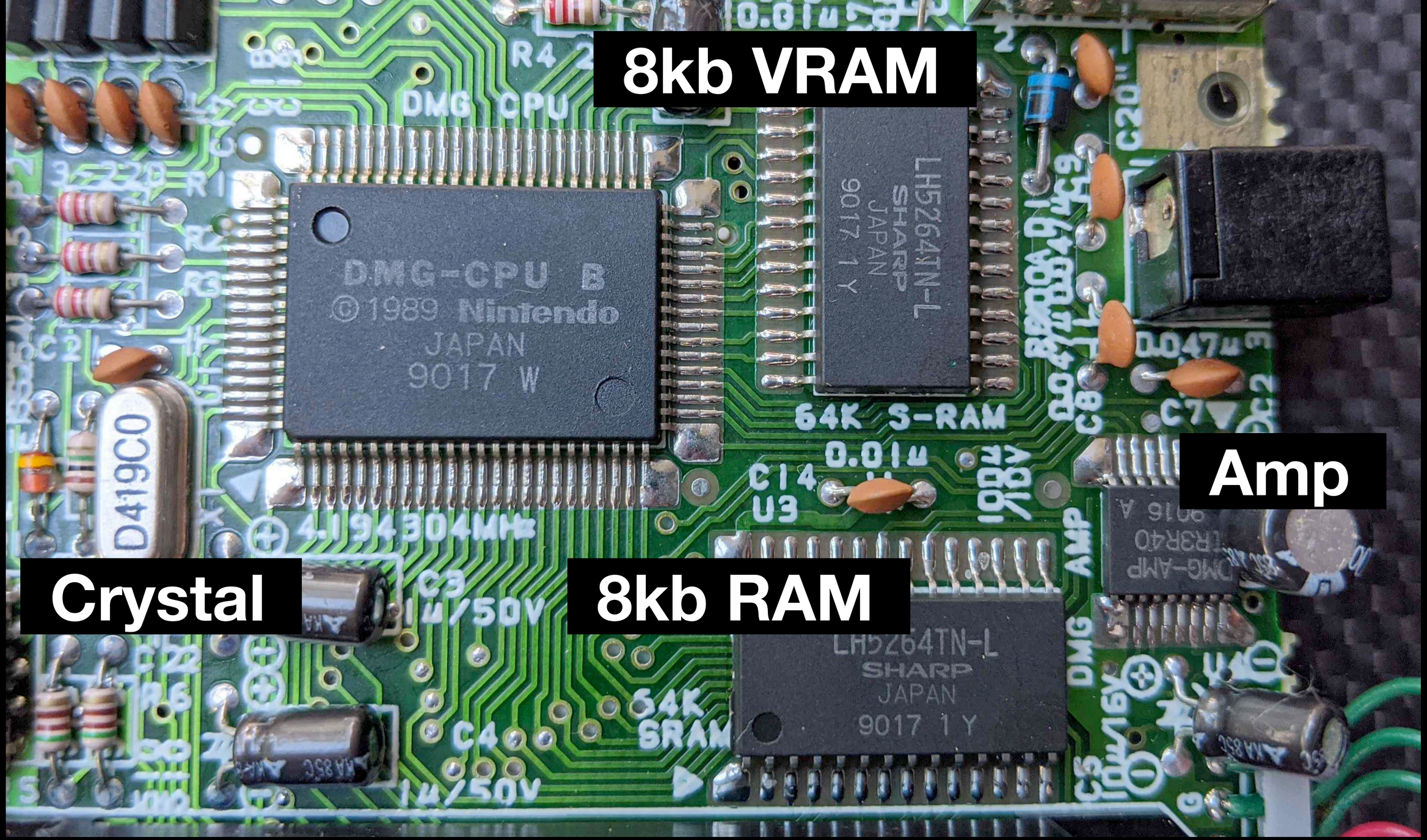


Photograph:

1. https://raphaelstaebler.medium.com/building-a-gameboy-from-scratch-part-2-the-cpu-d6986a5c6c74



Processing



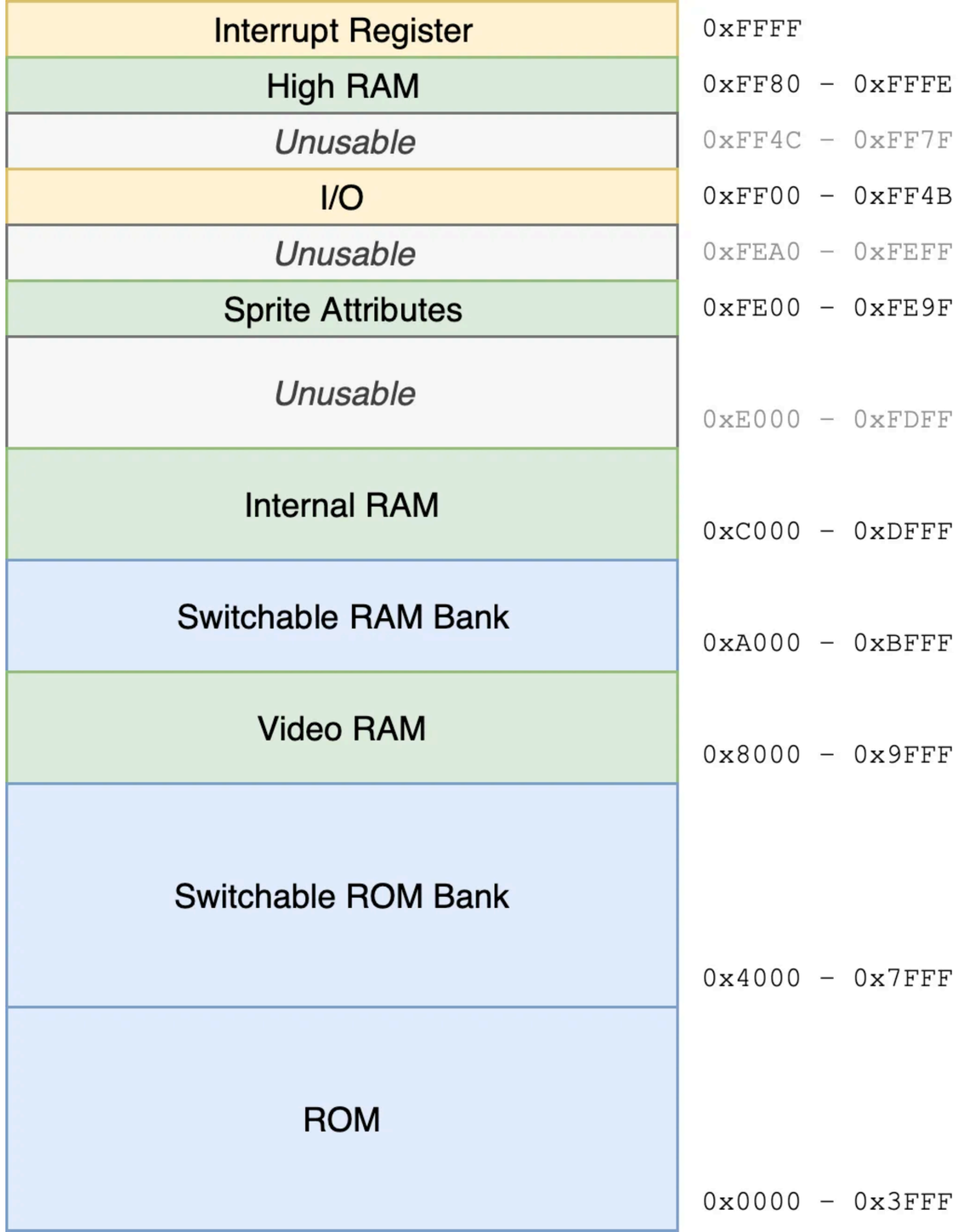
1. Retired from https://b13rg.github.io/Gameboy_DMG/

Memory Map (16-bit address bus)

- 0x0000 - 0x00FF: Boot ROM
- 0x0000 - 0x3FFF: Game ROM Bank 0
- 0x4000 - 0x7FFF: Game ROM Bank N
- 0x8000 - 0x97FF: Tile RAM (VRAM)
- 0x9800 - 0x9FFF: Background Map (VRAM)
- 0xA000 - 0xBFFF: Cartridge RAM
- 0xC000 - 0xDFFF: Working RAM (WRAM)
- 0xE000 - 0xFDFF: Echo RAM
- 0xFE00 - 0xFE9F: OAM (Object Attribute Memory)
- 0xFEA0 - 0xFEFF: Unused
- 0xFF00 - 0xFF7F: I/O Registers
- 0xFF80 - 0xFFFE: High RAM Area (HRAM)
- 0xFFFF: Interrupt Enabled Register

Memory Map

(Simplified)



Retired from raphaelstaebler.medium.com/memory-and-memory-mapped-i-o-of-the-gameboy-part-3-of-a-series-37025b40d89b

Work RAM (8 KB / WRAM)

- General purpose usage
- Four times larger NES Work RAM (2KB)

Display RAM (8 KB / VRAM)

- Contain most of the data to render graphics
- Basically tile data and tile maps

src/**mmu.rs**

```rust
const WRAM_SIZE: usize = 0x8000;
const ZRAM_SIZE: usize = 0x7F;

pub struct MemoryManagementUnit {
    wram: [u8; WRAM_SIZE],
    zram: [u8; ZRAM_SIZE],
    wrambank: usize,
    pub inte: u8,
    pub intf: u8,
    pub input: Input,
    pub gpu: Gpu,
    pub mbc: MemoryBankController,
}
```

src/**mmu.rs**

```
pub fn new(data: Vec<u8>) -> MemoryManagementUnit {
    let mbc = MemoryBankController::new(data);

    let mut res = MemoryManagementUnit {
        wram: [0; WRAM_SIZE],
        zram: [0; ZRAM_SIZE],
        wrambank: 1,
        inte: 0,
        intf: 0,
        input: Input::default(),
        gpu: Gpu::new(),
        mbc,
    };
```

src/**mmu.rs**

Map of the initial memory

```
res.write_byte(0×FF05, 0);
res.write_byte(0×FF06, 0);
res.write_byte(0×FF07, 0);
res.write_byte(0×FF10, 0×80);
res.write_byte(0×FF11, 0×BF);
res.write_byte(0×FF12, 0×F3);
res.write_byte(0×FF14, 0×BF);
res.write_byte(0×FF16, 0×3F);
res.write_byte(0×FF16, 0×3F);
res.write_byte(0×FF17, 0);
res.write_byte(0×FF19, 0×BF);
res.write_byte(0×FF1A, 0×7F);
res.write_byte(0×FF1B, 0×FF);
res.write_byte(0×FF1C, 0×9F);
res.write_byte(0×FF1E, 0×FF);
res.write_byte(0×FF20, 0×FF);
res.write_byte(0×FF21, 0);
res.write_byte(0×FF22, 0);
res.write_byte(0×FF23, 0×BF);
res.write_byte(0×FF24, 0×77);
res.write_byte(0×FF25, 0×F3);
res.write_byte(0×FF26, 0×F1);
res.write_byte(0×FF40, 0×91);
res.write_byte(0×FF42, 0);
res.write_byte(0×FF43, 0);
res.write_byte(0×FF45, 0);
res.write_byte(0×FF47, 0×FC);
res.write_byte(0×FF48, 0×FF);
res.write_byte(0×FF49, 0×FF);
res.write_byte(0×FF4A, 0);
res.write_byte(0×FF4B, 0);
```

src/**mmu.rs**

Note: This memory is incomplete and only works for our game

```
pub fn read_byte(&mut self, address: u16) → u8 {
        match address {
            0×0000..=0×7FFF ⇒ self.mbc.readrom(address),
            0×8000..=0×9FFF ⇒ self.gpu.read_byte(address),
            0×C000..=0×CFFF | 0×E000..=0×EFFF ⇒
self.wram[address as usize & 0×0FFF],
            0×D000..=0×DFFF | 0×F000..=0×FDFF ⇒ {
                self.wram[(self.wrambank * 0×1000) | address as
usize & 0×0FFF]
            }
            0×FE00..=0×FE9F ⇒ self.gpu.read_byte(address),
            0×FF00 ⇒ self.input.read_byte(),
            0×FF0F ⇒ self.intf | 0b11100000,
            0×FF40..=0×FF4F ⇒ self.gpu.read_byte(address),
            0×FF68..=0×FF6B ⇒ self.gpu.read_byte(address),
            0×FF70 ⇒ self.wrambank as u8,
            0×FF80..=0×FFFE ⇒ self.zram[address as usize &
0×007F],
            0×FFFF ⇒ self.inte,
            _ ⇒ 0×FF,
        }
    }
```

Memory Bank Controllers (MBC)

As the Game Boy 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 Game Boy itself).

Retired from https://gbdev.io/pandocs/MBCs.html

Memory Bank Controllers

Is necessary?

Small games of not more than 32 KiB ROM do not require a MBC chip for ROM banking. The ROM is directly mapped to memory at $0000-7FFF. Optionally up to 8 KiB of RAM could be connected at $A000-BFFF, using a discrete logic decoder in place of a full MBC chip.

However our game requires more than 32KiB!

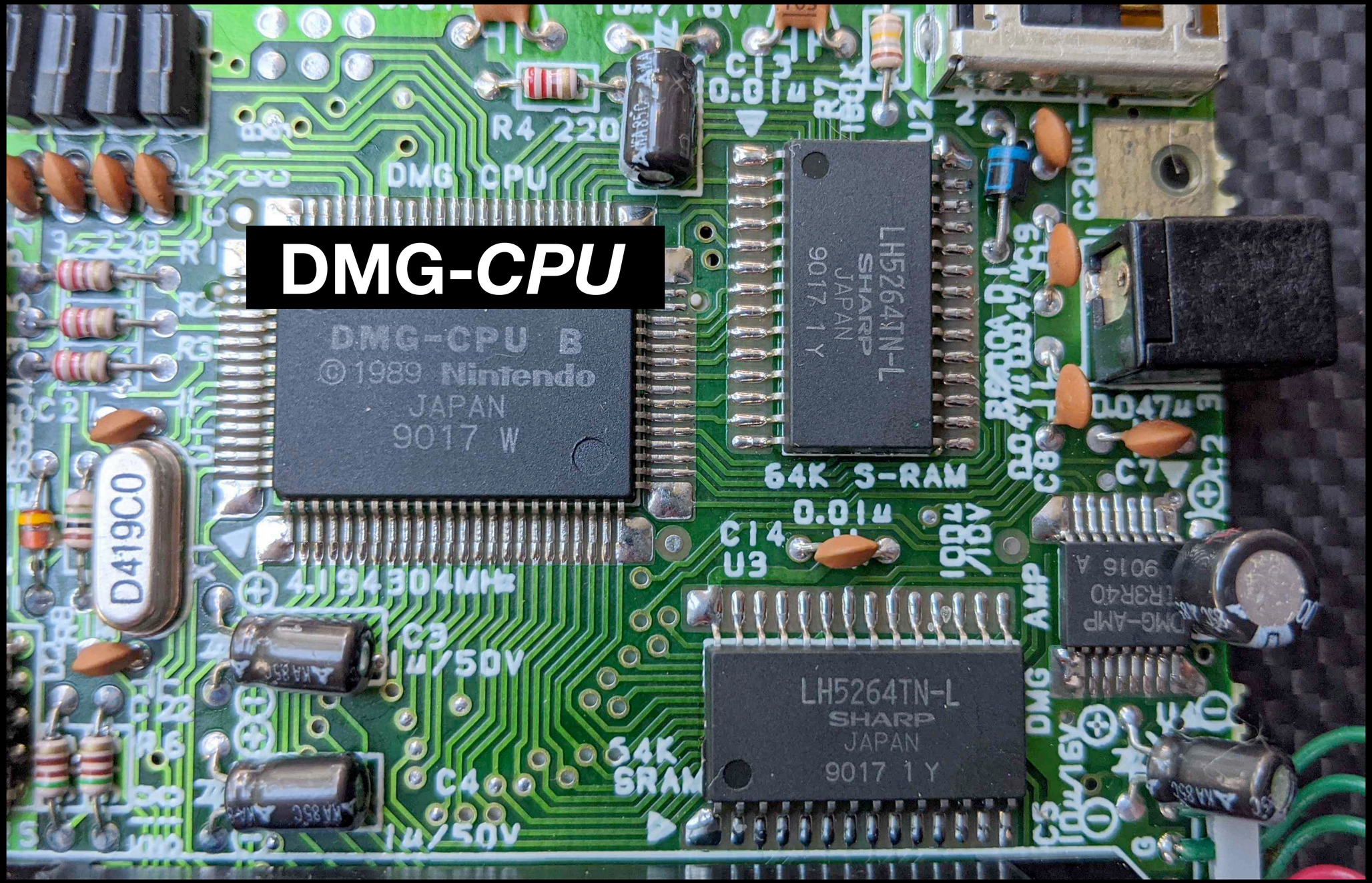
Retired from https://gbdev.io/pandocs/MBCs.html

Just a reminder that a kilobyte and a kibibyte are not the same.

The next available MBC for our game is MBC1.

We will implement a simple "variant" of MBC1 because in our game we only care about some very specific data from the rom

```
pub struct MemoryBankController {
    rom: Vec<u8>,
    rombank: usize,
    rombanks: usize,
}

impl MemoryBankController {
    pub fn new(rom: Vec<u8>) -> Self {
        Self {
            rom,
            rombank: 1,
            rombanks: 8,
        }
    }
    pub fn readrom(&self, a: u16) -> u8 { &0×FF }
    pub fn writerom(&mut self, a: u16, v: u8) {}
}
```



1. Retired from https://b13rg.github.io/Gameboy_DMG/

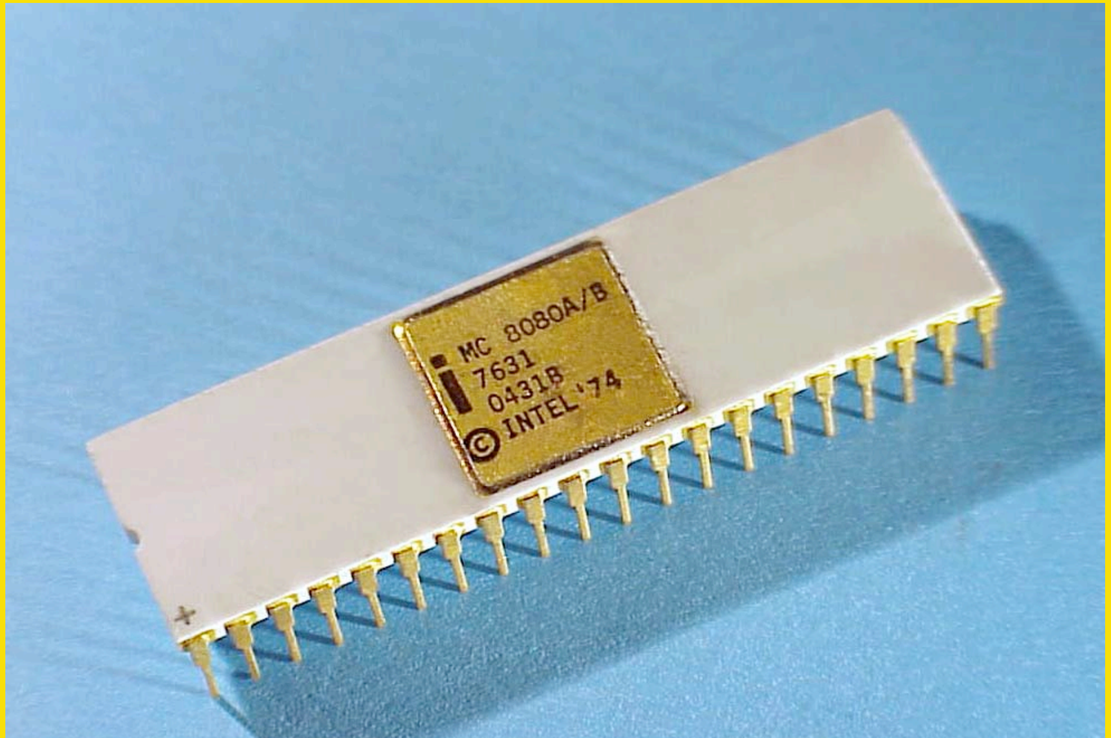
The name for the entire system on chip (SoC) is ***Sharp LR35902*** while the microprocessor name is ***Sharp SM83*** (8-bit CPU core).

* https://github.com/Gekkio/gb-research/tree/main/sm83-cpu-core

The Game Boy CPU is actually a hybrid between the Intel 8080 and the Zilog Z80.

The Z80 was designed to be binary compatible with the already existing Intel 8080. So, the instruction set found in the 8080 was also implemented by the Z80.

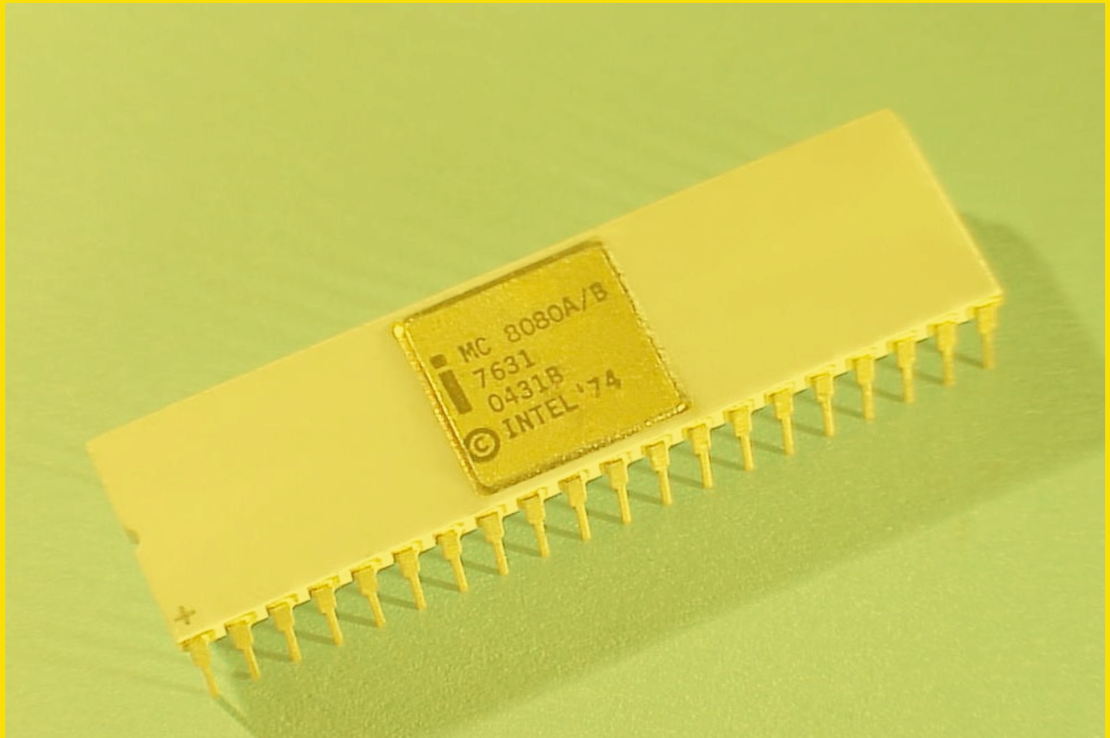
Intel 8080



Zilog Z80

Zilog Z80 introduced functionalities and features with binary compatibility to Intel 8080 (of course a lot those didn’t make through GB CPU).

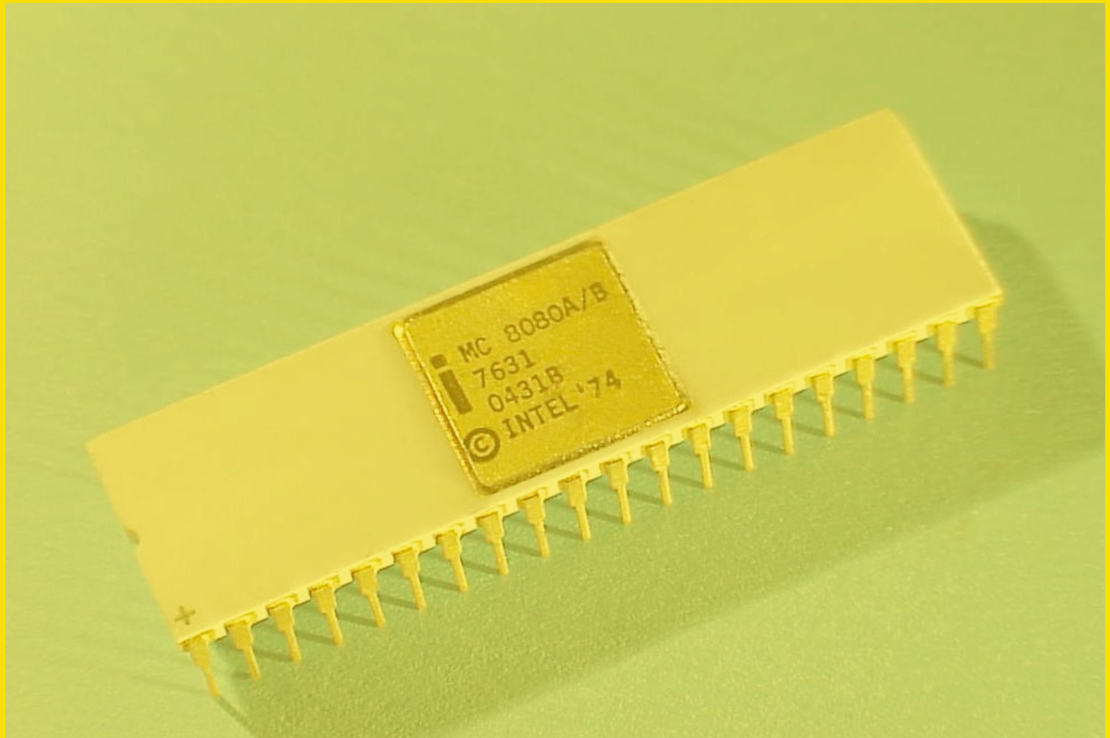
Intel 8080



Zilog Z80

From the nice features, one was very important for GB’s CPU: A special instruction that allowed for an extra 256 instruction set.

Intel 8080



Zilog Z80

If you have any interest in learn more about both microprocessors:

- https://www.zilog.com/docs/z80/um0080.pdf
- http://archive.computerhistory.org/resources/text/Oral_History/Zilog_Z80/102658073.05.01.pdf
- https://manualsbrain.com/en/manuals/2697505/
- https://archive.org/details/Mcs80_85FamilyUsersManual

Intel 8080

Zilog Z80

The processor is connected to most of the elements on the board.

It contains a single address space and doesn’t have extended address modes beyond 8-bits.

It also relies on a program counter to read the states of I/O from special places of mapped memory instead of having special instructions to access I/O input.

The Game Boy CPU (as Intel 8080 and Zilog Z80)
have an extremely simple way of executing a program
(in terms of interpretation of instructions for every byte the processor reads)

1. Bytes from memory are read according to the value held by the **Program Counter (‘PC’)** register
2. Eight 8-bit registers: **A, B, C, D, E, F, H, L**
3. Two 16-bit registers: **PC** and **SP (‘Stack Pointer’)**

src/cpu/**registers.rs**

```rust
pub struct Registers {
    pub a: u8,
    pub b: u8,
    pub c: u8,
    pub d: u8,
    pub e: u8,
    pub h: u8,
    pub l: u8,
    pub f: u8,
    pub pc: u16,
    pub sp: u16,
}
```

16-bit	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

The Flags Register (lower 8 bits of AF register)

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

Contains information about the result of the most recent instruction that has affected flags.

Retired from https://gbdev.io/pandocs/CPU_Registers_and_Flags.html

Gameboy CPU (LR35902) instruction set

Retired from pastraiser.com/cpu/gameboy/gameboy_opcodes.html

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 4 - - - -	LD BC,d16 3 12 - - - -	LD (BC),A 1 8 - - - -	INC BC 1 8 - - - -	INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B,d8 2 8 - - - -	RLCA 1 4 0 0 0 C	LD (a16),SP 3 20 - - - -	ADD HL,BC 1 8 - 0 H C	LD A,(BC) 1 8 - - - -	DEC BC 1 8 - - - -	INC C 1 4 Z 0 H -	DEC C 1 4 Z 1 H -	LD C,d8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x	STOP 0 2 4 - - - -	LD DE,d16 3 12 - - - -	LD (DE),A 1 8 - - - -	INC DE 1 8 - - - -	INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D,d8 2 8 - - - -	RLA 1 4 0 0 0 C	JR r8 2 12 - - - -	ADD HL,DE 1 8 - 0 H C	LD A,(DE) 1 8 - - - -	DEC DE 1 8 - - - -	INC E 1 4 Z 0 H -	DEC E 1 4 Z 1 H -	LD E,d8 2 8 - - - -	RRA 1 4 0 0 0 C
2x	JR NZ,r8 2 12/8 - - - -	LD HL,d16 3 12 - - - -	LD (HL+),A 1 8 - - - -	INC HL 1 8 - - - -	INC H 1 4 Z 0 H -	DEC H 1 4 Z 1 H -	LD H,d8 2 8 - - - -	DAA 1 4 Z - 0 C	JR Z,r8 2 12/8 - - - -	ADD HL,HL 1 8 - 0 H C	LD A,(HL+) 1 8 - - - -	DEC HL 1 8 - - - -	INC L 1 4 Z 0 H -	DEC L 1 4 Z 1 H -	LD L,d8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	JR NC,r8 2 12/8 - - - -	LD SP,d16 3 12 - - - -	LD (HL-),A 1 8 - - - -	INC SP 1 8 - - - -	INC (HL) 1 12 Z 0 H -	DEC (HL) 1 12 Z 1 H -	LD (HL),d8 2 12 - - - -	SCF 1 4 - 0 0 1	JR C,r8 2 12/8 - - - -	ADD HL,SP 1 8 - 0 H C	LD A,(HL-) 1 8 - - - -	DEC SP 1 8 - - - -	INC A 1 4 Z 0 H -	DEC A 1 4 Z 1 H -	LD A,d8 2 8 - - - -	CCF 1 4 - 0 0 C
4x	LD B,B 1 4 - - - -	LD B,C 1 4 - - - -	LD B,D 1 4 - - - -	LD B,E 1 4 - - - -	LD B,H 1 4 - - - -	LD B,L 1 4 - - - -	LD B,(HL) 1 8 - - - -	LD B,A 1 4 - - - -	LD C,B 1 4 - - - -	LD C,C 1 4 - - - -	LD C,D 1 4 - - - -	LD C,E 1 4 - - - -	LD C,H 1 4 - - - -	LD C,L 1 4 - - - -	LD C,(HL) 1 8 - - - -	LD C,A 1 4 - - - -
5x	LD D,B 1 4 - - - -	LD D,C 1 4 - - - -	LD D,D 1 4 - - - -	LD D,E 1 4 - - - -	LD D,H 1 4 - - - -	LD D,L 1 4 - - - -	LD D,(HL) 1 8 - - - -	LD D,A 1 4 - - - -	LD E,B 1 4 - - - -	LD E,C 1 4 - - - -	LD E,D 1 4 - - - -	LD E,E 1 4 - - - -	LD E,H 1 4 - - - -	LD E,L 1 4 - - - -	LD E,(HL) 1 8 - - - -	LD E,A 1 4 - - - -
6x	LD H,B 1 4 - - - -	LD H,C 1 4 - - - -	LD H,D 1 4 - - - -	LD H,E 1 4 - - - -	LD H,H 1 4 - - - -	LD H,L 1 4 - - - -	LD H,(HL) 1 8 - - - -	LD H,A 1 4 - - - -	LD L,B 1 4 - - - -	LD L,C 1 4 - - - -	LD L,D 1 4 - - - -	LD L,E 1 4 - - - -	LD L,H 1 4 - - - -	LD L,L 1 4 - - - -	LD L,(HL) 1 8 - - - -	LD L,A 1 4 - - - -
7x	LD (HL),B 1 8 - - - -	LD (HL),C 1 8 - - - -	LD (HL),D 1 8 - - - -	LD (HL),E 1 8 - - - -	LD (HL),H 1 8 - - - -	LD (HL),L 1 8 - - - -	HALT 1 4 - - - -	LD (HL),A 1 8 - - - -	LD A,B 1 4 - - - -	LD A,C 1 4 - - - -	LD A,D 1 4 - - - -	LD A,E 1 4 - - - -	LD A,H 1 4 - - - -	LD A,L 1 4 - - - -	LD A,(HL) 1 8 - - - -	LD A,A 1 4 - - - -
8x	ADD A,B 1 4 Z 0 H C	ADD A,C 1 4 Z 0 H C	ADD A,D 1 4 Z 0 H C	ADD A,E 1 4 Z 0 H C	ADD A,H 1 4 Z 0 H C	ADD A,L 1 4 Z 0 H C	ADD A,(HL) 1 8 Z 0 H C	ADD A,A 1 4 Z 0 H C	ADC A,B 1 4 Z 0 H C	ADC A,C 1 4 Z 0 H C	ADC A,D 1 4 Z 0 H C	ADC A,E 1 4 Z 0 H C	ADC A,H 1 4 Z 0 H C	ADC A,L 1 4 Z 0 H C	ADC A,(HL) 1 8 Z 0 H C	ADC A,A 1 4 Z 0 H C
9x	SUB B 1 4 Z 1 H C	SUB C 1 4 Z 1 H C	SUB D 1 4 Z 1 H C	SUB E 1 4 Z 1 H C	SUB H 1 4 Z 1 H C	SUB L 1 4 Z 1 H C	SUB (HL) 1 8 Z 1 H C	SUB A 1 4 Z 1 H C	SBC A,B 1 4 Z 1 H C	SBC A,C 1 4 Z 1 H C	SBC A,D 1 4 Z 1 H C	SBC A,E 1 4 Z 1 H C	SBC A,H 1 4 Z 1 H C	SBC A,L 1 4 Z 1 H C	SBC A,(HL) 1 8 Z 1 H C	SBC A,A 1 4 Z 1 H C
Ax	AND B 1 4 Z 0 1 0	AND C 1 4 Z 0 1 0	AND D 1 4 Z 0 1 0	AND E 1 4 Z 0 1 0	AND H 1 4 Z 0 1 0	AND L 1 4 Z 0 1 0	AND (HL) 1 8 Z 0 1 0	AND A 1 4 Z 0 1 0	XOR B 1 4 Z 0 0 0	XOR C 1 4 Z 0 0 0	XOR D 1 4 Z 0 0 0	XOR E 1 4 Z 0 0 0	XOR H 1 4 Z 0 0 0	XOR L 1 4 Z 0 0 0	XOR (HL) 1 8 Z 0 0 0	XOR A 1 4 Z 0 0 0
Bx	OR B 1 4 Z 0 0 0	OR C 1 4 Z 0 0 0	OR D 1 4 Z 0 0 0	OR E 1 4 Z 0 0 0	OR H 1 4 Z 0 0 0	OR L 1 4 Z 0 0 0	OR (HL) 1 8 Z 0 0 0	OR A 1 4 Z 0 0 0	CP B 1 4 Z 1 H C	CP C 1 4 Z 1 H C	CP D 1 4 Z 1 H C	CP E 1 4 Z 1 H C	CP H 1 4 Z 1 H C	CP L 1 4 Z 1 H C	CP (HL) 1 8 Z 1 H C	CP A 1 4 Z 1 H C
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ,a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ,a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A,d8 2 8 Z 0 H C	RST 00H 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z,a16 3 16/12 - - - -	PREFIX CB 1 4 - - - -	CALL Z,a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A,d8 2 8 Z 0 H C	RST 08H 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC,a16 3 16/12 - - - -		CALL NC,a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB d8 2 8 Z 1 H C	RST 10H 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C,a16 3 16/12 - - - -		CALL C,a16 3 24/12 - - - -		SBC A,d8 2 8 Z 1 H C	RST 18H 1 16 - - - -
Ex	LDH (a8),A 2 12 - - - -	POP HL 1 12 - - - -	LD (C),A 2 8 - - - -			PUSH HL 1 16 - - - -	AND d8 2 8 Z 0 1 0	RST 20H 1 16 - - - -	ADD SP,r8 2 16 0 0 H C	JP (HL) 1 4 - - - -	LD (a16),A 3 16 - - - -				XOR d8 2 8 Z 0 0 0	RST 28H 1 16 - - - -
Fx	LDH A,(a8) 2 12 - - - -	POP AF 1 12 Z N H C	LD A,(C) 2 8 - - - -	DI 1 4 - - - -		PUSH AF 1 16 - - - -	OR d8 2 8 Z 0 0 0	RST 30H 1 16 - - - -	LD HL,SP+r8 2 12 0 0 H C	LD SP,HL 1 8 - - - -	LD A,(a16) 3 16 - - - -	EI 1 4 - - - -			CP d8 2 8 Z 1 H C	RST 38H 1 16 - - - -

Prefix CB

Retired from pastraiser.com/cpu/gameboy/gameboy_opcodes.html

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 8 Z 0 0 C	RLC C 2 8 Z 0 0 C	RLC D 2 8 Z 0 0 C	RLC E 2 8 Z 0 0 C	RLC H 2 8 Z 0 0 C	RLC L 2 8 Z 0 0 C	RLC (HL) 2 16 Z 0 0 C	RLC A 2 8 Z 0 0 C	RRC B 2 8 Z 0 0 C	RRC C 2 8 Z 0 0 C	RRC D 2 8 Z 0 0 C	RRC E 2 8 Z 0 0 C	RRC H 2 8 Z 0 0 C	RRC L 2 8 Z 0 0 C	RRC (HL) 2 16 Z 0 0 C	RRC A 2 8 Z 0 0 C
1x	RL B 2 8 Z 0 0 C	RL C 2 8 Z 0 0 C	RL D 2 8 Z 0 0 C	RL E 2 8 Z 0 0 C	RL H 2 8 Z 0 0 C	RL L 2 8 Z 0 0 C	RL (HL) 2 16 Z 0 0 C	RL A 2 8 Z 0 0 C	RR B 2 8 Z 0 0 C	RR C 2 8 Z 0 0 C	RR D 2 8 Z 0 0 C	RR E 2 8 Z 0 0 C	RR H 2 8 Z 0 0 C	RR L 2 8 Z 0 0 C	RR (HL) 2 16 Z 0 0 C	RR A 2 8 Z 0 0 C
2x	SLA B 2 8 Z 0 0 C	SLA C 2 8 Z 0 0 C	SLA D 2 8 Z 0 0 C	SLA E 2 8 Z 0 0 C	SLA H 2 8 Z 0 0 C	SLA L 2 8 Z 0 0 C	SLA (HL) 2 16 Z 0 0 C	SLA A 2 8 Z 0 0 C	SRA B 2 8 Z 0 0 0	SRA C 2 8 Z 0 0 0	SRA D 2 8 Z 0 0 0	SRA E 2 8 Z 0 0 0	SRA H 2 8 Z 0 0 0	SRA L 2 8 Z 0 0 0	SRA (HL) 2 16 Z 0 0 0	SRA A 2 8 Z 0 0 0
3x	SWAP B 2 8 Z 0 0 0	SWAP C 2 8 Z 0 0 0	SWAP D 2 8 Z 0 0 0	SWAP E 2 8 Z 0 0 0	SWAP H 2 8 Z 0 0 0	SWAP L 2 8 Z 0 0 0	SWAP (HL) 2 16 Z 0 0 0	SWAP A 2 8 Z 0 0 0	SRL B 2 8 Z 0 0 C	SRL C 2 8 Z 0 0 C	SRL D 2 8 Z 0 0 C	SRL E 2 8 Z 0 0 C	SRL H 2 8 Z 0 0 C	SRL L 2 8 Z 0 0 C	SRL (HL) 2 16 Z 0 0 C	SRL A 2 8 Z 0 0 C
4x	BIT 0,B 2 8 Z 0 1 –	BIT 0,C 2 8 Z 0 1 –	BIT 0,D 2 8 Z 0 1 –	BIT 0,E 2 8 Z 0 1 –	BIT 0,H 2 8 Z 0 1 –	BIT 0,L 2 8 Z 0 1 –	BIT 0,(HL) 2 16 Z 0 1 –	BIT 0,A 2 8 Z 0 1 –	BIT 1,B 2 8 Z 0 1 –	BIT 1,C 2 8 Z 0 1 –	BIT 1,D 2 8 Z 0 1 –	BIT 1,E 2 8 Z 0 1 –	BIT 1,H 2 8 Z 0 1 –	BIT 1,L 2 8 Z 0 1 –	BIT 1,(HL) 2 16 Z 0 1 –	BIT 1,A 2 8 Z 0 1 –
5x	BIT 2,B 2 8 Z 0 1 –	BIT 2,C 2 8 Z 0 1 –	BIT 2,D 2 8 Z 0 1 –	BIT 2,E 2 8 Z 0 1 –	BIT 2,H 2 8 Z 0 1 –	BIT 2,L 2 8 Z 0 1 –	BIT 2,(HL) 2 16 Z 0 1 –	BIT 2,A 2 8 Z 0 1 –	BIT 3,B 2 8 Z 0 1 –	BIT 3,C 2 8 Z 0 1 –	BIT 3,D 2 8 Z 0 1 –	BIT 3,E 2 8 Z 0 1 –	BIT 3,H 2 8 Z 0 1 –	BIT 3,L 2 8 Z 0 1 –	BIT 3,(HL) 2 16 Z 0 1 –	BIT 3,A 2 8 Z 0 1 –
6x	BIT 4,B 2 8 Z 0 1 –	BIT 4,C 2 8 Z 0 1 –	BIT 4,D 2 8 Z 0 1 –	BIT 4,E 2 8 Z 0 1 –	BIT 4,H 2 8 Z 0 1 –	BIT 4,L 2 8 Z 0 1 –	BIT 4,(HL) 2 16 Z 0 1 –	BIT 4,A 2 8 Z 0 1 –	BIT 5,B 2 8 Z 0 1 –	BIT 5,C 2 8 Z 0 1 –	BIT 5,D 2 8 Z 0 1 –	BIT 5,E 2 8 Z 0 1 –	BIT 5,H 2 8 Z 0 1 –	BIT 5,L 2 8 Z 0 1 –	BIT 5,(HL) 2 16 Z 0 1 –	BIT 5,A 2 8 Z 0 1 –
7x	BIT 6,B 2 8 Z 0 1 –	BIT 6,C 2 8 Z 0 1 –	BIT 6,D 2 8 Z 0 1 –	BIT 6,E 2 8 Z 0 1 –	BIT 6,H 2 8 Z 0 1 –	BIT 6,L 2 8 Z 0 1 –	BIT 6,(HL) 2 16 Z 0 1 –	BIT 6,A 2 8 Z 0 1 –	BIT 7,B 2 8 Z 0 1 –	BIT 7,C 2 8 Z 0 1 –	BIT 7,D 2 8 Z 0 1 –	BIT 7,E 2 8 Z 0 1 –	BIT 7,H 2 8 Z 0 1 –	BIT 7,L 2 8 Z 0 1 –	BIT 7,(HL) 2 16 Z 0 1 –	BIT 7,A 2 8 Z 0 1 –
8x	RES 0,B 2 8 – – – –	RES 0,C 2 8 – – – –	RES 0,D 2 8 – – – –	RES 0,E 2 8 – – – –	RES 0,H 2 8 – – – –	RES 0,L 2 8 – – – –	RES 0,(HL) 2 16 – – – –	RES 0,A 2 8 – – – –	RES 1,B 2 8 – – – –	RES 1,C 2 8 – – – –	RES 1,D 2 8 – – – –	RES 1,E 2 8 – – – –	RES 1,H 2 8 – – – –	RES 1,L 2 8 – – – –	RES 1,(HL) 2 16 – – – –	RES 1,A 2 8 – – – –
9x	RES 2,B 2 8 – – – –	RES 2,C 2 8 – – – –	RES 2,D 2 8 – – – –	RES 2,E 2 8 – – – –	RES 2,H 2 8 – – – –	RES 2,L 2 8 – – – –	RES 2,(HL) 2 16 – – – –	RES 2,A 2 8 – – – –	RES 3,B 2 8 – – – –	RES 3,C 2 8 – – – –	RES 3,D 2 8 – – – –	RES 3,E 2 8 – – – –	RES 3,H 2 8 – – – –	RES 3,L 2 8 – – – –	RES 3,(HL) 2 16 – – – –	RES 3,A 2 8 – – – –
Ax	RES 4,B 2 8 – – – –	RES 4,C 2 8 – – – –	RES 4,D 2 8 – – – –	RES 4,E 2 8 – – – –	RES 4,H 2 8 – – – –	RES 4,L 2 8 – – – –	RES 4,(HL) 2 16 – – – –	RES 4,A 2 8 – – – –	RES 5,B 2 8 – – – –	RES 5,C 2 8 – – – –	RES 5,D 2 8 – – – –	RES 5,E 2 8 – – – –	RES 5,H 2 8 – – – –	RES 5,L 2 8 – – – –	RES 5,(HL) 2 16 – – – –	RES 5,A 2 8 – – – –
Bx	RES 6,B 2 8 – – – –	RES 6,C 2 8 – – – –	RES 6,D 2 8 – – – –	RES 6,E 2 8 – – – –	RES 6,H 2 8 – – – –	RES 6,L 2 8 – – – –	RES 6,(HL) 2 16 – – – –	RES 6,A 2 8 – – – –	RES 7,B 2 8 – – – –	RES 7,C 2 8 – – – –	RES 7,D 2 8 – – – –	RES 7,E 2 8 – – – –	RES 7,H 2 8 – – – –	RES 7,L 2 8 – – – –	RES 7,(HL) 2 16 – – – –	RES 7,A 2 8 – – – –
Cx	SET 0,B 2 8 – – – –	SET 0,C 2 8 – – – –	SET 0,D 2 8 – – – –	SET 0,E 2 8 – – – –	SET 0,H 2 8 – – – –	SET 0,L 2 8 – – – –	SET 0,(HL) 2 16 – – – –	SET 0,A 2 8 – – – –	SET 1,B 2 8 – – – –	SET 1,C 2 8 – – – –	SET 1,D 2 8 – – – –	SET 1,E 2 8 – – – –	SET 1,H 2 8 – – – –	SET 1,L 2 8 – – – –	SET 1,(HL) 2 16 – – – –	SET 1,A 2 8 – – – –
Dx	SET 2,B 2 8 – – – –	SET 2,C 2 8 – – – –	SET 2,D 2 8 – – – –	SET 2,E 2 8 – – – –	SET 2,H 2 8 – – – –	SET 2,L 2 8 – – – –	SET 2,(HL) 2 16 – – – –	SET 2,A 2 8 – – – –	SET 3,B 2 8 – – – –	SET 3,C 2 8 – – – –	SET 3,D 2 8 – – – –	SET 3,E 2 8 – – – –	SET 3,H 2 8 – – – –	SET 3,L 2 8 – – – –	SET 3,(HL) 2 16 – – – –	SET 3,A 2 8 – – – –
Ex	SET 4,B 2 8 – – – –	SET 4,C 2 8 – – – –	SET 4,D 2 8 – – – –	SET 4,E 2 8 – – – –	SET 4,H 2 8 – – – –	SET 4,L 2 8 – – – –	SET 4,(HL) 2 16 – – – –	SET 4,A 2 8 – – – –	SET 5,B 2 8 – – – –	SET 5,C 2 8 – – – –	SET 5,D 2 8 – – – –	SET 5,E 2 8 – – – –	SET 5,H 2 8 – – – –	SET 5,L 2 8 – – – –	SET 5,(HL) 2 16 – – – –	SET 5,A 2 8 – – – –
Fx	SET 6,B 2 8 – – – –	SET 6,C 2 8 – – – –	SET 6,D 2 8 – – – –	SET 6,E 2 8 – – – –	SET 6,H 2 8 – – – –	SET 6,L 2 8 – – – –	SET 6,(HL) 2 16 – – – –	SET 6,A 2 8 – – – –	SET 7,B 2 8 – – – –	SET 7,C 2 8 – – – –	SET 7,D 2 8 – – – –	SET 7,E 2 8 – – – –	SET 7,H 2 8 – – – –	SET 7,L 2 8 – – – –	SET 7,(HL) 2 16 – – – –	SET 7,A 2 8 – – – –

	x0	x1	
0x	NOP 1 4 - - - -	LD BC,d16 3 12 - - - -	LD (1 - -
1x	STOP 0 2 4	LD DE,d16 3 12	LD (1

```
0×00 ⇒ 1,
0×01 ⇒ {
    let value = self.memory.read_word(self.registers.pc);
    self.registers.pc = self.registers.pc.wrapping_add(2);
    cpu.registers.b = (value >> 8) as u8;
    cpu.registers.c = (value & 0×00FF) as u8;
    3
}
```

src/cpu/**mod.rs**

```
let byte = self.memory.read_byte(self.registers.pc);
self.registers.pc = self.registers.pc.wrapping_add(1);
let ticks = match byte {
    0x00 => 1,
    0x01 => { ld::bcnn(self); 3 }
    0x02 => { ld::bcm_a(self); 2 }
    0x03 => { data::incbc(self); 2 }

    // … (rest of the instructions)

    0xFF => { stack::rst(self, 0x38); 4 }
    _ => { panic!("{:#06x} not implemented", op);
}
self.memory.cycle(ticks * 4);
```

Interrupts

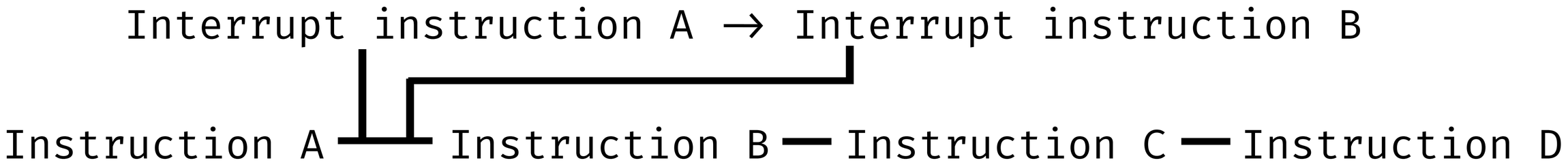
Interrupt register is just an 8 bit value consisting of flags (single bits) to indicate what kind of interrupts are enabled.

It interrupt the current program flow in response to certain events.

Interrupts

We saw the CPU executes the instruction the program counter is pointing to. However whenever an interrupt is put in action it will move the PC to the stack and run a opcodes based on that interrupt.

The Game Boy has interrupts for modules besides the CPU (like GPU and Button inputs).



Output

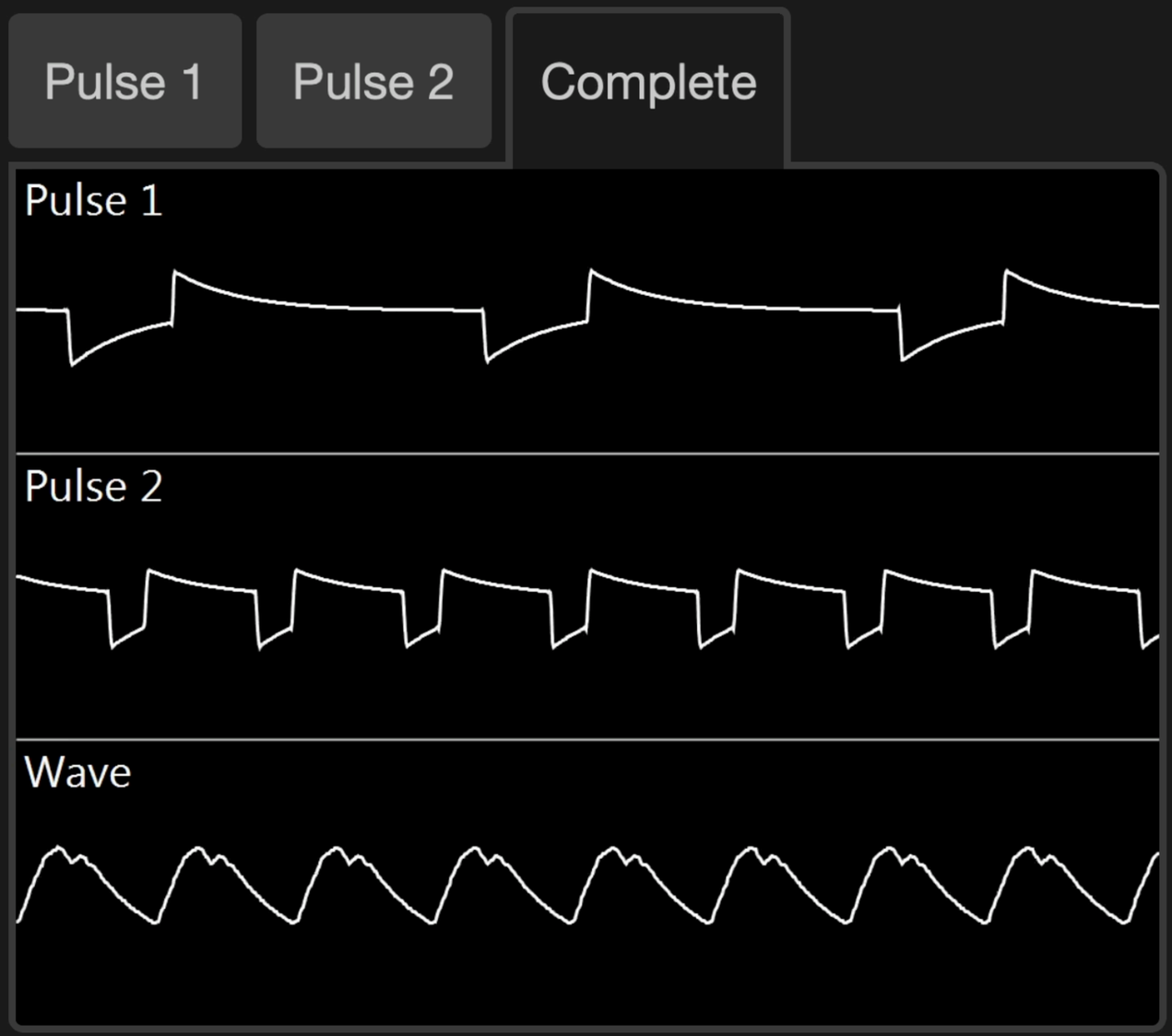
* Super Mario Land is a 1989 platform video game, and it was developed and published by Nintendo

Internal speaker with
mono sound output

headphone jack with support
for stereo sound

The audio system is carried out by the **Audio Processing Unit** (APU). APU is a Programmable Sound Generator with four channels.

Pulse waves have a very distinct *beep* sound that is mainly used for **melody or sound effects**.



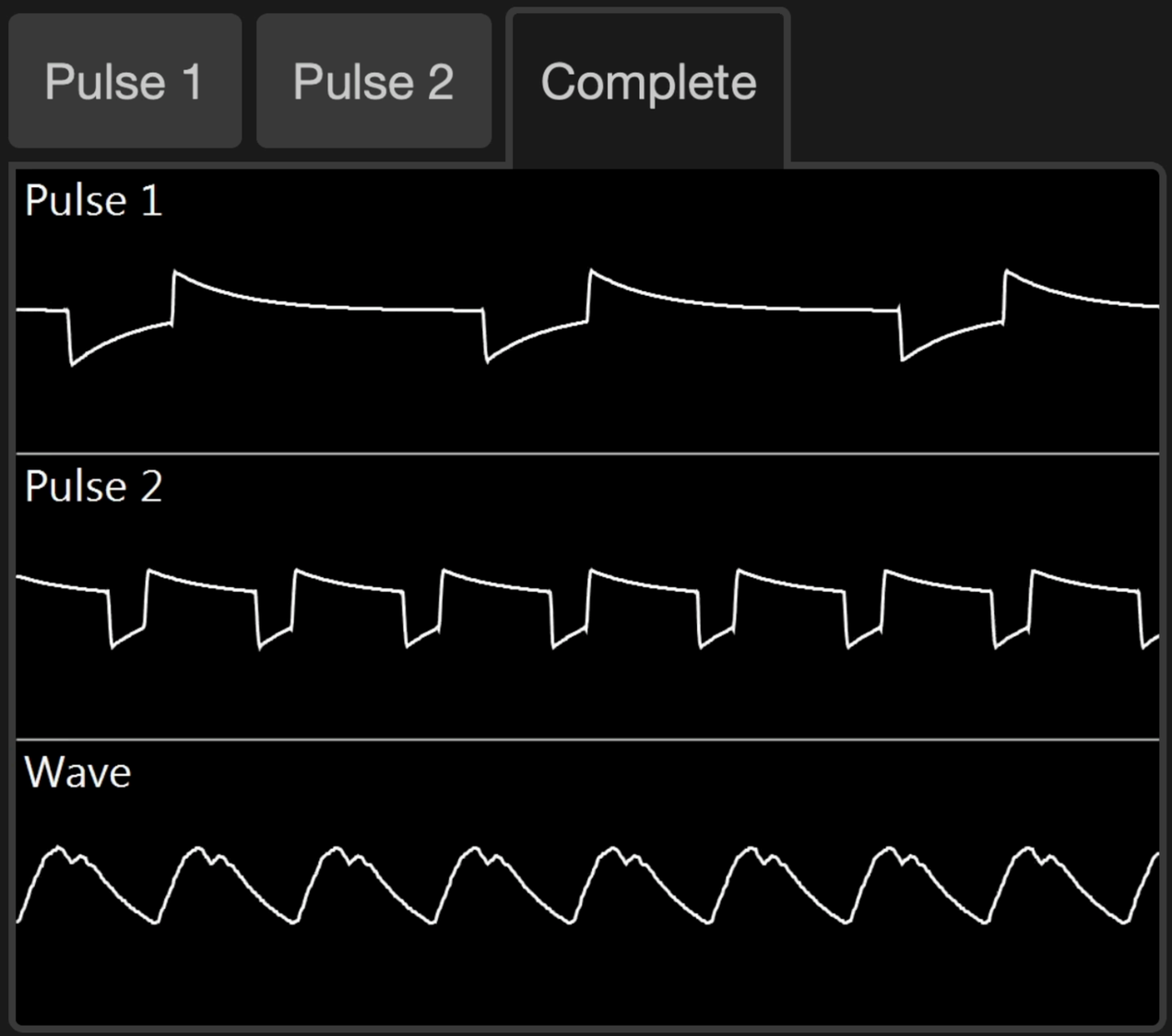
Oscilloscope view of all audio channels.

Pokemon Red/Blue (1996).

Retired from: https://www.copetti.org/writings/consoles/game-boy

The audio system is carried out by the **Audio Processing Unit** (APU). APU is a Programmable Sound Generator with four channels.

Pulse waves have a very distinct *beep* sound that is mainly used for **melody or sound effects**.



Oscilloscope view of all audio channels.

Pokemon Red/Blue (1996).

Retired from: https://www.copetti.org/writings/consoles/game-boy

160x144 pixel display

Liquid crystal screen (LCD)

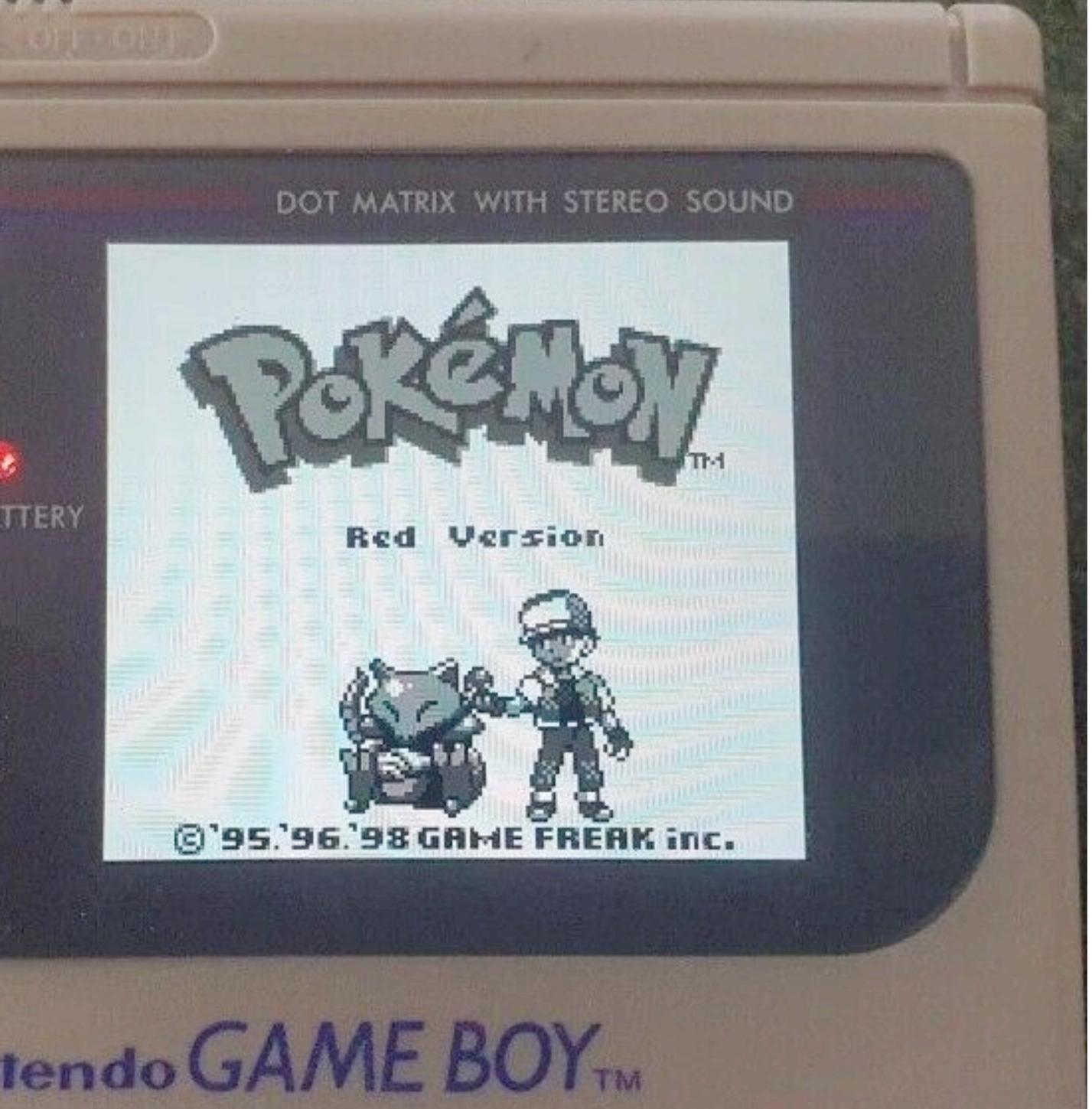
Original Game Boy Game Boy Pocket/Light

Original Game Boy Hex / Binary	0x0 00	0x1 01	0x2 10	0x3 11
Game Boy Pocket/Light Hex / Binary	0x0 00	0x1 01	0x2 10	0x3 11

The display uses a monochrome 4-shade palette.

Because the non-backlit LCD display background is greenish, this results in a "greenscale" graphic display.

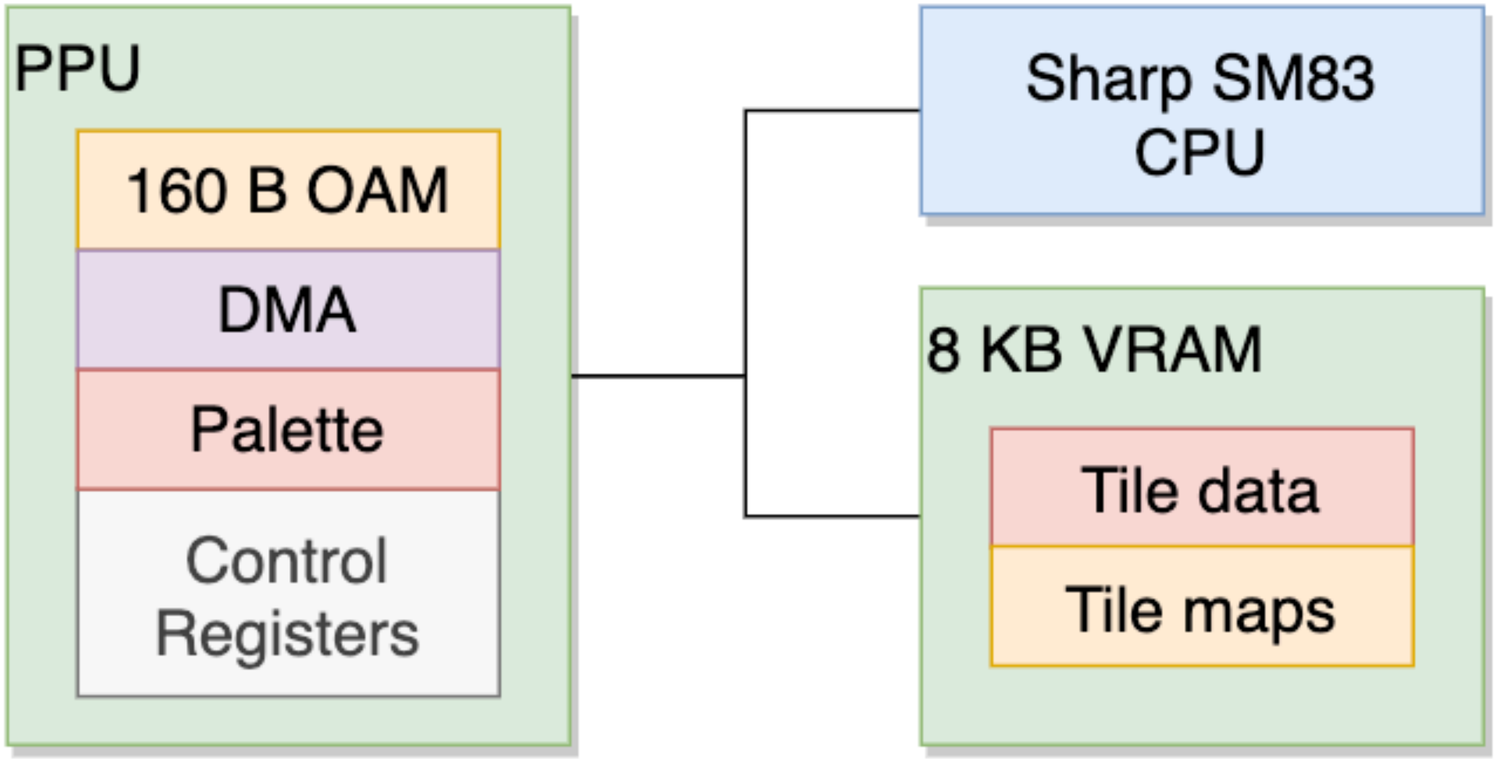
Retired from: https://en.wikipedia.org/wiki/List_of_video_game_console_palettes



For example:

This is what would look like if the classic gameboy had a IPS v4 Backlit LCD Screen.

All graphics calculations are done by the CPU, and then the Picture Processing Unit or ‘PPU’ renders them.



It uses tiles for rendering graphs, dividing by **background** and **sprites.**

Retired from: https://www.copetti.org/writings/consoles/game-boy

8x8 bitmaps stored in VRAM in a region called **Tile set**.

Tile set

In order to build the picture, tiles are referenced in another type of table known as **tile map.**

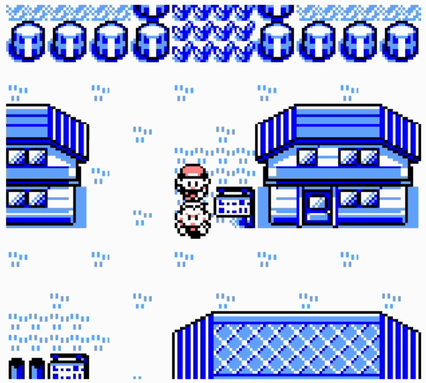
PPU*

LCD Control
LCD Display enable
Window Tile Map Address
Window Enable
BG & Window Tile Data
BG Tile Map Address
OBJ Size
OBJ Enable
BG Enable

LCDC Status
LYC=LY Interrupt
Mode 2 OAM Interrupt
Mode 1 V-Blank Interrupt
Mode 0 H-Blank Interrupt
LYC=LY Flag
Mode

Scroll Y
Scroll X
LCDC Y-Coordinate
LY Compare
DMA Transfer and Start
BG Palette
Object Palette 0
Object Palette 1
Window Y Position
Window X Position

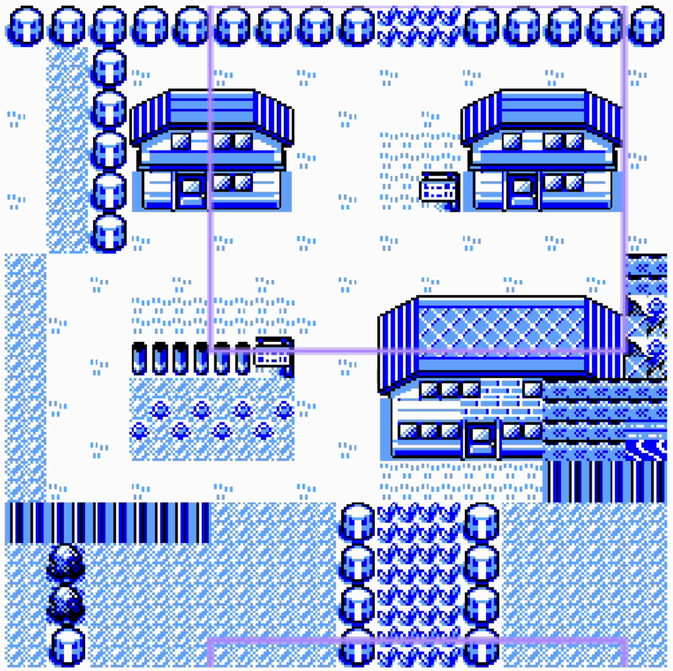
* We will need partial functionalities of the PPU for our game.



Background Map X

Tile Data X

Background Map



Tile Data

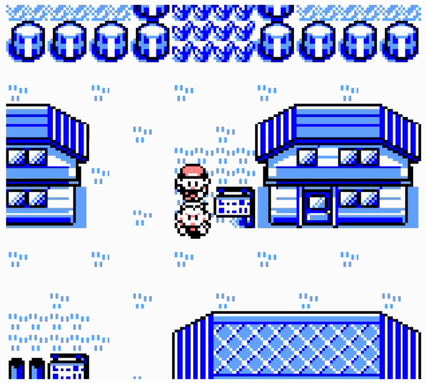
PPU contains all logic about display.
In this example you can see background map with scrolling

Background Map with Scrolling

(Using scroll Y and scroll X)

Basically it specify the coordinate of the screen's top-left pixel somewhere on the 256x256 pixel background map.

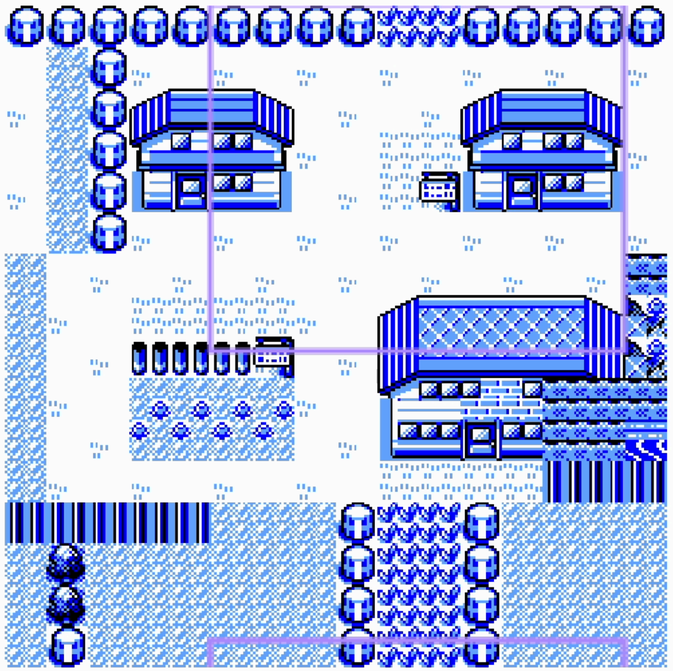
Retired from https://github.com/torch2424/wasmboy



Background Map X

Tile Data X

Background Map



Tile Data

PPU contains all logic about display.
In this example you can see background map with scrolling

Background Map with Scrolling

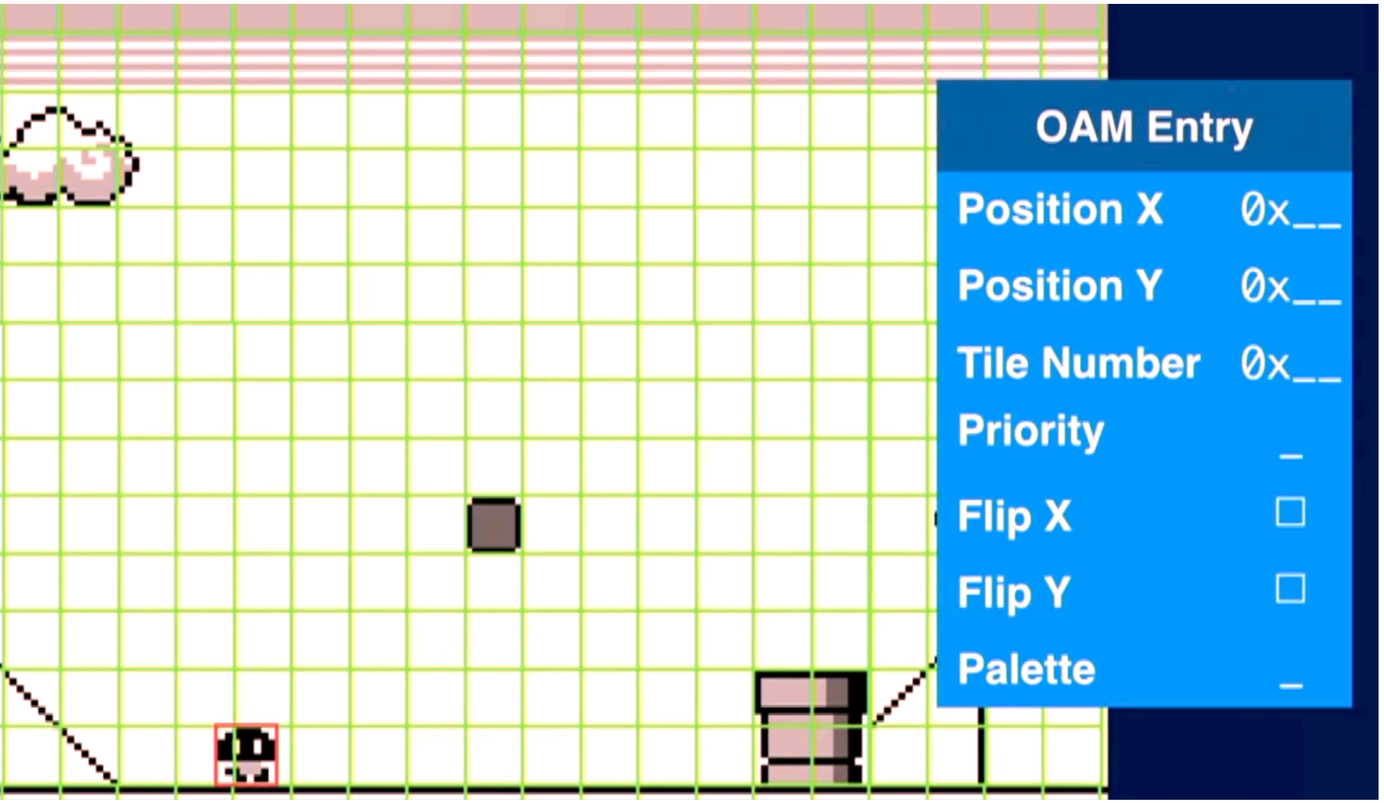
(Using scroll Y and scroll X)

Basically it specify the coordinate of the screen's top-left pixel somewhere on the 256x256 pixel background map.

Retired from https://github.com/torch2424/wasmboy

Object Attribute Memory (*OAM*)

The Game Boy PPU can display up to 40 movable objects (or sprites), each 8×8 or 8×16 pixels.



Retired from https://www.youtube.com/watch?v=HyzD8pNlpwI

Retired from https://gbdev.gg8.se/wiki/articles/OAM_DMA_tutorial

src/**ppu.rs**

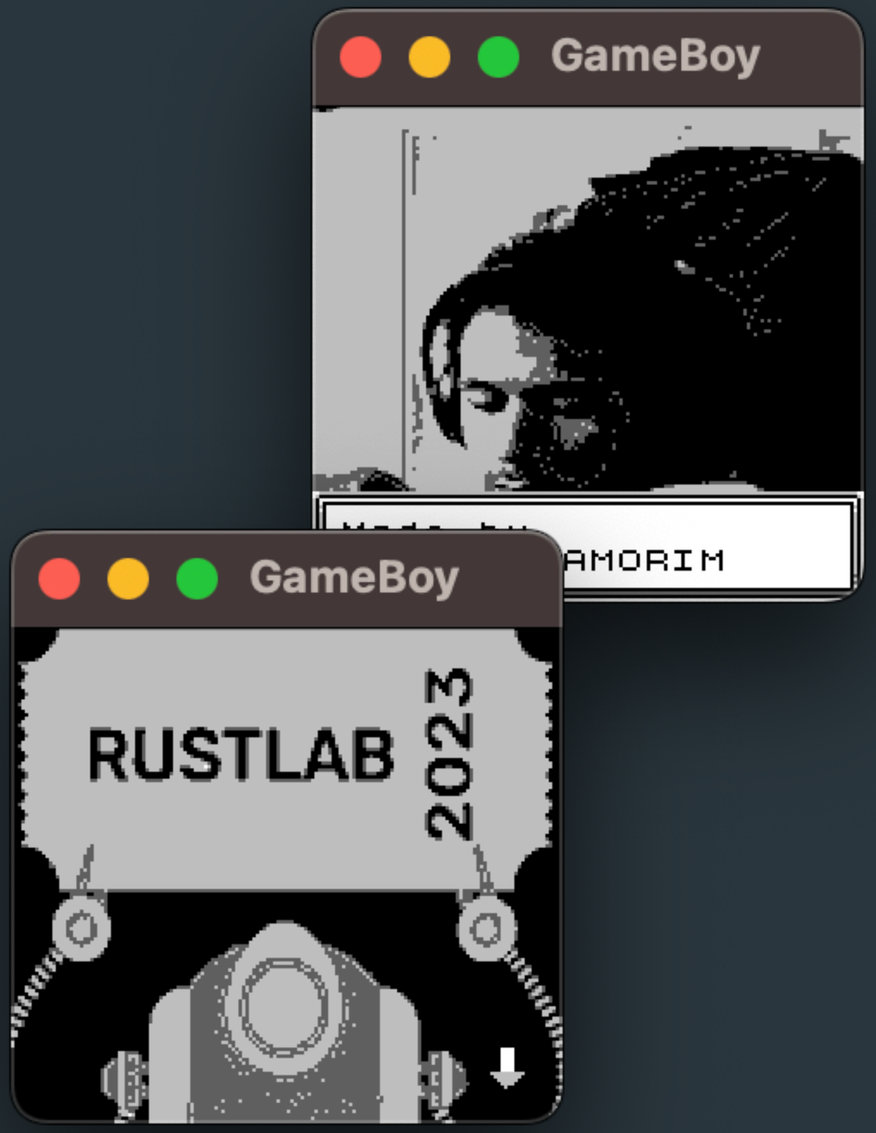
```
enum Mode {
    HBlank = 0×00,
    VBlank = 0×01,
    RdOam  = 0×02,
    RdVram = 0×03,
}
```

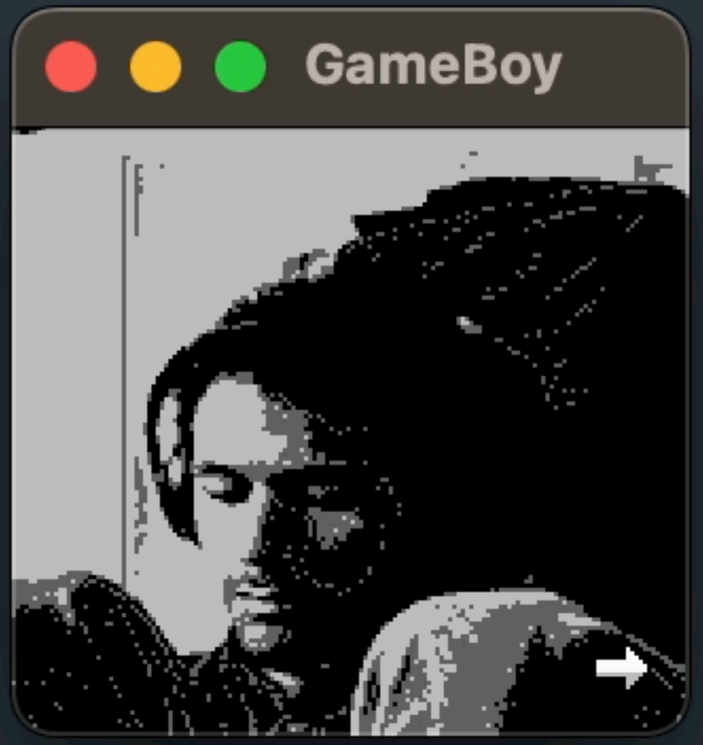
src/**ppu.rs**

```
pub fn read_byte(&self, address: u16) → u8 {
    match a {
        0×8000..=0×9FFF ⇒ {
            self.vram[(self.vrambank * 0×2000) |
            (address as usize & 0×1FFF)]
        }
        0×FE00..=0×FE9F ⇒
            self.voam[address as usize - 0×FE00],
        // …
```

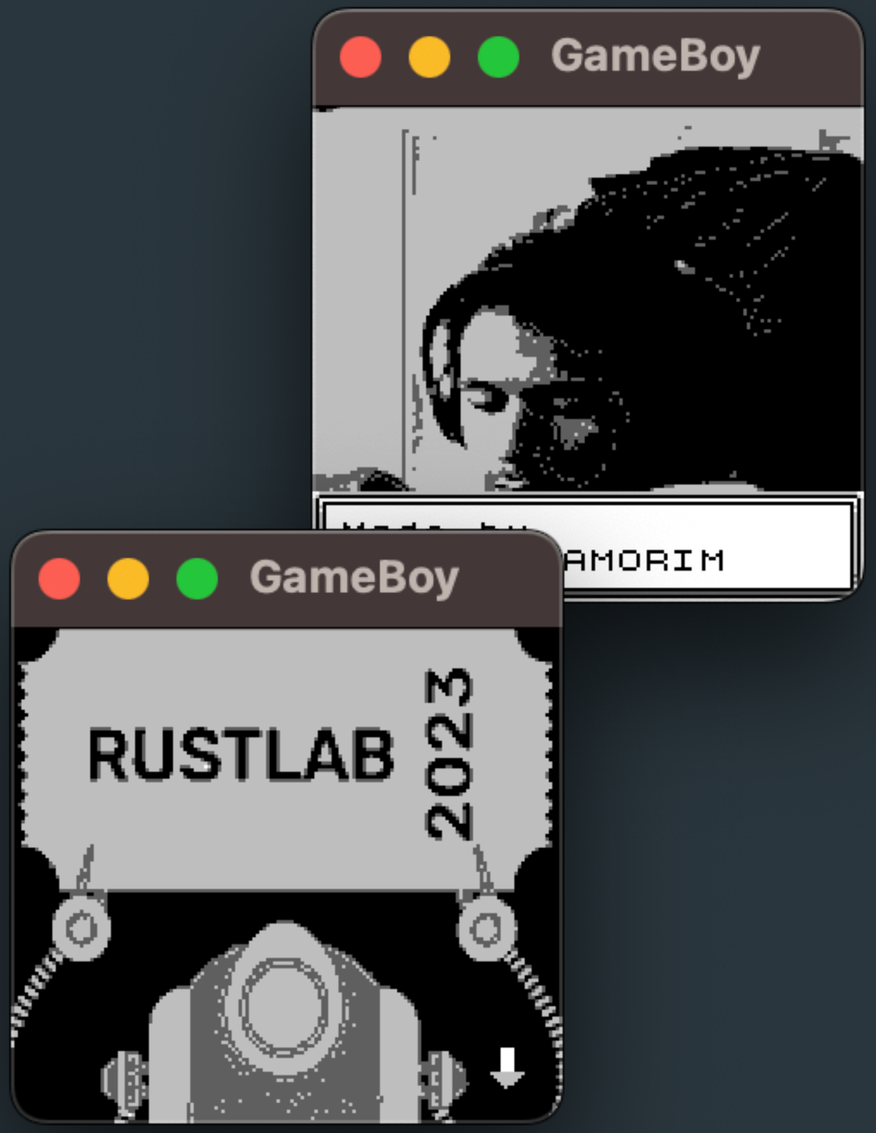
cargo run

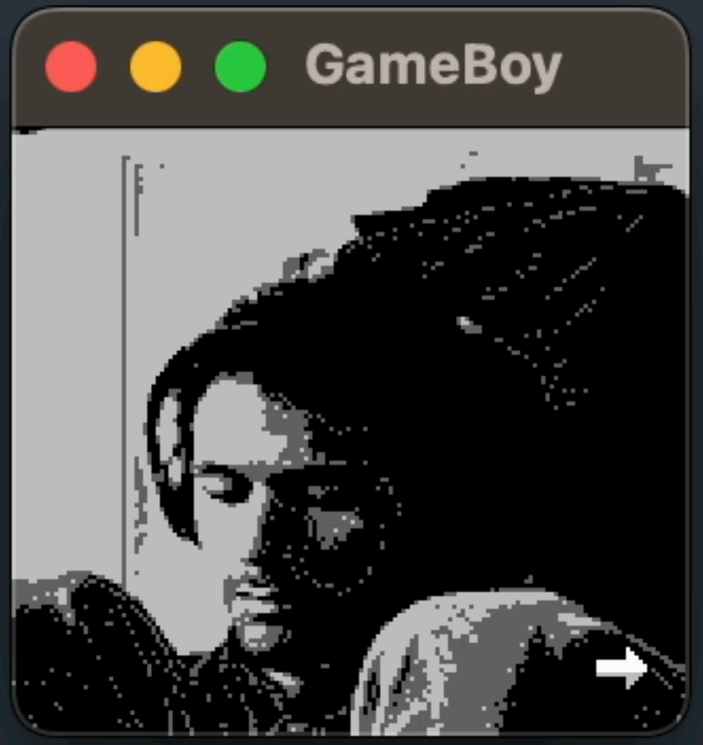
Desktop





Desktop



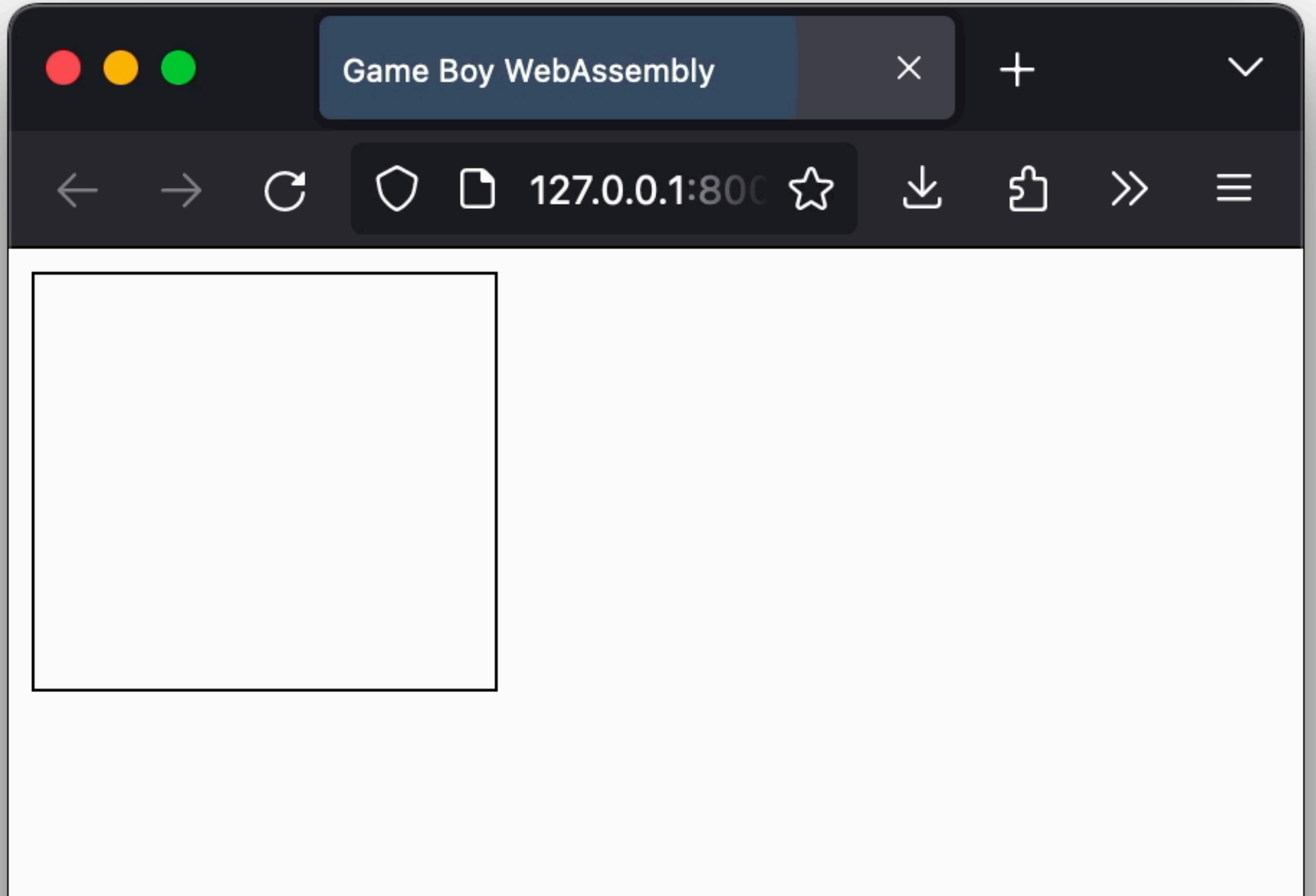


```
cargo install cargo-server
cargo install wasm-bindgen-cli
```

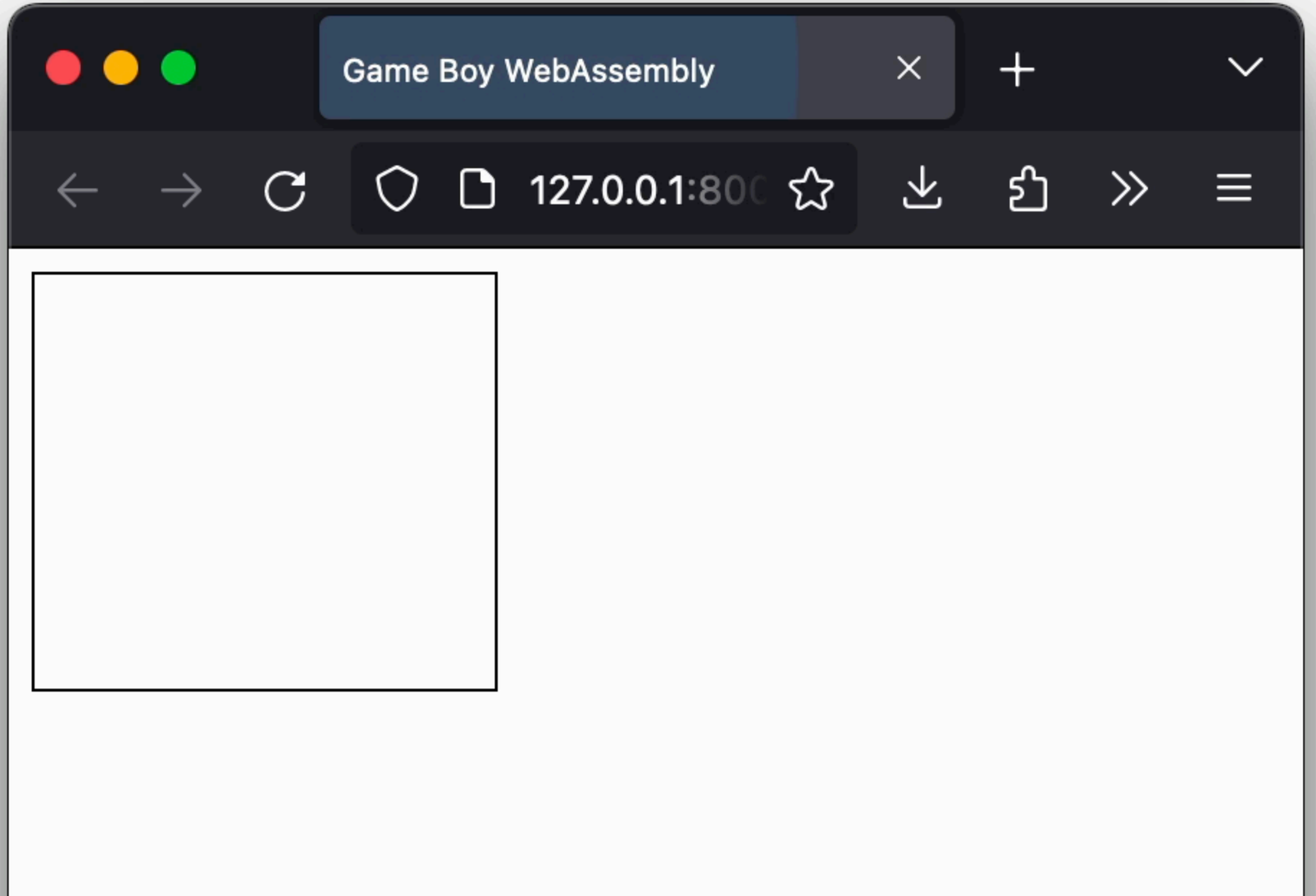
cargo build —target wasm32-unknown-unknown —lib

```
wasm-bindgen
  ./target/wasm32-unknown-unknown/debug/web.wasm
  —out-dir wasm —target web —no-typescript

cargo server —open
```

Web



Web



There is so much more

http://bgb.bircd.org/pandocs.htm
https://github.com/mvdnes/rboy
https://github.com/raphamorim/gameboy
https://github.com/alexcrichton/jba/tree/rust
https://github.com/gbdev/pandocs
http://imrannazar.com/GameBoy-Emulation-in-JavaScript:-The-CPU
https://multigesture.net/articles/how-to-write-an-emulator-chip-8-interpreter/
http://emubook.emulation64.com/
https://github.com/jawline/Mimic
https://www.youtube.com/watch?v=LqcEg3IVziQ
https://en.wikipedia.org/wiki/Zilog_Z80
https://en.wikipedia.org/wiki/Game_Boy
https://medium.com/@andrewimm/writing-a-game-boy-emulator-in-wasm-part-1-1ba023eb2c7c

There is so much more (2)

https://gbdev.gg8.se/wiki/articles/Gameboy_Bootstrap_ROM
https://github.com/yodalee/ruGameboy
https://www.youtube.com/watch?v=LqcEg3IVziQ
https://realboyemulator.wordpress.com/2013/01/01/the-nintendo-game-boy-1/
https://gbdev.gg8.se/wiki/articles/DMG_Schematics
https://chipmusic.org/forums/topic/13608/dmg-main-board-schematic-circuit-arduinoboy/
https://github.com/torch2424/wasmboy/
https://rylev.github.io/DMG-01/public/book/introduction.html
https://github.com/gbdev/awesome-gbdev
http://marc.rawer.de/Gameboy/Docs/GBProject.pdf
https://shonumi.github.io/dandocs.html
https://github.com/Baekalfen/PyBoy/blob/master/PyBoy.pdf
https://media.ccc.de/v/rustfest-rome-3-gameboy-emulator
https://github.com/rylev/DMG-01

Obrigado!

Thank you!