# Panzgb - a Gameboy emulator

Andrea Francesco Iuorio

15 Dicembre 2019





Un emulatore  dispositivo hardware o software che permette ad un sistema (host) di comportarsi come un altro (guest)

Gli emulatori non sono virtual machines:

- Un emulatore esegue software progettato per hardware diverso
- Una virtual machine implementa in software dell' hardware





Perch un emulatore Gameboy?
Hardware semplice rispetto a dispositivi piu moderni:

- Singola CPU, singolo core
- SoC tuttosommato semplice

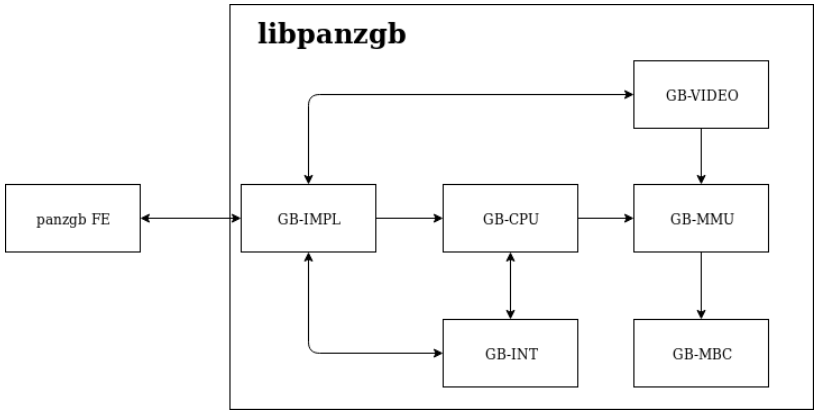
. . . ma che presenta tanti concetti utilizzati ancora oggi

- Memoria virtuale paginata
- CPU con interrupts, timers, trasferimenti DMA...





## Architettura



## Gameboy

- CPU Z80 a 8 bit (bus da 16 bit), 4.19 Mhz
- 8 Kb di RAM
- 8 Kb di VRAM
- Schermo a 4 colori da 2.16", 160x144, 59,73 Hz
- 4.2 watt a pieno utilizzo





## Cartuccie

- ROM da 32 Kb a 16 Mb (in commercio massimo 8 Mb)
- RAM da un massimo di 32 Kb
- Memory Bank Controller (MBC) che si occupa della comunicazione cartuccia-console
- Hardware extra come timers, coprocessori matematici ecc. ecc.





## Memoria virtuale

```
Interrupt Enable Register
---------------------------- FFFF
Internal RAM
---------------------------- FF80
Empty but unusable for I/O
---------------------------- FF4C
I/O ports
---------------------------- FF00
Empty but unusable for I/O
---------------------------- FEA0
Sprite Attrib Memory (OAM)
---------------------------- FE00
Echo of 8kB Internal RAM
---------------------------- E000
8kB Internal RAM
---------------------------- C000
8kB switchable RAM bank
---------------------------- A000
8kB Video RAM
---------------------------- 8000
16kB switchable ROM bank
---------------------------- 4000
16kB ROM bank #0
---------------------------- 0000
```

- ROM in pagine da 16kB
  - 0x0000 sempre pagina 0
  - 0x4000 pagina swappabile
- RAM in pagine da 8Kb
  - 0xA000 pagina RAM su cartuccia
  - 0xC000 RAM dispositivo (singola pagina)
- Ogni MBC ha un modo diverso per cambiare pagine





## GB-MMU

Il modulo MMU espone 2 metodi: readMemory e writeMemory usate per ogni accesso a memoria.

```
BYTE readMemory(gb *cpu, WORD addr) {
    if ((addr >= 0x0000) && (addr <= 0x3FFF)) {
        return cpu->cartridge[addr];
    }
    if ((addr >= 0x4000) && (addr <= 0x7FFF)) {
        WORD t = addr - 0x4000;
        return cpu->cartridge[t + (cpu->currentROMBank * 0x4000)];
    }

    else if ((addr >= 0xA000) && (addr <= 0xBFFF)) {
```

## GB-MBC

Ogni MBC possiede un modo diverso per cambiare le pagine mappate, ma in generale viene richiesta una scrittura di valori specifici nell' area ROM (0x0000 - 0x8000)

```
void writeMemory(gb *cpu, WORD addr, BYTE data) {
    /* This part is mapped on the rom, so read-only. It's used
     * by MBCs chips to change ROM page
     */
    if (addr < 0x8000) {
        cpu->changeBank(cpu, addr, data);
    }
```





```
/*Change ROM bank*/
else if ((addr >= 0x2000) && (addr < 0x4000)) {
    BYTE lower7 = data & 0x7F;
    cpu->currentROMBank = lower7;
    if (cpu->currentROMBank == 0x00)
        cpu->currentROMBank = 1;

}
```





## CPU emulation

Come emuliamo un programma?

- Rappresentiamo lo stato della macchina guest
- Una per volta, leggiamo una istruzione ed alteriamo lo stato

Vediamo un esempio concreto





```
struct gb_cpu {
    // The CPU registers
    WORD progCounter;
    WORD stack;

    BYTE A;
    BYTE B;
    BYTE C;
    BYTE D;
    BYTE E;
    BYTE F;
    BYTE H;
    BYTE L;

    BYTE master_interr_switch;
};
```

- CPU Z80 a 4.19 Mhz
- 8 registri general purpose da 8 bit
- 2 registri a 16 bit (SP e PC)
- Master Interrupt Switch
- Alcuni registri per I/O

## Fetch

Per il fetch, leggiamo il byte all' indirizzo del PC

```
BYTE executeGameBoy(gb *cpu) {
    BYTE opcode;
    BYTE numClock;

    if (cpu->cpuHalted != 0) {
        GET_BYTE_PC(cpu, opcode);
        numClock = executeOpcode(cpu, opcode);
    }

    increaseTimer(cpu, numClock);
    handleInterrupts(cpu);
    handleGraphic(cpu, numClock);

    return numClock;
}
```





## Decode and execute

Dobbiamo ora "capire" che istruzione sia e cambiare lo stato di conseguenza

```
BYTE executeOpcode(gb *cpu, BYTE opcode) {
    BYTE val = 0;
    switch (opcode) {
    case 0x00:
        return 4;
    case 0x06:
        GET_BYTE_PC(cpu, val)
        LOAD_8BIT(&(cpu->B), val);
        return 8;
    case 0x0E:
```





## Instruction timing

Cosa significa quel 4 che ritorniamo?

Ogni istruzione impiega un certo numero di cicli di clock: tenendo traccia di questo valore possiamo temporizzare l' esecuzione.

Tenere traccia del tempo  essenziale per i videogiochi!





## Instruction timing (cont.)

Tante cose dipendono dal timing preciso:

- refresh rate dello schermo
- fetch dei tasti premuti
- gestione interrupt
- trasferimenti memoria (es RAM - VRAM)

Ci sono giochi che si basano su tempistiche dell' hardware per fare ottimizzazioni

Per gestire il tempo:

```
while (1) {
    unsigned int timeStartFrame = SDL_GetTicks();
    getInput(gameboy, renderer);
    while (numOperation < NUM_OP_60HZ)
        numOperation += executeGameBoy(gameboy);
    numOperation -= NUM_OP_60HZ;
    renderScreen(gameboy, renderer, surface);
    float deltaT =
        (float)1000 / (59.7) -
        (float)(SDL_GetTicks() - timeStartFrame);
    if (deltaT > 0)
        SDL_Delay((unsigned int)deltaT);
}
```





## Gameboy PPU

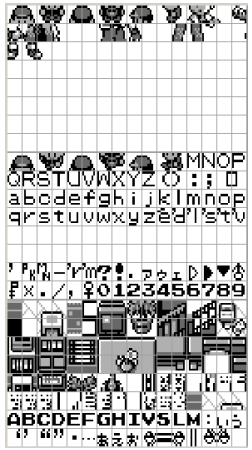
Il Gameboy possiede una PPU (Picture Processing Unit) con 8 kB di VRAM (0x8000 - 0x9FFF) ed una serie di registri interni
Concetti grafici base della PPU:

- Schermo a 4 colori: ogni colore identificabile con 2 bit
- Tile: quadretto di 8x8 pixel (16 byte) da disegnare
- Sprite: tile speciale che possiede un colore "trasparente". 8x8 pixel o 8x16 pixel





## Tile Map



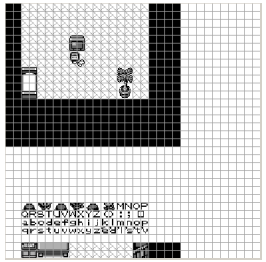
- Mappa da 192 tiles
- Ogni tile 8x8 pixel (16 byte)
- Alcune tiles sono sprites
- La Sprite Attribute Table ha informazioni come posizione, tile location per gli sprites





## Background Map

L' ultima parte della VRAM dedicata alla background map: mappa 32x32 (256x256 pixel) con indici della Tile Map



La BG Map da 256x256, ma lo schermo 160x144. Solo una parte mostrata su schermo.

## GB PPU

La PPU scorre la Background Map e la Sprite Attribute Map e colora i pixel in base ai colori che trova. Il tempo che la PPU

impiega per disegnare lo schermo  il refresh rate I giochi scrivono

direttamente nella Background Map e Tile Map per disegnare su schermo.





## V-Blank

Problema: e se il gioco scrive una tiles che sta venendo disegnata? Lo schermo del Gameboy  di 160x144 ma per la PPU lo schermo

ha dimensione 160x153. Mentre la PPU sta disegnando le 9 righe

virtuali, il gioco  sicuro di non corrompere l' immagine effettuando operazioni in VRAM Questo tempo  detto V-Blank





## V-Blank

Il V-Blank  un periodo 1.08 ms in cui il software pu scrivere sulla memoria video senza problemi Il software pu sapere se il V-Blank

attivo in due modi:

- Leggendo dal registro di stato della PPU
- Viene lanciato un interrupt quando inizia il V-Blank

Alcuni software ignorano questi check e si basano solo sul timing!





L' emulatore  disponibile su github:

`https://github.com/afiuorio/panzgb`

A tempo perso sto lavorando per supportare il Gameboy Color

- Stessa CPU ma con la possibilit di aumentare/diminuire il clock a piacere
- Memory map diversa (doppio della memoria video e RAM fisica)
- PPU molto piu complessa

... ma qualche risultato inizia a vedersi