TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

TIPE

L'émulation et la conservation des logiciels

LIAGRE Enzo

Introduction

Émulation

L'émulation est le processus par lequel une application reproduit le fonctionnement d'une machine ou d'un autre logiciel.

Exemples :

- Un émulateur de terminal
- Une machine virtuelle

Intérêt

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

```
ᛤ> ~/G/E/Gameboy ls Pokemon_Version_Rouge.gb
.rw-r--r-- 1,0M enzo 15 oct.   1998  Pokemon_Version_Rouge.gb
ᛤ> ~/G/E/Gameboy chmod 755 Pokemon_Version_Rouge.gb
ᛤ> ~/G/E/Gameboy ls Pokemon_Version_Rouge.gb
.rwxr-xr-x 1,0M enzo 15 oct.   1998  Pokemon_Version_Rouge.gb
ᛤ> ~/G/E/Gameboy ./Pokemon_Version_Rouge.gb
exec: Failed to execute process: './Pokemon_Version_Rouge.gb' the file could not be run by the o
perating system.
exec: Maybe the interpreter directive (#! line) is broken?
ᛤ> ~/G/E/Gameboy
```

Principe

On distingue deux méthodes pour l'émulation.

1. L'émulation de bas niveau (Low-level emulation)
   $\hookrightarrow$ On reproduit le fonctionnement de la machine en entier.
2. L'émulation de haut niveau (High-level emulation)
   $\hookrightarrow$ On reproduit ce que la machine permet.

On s'intéressera pour l'instant à l'émulation de bas niveau.

Low-level emulation (LLE)

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

Méthode

On étudie le sytème afin de savoir comment les composants fonctionnement, puis on les implémentes.

$\hookrightarrow$ On implémente des machines entières donc on privilégie un language de bas niveau.

L'émulation d'un jeu GameBoy

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

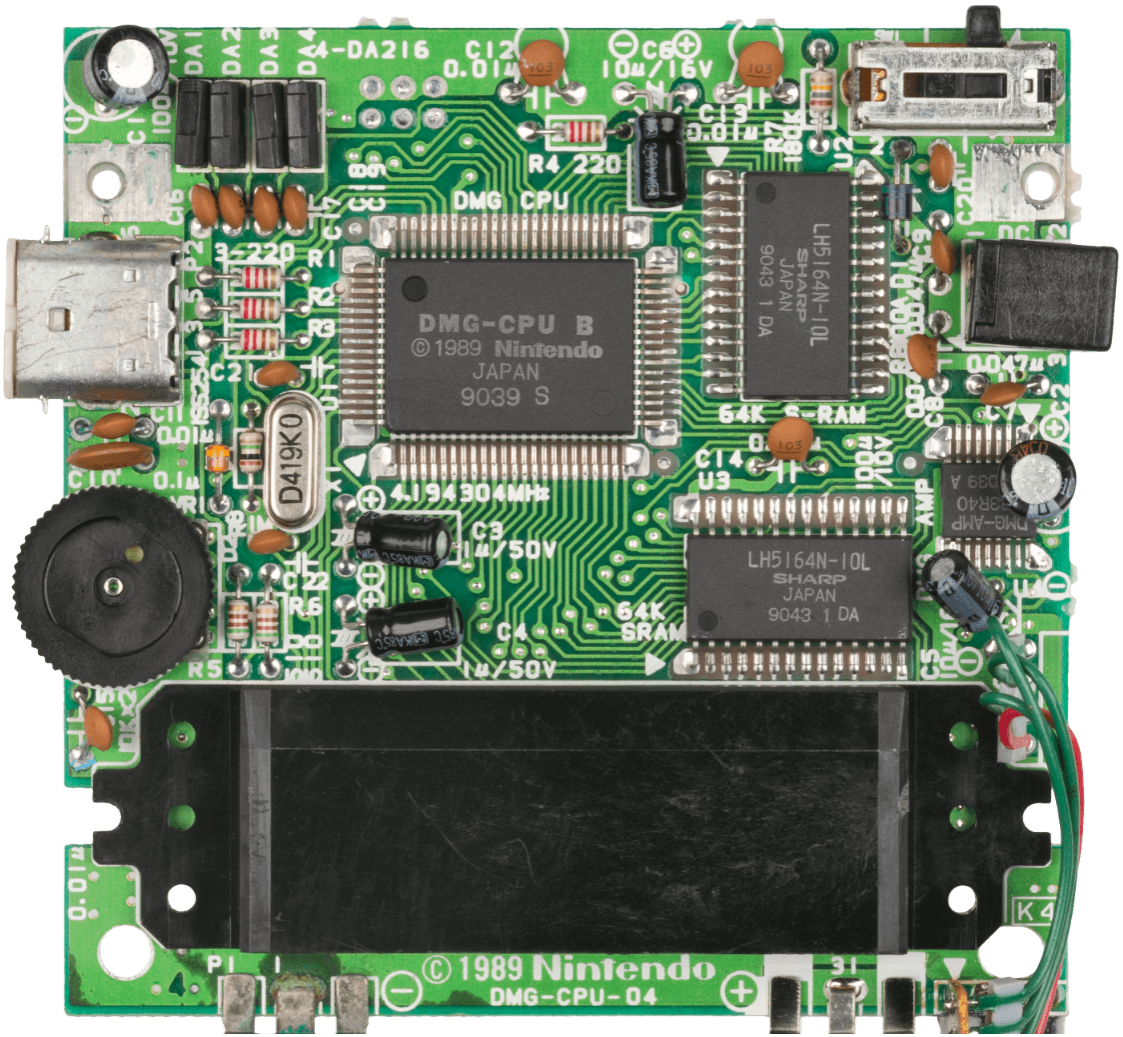
Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion



L'émulation d'un jeu GameBoy

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

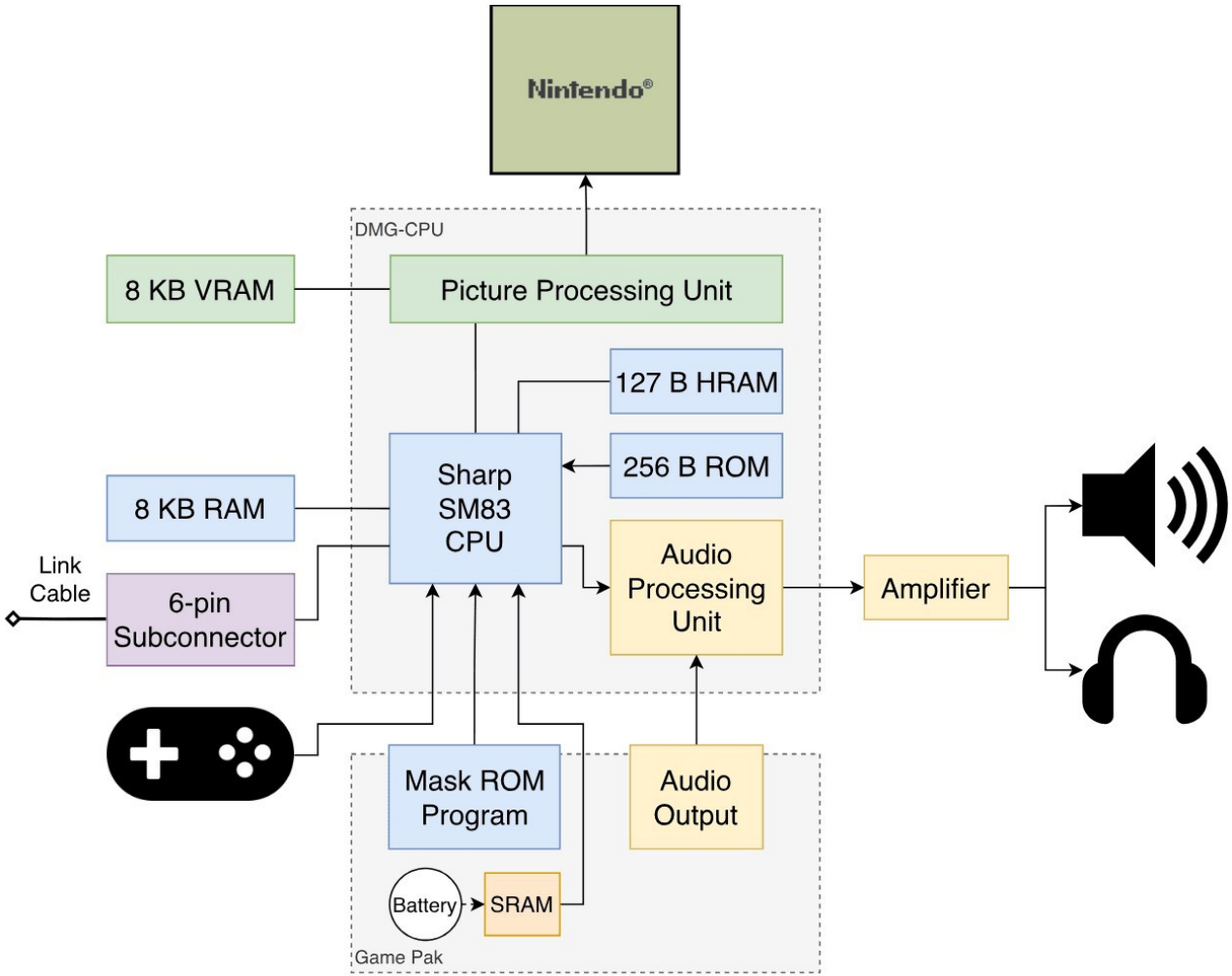
Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion



copetti.org © Rodrigo Copetti

La Structure

On suppose que la GameBoy n'est composée que

- d'un lecteur cartouche
- d'un CPU

On représente le lecteur cartouche par un tableau 8×16^3 entiers naturels codés sur $8-$bits.

Le CPU

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

Le processeur de la GameBoy est un Sharp SM83.
Celui-ci est 8−bit. Autrement dit ses régistre sont de taille 1 octet.

Registre

Un registre est un bloc de la mémoire interne du processeur. Il s'agit de la mémoire la plus rapide d'un ordinateur.

C'est dans ces bloc que le processeur effectue ses calcules.

L'organisation

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

Le Sharp SM83 contient 10 registres.

A	F
B	C
D	E
H	L

SP
PC

L'organisation

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

Le Sharp SM83 contient 10 registres.

A	$0 \rightarrow 255$	$0 \rightarrow 255$	F
B	$0 \rightarrow 255$	$0 \rightarrow 255$	F
C	$0 \rightarrow 255$	$0 \rightarrow 255$	F
D	$0 \rightarrow 255$	$0 \rightarrow 255$	E

SP

$0 \rightarrow 65535$
$0 \rightarrow 65535$

PC

Exécution

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

Si le CPU n'est qu'une structure avec ses registres représentés en variables, on a

```
λ> ~/G/E/Gameboy ./emugb.sh Pokemon_Version_Rouge.gb
ROM chargée (32768 octets)
Opcode 0x00 à 0x0100
Opcode 0xC3 à 0x0101
Opcode 0x50 à 0x0102
Opcode 0x01 à 0x0103
Opcode 0xCE à 0x0104
Opcode 0xED à 0x0105
Opcode 0x66 à 0x0106
Opcode 0x66 à 0x0107
Opcode 0xCC à 0x0108
Opcode 0x0D à 0x0109
λ> ~/G/E/Gameboy
```

Conclusion

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

En suivant la même méthode, on peut, en théorie, émuler n'importe quel logiciel.

Cependant, reproduire la machine en entier pose des problèmes de performance.

En prolongement

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

La suite du TIPE se portera sur l'émulation de haut niveau.

On pourrait envisager

- implementer l'émulateur en entier
- effectuer des test de performances entre LLE / HLE

Du négatif

TIPE

L'émulation et la conservation des logiciels

Présentation du sujet

Low-level emulation

Exemple : La GameBoy

Implémentation

Exécution

Conclusion

Il rest important de remarquer que

- Ce TIPE est très empirique
- L'émulation ne serait utile si tous les sytèmes avait un même systeme d'exploitation
- Il est légalement difficile de travailler sur ce sujet

Bibliographie

1. « Stack processor architecture and development methods suitable for dependable applications. » Mehdi Jallouli, Camille Diou, Fabrice Monteiro, Abbas Dandache.
2. « Game Boy : Complete Technical Reference » https://gekkio.fi, Révision 164.
3. L'article « Game Boy / Color Architecture – A Practical Analysis » écrit par Rodrigo Copetti www.copetti.org/writings/consoles/game-boy/.
4. La série « The Game Boy, a hardware autopsy » par JackTech https://www.youtube.com/@jacktech5101.

Ressources

1. Architecture du processeur « Sharp SM83 » https://gbdev.io/gb-opcodes//optables/.
2. Fichier ROM d'une cartouche de *Pokémon Version Rouge* développé par Game Freak.
3. Quelques illustrations de « Game Boy / Color Architecture – A Practical Analysis » écrit par Rodrigo Copetti www.copetti.org/writings/consoles/game-boy/.

emugb.sh


```
# Permet d'executer la compilation dans n'importe quel repertoire
pwd > ~/pwd_tmp_emugb_sh.txt

# Compilation cpu
cd ~/Workspace/tipe-2026/low_level/SoC/CPU
gcc -Werror -Wall -Wextra -fsanitize=address -c CPU.c

# Compilation gameboy
cd ../../Gameboy
gcc -Werror -Wall -Wextra -fsanitize=address -c Gameboy.c

# Compilation de l'emulateur
cd ..
gcc -Werror -Wall -Wextra -fsanitize=address -o emugb SoC/CPU/CPU.o Gameboy/
     Gameboy.o emu.c

# ouverture de la rom
./emugb $@ #/!\ requiert un chemin absolu

# retour au point de depart
echo ~/pwd_tmp_emugb_sh.txt | cd
rm ~/pwd_tmp_emugb_sh.txt
```

CPU.h I

```c
#ifndef __CPU_H__
#define __CPU_H__

// Inclusions
////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>

// types
////////////////////////////

typedef uint8_t u8;
typedef uint16_t u16;

typedef struct cpu_s* cpu;

typedef enum{
    A = -10,    F = -9,
    B = -8,     C = -7,
    D = -6,     E = -5,
    H = -4,     L = -3,
    SP = -2,    PC = -1,
} mem_empl ;
```

CPU.h II


```
// fonctions
////////////////////////////

// constructeur
cpu init_cpu();

// accesseur
u8 get_reg(cpu, mem_empl);

u16 get_SP(cpu);

u16 get_PC(cpu);

// accesseur
void set_reg(cpu, mem_empl, u8);

void set_SP(cpu, u16);

void set_PC(cpu, u16);


// destructeur
void free_cpu(cpu);

#endif
```

CPU.c I


```c
#include "CPU.h"

// types
////////////////////////////

struct cpu_s{
    u8 A;   u8 F;
    u8 B;   u8 C;
    u8 D;   u8 E;
    u8 H;   u8 L;
    u16 SP; u16 PC;
};


// fonctions
////////////////////////////

cpu init_cpu(){
    cpu sm83 = malloc(sizeof(struct cpu_s));
    assert(sm83 != NULL);
    sm83->A = 0x01;     sm83->F = 0xB0;
    sm83->B = 0x00;     sm83->C = 0x13;
    sm83->D = 0x00;     sm83->E = 0xD8;
    sm83->H = 0x01;     sm83->L = 0x4D;
    sm83->SP = 0xFFFE;  sm83->PC = 0x0100;
    return sm83;
}
```

CPU.c II


```
u8 get_reg(cpu c, mem_empl e){
    assert(c != NULL);
    switch (e) {
        case A:
            return c->A;
        case F:
            return c->F;
        case B:
            return c->B;
        case C:
            return c->C;
        case D:
            return c->D;
        case E:
            return c->E;
        case H:
            return c->H;
        case L:
            return c->L;
        default:
            assert(0);
    }
}
```

CPU.c III


```
u16 get_SP(cpu c){
    assert(c != NULL);
    return c->SP;
}
```

```
u16 get_PC(cpu c){
    assert(c != NULL);
    return c->PC;
}
```

CPU.c IV


```
void set_reg(cpu c, mem_empl e, u8 val){
    assert(c != NULL);
    switch (e) {
        case A:
            c->A = val;
            break;
        case F:
            c->F = val;
            break;
        case B:
            c->B = val;
            break;
        case C:
            c->C = val;
            break;
        case D:
            c->D = val;
            break;
        case E:
            c->E = val;
            break;
        case H:
            c->H = val;
            break;
        case L:
            c->L = val;
            break;
        default:
            assert(0);}}
```

CPU.c V


```
void set_SP(cpu c, u16 val){
    assert(c != NULL);
    c->SP = val;
}

void set_PC(cpu c, u16 val){
    assert(c != NULL);
    c->PC = val;
}

void free_cpu(cpu sm83){
    if(sm83 != NULL){
        free(sm83);
    }
}
```

SoC.h I

```
#ifndef __SOC_H__
#define __SOC_H__


#include "CPU/CPU.h"



#endif
```

Gameboy.h I

```c
#ifndef __GAMEBOY_H__
#define __GAMEBOY_H__

#define ROM_SIZE_MAX 0x8000



// Inclusions
//////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>

#include "../SoC/SoC.h"


// Types
//////////////////////////////

typedef uint8_t u8;
typedef uint16_t u16;


typedef struct gameboy_s* gameboy;
```

Gameboy.h II


```
// fonctions
////////////////////////////

gameboy init_gb();

u8* get_rom(gameboy);

cpu get_sm83(gameboy);

void free_gb(gameboy);

void load_rom(gameboy, const char*);

void emulate_cycle(gameboy);


#endif
```

Gameboy.c I


```c
#include "Gameboy.h"



// types
//////////////////////////////

struct gameboy_s {
    cpu sm83;
    u8* rom;
};



// fonctions
//////////////////////////////

gameboy init_gb(){
    gameboy gb = malloc(sizeof(struct gameboy_s));
    assert(gb != NULL);
    gb->sm83 = init_cpu();
    gb->rom = malloc(ROM_SIZE_MAX*(sizeof(u8)));
    assert(gb->rom != NULL);
    return gb;
}
```

Gameboy.c II


```
u8* get_rom(gameboy gb){
    assert(gb != NULL);
    return gb->rom;
}

cpu get_sm83(gameboy gb){
    assert(gb != NULL);
    return gb->sm83;
}

void free_gb(gameboy gb){
    if(gb != NULL){
        free_cpu(gb->sm83);
        free(gb->rom);
        free(gb);
    }
}

void emulate_cycle(gameboy gb) {
    uint16_t pc = get_PC(gb->sm83);
    uint8_t opcode = gb->rom[pc];
    printf("Opcode 0x%02X a 0x%04X\n", opcode, pc);

    set_PC(gb->sm83, get_PC(gb->sm83) + 1);
}
```

Gameboy.c III


```c
void load_rom(gameboy gb, const char* filename) {
    assert(gb != NULL);

    FILE* file = fopen(filename, "rb");
    if (!file) {
        perror("Erreur ouverture ROM");
        exit(1);
    }

    u8* rom = get_rom(gb);

    size_t read = fread(rom, 1, ROM_SIZE_MAX, file);
    fclose(file);

    if (read == 0) {
        fprintf(stderr, "Erreur : ROM vide\n");
        exit(1);
    }

    printf("ROM chargee (%zu octets)\n", read);
}
```

emu.c I


```c
#include "Gameboy/Gameboy.h"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        printf("Utilisation : %s fichier.gb\n", argv[0]);
        return 1;
    }

    gameboy gb = init_gb();
    load_rom(gb, argv[1]);

    for (int i = 0; i < 10; i++) {
        emulate_cycle(gb);
    }

    free_gb(gb);

    return 0;
}
```