DO178-B
31th January 2019

CHIP-8 Emulator

Raphaël Gault - gault_r@epita.fr
Guillaume Taquet Gaspérini - taquet_g@epita.fr
Tifanny Suyavong - suyavo_t@epita.fr
Grégoire Verdier - verdie_g@epita.fr

Software Requirements

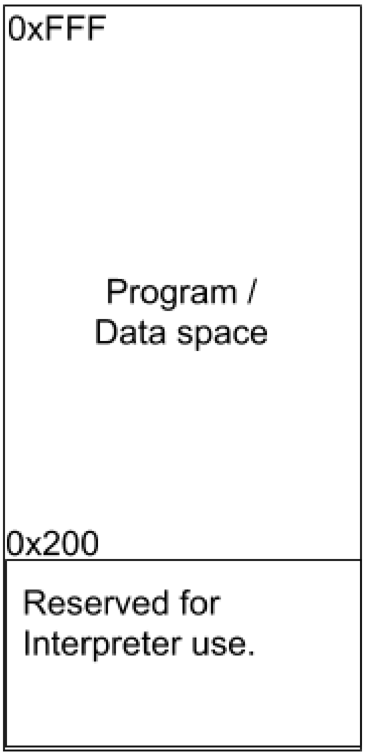
Level of criticality: E => Failure will have no effect for safety

Software requirement
The emulator should run Chip8 ROMs
The emulator should be written in Ada
The emulator should get user input via virtual keyboard
The emulator should run on STM32F4 Discovery board
The emulator should compile with Gnat 8.2.1 20181127

Software Architecture

Overall Architecture

Memory



The Chip-8 emulated processor is capable of accessing 4096 Bytes (4K) of memory (from 0x000 to 0xFFF).
The first 512 Bytes are reserved to the interpreter. The rom's code (program executed by the processor) should start at 0x200 (512).

Registers

The Chip-8 has 16 general purpose 8-bit registers, usually referred to as V*x*, where *x* is a hexadecimal digit (0 through F). There is also a 16-bit register called I. This register is generally used to store memory addresses, so only the lowest (rightmost) 12 bits are usually used.

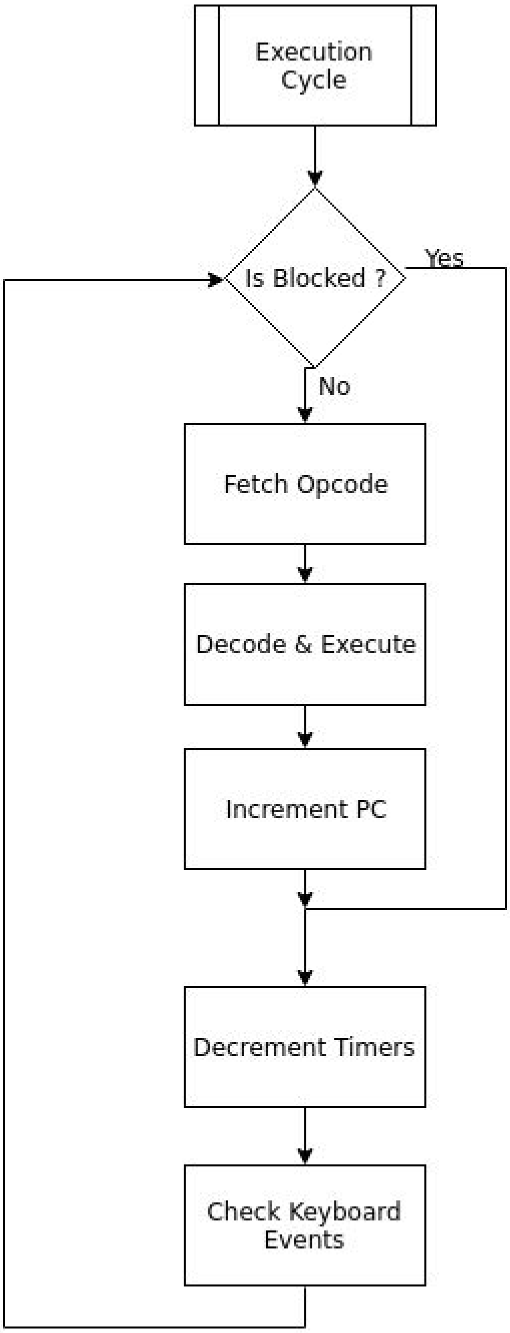
The VF register should not be used by any program, as it is used as a flag by some instructions.

Chip-8 also has two special purpose 8-bit registers, for the delay and sound timers. When these registers are non-zero, they are automatically decremented after each instruction cycle.

There are also some "pseudo-registers" which are not accessible from Chip-8 programs. The program counter (PC) should be 16-bit, and is used to store the currently executing address.

The stack is an array of 16 16-bit values, used to store the address that the interpreter should return to when finished with a subroutine. Chip-8 allows for up to 16 levels of nested subroutines.

Execution Flow



High Level Requirements

Identifier	Requirement
HLR.1	a keyboard should be displayed on the screen
HLR.2	the displayed keyboard should send the appropriate input value to the program when a key is pressed
HLR.3	the keyboard is displayed on two lines, each containing 8 keys ordered from lowest value to highest value
HLR.4	the emulated applications should be rendered in black and white
HLR.5	the default background color is black
HLR.6	when pressed, a key is highlighted (black font on white background) to provide a feedback to the user
HLR.7	any valid chip-8 rom can be run on the emulator
HLR.8	the game screen is bordered by a thin white border
HLR.9	multiple roms must be included by default on the emulator
HLR.10	at startup a 4x4 grid-menu allows the user to select the chip-8 rom to be executed on the emulator
HLR.11	sound of emulated applications is not supported

Low Level Requirements

Identifier	Requirement
LLR.1	the emulator should handle every chip-8 possible instruction
LLR.2	when a key is pressed the corresponding value in the keyboard buffer should be set
LLR.3	the Program Counter (PC) should always be between 0x200 and 0xFFF
LLR.4	the Program Counter should increase after each emulated instruction
LLR.5	the emulator should provide 64*32 pixels available to the applications
LLR.6	the hexa-decimal numbers sprites should be stored in the interpreter reserved space (starting at 0x0)
LLR.7	for each execution cycle, the opcode corresponding to the instruction is fetched in memory before being decoded and executed
LLR.8	instructions' bytes are stored in big-endian in memory
LLR.9	the processor should implement blocking instructions (keyboard event)
LLR.10	font characters should be 4 pixels wide and 5 pixel high

Traceability

HLR	LLR	Source Code
HLR.7	LLR.1	handlers.ads/adb
HLR.4	LLR.5	graphics.ads/adb
HLR.2	LLR.2	keybord.ads/adb
	LLR.4	main.adb
	LLR.6	registers.ads
	LLR.7	main.adb/registers.adb
	LLR.9	main.adb/class_f.adb

The code architecture is as follows:

```
├── src
│   ├── handlers
│   │   ├── class_e.adb
│   │   ├── class_e.ads
│   │   ├── class_eight.adb
│   │   ├── class_eight.ads
│   │   ├── class_f.adb
│   │   ├── class_f.ads
│   │   ├── handlers.adb
│   │   └── handlers.ads
│   ├── interpreter.adb
│   ├── interpreter.ads
│   ├── main.adb
│   ├── registers.adb
│   ├── registers.ads
│   ├── stack.adb
│   ├── stack.ads
│   ├── test.adb
│   ├── types.ads
│   └── user-interface
│       ├── graphics.adb
│       ├── graphics.ads
│       ├── keyboard.adb
│       ├── keyboard.ads
│       ├── menu.adb
│       └── menu.ads
```

High Level Test Cases

Test	HLR	Description
HLT.1	HLR.1	boot the board, select a ROM, a keyboard should be displayed on the screen
HLT.2	HLR.2	boot the board, select a ROM, use the keyboard to move
HLT.3	HLR.3	boot the board, select a ROM, the keyboard is displayed on two lines, each containing 8 keys ordered from lowest value to highest value
HLT.5	HLR.4 and HLR.5	boot the board, the font is white on a black background
HLT.6	HLR.6	boot the board, select a ROM, press on a key to view the highlighted feedback
HLT.7	HLR.7	flash a chip-8 valid ROM and boot the board to select it
HLT.8	HLR.8	boot the board, select a ROM, the screen is bordered by a thin white border
HLT.9	HLR.9	boot the board, multiple ROMs can be selected without external interaction
HLT.10	HLR.10	boot the board, a 4x4 grid-menu should be displayed and select a ROM to be executed on the emulator
HLT.11	HLR.11	boot the board, select a ROM with sound, no sound should be heard

Low Level Test Cases

Test	LLR	Description
LLT.2	LLR.2	press a key and check if the corresponding value is set in the keyboard buffer
LLT.3	LLR.3	check if the Program Counter (PC) is always between 0x200 and 0xFFF
LLT.4	LLR.4	emulate every instruction and check if the Program Counter is increased
LLT.5	LLR.5	check if the screen buffer size equals 64*32
LLT.6	LLR.6	check if the sprites are stored at the address 0x0
LLT.7	LLR.7	for each execution cycle, the opcode corresponding to the instruction is fetched in memory before being decoded and executed
LLT.8	LLR.8	instructions' bytes are stored in big-endian in memory
LLT.9	LLR.9	emulate blocking instructions and check the result
LLT.10	LLR.10	check if the constant for pixel sizes are correctly

Most of the low level requirements are checked by the contract programming paradigm which is used thanks to the Ada/Spark programming language.

Source Code

http://github.com/DakeyR/ada-chip8-emulator.git