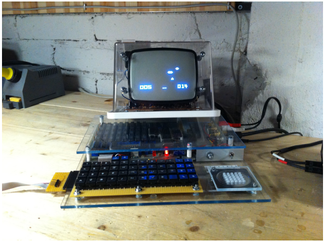
CHIP 8 emulator

Ronan Bocquillon, Pierre Gaucher, Nicolas Monmarché

(based on the Cowgod's CHIP 8 technical reference)

About CHIP 8

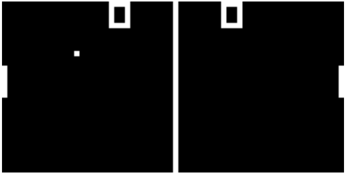
- CHIP 8 in a few words:
  - a 35-instruction interpreted programming language,
  - developped in the seventies, it allows video games to be programmed and run on low range computers of the time,
  - Pong, Space Invaders, Tetris, Pacman, among others...



A Telmac-1800 running CHIP 8 game Space Intercept [wikipedia].

About this project

- Your objective is to write a CHIP 8 emulator.
  1. To this end, you first need to simulate a machine made up of a virtual processor, some memory, a display, a keyboard and a speaker.
  2. You will then be able to interpret CHIP 8 instructions in terms of operations on this virtual machine.
  3. Putting all of this together within a classical fetch-decode-execute loop, you should be able to load and run complete programs/ROMs.



What you should be able to play in a few weeks.

Technical specifications

1. Memory
2. Processor
3. I/O devices

1. Memory
2. Processor
3. I/O devices

Memory

- CHIP 8 can address up to 4 KiB of RAM.
- The first 512 bytes (addresses from `0x000` to `0x1FF`) are where the original interpreter was located and are not used by programs.
- Therefore, CHIP 8 programs should be loaded from location `0x200`.

Processor

- CHIP 8 has:
  - 16 general purpose 8-bit registers, referred to as `V0`, `V1`, ..., `VF`,
  - a 16-bit register called `I`, generally used to store memory addresses,
  - 2 special purpose 8-bit registers, `DT` and `ST`, for the delay and sound timers,
  - a 16-bit register, called program counter or `PC`, used to store the address of the instruction currently being executed,
  - a stack — made up of a 16 16-bit values array and a 8-bit register named `SP` (pointing to to the topmost level of the stack) — to store the addresses that the interpreter should return to when a subroutine terminates.

Execution loop

- Initially, the program counter is set to 0x200, the location where the program is expected to start.
- The emulator then runs an infinite loop:
  1. at each step, the 16-bit instruction pointed to by the program counter is fetched from memory, interpreted and executed;
  2. then, the program counter is incremented by 2 so that it points to the following instruction.

Instructions

- Instructions are all 16-bit long, but may take different shapes.
- Here are some examples:
  - `6xkk`: sets $\texttt{Vx} := \texttt{kk}$,
  - `8xy4`: sets $\texttt{Vx} := \texttt{Vx} + \texttt{Vy}$ and $\texttt{VF} :=$ *carry* [operator],
  - `3xkk`: increments the program counter by 2 if $\texttt{Vx} = \texttt{kk}$ [skip-if],
  - `1nnn`: sets $\texttt{PC} := \texttt{nnn}$ [jump/loop],
  - `2nnn`: saves the current PC on the top of the stack, then branches to the subroutine $\texttt{PC} := \texttt{nnn}$ [call],
  - `Fx0A`: waits for a key press and stores the value of the key in `Vx`.

Keyboard

- The original target machine had a 16-key hexadecimal keypad.
- These 16 keys must be mapped to fit your keyboard or controller.

Speaker

- The delay and sound timers are automatically decremented at a rate of 60 Hz (unless they are already equal to zero)
- The CHIP 8 buzzer is expected to sound whenever the sound timer is greater than zero. The sound produced has only one tone.

Display

- CHIP uses a 64x32-pixel monochrome display.
- It draws graphics on screen through the use of sprites (that are groups of bytes representing the desired pictures).

```
[0]  11110000      ****
[1]  10010000      *  *
[2]  10010000      *  *
[3]  10010000      *  *
[4]  11110000      ****
```

The binary (sprite) and graphical (displayed) representations of "0".