Chip-8 Emulator

Technical reference by COWGOD: http://devernay.free.fr/hacks/chip8/C8TECH10.HTM

Other valuable links:

* https://raduangelescu.com/chip8emulatorjavascript.html
* http://mattmik.com/files/chip8/mastering/chip8.html
* http://blog.alexanderdickson.com/javascript-chip-8-emulator

Here's the simplified version:

**Memory:**

* Has 4096 bytes of memory; implemented using a JS array that is 4096-elements long and stores 8-bit integers
* Stores a list of opcodes (and other commands that draw sprites/graphics) from memory location 0x0000 to 0x01FF (the "interpreter" section of the memory)
* Stores the program loaded into the emulator from memory location 0x0200 to the end
* *Will not be big enough to hold all of the opcodes, if the program is too big*

**Processor**

* Runs at 500 Hz

**Registers:**

* The emulator has 16 registers that store 8-bit integers; implemented using 16 JS arrays that stores 8-bit integers.
* There is also a special I register.
* The registers are labeled from V0 to VF. VF is a special registers that is used for special stuffs (so we can't use that yet).
* These are used by the opcodes to perform VERY basic operations. For example, store the value in register Vx to register Vy.
* By manipulating the registers and using opcodes, we can do simple math operations and the like (just like a real processor).

**Program Counter (PC):**

* Shows which opcode we are currently executing when a program is running
* Increases by 2 each time to advance the program
* *Gives an error if the PC refers to a locked part of the memory*

**Stack Pointer:**

* Stores the old PC; useful when the PC is "jumping around" in the program and we need to backtrack the program when debugging

**Stack:**

* *Gives an error if the stack is full*

**Timers/Sounds**

* Chip-8 has two timers, a delay timer and a sound timer. Both timers hold an 8-bit number and will decrease by 1 at a rate of 60Hz when the value of the timers are non-zero.
* There are opcodes that can manipulate and retrieve the values of the timers.
* The registers can then use the values of the timers to make decisions, branching, and logic calculations.
* When the sound timer reaches 0, Chip-8 should make a "beep."
* *Timers are independent of the normal execution cycles and of the Fx0A opcode.*

**Keyboard Buffer**

* There are 16 unique keys that can be used for input.
* Normally, each key has a unique hexadecimal value.
* For our version, unlike the original one, the buttons are arranged in order from 0 to F, and the keyboard buttons are mapped respectively.
* *An error occurs if the value in Vx is too large.*
* *In analyze-mode, if the user wants to use button/keyboard inputs and simultaneously advance the Emulator one opcode at a time, the keyboard and the on-screen controls must be used at the same time in order for this to work. If the user runs into the Fx0A opcode, inputs are automatically disabled, and the user has to first exit analyze-mode in order to use button/keyboard inputs.*

**Display/Sprites**

**Automated Testing**

* The timer nor the main cycle does not run during this part.
* Because it is very hard (and somewhat illegal) to simulate keyboard inputs in a web browser, opcodes which are related to keyboard inputs are not automatically tested. Instead, these opcodes are manually tested.

**Notes:**

* If the program is too large, the memory won't be big enough to hold all the opcodes.
* sprite\_loc() should only allow Vx values from 0 to F, or should we?
* 1nnn and 2nnn opcodes execute the opcode at location nnn.
* Adjust the way keyboard inputs are received.
* As of today, the compile function does not export programs as a portable binary file.
* Analyze-mode and pause-mode do not conflict with each other.