## CHIP-8 Specifications

## Specifications

CHIP-8 has the following components:

* Memory: CHIP-8 has direct access to up to 4 kilobytes of RAM
* Display: 64 x 32 pixels monochrome, ie. black or white
* A program counter, often called just “PC”, which points at the current instruction in memory
* One 16-bit index register called “I” which is used to point at locations in memory
* A stack for 16-bit addresses, which is used to call subroutines/functions and return from them
* An 8-bit delay timer which is decremented at a rate of 60 Hz (60 times per second) until it reaches 0
* An 8-bit sound timer which functions like the delay timer, but which also gives off a beeping sound as long as it’s not 0
* 16 8-bit (one byte) general-purpose variable registers numbered 0 through F hexadecimal, ie. 0 through 15 in decimal, called V0 through VF
  + VF is also used as a flag register; many instructions will set it to either 1 or 0 based on some rule, for example using it as a carry flag

## Memory

The memory should be 4 kB (4 kilobytes, ie. 4096 bytes) large. CHIP-8’s index register and program counter can only address 12 bits (conveniently), which is 4096 addresses.

All the memory is RAM and should be considered to be writable. CHIP-8 games can, and do, modify themselves.

## Font

The CHIP-8 emulator should have a built-in font, with sprite data representing the hexadecimal numbers from 0 through F. Each font character should be 4 pixels wide by 5 pixels tall. These font sprites are drawn just like regular sprites .

**Registers**

Chip-8 has 16 general purpose 8-bit registers, usually referred to as Vx, 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.

## Stack

CHIP-8 has a [stack](https://en.wikipedia.org/wiki/Stack_(abstract_data_type)) (a common “last in, first out” data structure where you can either “push” data to it or “pop” the last piece of data you pushed). CHIP-8 uses it to call and return from subroutines (“functions”) and nothing else, so we will be saving addresses there; 16-bit numbers.

## Timers

There are two separate timer registers: The delay timer and the sound timer. They both work the same way; they’re one byte in size, and as long as their value is above 0, they should be decremented by one 60 times per second (ie. at 60 Hz). This is independent of the speed of the fetch/decode/execute loop .

The sound timer is special in that it should make the computer “beep” as long as it’s above 0.

Even though it’s called the “delay” timer, your interpreter should run as normal while it’s being decremented (the same goes for the sound timer). The CHIP-8 game will check the value of the timer and delay itself if it wants.

**Fetch/decode/execute loop**

An emulator’s main task is simple. It runs in an infinite loop, and does these three tasks in succession:

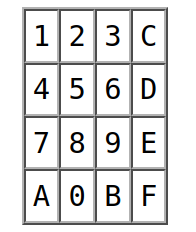
* Fetch the instruction from memory at the current PC (program counter)
* Decode the instruction to find out what the emulator should do
* Execute the instruction and do what it tells you.

**Timing**

A standard speed of around 700-900 CHIP-8 instructions per second fits well enough for most CHIP-8 programs . Different programs may work better at different instructions per seconds.

**Keyboard**

The computers which originally used the Chip-8 Language had a 16-key hexadecimal keypad with the following layout:

****

This layout must be mapped into various other configurations to fit the keyboards of today's platforms.

**Chip-8 Instructions**  
  
The original implementation of the Chip-8 language includes 36 different instructions, including math, graphics, and flow control functions.

All instructions are 2 bytes long and are stored most-significant-byte first. In memory, the first byte of each instruction should be located at an even addresses. If a program includes sprite data, it should be padded so any instructions following it will be properly situated in RAM.

**Standard Chip-8 instructions :**  
  
00E0 - CLS  
Clear the display.  
  
00EE - RET  
Return from a subroutine.

The interpreter sets the program counter to the address at the top of the stack, then subtracts 1 from the stack pointer.  
  
  
**1nnn - JP addr**  
Jump to location nnn.  
  
The interpreter sets the program counter to nnn.  
  
  
**2nnn - CALL addr**Call subroutine at nnn.  
  
The interpreter increments the stack pointer, then puts the current PC on the top of the stack. The PC is then set to nnn.  
  
  
**3xkk - SE Vx, byte**Skip next instruction if Vx = kk.  
  
The interpreter compares register Vx to kk, and if they are equal, increments the program counter by 2.  
  
  
**4xkk - SNE Vx, byte**Skip next instruction if Vx != kk.  
  
The interpreter compares register Vx to kk, and if they are not equal, increments the program counter by 2.  
  
  
**5xy0 - SE Vx, Vy**Skip next instruction if Vx = Vy.  
  
The interpreter compares register Vx to register Vy, and if they are equal, increments the program counter by 2.  
  
  
**6xkk - LD Vx, byte**Set Vx = kk.  
  
The interpreter puts the value kk into register Vx.  
  
  
**7xkk - ADD Vx, byte**Set Vx = Vx + kk.  
  
Adds the value kk to the value of register Vx, then stores the result in Vx.   
  
**8xy0 - LD Vx, Vy**Set Vx = Vy.  
  
Stores the value of register Vy in register Vx.  
  
  
**8xy1 - OR Vx, Vy**Set Vx = Vx OR Vy.  
  
Performs a bitwise OR on the values of Vx and Vy, then stores the result in Vx. A bitwise OR compares the corrseponding bits from two values, and if either bit is 1, then the same bit in the result is also 1. Otherwise, it is 0.   
  
  
**8xy2 - AND Vx, Vy**Set Vx = Vx AND Vy.  
  
Performs a bitwise AND on the values of Vx and Vy, then stores the result in Vx. A bitwise AND compares the corrseponding bits from two values, and if both bits are 1, then the same bit in the result is also 1. Otherwise, it is 0.   
  
  
**8xy3 - XOR Vx, Vy**Set Vx = Vx XOR Vy.  
  
Performs a bitwise exclusive OR on the values of Vx and Vy, then stores the result in Vx. An exclusive OR compares the corrseponding bits from two values, and if the bits are not both the same, then the corresponding bit in the result is set to 1. Otherwise, it is 0.   
  
  
**8xy4 - ADD Vx, Vy**Set Vx = Vx + Vy, set VF = carry.  
  
The values of Vx and Vy are added together. If the result is greater than 8 bits (i.e., > 255,) VF is set to 1, otherwise 0. Only the lowest 8 bits of the result are kept, and stored in Vx.  
  
  
**8xy5 - SUB Vx, Vy**Set Vx = Vx - Vy, set VF = NOT borrow.  
  
If Vx > Vy, then VF is set to 1, otherwise 0. Then Vy is subtracted from Vx, and the results stored in Vx.  
  
  
**8xy6 - SHR Vx {, Vy}**Set Vx = Vx SHR 1.  
  
If the least-significant bit of Vx is 1, then VF is set to 1, otherwise 0. Then Vx is divided by 2.  
  
  
**8xy7 - SUBN Vx, Vy**Set Vx = Vy - Vx, set VF = NOT borrow.  
  
If Vy > Vx, then VF is set to 1, otherwise 0. Then Vx is subtracted from Vy, and the results stored in Vx.  
  
  
**8xyE - SHL Vx {, Vy}**Set Vx = Vx SHL 1.  
  
If the most-significant bit of Vx is 1, then VF is set to 1, otherwise to 0. Then Vx is multiplied by 2.  
  
  
**9xy0 - SNE Vx, Vy**Skip next instruction if Vx != Vy.  
  
The values of Vx and Vy are compared, and if they are not equal, the program counter is increased by 2.  
  
  
**Annn - LD I, addr**Set I = nnn.  
  
The value of register I is set to nnn.  
  
  
**Bnnn - JP V0, addr**Jump to location nnn + V0.  
  
The program counter is set to nnn plus the value of V0.  
  
  
**Cxkk - RND Vx, byte**Set Vx = random byte AND kk.  
  
The interpreter generates a random number from 0 to 255, which is then ANDed with the value kk. The results are stored in Vx.  
  
**Dxyn - DRW Vx, Vy, nibble**Display n-byte sprite starting at memory location I at (Vx, Vy), set VF = collision.  
  
The interpreter reads n bytes from memory, starting at the address stored in I. These bytes are then displayed as sprites on screen at coordinates (Vx, Vy). Sprites are XORed onto the existing screen. If this causes any pixels to be erased, VF is set to 1, otherwise it is set to 0. If the sprite is positioned so part of it is outside the coordinates of the display, it wraps around to the opposite side of the screen.  
  
  
**Ex9E - SKP Vx**Skip next instruction if key with the value of Vx is pressed.  
  
Checks the keyboard, and if the key corresponding to the value of Vx is currently in the down position, PC is increased by 2.  
  
  
**ExA1 - SKNP Vx**Skip next instruction if key with the value of Vx is not pressed.  
  
Checks the keyboard, and if the key corresponding to the value of Vx is currently in the up position, PC is increased by 2.  
  
  
**Fx07 - LD Vx, DT**Set Vx = delay timer value.  
  
The value of DT is placed into Vx.  
  
  
**Fx0A - LD Vx, K**Wait for a key press, store the value of the key in Vx.  
  
All execution stops until a key is pressed, then the value of that key is stored in Vx.  
  
  
**Fx15 - LD DT, Vx**Set delay timer = Vx.  
  
DT is set equal to the value of Vx.  
  
  
**Fx18 - LD ST, Vx**Set sound timer = Vx.  
  
ST is set equal to the value of Vx.  
  
  
**Fx1E - ADD I, Vx**Set I = I + Vx.  
  
The values of I and Vx are added, and the results are stored in I.  
  
  
**Fx29 - LD F, Vx**Set I = location of sprite for digit Vx.  
  
The value of I is set to the location for the hexadecimal sprite corresponding to the value of Vx.  
  
**Fx33 - LD B, Vx**Store BCD representation of Vx in memory locations I, I+1, and I+2.  
  
The interpreter takes the decimal value of Vx, and places the hundreds digit in memory at location in I, the tens digit at location I+1, and the ones digit at location I+2.  
  
  
**Fx55 - LD [I], Vx**Store registers V0 through Vx in memory starting at location I.  
  
The interpreter copies the values of registers V0 through Vx into memory, starting at the address in I.  
  
  
**Fx65 - LD Vx, [I]**Read registers V0 through Vx from memory starting at location I.  
  
The interpreter reads values from memory starting at location I into registers V0 through Vx.