Cowgod's

Chip-8

Technical Reference v1.0

0.0 - Table of Contents

0.1 - Using This Document [TOC]

While creating this document, I took every effort to try to make it easy to read, as well as easy to find what you're looking for.

In most cases, where a hexadecimal value is given, it is followed by the equivalent decimal value in parenthesis. For example, "0x200 (512)."

In most cases, when a word or letter is italicized, it is referring to a variable value, for example, if I write "V*x*," the *x* reffers to a 4-bit value.

The most important thing to remember as you read this document is that every [TOC] link will take you back to the Table Of Contents. Also, links that you have not yet visited will appear in blue, while links you have used will be gray.

1.0 - About Chip-8 [TOC]

Whenever I mention to someone that I'm writing a Chip-8 interpreter, the response is always the same: "What's a Chip-8?"

Chip-8 is a simple, interpreted, programming language which was first used on some do-it-yourself computer systems in the late 1970s and early 1980s. The COSMAC VIP, DREAM 6800, and ETI 660 computers are a few examples. These computers typically were designed to use a television as a display, had between 1 and 4K of RAM, and used a 16-key hexadecimal keypad for input. The interpreter took up only 512 bytes of memory, and programs, which were entered into the computer in hexadecimal, were even smaller.

In the early 1990s, the Chip-8 language was revived by a man named Andreas Gustafsson. He created a Chip-8 interpreter for the HP48 graphing calculator, called Chip-48. The HP48 was lacking a way to easily make fast games at the time, and Chip-8 was the answer. Chip-48 later begat Super Chip-48, a modification of Chip-48 which allowed higher resolution graphics, as well as other graphical enhancements.

Chip-48 inspired a whole new crop of Chip-8 interpreters for various platforms, including MS-DOS, Windows 3.1, Amiga, HP48, MSX, Adam, and ColecoVision. I became involved with Chip-8 after stumbling upon Paul Robson's interpreter on the World Wide Web. Shortly after that, I began writing my own Chip-8 interpreter.

This document is a compilation of all the different sources of information I used while programming my interpreter.

2.0 - Chip-8 Specifications [TOC]

This section describes the Chip-8 memory, registers, display, keyboard, and timers.

2.1 - Memory [TOC]

The Chip-8 language is capable of accessing up to 4KB (4,096 bytes) of RAM, from location 0x000 (0) to 0xFFF (4095). The first 512 bytes, from 0x000 to 0x1FF, are where the original interpreter was located, and should not be used by programs.

Most Chip-8 programs start at location 0x200 (512), but some begin at 0x600 (1536). Programs beginning at 0x600 are intended for the ETI 660 computer.

Memory Map:

```
+---------------+= 0xFFF (4095) End of Chip-8 RAM
|               |
|               |
|               |
|               |
|               |
| 0x200 to 0xFFF|
|     Chip-8    |
| Program / Data|
|     Space     |
|               |
|               |
|               |
+- - - - - - - -+= 0x600 (1536) Start of ETI 660 Chip-8 programs
|               |
|               |
|               |
+---------------+= 0x200 (512) Start of most Chip-8 programs
| 0x000 to 0x1FF|
| Reserved for  |
|  interpreter  |
+---------------+= 0x000 (0) Start of Chip-8 RAM
```

2.2 - Registers [TOC]

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. See section 3.0, Instructions for details.

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 at a rate of 60Hz. See the section 2.5, Timers & Sound, for more information on these.

There are also some "pseudo-registers" which are not accessable from Chip-8 programs. The program counter (PC) should be 16-bit, and is used to store the currently executing address. The stack pointer (SP) can be 8-bit, it is used to point to the topmost level of the stack.

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

2.3 - Keyboard [TOC]

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

1	2	3	C
4	5	6	D
7	8	9	E
A	0	B	F

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

2.4 - Display [TOC]

The original implementation of the Chip-8 language used a 64x32-pixel monochrome display with this format:

(0,0)	(63,0)
(0,31)	(63,31)

Some other interpreters, most notably the one on the ETI 660, also had 64x48 and 64x64 modes. To my knowledge, no current interpreter supports these modes. More recently, Super Chip-48, an interpreter for the HP48 calculator, added a 128x64-pixel mode. This mode is now supported by most of the interpreters on other platforms.

Chip-8 draws graphics on screen through the use of sprites. A sprite is a group of bytes which are a binary representation of the desired picture. Chip-8 sprites may be up to 15 bytes, for a possible sprite size of 8x15.

Programs may also refer to a group of sprites representing the hexadecimal digits 0 through F. These sprites are 5 bytes long, or 8x5 pixels. The data should be stored in the interpreter area of Chip-8 memory (0x000 to 0x1FF). Below is a listing of each character's bytes, in binary and hexadecimal:

"0"	Binary	Hex	"1"	Binary	Hex
****	11110000	0xF0	*	00100000	0x20
* *	10010000	0x90	**	01100000	0x60
* *	10010000	0x90	*	00100000	0x20
* *	10010000	0x90	*	00100000	0x20
****	11110000	0xF0	***	01110000	0x70

"2"	Binary	Hex	"3"	Binary	Hex
****	11110000	0xF0	****	11110000	0xF0
*	00010000	0x10	*	00010000	0x10
****	11110000	0xF0	****	11110000	0xF0
*	10000000	0x80	*	00010000	0x10
****	11110000	0xF0	****	11110000	0xF0

"4"	Binary	Hex	"5"	Binary	Hex
* *	10010000	0x90	****	11110000	0xF0
* *	10010000	0x90	*	10000000	0x80
****	11110000	0xF0	****	11110000	0xF0
*	00010000	0x10	*	00010000	0x10
*	00010000	0x10	****	11110000	0xF0

"6"	Binary	Hex	"7"	Binary	Hex
****	11110000	0xF0	****	11110000	0xF0
*	10000000	0x80	*	00010000	0x10
****	11110000	0xF0	*	00100000	0x20
* *	10010000	0x90	*	01000000	0x40
****	11110000	0xF0	*	01000000	0x40

"8"	Binary	Hex	"9"	Binary	Hex
****	11110000	0xF0	****	11110000	0xF0
* *	10010000	0x90	* *	10010000	0x90
****	11110000	0xF0	****	11110000	0xF0
* *	10010000	0x90	*	00010000	0x10
****	11110000	0xF0	****	11110000	0xF0

"A"	Binary	Hex	"B"	Binary	Hex
****	11110000	0xF0	***	11100000	0xE0
* *	10010000	0x90	* *	10010000	0x90
****	11110000	0xF0	***	11100000	0xE0
* *	10010000	0x90	* *	10010000	0x90
* *	10010000	0x90	***	11100000	0xE0

"C"	Binary	Hex	"D"	Binary	Hex
****	11110000	0xF0	***	11100000	0xE0
*	10000000	0x80	* *	10010000	0x90
*	10000000	0x80	* *	10010000	0x90
*	10000000	0x80	* *	10010000	0x90
****	11110000	0xF0	***	11100000	0xE0

"E"	Binary	Hex	"F"	Binary	Hex
****	11110000	0xF0	****	11110000	0xF0
*	10000000	0x80	*	10000000	0x80
****	11110000	0xF0	****	11110000	0xF0
*	10000000	0x80	*	10000000	0x80
****	11110000	0xF0	*	10000000	0x80

2.5 - Timers & Sound [TOC]

Chip-8 provides 2 timers, a delay timer and a sound timer.

The delay timer is active whenever the delay timer register (DT) is non-zero. This timer does nothing more than subtract 1 from the value of DT at a rate of 60Hz. When DT reaches 0, it deactivates.

The sound timer is active whenever the sound timer register (ST) is non-zero. This timer also decrements at a rate of 60Hz, however, as long as ST's value is greater than zero, the Chip-8 buzzer will sound. When ST reaches zero, the sound timer deactivates.

The sound produced by the Chip-8 interpreter has only one tone. The frequency of this tone is decided by the author of the interpreter.

3.0 - Chip-8 Instructions [TOC]

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

Super Chip-48 added an additional 10 instructions, for a total of 46.

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.

This document does not yet contain descriptions of the Super Chip-48 instructions. They are, however, listed below.

In these listings, the following variables are used:

nnn or *addr* - A 12-bit value, the lowest 12 bits of the instruction
n or *nibble* - A 4-bit value, the lowest 4 bits of the instruction
x - A 4-bit value, the lower 4 bits of the high byte of the instruction
y - A 4-bit value, the upper 4 bits of the low byte of the instruction
kk or *byte* - An 8-bit value, the lowest 8 bits of the instruction

3.1 - Standard Chip-8 Instructions [TOC]

0nnn - SYS addr
Jump to a machine code routine at *nnn*.

This instruction is only used on the old computers on which Chip-8 was originally implemented. It is ignored by modern interpreters.

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 V*x* = *kk*.

The interpreter compares register V*x* to *kk*, and if they are equal, increments the program counter by 2.

4xkk - SNE Vx, byte
Skip next instruction if V*x* != *kk*.

The interpreter compares register V*x* to *kk*, and if they are not equal, increments the program counter by 2.

5xy0 - SE Vx, Vy
Skip next instruction if V*x* = V*y*.

The interpreter compares register V*x* to register V*y*, and if they are equal, increments the program counter by 2.

6xkk - LD Vx, byte
Set V*x* = *kk*.

The interpreter puts the value *kk* into register V*x*.

7xkk - ADD Vx, byte
Set V*x* = V*x* + *kk*.

Adds the value *kk* to the value of register V*x*, then stores the result in V*x*.

8xy0 - LD Vx, Vy
Set V*x* = V*y*.

Stores the value of register V*y* in register V*x*.

8xy1 - OR Vx, Vy
Set V*x* = V*x* OR V*y*.

Performs a bitwise OR on the values of V*x* and V*y*, then stores the result in V*x*. 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 V*x* = V*x* AND V*y*.

Performs a bitwise AND on the values of V*x* and V*y*, then stores the result in V*x*. 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 V*x* = V*x* XOR V*y*.

Performs a bitwise exclusive OR on the values of V*x* and V*y*, then stores the result in V*x*. 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 V*x* = V*x* + V*y*, set VF = carry.

The values of V*x* and V*y* 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 V*x*.

8xy5 - SUB Vx, Vy
Set V*x* = V*x* - V*y*, set VF = NOT borrow.

If V*x* > V*y*, then VF is set to 1, otherwise 0. Then V*y* is subtracted from V*x*, and the results stored in V*x*.

8xy6 - SHR Vx {, Vy}
Set V*x* = V*x* SHR 1.

If the least-significant bit of V*x* is 1, then VF is set to 1, otherwise 0. Then V*x* is divided by 2.

8xy7 - SUBN Vx, Vy
Set V*x* = V*y* - V*x*, set VF = NOT borrow.

If V*y* > V*x*, then VF is set to 1, otherwise 0. Then V*x* is subtracted from V*y*, and the results stored in V*x*.

8xyE - SHL Vx {, Vy}
Set V*x* = V*x* SHL 1.

If the most-significant bit of V*x* is 1, then VF is set to 1, otherwise to 0. Then V*x* is multiplied by 2.

9xy0 - SNE Vx, Vy
Skip next instruction if V*x* != V*y*.

The values of V*x* and V*y* 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 V*x* = 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 V*x*. See instruction 8xy2 for more information on AND.

Dxyn - DRW Vx, Vy, nibble
Display *n*-byte sprite starting at memory location I at (V*x*, V*y*), 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 (V*x*, V*y*). 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. See instruction 8xy3 for more information on XOR, and section 2.4, Display, for more information on the Chip-8 screen and sprites.

Ex9E - SKP Vx
Skip next instruction if key with the value of V*x* is pressed.

Checks the keyboard, and if the key corresponding to the value of V*x* is currently in the down position, PC is increased by 2.

ExA1 - SKNP Vx
Skip next instruction if key with the value of V*x* is not pressed.

Checks the keyboard, and if the key corresponding to the value of V*x* is currently in the up position, PC is increased by 2.

Fx07 - LD Vx, DT
Set V*x* = delay timer value.

The value of DT is placed into V*x*.

Fx0A - LD Vx, K
Wait for a key press, store the value of the key in V*x*.

All execution stops until a key is pressed, then the value of that key is stored in V*x*.

Fx15 - LD DT, Vx
Set delay timer = V*x*.

DT is set equal to the value of V*x*.

Fx18 - LD ST, Vx
Set sound timer = V*x*.

ST is set equal to the value of V*x*.

Fx1E - ADD I, Vx
Set I = I + V*x*.

The values of I and V*x* are added, and the results are stored in I.

Fx29 - LD F, Vx
Set I = location of sprite for digit V*x*.

The value of I is set to the location for the hexadecimal sprite corresponding to the value of V*x*. See section 2.4, Display, for more information on the Chip-8 hexadecimal font.

Fx33 - LD B, Vx
Store BCD representation of V*x* in memory locations I, I+1, and I+2.

The interpreter takes the decimal value of V*x*, 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 V*x* in memory starting at location I.

The interpreter copies the values of registers V0 through V*x* into memory, starting at the address in I.

Fx65 - LD Vx, [I]
Read registers V0 through V*x* from memory starting at location I.

The interpreter reads values from memory starting at location I into registers V0 through V*x*.

3.2 - Super Chip-48 Instructions [TOC]

00Cn - SCD nibble
00FB - SCR
00FC - SCL
00FD - EXIT
00FE - LOW
00FF - HIGH
Dxy0 - DRW Vx, Vy, 0
Fx30 - LD HF, Vx
Fx75 - LD R, Vx
Fx85 - LD Vx, R

4.0 - Interpreters [TOC]

Below is a list of every Chip-8 interpreter I could find on the World Wide Web:

Title	Version	Author	Platform(s)
Chip-48	2.20	Anrdreas Gustafsson	HP48
Chip8	1.1	Paul Robson	DOS
Chip-8 Emulator	2.0.0	David Winter	DOS
CowChip	0.1	Thomas P. Greene	Windows 3.1
DREAM MON	1.1	Paul Hayter	Amiga
Super Chip-48	1.1	Based on Chip-48, modified by Erik Bryntse	HP48
Vision-8	1.0	Marcel de Kogel	DOS, Adam, MSX, ColecoVision

5.0 - Credits [TOC]

This document was compiled by Thomas P. Greene.

Sources include:

- My own hacking.
- E-mail between David Winter and myself.
- David Winter's Chip-8 Emulator documentation.
- Christian Egeberg's Chipper documentation.
- Marcel de Kogel's Vision-8 source code.
- Paul Hayter's DREAM MON documentation.
- Paul Robson's web page.
- Andreas Gustafsson's Chip-48 documentation.

August 30, 1997 06:00:00