CHIP-8

From Wikipedia, the free encyclopedia

CHIP-8 is an interpreted programming language, developed by the late Joseph Weisbecker. It was initially used on the COSMAC VIP and Telmac 1800 8-bit microcomputers in the mid-1970s. CHIP-8 programs are run on a CHIP-8 virtual machine. It was made to allow video games to be more easily programmed for said computers.

Roughly twenty years after CHIP-8 was introduced, derived interpreters appeared for some models of graphing calculators (from the late 1980s onward, these handheld devices in many ways have more computing power than most mid-1970s microcomputers for hobbyists).

Contents

- 1 CHIP-8 applications
- 2 CHIP-8 today
- 3 Virtual machine description
  - 3.1 Memory
  - 3.2 Registers
  - 3.3 The stack
  - 3.4 Timers
  - 3.5 Input
  - 3.6 Graphics and sound
  - 3.7 Opcode table
- 4 Additional Resources

CHIP-8 applications

There are a number of classic video games ported to CHIP-8, such as Pong, Space Invaders, Tetris, and Pac-Man. There's also a random maze generator available. These programs are reportedly placed in the public domain, and can be easily found on the Internet.

CHIP-8 today

There is a CHIP-8 implementation for almost every platform imaginable today, as well as some development tools. Despite this, there are only a small number of games for the CHIP-8. Perhaps this is because many people view the CHIP-8 as a good first emulator project, and because of that, many CHIP-8 implementations have appeared.

CHIP-8 has a descendant called SCHIP (Super Chip), introduced by Erik Bryntse. In 1990, a CHIP-8 interpreter called CHIP-48 was made for HP-48 graphing calculators so that games could be programmed more easily. Its extensions to CHIP-8 are what became known as SCHIP. It features a larger resolution and several additional opcodes which make programming easier. If it were not for the development of the CHIP-48 interpreter, CHIP-8 would not be as well known today.

The next most influential developments (which popularized S/CHIP-8 on many other platforms) were David Winter's emulator, dissassembler, and extended technical documentation. It laid out a complete list of undocumented opcodes and features, and was distributed across many hobbyist forums. Many of the emulators listed below had these works as a starting point.

Virtual machine description

Memory

CHIP-8's memory addresses range from 200h to FFFh, making for 3,584 bytes. The reason for the memory starting at 200h is that on the Cosmac VIP and Telmac 1800, the first 512 bytes are reserved for the interpreter. On those machines, the uppermost 256 bytes (F00h-FFFh on a 4K machine) were reserved for display refresh, and the 96 bytes below that (EA0h-EFFh) were reserved for the call stack, internal use, and the variables.

Registers

CHIP-8 has 16 8-bit data registers named from V0 to VF. The VF register doubles as a carry flag.

The address register, which is named I, is 16 bits wide and is used with several opcodes that involve memory operations.

The stack

The stack is only used to store return addresses when subroutines are called. The original 1802 version allocated 48 bytes for up to 12 levels of nesting; modern implementations normally have at least 16 levels.

Timers

CHIP-8 has two timers. They both count down at 60 hertz, until they reach 0.

- Delay timer: This timer is intended to be used for timing the events of games. Its value can be set and read.
- Sound timer: This timer is used for sound effects. When its value is nonzero, a beeping sound is made.

Input

Input is done with a hex keyboard that has 16 keys which range from 0 to F. The '8', '4', '6', and '2' keys are typically used for directional input. Three opcodes are used to detect input. One skips an instruction if a specific key is pressed, while another does the same if a specific key is *not* pressed. The third waits for a key press, and then stores it in one of the data registers.

Graphics and sound

Display resolution is 64×32 pixels, and color is monochrome. Graphics are drawn to the screen solely by drawing sprites, which are 8 pixels wide and may be from 1 to 15 pixels in height. Sprite pixels that are set flip the color of the corresponding screen pixel, while unset sprite pixels do nothing. The carry flag (VF) is set to 1 if any screen pixels are flipped from set to unset when a sprite is drawn.

As previously described, a beeping sound is played when the value of the sound timer is nonzero.

Opcode table

CHIP-8 has 35 opcodes, which are all two bytes long. They are listed below, in hexadecimal and with the following symbols:

- NNN: address
- NN: 8-bit constant

- N: 4-bit constant
- X and Y: registers

Opcode	Explanation
0NNN	Calls RCA 1802 program at address NNN.
00E0	Clears the screen.
00EE	Returns from a subroutine.
1NNN	Jumps to address NNN.
2NNN	Calls subroutine at NNN.
3XNN	Skips the next instruction if VX equals NN.
4XNN	Skips the next instruction if VX doesn't equal NN.
5XY0	Skips the next instruction if VX equals VY.
6XNN	Sets VX to NN.
7XNN	Adds NN to VX.
8XY0	Sets VX to the value of VY.
8XY1	Sets VX to VX or VY.
8XY2	Sets VX to VX and VY.
8XY3	Sets VX to VX xor VY.
8XY4	Adds VY to VX. VF is set to 1 when there's a carry, and to 0 when there isn't.
8XY5	VY is subtracted from VX. VF is set to 0 when there's a borrow, and 1 when there isn't.
8XY6	Shifts VX right by one. VF is set to the value of the least significant bit of VX before the shift. [1]
8XY7	Sets VX to VY minus VX. VF is set to 0 when there's a borrow, and 1 when there isn't.
8XYE	Shifts VX left by one. VF is set to the value of the most significant bit of VX before the shift. [1]
9XY0	Skips the next instruction if VX doesn't equal VY.
ANNN	Sets I to the address NNN.
BNNN	Jumps to the address NNN plus V0.
CXNN	Sets VX to a random number and NN.
DXYN	Draws a sprite at coordinate (VX, VY) that has a width of 8 pixels and a height of N pixels. As described above, VF is set to 1 if any screen pixels are flipped from set to unset when the sprite is drawn, and to 0 if that doesn't happen.
EX9E	Skips the next instruction if the key stored in VX is pressed.
EXA1	Skips the next instruction if the key stored in VX isn't pressed.
FX07	Sets VX to the value of the delay timer.
FX0A	A key press is awaited, and then stored in VX.
FX15	Sets the delay timer to VX.
FX18	Sets the sound timer to VX.
FX1E	Adds VX to I.
FX29	Sets I to the location of the sprite for the character in VX. Characters 0-F (in hexadecimal) are represented by a 4x5 font.
FX33	Stores the BCD representation of VX at the addresses I, I plus 1, and I plus 2.
FX55	Stores V0 to VX in memory starting at address I. [2]

FX65	Fills V0 to VX with values from memory starting at address I. [2]

1. ^ *a* *b* On the original interpreter, the value of VY is shifted, and the result is stored into VX. On current implementations, Y is ignored.
2. ^ *a* *b* On the original interpreter, when the operation is done, I=I+X+1.