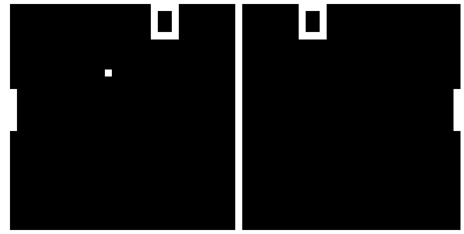
WIKIPEDIA

CHIP-8

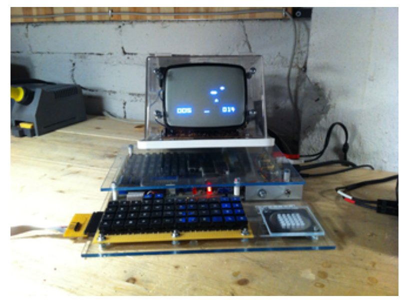
CHIP-8 is an interpreted programming language, developed by Joseph Weisbecker. It was initially used on the COSMAC VIP and Telmac 1800 8-bit microcomputers in the mid-1970s. CHIP-8 programs are run on a CHIP-8 virtual machine. It was made to allow video games to be more easily programmed for these computers.



Screenshot of *Pong* implemented in CHIP-8

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).

An active community of users and developers existed in the late 1970s, beginning with ARESCO's "VIPer" newsletter whose first three issues revealed the machine code behind the CHIP-8 interpreter.[1]



Telmac 1800 running CHIP-8 game Space Intercept (Joseph Weisbecker, 1978)

Contents

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 are also applications like a random maze generator and Conway's Game of Life. These programs are reportedly placed in the public domain, and can be easily found on the Internet.

CHIP-8 extensions

During the 1970s and 1980s, CHIP-8 users shared changes and extensions to the CHIP-8 interpreter in the VIPer magazine. These extensions included CHIP-10 and Hi-Res CHIP-8, which introduced a higher resolution than the standard 64x32, and CHIP-8C and CHIP-8X, which extended the monochrome display capabilities to support limited color, among other features.[2] These extensions were mostly backwards compatible, as they were based on the original interpreter, although some repurposed rarely used opcodes for new instructions.[3]

In 1990, a CHIP-8 interpreter called CHIP-48 was made for HP-48 graphing calculators so games could be programmed more easily. Erik Bryntse later created another interpreter based on CHIP-48, called SCHIP, S-CHIP or Super-Chip. SCHIP extended the CHIP-8 language with a larger resolution and several additional opcodes meant to make programming easier.[4] If it were not for the development of the CHIP-48 interpreter, CHIP-8 would not be as well known today.

David Winter's emulator, disassembler, and extended technical documentation popularized CHIP-8/SCHIP on many other platforms. It laid out a complete list of undocumented opcodes and features[5], and was distributed across many hobbyist forums. Many emulators used these works as a starting point.

However, CHIP-48 subtly changed the semantics of a few of the opcodes, and SCHIP continued to use those new semantics in addition to changing other opcodes. Many online resources about CHIP-8 propagate these new semantics, so many modern CHIP-8 games are not backward compatible with the original CHIP-8 interpreter for the COSMAC VIP, even if they don't specifically use the new SCHIP extensions.[6]

CHIP-8 today

There is a CHIP-8 implementation for almost every platform, as well as some development tools. Despite this, there are only a small number of games for the CHIP-8.

While CHIP-8 and SCHIP have commonly been implemented as emulators, a pure hardware implementation (written in the Verilog language) also exists for certain FPGA boards.

Virtual machine description

Memory

CHIP-8 was most commonly implemented on 4K systems, such as the Cosmac VIP and the Telmac 1800. These machines had 4096 (0x1000) memory locations, all of which are 8 bits (a byte) which is where the term CHIP-8 originated. However, the CHIP-8 interpreter itself occupies the first 512 bytes of the memory space on these machines. For this reason, most programs written for the original system begin at memory location 512 (0x200) and do not access any of the memory below the location 512 (0x200). The uppermost 256 bytes (0xF00-0xFFF) are reserved for display refresh, and the 96 bytes below that (0xEA0-0xEFF) were reserved for the call stack, internal use, and other variables.

In modern CHIP-8 implementations, where the interpreter is running natively outside the 4K memory space, there is no need to avoid the lower 512 bytes of memory (0x000-0x200), and it is common to store font data there.

Registers

CHIP-8 has 16 8-bit data registers named V0 to VF. The VF register doubles as a flag for some instructions; thus, it should be avoided. In an addition operation, VF is the carry flag, while in subtraction, it is the "no borrow" flag. In the draw instruction VF is set upon pixel collision.

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 RCA 1802 version allocated 48 bytes for up to 12 levels of nesting[7]; modern implementations usually have more[8][9].

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 ranging 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

Original CHIP-8 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 are XOR'd with corresponding screen pixels. In other words, 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 and set to 0 otherwise. This is used for collision detection.

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 and stored big-endian. The opcodes are listed below, in hexadecimal and with the following symbols:

- NNN: address
- NN: 8-bit constant
- N: 4-bit constant
- X and Y: 4-bit register identifier
- PC : Program Counter
- I : 16bit register (For memory address) (Similar to void pointer)
- VN: One of the 16 available variables. N may be 0 to F (hexadecimal)

Opcode	Type	C Pseudo	Explanation
0NNN	Call		Calls RCA 1802 program at address NNN. Not necessary for most ROMs.
00E0	Display	disp_clear()	Clears the screen.
00EE	Flow	return;	Returns from a subroutine.
1NNN	Flow	goto NNN;	Jumps to address NNN.
2NNN	Flow	*(0xNNN)()	Calls subroutine at NNN.
3XNN	Cond	if(Vx==NN)	Skips the next instruction if VX equals NN. (Usually the next instruction is a jump to skip a code block)
4XNN	Cond	if(Vx!=NN)	Skips the next instruction if VX doesn't equal NN. (Usually the next instruction is a jump to skip a code block)
5XY0	Cond	if(Vx==Vy)	Skips the next instruction if VX equals VY. (Usually the next instruction is a jump to skip a code block)
6XNN	Const	V*x* = NN	Sets VX to NN.
7XNN	Const	V*x* += NN	Adds NN to VX. (Carry flag is not changed)
8XY0	Assign	V*x*=Vy	Sets VX to the value of VY.
8XY1	BitOp	Vx=Vx|V*y*	Sets VX to VX or VY. (Bitwise OR operation)
8XY2	BitOp	Vx=V*x*&V*y*	Sets VX to VX and VY. (Bitwise AND operation)
8XY3[a]	BitOp	Vx=Vx^Vy	Sets VX to VX xor VY.
8XY4	Math	Vx += Vy	Adds VY to VX. VF is set to 1 when there's a carry, and to 0 when there isn't.
8XY5	Math	Vx -= Vy	VY is subtracted from VX. VF is set to 0 when there's a borrow, and 1 when there isn't.
8XY6[a]	BitOp	Vx>>=1	Stores the least significant bit of VX in VF and then shifts VX to the right by 1.[b]
8XY7[a]	Math	Vx=Vy-Vx	Sets VX to VY minus VX. VF is set to 0 when there's a borrow, and 1 when there isn't.
8XYE[a]	BitOp	Vx<<=1	Stores the most significant bit of VX in VF and then shifts VX to the left by 1.[b]
9XY0	Cond	if(Vx!=Vy)	Skips the next instruction if VX doesn't equal VY. (Usually the next instruction is a jump to skip a code block)
ANNN	MEM	I = NNN	Sets I to the address NNN.
BNNN	Flow	PC=V0+NNN	Jumps to the address NNN plus V0.
CXNN	Rand	Vx=rand()&NN	Sets VX to the result of a bitwise and operation on a random number (Typically: 0 to 255) and NN.
DXYN	Disp	draw(Vx,Vy,N)	Draws a sprite at coordinate (VX, VY) that has a width of 8 pixels and a height of N pixels. Each row of 8 pixels is read as bit-coded starting from memory location I; I value doesn’t change after the execution of this instruction. 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	KeyOp	if(key()==Vx)	Skips the next instruction if the key stored in VX is pressed. (Usually the next instruction is a jump to skip a code block)
EXA1	KeyOp	if(key()!=Vx)	Skips the next instruction if the key stored in VX isn't pressed. (Usually the next instruction is a jump to skip a code block)
FX07	Timer	Vx = get_delay()	Sets VX to the value of the delay timer.
FX0A	KeyOp	Vx = get_key()	A key press is awaited, and then stored in VX. (Blocking Operation. All instruction halted until next key event)
FX15	Timer	delay_timer(Vx)	Sets the delay timer to VX.
FX18	Sound	sound_timer(Vx)	Sets the sound timer to VX.
FX1E	MEM	I +=Vx	Adds VX to I. VF is set to 1 when there is a range overflow (I+VX>0xFFF), and to 0 when there isn't.[c]
FX29	MEM	I=sprite_addr[Vx]	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	BCD	set_BCD(Vx);	Stores the binary-coded decimal representation of VX, with the most significant of three

		*(I+0)=BCD(3); *(I+1)=BCD(2); *(I+2)=BCD(1);	digits at the address in I, the middle digit at I plus 1, and the least significant digit at I plus 2. (In other words, take the decimal representation of VX, place 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	MEM	reg_dump(Vx,&I)	Stores V0 to VX (including VX) in memory starting at address I. The offset from I is increased by 1 for each value written, but I itself is left unmodified.[d]
FX65	MEM	reg_load(Vx,&I)	Fills V0 to VX (including VX) with values from memory starting at address I. The offset from I is increased by 1 for each value written, but I itself is left unmodified.[d]

Notes

a. The logical opcodes 8XY3, 8XY6 and 8XYE were not documented in the original CHIP-8 specification, as all the 8000 opcodes were dispatched to instructions in the 1802's ALU, and not located in the interpreter itself; these three additional opcodes were therefore presumably unintentional functionality.

b. CHIP-8's opcodes 8XY6 and 8XYE (the bit shift instructions), which were in fact undocumented opcodes in the original interpreter, shifted the value in the register VY and stored the result in VX. The CHIP-48 and SCHIP implementations instead ignored VY, and simply shifted VX.[6]

c. This is an undocumented feature of the CHIP-8 and used by the Spacefight 2091! game.

d. In the original CHIP-8 implementation, and also in CHIP-48, I is left incremented after this instruction had been executed. In SCHIP, I is left unmodified.

References

1. "*VIPER* for RCA VIP owner" (https://books.google.com/books?id=IT4EAAAAMBAJ&pg=PA9&lpg=PA9&dq=aresco+viper#v=onepage&q=aresco%20viper&f=false). *Intelligent Machines Journal (InfoWorld)*. InfoWorld Media Group. 1978-12-11. p. 9. Retrieved 2010-01-30.
2. https://github.com/mattmikolay/chip-8/wiki/CHIP-8-Extensions-Reference
3. https://github.com/trapexit/chip-8_documentation
4. https://github.com/Chromatophore/HP48-Superchip
5. http://vanbeveren.byethost13.com/stuff/CHIP8.pdf
6. https://github.com/JohnEarnest/Octo/blob/gh-pages/docs/SuperChip.md#compatibility
7. *RCA COSMAC VIP CDP18S711 Instruction Manual* (https://archive.org/details/bitsavers_rcacosmacCManual1978_6956559/page/n36/mode/2up). Somerville: RCA Solid State Division. 1978. p. 36.
8. "Cowgod's Chip-8 Technical Reference" (http://devernay.free.fr/hacks/chip8/C8TECH10.HTM#2.2). *devernay.free.fr*. Retrieved 2020-02-03.
9. Mikolay, Matthew. "Mastering CHIP-8 : Soubroutines" (http://mattmik.com/files/chip8/mastering/chip8.html). *mattmik.com*. Retrieved 2020-02-03.

Further reading

- "RCA COSMAC VIP CDP18S711 Instruction Manual," RCA Solid State Division, Somerville, NJ 08776, February 1978. Part VIP-311. pp. 13–18, 35-37.
- BYTE magazine, December 1978 (https://archive.org/details/byte-magazine-1978-12), pp. 108–122. "An Easy Programming System," by Joseph Weisbecker. Describes CHIP-8 with specific example of a rocketship and UFO shooting-gallery game.
- Archive of Chip8.com (https://web.archive.org/web/20131030034311/http://www.chip8.com/) Website dedicated to CHIP-8 and related systems. Maintains the most complete collection of CHIP-8 programs on the net.
- Mastering CHIP-8 (http://mattmik.com/files/chip8/mastering/chip8.html), an accurate reference to the original CHIP-8 instruction set
- David Winter's CHIP-8 (http://www.pong-story.com/chip8) Emulator, utilities and games.
- BytePusher (http://esolangs.org/wiki/BytePusher) A minimalist virtual machine inspired by the CHIP-8.

- RCA COSMAC group on Yahoo (http://dir.groups.yahoo.com/group/rcacosmac/), with authorized scans of the VIPER magazine.
- OChip8 (https://web.archive.org/web/20141108023643/http://ochip8.com/) A CHIP-8 emulator in a browser
- Dream 6800 (http://www.mjbauer.biz/DREAM6800.htm) The popular Dream 6800 Microcomputer featured in Electronics Australia in 1979 ran CHIP-8.
- FPGA SuperChip (https://bitbucket.org/csoren/fpga-chip8) A Verilog implementation of the SCHIP specification.
- Octo (https://johnearnest.github.io/Octo/) is an Online CHIP-8 IDE, Development System, Compiler/Assembler and Emulator, with a proprietary scripting language
- Cowgod's Chip-8 (http://devernay.free.fr/hacks/chip8/C8TECH10.HTM) Technical Reference (CHIP-48/SCHIP)
- Matt Mikolay *CHIP-8 Extensions Reference* (http://www.mattmik.com/files/chip8/extensions/CHIP8ExtensionsReference.pdf)
- *CHIP-8.com* (https://storage.googleapis.com/wzukusers/user-34724694/documents/5c83d6a5aec8eZ0cT194/CHIP-8%20Classic%20Manual%20Rev%201.3.pdf) CHIP-8 Classic Computer Manual