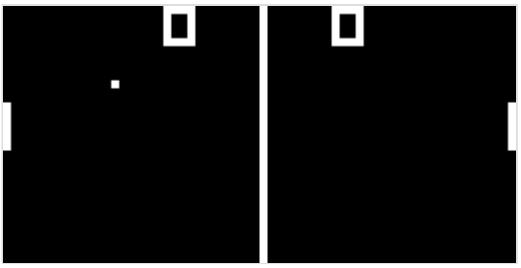
CHIP-8

CHIP-8 is an interpreted programming language, developed by Joseph Weisbecker on his 1802 microprocessor. It was initially used on the COSMAC VIP and Telmac 1800, which were 8-bit microcomputers made in the mid-1970s.



Screenshot of *Pong* implemented in CHIP-8

CHIP-8 was designed to be easy to program for, as well as using less memory than other programming languages like BASIC. [1]

Interpreters have been made for many devices, such as home computers, microcomputers, graphing calculators, mobile phones, and video game consoles. [2] [3]

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

Community

CHIP-8 has been used on a wide range of devices over time; the first community to use CHIP-8 started in the 1970s on microcomputers. They shared extensions and programs in newsletters such as ARESCO's VIPER for COSMAC VIP users or DREAMER for DREAM 6800 users[4]. In the VIPER newsletter, the first three issues detailed the machine code for the CHIP-8 interpreter for the VIP. [5]

In the 1990s, CHIP-8 interpreters started to be created for graphing calculators. Notable examples include CHIP-48 and SCHIP for the HP-48. [6]

CHIP-8 applications include original games, demos, as well as recreations of popular games from other systems. [7] With some CHIP-8 applications being in the public domain, using licenses like the Creative Commons Zero license. [8] [9]

CHIP-8 extensions and variations

During the 1970s and 1980s, CHIP-8 users shared CHIP-8 programs, but also changes and extensions to the CHIP-8 interpreter, like in the VIPER magazine for COSMAC VIP. 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. [10] These extensions were mostly backwards compatible, as they were based on the original interpreter, although some repurposed rarely used opcodes for new instructions. [11]

In 1979, Electronics Australia ran a series of articles on building a kit computer similar to the COSMAC VIP, based on the Motorola 6800 architecture. [12] This computer, the DREAM 6800, came with its own version of CHIP-8. A newsletter similar to VIPER, called DREAMER[13], was used to share CHIP-8 games for this interpreter. In 1981, Electronics Today International (ETI) ran a series of articles on building a computer, the ETI-660, which was also very similar to the VIP (and used the same microprocessor). ETI ran regular ETI-660 and general CHIP-8 columns[14] until 1985.

In 1990, a CHIP-8 interpreter called CHIP-48 was made by Andreas Gustafsson [15] for HP-48 graphing calculators. Erik Bryntse later created another interpreter based on CHIP-48,[16] titled "SUPER-CHIP", often shortened to SCHIP or S-CHIP. SCHIP extended the CHIP-8 language with a larger resolution and several additional opcodes meant to make programming easier. [17]

David Winter's interpreter, disassembler, and extended technical documentation popularized CHIP-8/SCHIP on other platforms. It laid out a complete list of undocumented opcodes and features[18] and was distributed across hobbyist forums. Many interpreters 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 backwards compatible with the original CHIP-8 interpreter for the COSMAC VIP, even if they do not specifically use the new SCHIP extensions. [19]

Some extensions take opcodes or behavior from multiple extensions, like XO-CHIP which takes some from behavior from SCHIP and CHIP-8E. [20]

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 (0x000-0x1FF), 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 12 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;[21] modern implementations usually have more.[22][23]

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. Its value can only be set.

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 : 12bit register (For memory address) (Similar to void pointer);
- VN: One of the 16 available variables. N may be 0 to F (hexadecimal);

There have been many implementations of the CHIP-8 instruction set since 1978. The following specification is based on the SUPER-CHIP specification from 1991 (but without the additional opcodes that provide extended functionality), as that is the most commonly encountered extension set today. Footnotes denote incompatibilities with the original CHIP-8 instruction set from 1978.

Opcode	Type	C Pseudocode	Explanation
0NNN	Call		Calls machine code routine (RCA 1802 for COSMAC VIP) at address NNN. Not necessary for most ROMs.[24]
00E0	Display	`disp_clear()`	Clears the screen.[24]
00EE	Flow	`return;`	Returns from a subroutine.[24]
1NNN	Flow	`goto NNN;`	Jumps to address NNN.[24]
2NNN	Flow	`*(0xNNN)()`	Calls subroutine at NNN.[24]
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).[24]
4XNN	Cond	`if (Vx != NN)`	Skips the next instruction if VX does not equal NN (usually the next instruction is a jump to skip a code block).[24]
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).[24]
6XNN	Const	`Vx = NN`	Sets VX to NN.[24]
7XNN	Const	`Vx += NN`	Adds NN to VX (carry flag is not changed).[24]
8XY0	Assig	`Vx = Vy`	Sets VX to the value of VY.[24]
8XY1	BitOp	`Vx |= Vy`	Sets VX to VX or VY. (bitwise OR operation).[24]
8XY2	BitOp	`Vx &= Vy`	Sets VX to VX and VY. (bitwise AND operation).[24]
8XY3[a]	BitOp	`Vx ^= Vy`	Sets VX to VX xor VY.[24]
8XY4	Math	`Vx += Vy`	Adds VY to VX. VF is set to 1 when there's an overflow, and to 0 when there is not.[24]
8XY5	Math	`Vx -= Vy`	VY is subtracted from VX. VF is set to 0 when there's an underflow, and 1 when there is not. (i.e. VF set to 1 if VX >= VY and 0 if not).[24]
8XY6[a]	BitOp	`Vx >>= 1`	Shifts VX to the right by 1, then stores the least significant bit of VX prior to the shift into VF.[b][24]
8XY7[a]	Math	`Vx = Vy - Vx`	Sets VX to VY minus VX. VF is set to 0 when there's an underflow, and 1 when there is not. (i.e. VF set to 1 if VY >= VX).[24]
8XYE[a]	BitOp	`Vx <<= 1`	Shifts VX to the left by 1, then sets VF to 1 if the most significant bit of VX prior to that shift was set, or to 0 if it was unset.[b][24]
9XY0	Cond	`if (Vx != Vy)`	Skips the next instruction if VX does not equal VY. (Usually the next instruction is a jump to skip a code block).[24]
ANNN	MEM	`I = NNN`	Sets I to the address NNN.[24]
BNNN	Flow	`PC = V0 + NNN`	Jumps to the address NNN plus V0.[24]
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.[24]
DXYN	Display	`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 does not 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 does not happen.[24]
EX9E	KeyOp	`if (key() == Vx)`	Skips the next instruction if the key stored in VX(only consider the lowest nibble) is pressed (usually the next instruction is a jump to skip a code block).[24]
EXA1	KeyOp	`if (key() != Vx)`	Skips the next instruction if the key stored in VX(only consider the lowest nibble) is not pressed (usually the next instruction is a jump to skip a code block).[24]
FX07	Timer	`Vx = get_delay()`	Sets VX to the value of the delay timer.[24]
FX0A	KeyOp	`Vx = get_key()`	A key press is awaited, and then stored in VX (blocking operation, all instruction halted until next key event, delay and sound timers should continue processing).[24]
FX15	Timer	`delay_timer(Vx)`	Sets the delay timer to VX.[24]
FX18	Sound	`sound_timer(Vx)`	Sets the sound timer to VX.[24]
FX1E	MEM	`I += Vx`	Adds VX to I. VF is not affected.[c][24]
FX29	MEM	`I = sprite_addr[Vx]`	Sets I to the location of the sprite for the character in VX(only consider the lowest nibble). Characters 0-F (in hexadecimal) are represented by a 4x5 font.[24]

FX33	BCD	`set_BCD(Vx)`
		`*(I+0) = BCD(3);`
		`*(I+1) = BCD(2);`
		`*(I+2) = BCD(1);`
			Stores the binary-coded decimal representation of VX, with the hundreds digit in memory at location in I, the tens digit at location I+1, and the ones digit at location I+2.[24]

FX55	MEM	`reg_dump(Vx, &I)`	Stores from 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][24]
FX65	MEM	`reg_load(Vx, &I)`	Fills from V0 to VX (including VX) with values from memory, starting at address I. The offset from I is increased by 1 for each value read, but I itself is left unmodified.[d][24]

See also

- Fantasy video game console

Notes

a. The logical opcodes 8XY3, 8XY6, 8XY7 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 four 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.[19]

c. Most CHIP-8 interpreters' FX1E instructions do not affect VF, with one exception: the CHIP-8 interpreter for the Commodore Amiga sets VF to 1 when there is a range overflow

(I+VX>0xFFF), and to 0 when there is not.[25] The only known game that depends on this behavior is Spacefight 2091!, while at least one game, Animal Race, depends on VF not being affected.

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. "An Easy Programming System". *BYTE*. Vol. 3, no. 12. December 1978. p. 108.
2. "The CHIP-8 Emulator HomePage" (https://pong-story.com/chip8/).
3. "Nintendo Game & Watch hacking scene brings Pokémon, CHIP-8 and more to the $50 handheld" (https://liliputing.com/nintendo-games-watch-hacking-scene-brings-pokemon-chip-8-and-more-to-the-50-handheld/). 8 December 2020.
4. https://archive.org/details/dreamer_newsletter_01/mode/2up
5. "*VIPER* for RCA VIP owner" (https://books.google.com/books?id=IT4EAAAAMBAJ&q=arescо+viper&pg=PA9). *Intelligent Machines Journal (InfoWorld)*. InfoWorld Media Group. 1978-12-11. p. 9. Retrieved 2010-01-30.
6. "[What I Learned About] Python and Emulators [by] Making a Chip-8 Emulator" (https://medium.com/@tobywhughes/what-i-learned-about-python-and-emulators-by-making-a-chip-8-emulator-f6ac7f25e095). 15 December 2019.
7. "The CHIP-8 Emulator HomePage" (https://pong-story.com/chip8/).
8. "Chip-8 Public Domain ROMs - Zophar's Domain" (https://www.zophar.net/pdroms/chip8.html).
9. CHIP-8 Archive (https://johnearnest.github.io/chip8Archive/)
10. CHIP-8 Extensions Reference (https://github.com/mattmikolay/chip-8/wiki/CHIP%E2%80%908-Extensions-Reference) by Matthew Mikolay
11. https://github.com/trapexit/chip-8_documentation
12. https://archive.org/stream/EA1979/EA%201979-05%20May#page/n85/mode/2up
13. https://archive.org/details/dreamer_newsletter_01/mode/2up
14. https://archive.org/stream/ETIA1981/ETI%201981-11%20November#page/n113/mode/2up
15. "CHIP-48 source code" (https://github.com/Chromatophore/HP48-Superchip/blob/master/binaries/source/c48_source.txt). *GitHub*.
16. SCHIP 1.1 documentation (http://devernay.free.fr/hacks/chip8/schip.txt)
17. HP48-Superchip (https://github.com/Chromatophore/HP48-Superchip) Repository on GitHub
18. CHIP8.pdf (https://web.archive.org/web/20140825173007/http://vanbeveren.byethost13.com/stuff/CHIP8.pdf) by David WINTER (HPMANIAC)
19. Mastering SuperChip (https://github.com/JohnEarnest/Octo/blob/gh-pages/docs/SuperChip.md#compatibility) Compatibility section
20. "XO-CHIP Specification by John Earnest" (https://github.com/JohnEarnest/Octo/blob/gh-pages/docs/XO-ChipSpecification.md). *GitHub*.
21. *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.
22. Greene, Thomas P. (1997-08-30). "Cowgod's Chip-8 Technical Reference" (http://devernay.free.fr/hacks/chip8/C8TECH10.HTM). *devernay.free.fr*. Archived (https://web.archive.org/web/20240103155311/http://devernay.free.fr/hacks/chip8/C8TECH10.HTM) from the original on 2024-01-03. Retrieved 2020-02-03.

23. Mikolay, Matthew. "Mastering CHIP-8 : Subroutines" (http://mattmik.com/files/chip8/mastering/chip8.html). *mattmik.com*. Retrieved 2020-02-03.
24. Schümann, Steffen "Gulrak". "CHIP-8 Variant Opcode Table" (https://chip8.gulrak.net/). *chip8.gulrak.net*.
25. "FX1E and VF · Issue #2 · Chromatophore/HP48-Superchip · GitHub" (https://github.com/Chromatophore/HP48-Superchip/issues/2). *GitHub*.

Further reading

- CHIP-8 Variant Opcode Table (https://chip8.gulrak.net/) an accurate reference to the opcodes for CHIP-8 and several extensions, including CHIP-48, SCHIP 1.0, 1.1, XO-CHIP.
- Mastering CHIP-8 (http://mattmik.com/files/chip8/mastering/chip8.html), an accurate reference to the original CHIP-8 instruction set.
- Cowgod's Chip-8 Technical Reference (http://devernay.free.fr/hacks/chip8/C8TECH10.HTM) contains inaccuracies and omissions in several instructions.
- Matt Mikolay *CHIP-8 Extensions Reference* (http://www.mattmik.com/files/chip8/extensions/CHIP8ExtensionsReference.pdf)
- RCA COSMAC group on Yahoo (https://archive.today/20130412050150/http://dir.groups.yahoo.com/group/rcacosmac/), with authorized scans of the VIPER magazine.
- *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
- Archive of Chip8.com (https://web.archive.org/web/20131030034311/http://www.chip8.com/) Website dedicated to CHIP-8 and related systems. Maintained a large collection of CHIP-8 programs on the net.
- "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.

External links

- 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
- Cadmium (https://github.com/gulrak/cadmium) is a CHIP-8 interpreter for computers, with a GUI which allows for selecting accurate emulation of different extensions and implementations of CHIP-8
- Silicon8 (https://timendus.github.io/silicon8/) A CHIP-8 emulator for web browsers.