CHIP-8 extensions and compatibility

Preamble

This document is intended as a well-sourced and exhaustive documentation of all extensions to CHIP-8. They all use the standard CHIP-8 interpreter as a base, mentioning what additions and changes they make to that common base.

This list is mostly for historical purposes; few of the extensions listed here had a considerable number of programs written for them. It could serve as a reference for CHIP-8 emulator/interpreter developers who want to support the various incompatible instructions and behaviors that have cropped up in different interpreters over the decades. Perhaps the instructions listed here can also inspire someone to make their own modern CHIP-8 extension (of which there have been a couple, see the bottom of the list).

A note on modern implementations

The only CHIP-8 extensions in active use today (beyond basic CHIP-8) are SUPER-CHIP 1.1 and the relative newcomer XO-CHIP.

When SUPER-CHIP for the HP48 calculators revived CHIP-8 in the 1990s, it introduced several incompatibilities with the original CHIP-8 interpreters from the 1970s and '80s. These incompatibilities have persisted until today, when CHIP-8 has become the de facto "Hello, world!" project for emulator developers. Even modern, basic CHIP-8 interpreters and games that do not support the SUPER-CHIP extensions will assume the "quirky" SUPER-CHIP behavior.

This means that many of the original pre-1990 CHIP-8 games do not run in modern interpreters. For an overview of CHIP-8 games and their compatibility, see the CHIP-8 database project.

A note on machine code routines

In the original CHIP-8 interpreter, instructions on the form `0NNN` would execute a 1802 machine code routine located at address `0NNN`. The instructions `00E0` (clear screen) and `00EE` (return from CHIP-8 subroutine) are both such routines. As they were explicitly listed in the instruction table, I treat them as regular instructions here. The same goes for any such subroutines that were added in extended interpreters. After all, what is really the difference between a machine code routine and a "regular" CHIP-8 instruction, apart from how the interpreter dispatches them? If a machine code instruction was exposed to the users of the interpreters, we consider it a CHIP-8 instruction for the purpose of this document.

Some CHIP-8 interpreters for other microcomputers during the late 1970s and early '80s also supported calling machine code routines for their specific CPUs with a general `0NNN` instruction, but this is not well documented and few programs took advantage of it. For this reason, this instruction is generally not listed below; more research would be needed to say conclusively what interpreters support it.

Sources

The list builds on these modern documents, in addition to historical sources:

- A collection of documentation on the CHIP-8 and related (https://github.com/trapexit/chip-8_documentation) – Lists the instructions of the most well-known extensions
- CHIP-8 Extensions Reference (https://github.com/mattmikolay/chip-8/wiki/CHIP%E2%80%908-Extensions-Reference)
- SCHIP Compatibility (https://github.com/Chromatophore/HP48-Superchip)
- xChip manual (https://github.com/mehcode/xchip/blob/master/docs/manual.md)

Commonalities and notation

CHIP-8 interpreters typically (unless otherwise stated) have the following components:

- `200` (512) bytes of reserved memory for the interpreter, followed by the loaded CHIP-8 program, all in RAM (self-modifying code allowed)
- A 12-bit (or occasionally larger) program counter that starts at address `200`
- A stack of addresses, which is only used for subroutines
- A 12-bit (or occasionally larger) addressing register called `I`
- 16 8-bit registers or variables called `V0` – `VF`; `VF` is used as a flag register for certain instructions
- Two 8-bit counters called the "delay timer" (read/white) and "sound timer" (write only) that both count down at 60 Hz; if the sound timer is non-zero, an audible beep is usually heard
- A frame buffer that is automatically copied to the screen bit-to-pixel each frame, usually manipulated by a single drawing instruction (and a "clear screen" instruction)

All CHIP-8 instructions are two bytes long, written as a hexadecimal number. The opcode is recognized by the first hexadecimal digit (or nibble) and sometimes also additional digits. The arguments are designated as follows:

- `X` : One of CHIP-8's 16 registers/variables, `V0` – `VF`
- `Y` : One of CHIP-8's 16 registers/variables, `V0` – `VF`
- `N` : A 4-bit number, ie. a nibble
- `NN` : An 8-bit number, ie. a byte
- `NNN` : A 12-bit memory address
- `NNNN` : A 16-bit memory address (only found in certain extensions)

What this list does not cover

Systems inspired by, but completely incompatible with, CHIP-8 – such as CHIP-12 (http://web.archive.org/web/20170408173026/http://marcpic.com/CHIP12), CHIP-16 (https://github.com/chip16/chip16/wiki) and BytePusher (https://esolangs.org/wiki/BytePusher) - are not listed here.

CHIP-8

The original CHIP-8 interpreter created for the RCA COSMAC VIP by Joseph Weisbecker. It was first detailed in VIPER volume 1, issue 1, 1978.

The instructions `8XY3`, `8XY6`, `8XY7` and `8XYE` were all part of the interpreter from the beginning, but they were initially undocumented. They were quickly found, however, and revealed in VIPER issue 2.

Instructions

- `0NNN` : Execute RCA 1802 machine language routine at address `NNN`
- `00E0` : Clear the screen
- `00EE` : Return from subroutine
- `1NNN` : Jump to address `NNN`
- `2NNN` : Call subroutine at address `NNN`
- `3XNN` : Skip the following instruction if the value of register VX equals `NN`
- `4XNN` : Skip the following instruction if the value of register VX is not equal to `NN`
- `5XY0` : Skip the following instruction if the value of register VX is equal to the value of register VY
- `6XNN` : Set VX to `NN`
- `7XNN` : Add `NN` to VX
- `8XY0` : Set VX to the value in VY
- `8XY1` : Set VX to VX OR VY
- `8XY2` : Set VX to VX AND VY
- `8XY3` : Set VX to VX XOR VY
- `8XY4` : Add the value of register VY to register VX. Set VF to 01 if a carry occurs. Set VF to 00 if a carry does not occur
- `8XY5` : Subtract the value of register VY from register VX. Set VF to 00 if a borrow occurs. Set VF to 01 if a borrow does not occur
- `8XY6` : Store the value of register VY shifted right one bit in register VX. Set register VF to the least significant bit prior to the shift
- `8XY7` : Set register VX to the value of VY minus VX. Set VF to 00 if a borrow occurs. Set VF to 01 if a borrow does not occur
- `8XYE` : Store the value of register VY shifted left one bit in register VX. Set register VF to the most significant bit prior to the shift
- `9XY0` : Skip the following instruction if the value of register VX is not equal to the value of register VY
- `ANNN` : Store memory address NNN in register I
- `BNNN` : Jump to address NNN + V0
- `CXNN` : Set VX to a random number with a mask of NN
- `DXYN` : Draw a sprite at position VX, VY with N bytes of sprite data starting at the address stored in I. Set VF to 01 if any set pixels are changed to unset, and 00 otherwise

- `EX9E` : Skip the following instruction if the key corresponding to the hex value currently stored in register VX is pressed
- `EXA1` : Skip the following instruction if the key corresponding to the hex value currently stored in register VX is not pressed
- `FX07` : Store the current value of the delay timer in register VX
- `FX0A` : Wait for a keypress and store the result in register VX
- `FX15` : Set the delay timer to the value of register VX
- `FX18` : Set the sound timer to the value of register VX
- `FX1E` : Add the value stored in register VX to register I
- `FX29` : Set I to the memory address of the sprite data corresponding to the hexadecimal digit stored in register VX
- `FX33` : Store the binary-coded decimal equivalent of the value stored in register VX at addresses I, I+1, and I+2
- `FX55` : Store the values of registers V0 to VX inclusive in memory starting at address I. I is set to I + X + 1 after operation
- `FX65` : Fill registers V0 to VX inclusive with the values stored in memory starting at address I. I is set to I + X + 1 after operation

Compatibility notes

- The original CHIP-8 interpreter occupied the first 512 bytes of memory, loading in CHIP-8 programs starting at memory location `0200`.
- The execution of a CHIP-8 program did in fact start at `01FC`, executing the following two machine code instructions located at the end of the interpreter: `00E0` (clear screen) and `004B` (turn the display on). These were both machine code routines.
- The final 352 bytes of memory were reserved for "variables and screen refresh" (ie. the CHIP-8 "registers", stack and a frame buffer). On a COSMAC VIP with 2048 bytes of RAM, this means addresses `0700` to `07FF` were reserved; with 4096 bytes of RAM, `0E90` to `0FFF` were reserved.
- The original interpreter supports 12 stack entries.
- `DXYN` waits for the display interrupt/vertical blanking period before drawing to the screen. This slows down execution, but eliminates sprite tearing.
- `DXYN` does a modulo of the values in VX and VY before drawing, meaning that it wraps sprites that would be drawn in their entirety outside of the display area.
- `DXYN` clips sprites that are partially drawn outside of the display area.
- The sound timer on a COSMAC VIP would not respond to a value of 1.

The COSMAC VIP hexadecimal keypad had the following layout:

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

CHIP-8 1/2

An interpreter written by Peter K. Morrison, mentioned in VIPER volume 1, issue 2, 1977. The code for this interpreter is not given. It is incompatible with CHIP-8, as it moves several instructions around in memory. Of note here is the expansion of the display instruction.

New instructions

- `NXMM` : Branch to MM if VX = 0 or VX != 0.

Altered instructions

- The `FX` series of instructions are moved to page 2 of the memory, allowing a full page of this instruction type.
- `5XY0` and `9XY0` are combined into one opcode (details unclear).
- `EXA1` and `EX9E` are moved to the `FX` series of instructions (presumably to `FXA1` and `FX9E`).
- The display instruction was expanded to include OR, AND, XOR and test functions (details unclear).

CHIP-8I

This is a modification of CHIP-8 to provide I/O instructions, by Rick Simpson. It was detailed in VIPER issue 3.

In addition to the new instructions below, it was suggested that the Q-line on the COSMAC VIP (the sound output) be connected to the add-on card you wish to interface with, and that setting the sound timer to 1 (which would not produce an audible tone on the COSMAC VIP) would signal to the card that it can consume input. Likewise, the EF4 line on the COSMAC VIP would be connected to the card and used to signal to the VIP that input should be consumed.

This could be used with an external keyboard to allow ASCII input.

New instructions

- `B0NN` : Output NN to port
- `B1X0` : Output VX to port
- `B1X1` : Wait for input (EF4 line is low) and then set VX to input from port

Altered instructions

- `BNNN` (jump to `0NNN` + V0) is removed.

CHIP-8 II aka. Keyboard Kontrol

A modification of CHIP-8 I which adds support for real-time games and two player games. It was made by Tom Swan, and detailed in VIPER volume 2, issue 4.

New instructions

The same as for CHIP-8 I above, and:

- `FX00` : Set VX to input from port (same as `B1X1` , but does not halt)

CHIP-8III

A modification of CHIP-8I, CHIP-8 II and CHIP-8 with I/O port driver routine. It was made by John Chmielewski and detailed in VIPER volume 2, issue 7.

New instructions

- `FX00` : Set VX to input from port if key is pressed (EF4 is low), or to 0 otherwise
- `FXF2` : Wait for input (EF4 is low) and then set VX to input from port
- `FXF9` : Output VX to port (X is V0–VE)

Compatibility notes

- The VF register could not be used with `FXF9` .

Two-page display for CHIP-8

A modified CHIP-8 interpreter that increases the resolution to 64 x 64, by Andy Modla and Jef Winsor. It is detailed in VIPER volume 1, issue 3. It was used in several programs presented in Tom Swan's book *PIPS FOR VIPS*.

New instructions

- `0230` : Clear screen

Compatibility notes

- CHIP-8 programs are loaded at `0260` .
- The new two-page clear screen routine is located at `0230` , and some games seem to rely on this by calling it instead of `00E0` , although (it seems to me) the interpreter still supports using the old `00E0` opcode for this.
- The interpreter still starts execution at address `01FC` like before; address `0200` holds a `1260` instruction to skip over the new interrupt and clear screen routines. Many newer interpreters (https://github.com/dmatlack/chip8/blob/master/roms/hires/!hires_information.txt) seem to rely on this to recognize a game written for the two-page interpreter, since it seems some games seem to actually include the parts of the interpreter from `0200` – `0260` in their binary file.

CHIP-8C

This was a color-language addition to CHIP-8, which was advertised in VIPER issue 2 as coming out in "late October" of 1978, but it was apparently never released. It would require an RCA Color Board VP-590, and support three background colors and eight foreground colors. It's likely it was supplanted by CHIP-8X.

According to Ipso Facto issue 12, it had 5 new instructions. CHIP-8X has only 4 instructions pertaining to color.

CHIP-10

This is a modification to the original CHIP-8 interpreter that expands the resolution to 128 x 64, detailed in VIPER issue 7 and Ipso Facto issue 10. It was created by Ben H. Hutchinson, Jr.

It required a hardware add-on to expand the horizontal resolution, and reduced the pixel height from four to two scanlines. It's called "CHIP-10" since it controls 10 bits worth of display memory rather than 8. The hardware requirements were outlined in Ipso Facto issue 11.

Compatibility notes

- The memory addresses `055F` to `0FFF` are reserved.

Altered instructions

- `00E0`
- `DXYN` : Looks at the bottom 7 instead of 6 bits of X, and the bottom 6 instead of 5 bits of Y. Sprites obviously no longer wrap at the same coordinates, so not all CHIP-8 programs run without modification.

CHIP-8 modification for saving and restoring variables

By John Bennett, from VIPER volume 1, issue 10.

New instructions

- `FXF2` `FY55` : Store VX–VY to memory location I.
- `FXF2` `FY65` : Load VX–VY from memory location I.

Altered instructions

- `FX55` and `FX65` should no longer be used without a preceding `FXF2` instruction.

Improved CHIP-8 modification for saving and restoring variables

---

By John Bennett, from VIPER volume 2, issue 2.

New instructions

- `FXY0` : Store VX–VY to memory location I. I is changed.
- `FXY1` : Load VX–VY from memory location I. I is changed.

Altered instructions

- `FX55` and `FX65` are replaced with the new opcodes.

CHIP-8 modification with relative branching

By Wayne Smith, in VIPER volume 2, issue 1.

New instructions

- `BFNN` : Jump to current address + NN bytes
- `BBNN` : Jump to current address - NN bytes

Altered instructions

- `BNNN` is removed.

Another CHIP-8 modification with relative branching

By Tom Swan, in VIPER volume 2, issue 5.

New instructions

- `FXA4` : Jump to current address + the number of instructions in VX
- `FXAE` : Jump to current address - the number of instructions in VX (current instruction is counted)

CHIP-8 modification with fast, single-dot DXYN

By Wayne Smith, in VIPER volume 2, issue 1.

Altered instructions

- `DXYN` now ignores N, and draws a single pixel at location VX,VY.

CHIP-8 with I/O port driver routine

By James Barnes, in VIPER volume 2, issue 2.

New instructions

- `FXF2` : Transfer input data to VX
- `FXF5` : Transfer output data from VX

CHIP-8 8-bit multiply and divide

By Wayne E. Smith, Jr. Detailed in VIPER volume 2, issue 3.

New instructions

- `BXY0` : VF, VX = VX * VY
- `BXYN` (where N > 0): VX = VX / VY (VF = remainder)

Altered instructions

- `BNNN` is removed.

HI-RES CHIP-8 (four-page display)

Hi-res CHIP-8 increases the screen resolution from 64 x 32 to 64 x 128 by reducing each pixel's height from 4 to 2 scanlines. It was created by Tom Swan and detailed in VIPER volume 2, issue 6.

New instructions

- `0200` : Clear screen
- `0216` : Protect the number of pages in `V0` (0-4) from subsequent calls to `0200`, from the bottom of the screen up. Can be used to keep the top 32, 64 or 96 horizontal pixels persistent.

Altered instructions

- `00E0` has been deprecated

Compatibility notes

- Hi-res CHIP-8 programs are loaded starting from `0244`.
- This interpreter runs much faster on a COSMAC VIP than standard CHIP-8.

HI-RES CHIP-8 with I/O

An extension of Hi-res CHIP-8 by Tom Swan which adds some I/O instructions and the MESSAGER program from the first PIPS FOR VIPS book. It was detailed in the appendix to PIPS FOR VIPS IV, as well as VIPER volume 5, issue 3 which featured excerpts from that book.

New instructions

- `0200` : Clear screen
- `0216` : Protect the number of pages in `V0` (0-4) from subsequent calls to `0200`, from the bottom of the screen up. Can be used to keep the top 32, 64 or 96 horizontal pixels persistent.
- `0244 DXYN` : Prints the null-terminated ASCII string pointed at by I, starting at coordinates VX,VY.
- `BXA7` : Output VX to port (X != F)
- `BXA9` : Load VX with input from port
- `BXAB` : Wait for keypress, then load VX with input from port
- `BXB1` : Strobe (set and reset Q-line)

Altered instructions

- `00E0` has been deprecated
- `BNNN` is removed

Compatibility notes

- The interpreter itself and the MESSAGER program occupies memory addresses `0000` – `03FF`, and there's an ASCII character set from `0300` – `04FF`. Therefore, CHIP-8 programs are loaded starting from `0500`.
- With 4K memory, the display buffers occupy `0800` – `0BFF` and `0C00` – `0FFF`, leaving only `0500` – `07FF` for CHIP-8 programs unless additional memory is installed (in which case the interpreter must be modified to relocate the buffers).
- Memory `0FB0` – `0FFF` is in any case reserved for use by the VIP operating system.

HI-RES CHIP-8 with page switching

A modification of Hi-res CHIP-8 which adds a display buffer to memory, allowing the entire display to be flipped for smooth animation multi-sprite drawings, among other things. It was made by Tom Swan and is detailed in VIPS FOR PIPS IV and VIPER volume 5, issue 3.

New instructions

- `0200` : Clear inactive display buffer
- `0216` : Toggle which display buffer is active and displayed on the screen
- `BXA7` : Output VX to port (X != F)
- `BXA9` : Load VX with input from port
- `BXAB` : Wait for keypress, then load VX with input from port
- `BXB1` : Strobe (set and reset Q-line)

Altered instructions

- `00E0` has been deprecated
- `BNNN` is removed
- `DXYN` draws to the inactive display buffer

Compatibility notes

- The display buffers are not initialized when the interpreter starts; `0200` must be called for each page to erase them before using them.
- `DXYN` no longer clips sprites drawn off the bottom of the screen correctly. A sprite will wrap from the bottom of page #1 to the top of the visible page #2, and a sprite drawn off the bottom of page #2 might overwrite part of the CHIP-8 interpreter itself.

CHIP-8E

This is a rewritten CHIP-8 interpreter which incorporates many additions from previous extensions. It was written by Gilles Detillieux and detailed in VIPER volume 2, issue 8/9.

New instructions

- `00ED` : Stop execution
- `0151` : Stops execution when timer = 0
- `00F2` : No operation
- `0188` : Skip next instruction
- `5XY1` : Skip if VX > VY
- `5XY2` : Saves VX–VY to memory location in I
- `5XY3` : Loads VX–VY from memory location in I
- `BBNN` : Jump to current memory location - NN
- `BFNN` : Jump to current memory location + NN
- `FX03` : Transfer output data from VX to port 3
- `FX1B` : Skip the number of bytes in VX
- `FX4F` : Timer = VX, then wait until timer = 0
- `FXE3` : Wait for input (EF4 line is low) from port 3, then load into VX
- `FXE7` : Read input from port 3 into VX

CHIP-8 with improved BNNN

A short modification by George Ziniewicz from VIPER volume 2, issue 8/9.

New instructions

- `FXF2 BNNN` : Jump to NNN + VX

Altered instructions

- `BNNN` should no longer be used on its own, unless you know the value of 1802 register RD.0

CHIP-8 scrolling routine

A short machine language subroutine that scrolls the display upwards by 1 pixel. Written by Tom Swan, detailed in VIPER volume 3, issue 1.

CHIP-8X

The first official CHIP-8 extension by RCA, released in 1980. It added support for the Color Card, Simple Sound and expansion hex keyboard. It was detailed in

The instruction listing for CHIP-8X still did not include the undocumented opcodes.

New instructions

- `FXF8` : Output VX to port
- `FXFB` : Wait for input (EF4 is low) and load into VX
- `EXF2` : Skip next instruction if the key in VX is pressed on hexadecimal keyboard 2
- `EXF5` : Skip next instruction if the key in VX is not pressed on hexadecimal keyboard 2
- `BXY0` : Set the foreground color of the pixel area defined by VX and VX+1 to the color defined in VY (VY <= 7, where values correspond to black, red, blue, violet, green, yellow, aqua and white, respectively). The display is split into 8 x 8 zones (8 x 4 pixels each); the least significant nibble of VX specifies the horizontal position of the left-most zone, and the most significant nibble of VX specifies the extra number of horizontal zones to color (ie. a value of 0 will color one zone). Ditto for VX+1, but with vertical zones.
- `BXYN` : Set the foreground color of the pixel area where VX is the horizontal coordinate and VX+1 is the vertical, for 8 horizontal pixels (similar to DXYN), to the color defined in VY. (N > 0)
- `02A0` : Step background color (cycles between blue, black, green and red)
- `5XY1` : Adds each nibble in VX to each nibble in VY, and stores the result in VX modulus 8

Altered instructions

- `BNNN` is removed.

Compatibility notes

- CHIP-8X programs begin at address `0300`.
- Execution of a CHIP-8X program started at `02FA` (recall that CHIP-8 started execution at `01FC`), where it called machine code routine `0280` which looked for a color map at `0C00` to determine whether it was running on a system with a Color Card or not.

Two-page display for CHIP-8X

A modification of the two-page display for CHIP-8 (by Andy Modle and Jef Winsor, detailed above) which adds CHIP-8X functionality. In other words, it is a CHIP-8X with 64 x 64 resolution. It required hardware modifications. It is written by Jeff Jones and is detailed in VIPER volume 4, issue 3.

New instructions

- Same as CHIP-8X, except `BXY0`
- `00F0` : Return from subroutine (replaces `00EE`)

Altered instructions

- `BXY0` is removed
- `00EE` is removed
- `5XY1` can handle numbers up to FF

Hi-res CHIP-8X

A modification of Hi-res CHIP-8 (by Tom Swan, detailed above) which adds CHIP-8X functionality. In other words, it is a CHIP-8X with 64 x 128 resolution. It required hardware modifications. It is written by Jeff Jones and is detailed in VIPER volume 4, issue 3.

New instructions

- Same as CHIP-8X, except `BXY0`
- Same as Hi-res CHIP-8

Altered instructions

- `BXY0` is removed
- `5XY1` can handle numbers up to FF

Compatibility notes

- A slight disturbance would occur on the screen every time `5XYN` was executed.

CHIP-8Y

Bob Casey's CHIP-8 with I/O modifications. Detailed in VIPER volume 3, issue 1. Similar to CHIP-8I and CHIP-8X, but compatible with CHIP-8.

New instructions

- `FXF3` : Wait for input (EF4 line is low) and then set VX to input from port
- `FXF8` : Output VX to port

CHIP-8 "Copy to Screen"

By Tom Swan in VIPER volume 3, issue 4.

Altered instructions

- `DXYN` now draws pixel values directly to the screen, instead of using XOR.

CHIP-BETA

Ron Applebach's interpreter, mentioned in VIPER volume 3, issue 5. It features a 64 x 64 resolution, plus the I/O instructions from CHIP-8X to run Simple Sound and player 2 hexadecimal keyboard. Presumably similar to the two-page CHIP-8X interpreter.

CHIP-8M

A CHIP-8 interpreter that can send International Morse Code, written by Steven Vincent Gunhouse and detailed in VIPER volume 4, issue 5.

New instructions

- `027A` : Initialize Morse output registers
- `0280` : Send long space
- `0288` : Wait for end of current character
- `FX00` : Output VX to output port
- `FXBC` : Output Morse code of ASCII character in VX
- `FXC8` : Output Morse code of hexadecimal (least significant nibble) in VX
- `FXF2` : Input from standard, parallel ASCII keyboard

Compatibility notes

- The interrupt routine is longer, as it's used for timing the morse code, so programs will run a little slower.
- CHIP-8M programs start at `0300`.
- Since the hexadecimal keypad and `FX18` (the regular sound timer instruction) both use the Q line, and the Morse output is buffered for timing and interrupt, it is necessary to explicitly separate Morse code output and sound/keypad instructions. You must use either `0280` or `0288` after `FX18` and before `FXBC` or `FXC8` to separate regular sound from Morse. The sound timer is also unusable during Morse output for the same reason, so an invalid ASCII character (such as 01) or `0288` must be used after Morse code and before `FX18` or keypad instructions.
- The original listing in VIPER does not have a pre-set time constant, ie. the speed of the Morse code tones. There's no default speed listed, just a placeholder. The time constant is located at address `024E` and so can be set by programs with the instructions `A24E 60NN F055` or similar. A value of 0 should be about 55 wpm, 0A is 5 wpm, 04 is 11 wpm, 03 is 14 wpm, 02 is just over 18 wpm, and 01 is 27.5 wpm.

Multiple Nim interpreter

Detailed in VIPER volume 4, issue 5, this is a modified CHIP-8 interpreter that is used to play the game "Multiple Nim" (a variant of the well-known mathematical strategy game Nim (https://en.wikipedia.org/wiki/Nim)).

The changes in this interpreter are not specified, but the machine code for it is given, so if someone has some time to kill it should be possible to find out.

Double Array Modification

A modification to any CHIP-8 interpreter which allows easy lookup in a two-dimensional array. It was written by Ron Applebach and detailed in VIPER volume 4, issue 6.

The new instruction replaces any old instruction, but the article suggests replacing `BNNN` as that was not commonly used, and available in the standard CHIP-8 interpreter.

New instructions

- `BXYN` : Sets I to the memory location that holds the value VX,VY in the two-dimensional array in page N

Altered instructions

- `BNNN` is removed

Compatibility notes

- CHIP-8 programs begin at address `0240`.
- The display buffer for CHIP-8 was originally located on page F, so the instruction `BXYF` would address an 8-pixel (1 byte) wide column on the screen and allow changing it directly. All the included example programs used page F to demonstrate the routine. Modern interpreters likely do not store the display buffer in actual, addressable memory.

CHIP-8 for DREAM 6800 (CHIPOS)

Created by Michael Bauer. Described in Electronics Australia, May 1979 (https://archive.org/stream/EA1979/EA%201979-05%20May#page/n85/mode/2up). The DREAM 6800 had its own newsletter, DREAMER, which ran for 19 issues from 1980 and into 1982 and provided many CHIP-8 programs.

The interpreter is run from a monitor program called CHIPOS, sometimes used as the name for the interpreter itself.

New instructions

- `0000`: No operation
- `F000`: Stop interpreter and return to CHIPOS
- `DXY0`: Draws an 8 x 16 pixel high sprite; otherwise same as `DXYN`

Altered instructions

- `FX0A`: As before, but if you press the additional FN key on the keypad, VX is set to `8C`. There is an audible beep when pressing a key at this instruction (not at the others).
- `8XY3`, `8XY6`, `8XY7`, `8XYE`: The undocumented instructions for the COSMAC VIP interpreter were not implemented.

Compatibility notes

- All `00NN` instructions where `NN` is not `E0` (erase screen) or `EE` (return) are actually no-ops (since CHIPOS uses many of the RAM addresses in the zero page for internal housekeeping, it also doesn't want to provide this memory for `0NNN` machine code subroutines), but `0000` is the no-op instruction supplied in the manual.
- All `FXNN` instructions that didn't exist in the original CHIP-8 interpreter exit to the monitor, but `F000` is the one supplied in the manual.
- The delay timer is decremented every frame, regardless of whether it's 0 or not; on the COSMAC VIP, it would only decrement as long as it was over 0. Its value will therefore go from 0 to 255, and any CHIP-8 game that checks if the delay timer is 0 without a busy loop will need to get lucky to get the right timing.
- The CHIP-8 interpreter is paused while the sound beeper is playing; on the COSMAC VIP, play would continue while the beep sounded.

The prototype DREAM 6800 shown in the *Electronics Australia* article (see Michael Bauer's DREAM 6800 Archive website (http://www.mjbauer.biz/DREAM6800.htm)) used the following keypad layout:

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

This is also the layout used by the CHIP-8 Classic reproduction computer (see below).

However, most DREAM computers used off-the-shelf Digitran keypads, with the following layout:

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

This became the "standard" layout used by all programs in the *DREAMER* magazine. It's also the layout used by David Fry's 40th anniversary DREAM 6800 reproduction computer.

CHIP-8 with logical operators for DREAM 6800 (CHIPOSLO)

A modified version of the DREAM 6800 interpreter, with the missing four undocumented instructions added, made by Tobias V. Langhoff. GitHub repository (https://github.com/tobiasvl/chiposlo).

New instructions

- `8XY3` : Set VX to VX XOR VY
- `8XY6` : Store the value of register VY shifted right one bit in register VX. Set register VF to the least significant bit prior to the shift
- `8XY7` : Set register VX to the value of VY minus VX. Set VF to 00 if a borrow occurs. Set VF to 01 if a borrow does not occur
- `8XYE` : Store the value of register VY shifted left one bit in register VX. Set register VF to the most significant bit prior to the shift

Compatibility notes

- Any non-existing `8XYN` instructions (`8XY8` through `8XYD` , plus `8XYF`) have undefined behavior and will likely crash the interpreter.

CHIP-8 for DREAM 6800 with joystick

A modification of CHIP-8/CHIPOS for the DREAM 6800 with an added joystick, described in DREAMER #3.

- An interrupt running 12.5 times a second polls the joystick
- Register VC will at all times contain the joystick's X coordinate, and VD will contain the Y coordinate

2K CHIPOS for DREAM 6800

An extended version of the CHIP-8 interpreter running in CHIPOS for a modified 2K EPROM version of the DREAM 6800. Created by Keith Semrad and detailed in DREAMER issue #19, January 1982.

It has several new instructions, but their usage is not very beginner-friendly; almost all of them take a value in VX that is then looked up in one giant look-up table. Values are therefore not human-readable without a reference.

New instructions

- `FX17` : Like `FX18`, but 1200 Hz
- `FX16` : Time delay for X value in look-up table; values `10` – `1F` map to values between 100 ms to 1.5 hours
- `FX30` : Roll screen up or down for VX value in look-up table; a value of `60` – `7F` rolls up 1–31 pixels, and a value of `80` – `9F` rolls down 1–31 pixels, respectively
- `FXEE` : Complement screen, part or full, for value in look-up table
- `FXFC` : Fill screen with value of X
- `FXDA` : Display ASCII characters, roll, complement, etc, see table
- `8XY3` : Set VX to VX XOR VY (one of the undocumented instructions for the COSMAC VIP version)

CHIP-8 for ETI-660

A CHIP-8 interpreter for the ETI-660 Learner's Microcomputer, detailed in *Electronics Today International*, November 1981, page 115 (https://archive.org/stream/ETIA1981/ETI%201981-11%20November#page/n114/mode/2up). This computer was an upgraded version of the HUG1802 computer from New Zealand, of which not much is known today.

Although not stated explicitly in the above source, the ETI-660 and its version of CHIP-8 in fact supported a resolution of 64 x 48.

In the "Hints for CHIP-8 Programmers" column, which started in the *ETI* issue of December 1982, the ETI-660 version of the CHIP-8 language is called the "CHIP 8.D3 dialect". The original COSMAC VIP version is called "CHIP 8.D1", but it's not clear what "CHIP 8.D2" refers to (although it's stated to load programs from `0200` like on the VIP); presumably the DREAM 6800 version.

New instructions

- `FX00` : Set pitch of sound signal
- `0000` : Return to monitor (exit interpreter)
- `00F8` : Turn display on
- `00FC` : Turn display off
- `00FF` : Do nothing

Compatibility notes

- CHIP-8 programs are loaded at `0600` to `07FF`.
- The resolution is 64 x 48.

The ETI-660 had a “standard” keyboard with the following layout:

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

But many used off-the-shelf hexadecimal keypads instead, with the same layout as the Digitran keypad:

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

CHIP-8 with color support for ETI-660

This adds color support to the CHIP-8 interpreter for ETI-660, which required hardware modifications. It was detailed in *Electronics Today International*, April 1982, page 88 (https://archive.org/stream/ETIA1982/ETI%201982-04%20April#page/n87/mode/2up), and was in principle very similar to CHIP-8X (and presumably CHIP-8C) for the COSMAC VIP although its instruction structure was different.

New instructions

- `07A2` : Step background color (cycles between blue, black, green and red)
- `07C1` : Enable foreground color instructions
- `27AB` : Set the foreground color of the 8 x 2 (two-byte) pixel area defined horizontally by VE (0–7) and vertically by VF (0–23, as the vertical resolution is 48) to the color defined in VD (VD <= 7, where values correspond to black, red, blue, violet, green, yellow, pale blue, and white, respectively). Using this instruction requires `07C1` to have been called already.

Compatibility notes

- Memory addresses `07A2` through `07FF` are reserved.
- The screen is initialized with a blue background

CHIP-8 for ETI-660 with high resolution

A version of the CHIP-8 for ETI-660 with color, this interpreter added support for a 64 x 64 resolution and required hardware modifications. It was detailed in *Electronics Today International*, February 1984 (https://archive.org/stream/ETIA1984/ETI%201984-02%20February#page/n87/mode/1up) and written by Bill Kreykes.

It moves the color instructions around in memory, so they have new opcodes. They also use different registers.

New instructions

- `049F` : Step background color (cycles between blue, black, green and red)
- `04A2` : Enable foreground color instructions
- `04B2` : Set the foreground color of the 8 x 2 (two-byte) pixel area defined horizontally by V1 (0–7) and vertically by V2 (0–32, as the vertical resolution is 64) to the color defined in V0 (V0 <= 7, where values correspond to black, red, blue, violet, green, yellow, pale blue, and white, respectively). Using this instruction requires `04A2` to have been called already.

Altered instructions

- `07A2` , `07C1` and `27AB` are removed.

Compatibility notes

- CHIP-8 programs are loaded from `0700` . As well as `0700` – `0FFF` , the memory region `0480` – `04EF` is also available.
- The display buffer now starts at `04F0` instead of `0480` .

CHIP-8 for COSMAC ELF

Described in *Programs for the COSMAC ELF – Interpreters*, by Paul C. Moews.

New instructions

- `5XY1` : Skip if VX > VY
- `5XY2` : Skip if VX < VT
- `5XY3` : Skip if VX != VY
- `9XY1` : VF, VX = VX * VY
- `9XY2` : VX = VX / VY (VF = remainder)
- `9XY3` : Convert 16-bit number in VX, VY to BCD stored at I, I+1, I+2, I+3 and I+4 (I does not change)
- `FX94` : Set I to the memory location of the 3-byte ASCII character in VX (0–64)
- `FFFF NMMM` : Jump to NMMM; set field to N and jump to MMM
- `FX75` : Output VX to hex display

Altered instructions

- `FX29` does not change I.

CHIP-VDU / CHIP-8 for the ACE VDU

A port of the CHIP-8 interpreter to the ACE VDU board, by Tony Hill. It added two long branch instructions to get input from the ACE keyboard, and provided a resolution of 128 x 64. Described in Ipso Facto, issue 35, June 1983.

Issue 36 provides some modifications to old games that don't work with the new resolution; these modifications could be used for other extensions that provide higher resolution but lack instructions to support more standard lower resolutions, such as CHIP-10.

New instructions

These are not instructions per se, but jumps, but since the interpreter is located at address `1000` on the ACE they work as machine code instructions.

- `110D` : Gets a single hex digit and puts it in the ACE's D register.
- `119C` : DF is set to 0 if no key is pressed and 1 if a key is pressed; if so, the key value is put in D.

Compability notes

The interpreter is located at address `1000` to `12EE` . Execution starts at `1000` . The CHIP-8 program is still loaded at `0200` .

CHIP-8 AE (ACE Extended)

CHIP-8 AE was a major extension for the ACE by Larry Owen, Tony Hill and Mike E. Franklin. It is described in the newsletter Ipso Facto, issue #40, May 1984, with a disassembler provided in issue #41 of July 1984. There was one version for the 1861 Pixie chip, and one for the ACE VDU. It provides the following features:

- It has modes with higher resolution than 64 x 32.
- Instead of just 16 variables, V0–VF, it has 16 banks of 16 variables each. One bank can be active at a time. All instructions that operate on variable registers will operate on the ones in the currently active bank (with the exception of `BNNN`, which always uses V0 in bank 0).
- In addition to the single original CHIP-8 timer, there are 16 new timers.
- Instead of 16 keyboard keys, it supports 256 keys. It also supports drawing 256 ASCII characters.
- Some instructions from CHIP-8E are included.

New instructions

- `000D` : Lo-res mode, 64 x 32 resolution (with the Pixie chip, the display is turned off and must manually be turned back on with `00DE`)
- `0010` : Mid-res mode, 64 x 64 resolution (with the Pixie chip, the display is turned off and must manually be turned back on with `00DE`)
- `0013` : Hi-res mode, either 128 x 64 with VDU or 64 x 128 with Pixie chip (in the latter case, the display is turned off and must manually be turned back on with `00DE`)
- `0071` : Skip if hit on erase
- `0075` : Skip if hit on draw
- `007F` : Reset program: Wait for keypress, then jump to `0200`
- `00DE` : Turn on display (Pixie chip only)
- `5XY1` : Skip if VX > VY
- `5XT2` : Skip if timer T == 0
- `8XT8` : Timer T = VX
- `8XT9` : VX = timer T
- `9XY1` : Skip if VX < VY
- `9XT2` : Skip if timer T != 0
- `EXA3` : Skip if VX != keyboard key (full byte)
- `EXAD` : Skip if VX == keyboard key (full byte)
- `EXB5` : Skip if VX > keyboard hex key (nibble)
- `EXB7` : Skip if VX > keyboard key (full byte)
- `EXBF` : Skip if VX < keyboard hex key (nibble)
- `EXC1` : Skip if VX < keyboard key (full byte)
- `FX0E` : VX = keyboard key (full byte)
- `FX2C` : Point I at 5 byte sprite for VX most significant nibble
- `FX50` : Clear most significant nibble of VX; VX &= `0F`
- `FN74` : Store N bytes pointed at by I into "I storage" (I is changed)
- `FX95` : Point I at 7 byte sprite for ASCII character in VX
- `FNC2` : Make variable bank C active bank
- `FNC6` : The address I points to is stored in I storage location N
- `FNCA` : The byte stored in I storage location N is loaded in I
- `FNCE` : Swaps the address I points to with I storage location N
- `FXD4` : I -= VX

- `FXFA` : Pixie chip only: The high byte at the start of the display memory is set to VX, and the display is turned off

Altered instructions

- `00E0` : Clears the display as before, but if using the Pixie chip, only the display memory that is currently displayed (depending on resolution mode) is cleared, and also the display is turned on afterwards.
- `BNNN` : Not altered per se, but unlike all other instructions that use variables, this will always jump based on V0 in bank 0, regardless of the currently active bank.
- `DXYN` : Now stores two new flags (in addition to VF), "hit on erase" and "hit on draw", which are always mutually exclusive.
- `FX0A` : Waits on keypress as before, but only stores 0–9 in VX; the keys A–F can not be recognized.

Compatibility notes

The major issue of compatibility is the keyboard. CHIP-8 AE is written for an ASCII keyboard, not a hexadecimal keyboard, and it provides no mapping. The numerical keys all map to the same value, but in order to use the A-F hexadecimal key inputs that some games might expect, the user would have to either use the keys that had ASCII values that corresponded to those hexadecimal values (J, K, L, M, N and O), or patch the CHIP-8 programs to use different keys.

Dreamcards Extended CHIP-8 V2.0

Introduced by Lindsay R. Ford of Dreamcards for the Microbee computer in *Electronics Today International*, October 1984, page 122 (https://archive.org/stream/ETIA1984/ETI%201984-10%20October#page/n121/mode/2up), this was a major extension based on CHIP-8 for the ETI-660.

The Microbee was a computer supporting a screen resolution of 512 x 256 pixels. In order to utilize this larger resolution, each individual CHIP-8 pixel was drawn as a character using the Microbee's hardware cursor in the PCG (Synartek 6545 Programmable Character Generator), an 8 x 16 bitmap/sprite. This cursor could be custom, and an ASCII character could also be printed that didn't adhere to the actual pixel grid of the game. There was also an instruction for drawing a dithered checkerboard-patterned cursor, called the "half-tone cursor".

In addition, it supported advanced sound generation, and joysticks. It was also specifically made to support CHIP-8 programs written for the different popular interpreters of the day, with an instruction to switch resolutions.

Finally, there was also a CHIP-8/BASIC integration, with a "compiler"/"decompiler", which converted a CHIP-8 binary program to `REM` comments in a BASIC program. This meant that BASIC programs could have CHIP-8 "subroutines"; the BASIC interpreter could run the CHIP-8

interpreter for embedded code, and the two could exchange data with each other. Therefore, CHIP-8 programs for this interpreter were often given as BASIC code listings.

New instructions

- `0000` : No operation
- `00FF` : No operation
- `8XY7` : Output VX to port VY (VF is cleared)
- `8XY8` : Load VX with input from port VY (VF is cleared)
- `F000` : Stop
- `F001` : Toggle CHIP-8 cursor between full-tone and half-tone (dithered); if a custom cursor has been set, the cursor will be set to standard full-tone
- `F002` : Wait for vertical blanking period ("toggle display switch")
- `F003` : Change screen format (64 x 32, 64 x 48, 64 x 64)
- `FX05` : Set language execution speed/rate to VX; a value of 0 is treated as 256, which is also the default
- `FX25` : Set tone frequency to VX (0–F; default is 8)
- `FX35` : Set the tone synthesizer SYNTHA strobing frequency to VX
- `FX45` : Set the tone synthesizer SYNTHB strobing range/depth to VX; if VX is 80, the strobe range will continually vary at random
- `FX75` : Load custom 8 x 16 PCG cursor VX (0–6) from I (I is incremented by 16 after execution)
- `FX85` : Replace cursor with custom cursor VX (0–6)
- `FX95` : Pseudo-poke/store V0 to VX in imaginary CHIP-8 screen; intended as a substitute for older CHIP-8 programs that used `FX55` to poke data directly into the screen buffer, which is no logner possible with that instruction (I is incremented by X + 1)
- `FXY4` : Set program page ± X, data page ± Y (*not* the values in VX and VY!); program page change will not take effect until encountering a `0NNN`, `1NNN`, `2NNN` or `BNNN` instruction (meaning that normal execution that runs off a page will actually jump back to the start of the page), and data page change will not take effect until `ANNN`
- `FXY6` : Set VX/VY to display coincidence coordinates; the *first* screen coordinates where a collision occured during the *last* display instruction that caused a collision
- `FXYB` : Limit joystick range to VX/VY
- `FXYC` : Read joystick movement; a joystick movement horizontally or vertically will increment or decrement VX or VY, and the status of the fire button will set VF to 1 or 0
- `FXYD` : Display ASCII string starting at I, at VX/VY; string can contain characters in the range 20–7E plus CR and LF, and are terminated by EOT. The sequence "|XA" will repeat the A character X+1 times (1–40). Wraps around the screen like `DXYN`. Will continue to operate past a Data Page, but does not alter the Page Buffer.

- `FXYF` / `FXY0` : No operation, but reserved for future color expansions

Altered instructions

- `0NNN` : Calls Z80 machine language routine at address `0NNN`, as long as `NNN` is larger than `0FF` (any instruction starting with `00` is a no-op). The contents of VB and VC are transferred to the Z80 registers B and C, and upon return to CHIP-8, VB and VC will hold the values of Z80 registers B and C. Does not require the address to be even.
- `DXYN` : As before, but in hi-res screen format: Instead of turning on and off individual pixels, this instruction now addresses a 64 x 32 grid of 8 x 16 “cursor” bitmaps. If the cursor to be drawn is the same as the existing one, that cursor will be erased. However, if they are different, the old cursor will be replaced by the new, except that a full-tone and a half-tone cursor will result in a full-tone, and full/half-tone cursors can’t be printed over custom cursors (but they can be printed over alphanumerics). VF will be set as usual on collision.
- `FX07` : Produces a 20 millisecond delay, decrements the timer, and then stores that value in VX

Compatibility notes

- This interpreter specifically includes the undocumented instructions `8XY3` (XOR) and `8XY6` (shift right), but its `8XY7` is used for something else rather than shift left (but `8XX4` can be used instead), and `8XYE` is missing outright.
- The Microbee has 64K of memory; the first `0FFF` bytes are usually considered reserved for BASIC, so CHIP-8 programs will usually be loaded from as high as `2000` or `3000`. In addition, the most significant nibble is not itself a part of the program counter, but stored separately. The program counter is therefore only 12 bytes, and an overflow will not increment the data page. The data page can be changed with the `FXY4` instruction.
- All instructions starting with `00`, except `00E0` and `00EE`, are in fact no-ops. The manual mentions `0000` and `00FF` specifically because they were used by the interpreters for DREAM 6800 and ETI-660, respectively.
- All instructions to be executed must be at even addresses. If the program counter is ever pointed at an odd address, it will stop as if a `F000` was encountered. The manual claims that "most earlier interpreters" would crash in this case. TODO: Which ones?
- The call stack has 16 levels, and is protected from both underflow and overflow. If an `00EE` is encountered when the stack is empty, the instruction is treated as an `F000` and the program will stop. If a subroutine is called when the stack is full, the `2NNN` instruction is treated as a no-op.
- If `00EE` returns to a different program page (as changed by `FXY4`), the active program page will automatically change.
- A sound timer value of 1 as set by `FX18` will work fine (unlike on the COSMAC VIP), and setting either timer to 0 actually sets it to 256.
- There are no interrupts; the delay timer will not actually decrement the timer unless `FX07` is encountered. As most delays are implemented using "busy loops" that check the delay timer for the value 0, this probably has little practical impact.

Amiga CHIP-8 interpreter

The CHIP-8 interpreter for the Commodore Amiga was first released in 1990 by Paul Hayter. It was inspired by the DREAM 6800 interpreter, and also included a similar monitor program (called "DREAM Mon") to input and run programs.

This was perhaps the first "emulator"-like interpreter. It ran the CHIP-8 environment in a virtual machine, simulating the memory instead of using actual, mapped memory.

New instructions

- `0000` : No operation
- `F000` : Stop

Altered instructions

- `FX1E` : If the result of `VX + I` overflows

Compatibility notes

- Of the undocumented instructions, `8XY6`, `8XY7` and `8XYE` are missing from the interpreter (although `8XY3` is included).
- The sound timer was not yet implemented in version 1.1, which is the version that seems to be available online.
- CHIP-8 programs were saved in `.c8` files. A filename that ended in `2.c8` indicated that it should be loaded from `0200` (like games for the COSMAC VIP or DREAM 6800); if it ended in `6.c8` it should be loaded from `0600` (like the ETI-660).

CHIP-48

Created by Andreas Gustafsson for the HP48 graphing calculators. Posted on comp.sys.handhelds in September 1990 (https://groups.google.com/d/topic/comp.sys.handhelds/zv7XZKIDS34/discussion), source code here (https://groups.google.com/d/topic/comp.sys.handhelds/IhEF9Da2RJk/discussion).

This is mostly a re-implementation of CHIP-8, but contains a crucial difference in the bit shifting instructions' semantics, as well as a bug in `FX55` / `FX65`.

New instructions

- `BXNN` : Jump to address `XNN` + the value in VX (instead of address `NNN` + the value in V0). Possibly a bug, see this analysis (https://github.com/Chromatophore/HP48-Superchip/blob/master/investigations/quirk_jump0.md).

Altered instructions

- `BNNN` is replaced by `BXNN` (see above)
- `FX55` / `FX65` no longer increment `I` correctly; it is incremented by one less than it should. If X is 0, it is not incremented at all, as noted in the CHIPPER assembler documentation (https://groups.google.com/forum/#!searchin/comp.sys.hp48/chip-8\|sort:date/comp.sys.hp48/e7In51mOgHY/8tR3ZKeX9FUJ)
- `8XY6` / `8XYE` shift VX and ignore VY

Compatibility notes

- The stack is limited to 16 entries.
- The memory is limited to 4K, from `000` to `FFF`.
- The first 17 bytes of the program were visible as ASCII charactersi (actually, a modified Latin-1 (https://www.drehersoft.com/mapping-hp48-text-to-unicode/)) in the HP48's interface, and so often contain a jump (ie. the two bytes `12NN`, displayed as two block characters) followed by the title of the game and sometimes the author.

SUPER-CHIP 1.0

Created by Erik Bryntse in 1991 for the HP 48S and HP 48SX graphing calculators, based on CHIP-48. It was announced on comp.sci.handhelds May 16, 1991 (https://groups.google.com/d/topic/comp.sys.handhelds/RuzVNccds2Q/discussion) (with an errata (https://groups.google.com/d/topic/comp.sys.handhelds/fPUzuAkDdVs/discussion)). Also known as SCHIP and S-CHIP.

Adds a high-resolution mode of 128 x 64 as well as persistent memory in the HP48's "RPL" memory.

New instructions

- `00FD` : Exit interpreter
- `00FE` : Disable high-resolution mode
- `00FF` : Enable high-resolution mode
- `DXY0` : Draw 16 x 16 sprite (only if high-resolution mode is enabled)
- `FX75` : Store V0..VX in RPL user flags (X <= 7)
- `FX85` : Read V0..VX from RPL user flags (X <= 7)

Altered instructions

Same as CHIP-48, plus:

- `FX29` : Point I to 5-byte font sprite as in CHIP-8, but if the high nibble in VX is 1 (ie. for values between `10` and `19` in hex) it will point I to a 10-byte font sprite for the digit in the lower nibble of VX (only digits 0-9)

Compatibility notes

- In low-resolution mode (ie. the original 64 x 32), the screen memory should still be represented as 128 x 64 with each "pixel" being represented by 2 x 2 pixels. This means that switching between modes produces no visual effect, and the display is not cleared. In fact, the low-resolution mode is simply a special mode where `DXYN` 's coordinates are doubled. This has more significance in SUPER-CHIP 1.1.
- Memory was uninitialized and random at startup.
- The final byte is reserved, probably because of an off-by-one error, and utilizing it will crash the program.

SUPER-CHIP 1.1

An extension to SUPER-CHIP, which adds scrolling instructions. It was announced by Erik Bryntse on May 24, 1991 on comp.sys.handhelds (https://groups.google.com/forum/#!searchin/comp.sys.handhelds/super$20chip/comp.sys.handhelds/sDY9zFb6KUo/JcYBK2_yerMJ).

New instructions

- `00CN` : Scroll display N pixels down; in low resolution mode, N/2 pixels
- `00FB` : Scroll right by 4 pixels; in low resolution mode, 2 pixels
- `00FC` : Scroll left by 4 pixels; in low resolution mode, 2 pixels
- `FX30` : Point I to 10-byte font sprite for digit VX (only digits 0-9)

Altered instructions

Same as SUPER-CHIP 1.0, but:

- `FX29` works like the regular CHIP-8 font instruction; the larger 10-byte font sprites have their own, dedicated `FX30` instruction now
- `FX55` / `FX65` no longer increment `I` at all.

Compatibility notes

- As in SUPER-CHIP 1.0, the low-resolution (64x32) mode is simply a special mode where `DXYN`'s coordinates are doubled, and each "pixel" is represented by 2x2 on-screen pixels. This in fact means that using the instruction `00C1`, a game is able to scroll a "half-pixel" in low-resolution mode. Read more about this here. (https://github.com/Chromatophore/HP48-Superchip/blob/master/investigations/quirk_display.md)
- In high resolution mode, `DXYN` / `DXY0` sets VF to the number of rows that either collide with another sprite or are clipped by the bottom of the screen. The original CHIP-8 interpreter only set VF to 1 if there was a collision.

GCHIP

S-CHIP (SUPER-CHIP) ported to the HP 48G and HP 48GX calculator. It functions like S-CHIP, except that two S-CHIP instructions don't work due to an oversight.

Altered instructions

`FX75` and `FX85` (save and load RPL user flags) crash the interpreter.

SCHIP Compatibility (SCHPC) and GCHIP Compatibility (GCHPC)

A modified version of SUPER-CHIP 1.1 that attempts to fix all the incompatibilities listed above. Repository here, with detailed research and patches. (https://github.com/Chromatophore/HP48-Superchip)

New instructions

Same as SCHIP 1.1.

Compatibility notes

Unlike SCHIP 1.1, all instructions behave as they did in CHIP-8.

For GHCPC: `FX75` and `FX85` now work correctly (like they did in SCHIP, and unlike GCHIP).

VIP2K CHIP-8

A CHIP-8 interpreter for the VIP2K (http://www.sunrise-ev.com/vip2k.htm), a 40th anniversary reproduction of the COSMAC VIP computer. Manual for the VIP2K CHIP-8 interpreter. (http://www.sunrise-ev.com/photos/1802/Chip8interpreter.pdf)

New instructions

- `00NN` : Call 1802 machine language routine at address `70NN` (rather than `80NN` like `0NNN` does)

Altered instructions

- The instruction `00NN`

Compatibility notes

- CHIP-8 programs are loaded from `8200` . All memory-related instructions are relative to the memory at `8000` .

SUPER-CHIP with scroll up

In massung's CHIP-8 emulator (https://massung.github.io/CHIP-8/), which in fact is a SUPER-CHIP 1.1-compatible interpreter, there is support for the extra instruction `00BN` , which scrolls up (as a mirror of the standard SCHIP `00CN` instruction for scrolling down).

This instruction, which never existed in SUPER-CHIP, has been tracked to trapexit's CHIP-8 documentation (https://github.com/trapexit/chip-8_documentation#opcodes). The author does not remember where it originally came from (https://github.com/JohnEarnest/Octo/issues/68#issuecomment-338436036).

See also XO-CHIP below, which adds this same instruction with the opcode `00DN` instead.

New instructions

- `00BN` : Scroll display N pixels up; in low resolution mode, N/2 pixels

chip8run

A Linux interpreter by Peter Miller, created as part of the chip8 package. It supports both CHIP-8 and SCHIP games, and includes an instruction to toggle one incompatibility between the versions.

New instructions

- `001N` : Exit interpreter with the exit status N
- `00FA` : Set compatibility mode for `FX55` and `FX65` ; running this changes the behavior of those instructions so they no longer change `I`

Mega-Chip

A superset of SCHIP created by Revival Studios in 2007.

- CLS will be used for timing/flipping/noflickering purpose
- Emulationspeed/Num.ticks per frame will be fixed (1000 ticks????) OR timed to CLS
- LDHI I,nnnnnnn instruction allows 24-bit adressing, always follow LDHI by a NOP instruction.
- LDPAL Will load 32-bit colors to an internal mchip8 pallete at position 1. Color 0 is always black/transparent.
- SPRW/SPRH can be 0..255 , use 0 for a width/height of 256
- SPRW/SPRH will overwrite n-setting for non-character sprites.
- ALPHA/FADE will effect the fade-factor of the screenbuffer (0..255)

New instructions

- `0010` : Disable Megachip mode
- `0011` : Enable Megachip mode
- `01NN` : Load I with NN concatenated with the next byte
- `02NN` : Load NN-colors palette at I
- `03NN` : Set sprite width to NN
- `04NN` : Set sprite height to NN
- `05NN` : Set screen alpha to NN
- `060N` : Play sound at I, loop if N
- `0700` : Stop sound
- `080N` : Set sprite blend mode to N (0=normal,1=25%,2=50%,3=75%,4=addative,5=multiply)
- `00BN` : Scroll display N pixels up

Altered instructions

Same as SCHIP

XO-CHIP

An extension for SCHIP created by John Earnest in 2014.

XO-CHIP supports audio and 64 kb of memory, which is usable mainly for graphics and audio (addressable only by I). It also has one extra buffer ("plane") of display memory, which works identical to the regular one. Both planes are displayed "on top" of each other, and they can be drawn in different colors. Pixels that are turned on in both planes can be drawn in another color. Clear, draw and scroll instructions will only affect the currently selected plane(s).

Note that XO-CHIP is mainly supported by John Earnest's own Octo assembler, which supports "macros" for comparison operators (less than (or equal), greater than (or equal)) but which assemble down to regular CHIP-8 bytecode instead of dedicated instructions.

New instructions

- `00DN` : Scroll up N pixels
- `5XY2` : Save VX..VY to memory starting at I (same as CHIP-8E); order can be ascending or descending; does not increment I
- `5XY3` : Load VX..VY from memory starting at I (same as CHIP-8E); order can be ascending or descending; does not increment I
- `F000 NNNN` : Load I with 16-bit address NNNN
- `FN01` : Select drawing planes by bitmask (0 planes, plane 1, plane 2 or both planes (3))
- `F002` : Store 16 bytes in audio pattern buffer, starting at I, to be played by the sound buzzer
- `FX3A` : Set the pitch register to the value in `VX` .

Altered instructions

- Skip instructions will skip over the entire double-wide `F000 NNNN` instruction.
- Clear, scroll and draw instructions only apply to the selected drawing plane.
- `00FE` and `00FF` , which switch between low and high resolution, will clear the screen as well.
- `FX75` : As in SUPER-CHIP, store V0..VX in RPL user flags, but X is not limited to 7
- `FX85` : As in SUPER-CHIP, read V0..VX from RPL user flags, but X is not limited to 7

Compatibility notes

- Switching resolution mode erases the screen, unlike on SCHIP.
- Audio patterns are a series of 1-bit samples played at a rate of 4000*2^((`pitch` -64)/48) Hertz, or samples per second, where `pitch` is the pitch register.
- The pitch register is initialized to 64, making the default sample rate 4000 Hz. audio pattern playback rate to 4000*2^((vx -64)/48)Hz.
- The audio pattern buffer is restricted to 16 bytes in Octo.
- The audio pattern buffer is not necessarily cleared on program start, although Octo does so.
- The audio pattern buffer is loaded when `F002` is called. Subsequent rewrites of the memory that `I` pointed to at that time are not reflected in the buffer.
- The playback offset of the audio pattern buffer only resets when the sound timer reaches 0 (either by itself, or by being set explicitly).

Octo

Octo is a CHIP-8 interpreter that supports regular CHIP-8, SUPER-CHIP and XO-CHIP. For the most part it is compatible with those implementations.

New instructions

- `0000` : Halts the interpreter

Altered instructions

- `DXY0` : Draws a 16x16 sprite like SUPER-CHIP does, but it does so even in low-resolution mode
- `FX30` : Sets I to the address of large 10-byte font sprites for all the hexadecimal values from 0–F, not just 0–9 like SUPER-CHIP

Compatibility notes

- Switching resolution mode erases the screen, like XO-CHIP dictates, but Octo does so even with SCHIP games.

CHIP-8 Classic / Color

The CHIP-8 Classic and CHIP-8 Color is a microcontroller computer created by StandAloneComputers in 2019. It's inspired by the DREAM 6800 interpreter, and includes a similar monitor program, also called CHIPOS. It also adds an ASCII character set, I/O, and color support. See https://chip-8.com.

New instructions

- `0000` : No operation
- `F000` : Stop and return to monitor (CHIPOS)
- `FX17` : Set pitch of tone generator to value in VX
- `FX2A` : Point I at 7 byte sprite for ASCII character in VX
- `FX70` : Output data in VX to RS485 port
- `FX71` : Wait for input from RS485 port and put it in VX
- `FX72` : Set RS485 baud rate
- `FX75` : Set VX to RRRGGGBB format (only with CHIP-8 Color firmware)

Altered instructions

- `8XY6`, `8XY7` and `8XYE` are not listed in the instruction set and might not be supported. (Note that the fourth instruction that was originally undocumented in the COSMAC VIP interpreter, `8XY3`, is listed.)

Compatibility notes

The hexadecimal keypad uses the prototype DREAM 6800 layout:

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