CHIP-8 Assembler Reference

The assembler is case-insensitive. Lowercase character input is treated as uppercase. In this sheet, **bold** means a value to be entered exactly as shown, while *italic* means a value to be chosen by you.

Input rules

Statements

Each statement consists of these parts, all of which are optional:

- A label, which is an alphabetic character followed by zero or more alphanumeric characters, followed by a colon. An underscore may be used as an alphabetic. Examples: Q:, DO_DRAW_2:.
- An opcode, either an instruction name such as JP or LD, or a directive such as EQU.
- One or more expressions (depending on the opcode) separated by commas. Examples: v2, v7 or buflen % #10.
- A semicolon and a comment, for example; increment x coord.

Examples:

```
; just the comment
EXIT: ; label, comment, no opcode or operands
SUB START: LD v3, #FF ; all four parts
```

Reserved Names

The following names are reserved and may not be used as labels.

- All opcodes and directives are reserved. For example, a line that starts with **SUB** will be taken as a subtract opcode, not a label for a subroutine.
- Data register names, which are: v0, v1, ... v9, vA, vB, vC, vD, vE and vF.
- The I register is named **I**.
- The timer countdown register is named **DT**.
- The sound countdown register is named **ST**.
- In one instruction, the keyboard is referred to as **K**.

Strings

String data is ASCII characters enclosed in single quotation marks, 'LIKE THIS' Use a pair of quotation marks to include a single quotation mark in the string: 'THAT''S HOW'.

Lowercase letters are forced to uppercase. Non-ASCII characters are not supported. (Hey, it's 1975!)

Numeric Values

NT 1

Numeric values are written as:

- Decimal, 27
- Hexadecimal when preceded by a hashmark, #1B.
- Octal when preceded by an at-sign, @177.
- Binary when preceded by a dollarsign, \$00011011. When writing binary, you may use a decimal point in place of a zero, which makes it easier to define sprite values: \$...11.11.
- A label represents the value of its address, or the value assigned by the EQU directive.

Any numeric literal is padded on the left with zeros to make a 16-bit integer before use.

Expressions

Numeric expressions are allowed using these operators, listed from highest priority to lowest:

- Parentheses (and)
- + value unary plus
- - value unary minus
- ~ value bitwise NOT
- value! exp power-of
- *value* < *count* shift-left
- *value* > *count* shift-right
- *value* **value* multiply
- value / value divide
- value + value add
- value value subtract
- value & value bitwise AND
- value | value bitwise OR
- *value* ^ *value* bitwise XOR
- value % value modulus (remainder)

All operations are performed on 16-bit integers and return 16-bit integers.

Directives

Directives control the assembly process. The following directives are allowed:

- **DA** *string* The ASCII bytes representing the characters are assembled.
- **DB** expression [, expression...] One or more bytes are assembled.
- **DS** *expression* Reserve *expression* bytes of space.
- name **EQU** expression Give the value of expression to name. You may = in place of **EQU**.
- **ORG** *expression* Set the assembly location to *expression*, possibly skipping over undefined space or possibly returning to assemble over bytes assembled previously.

Space reserved by **DS** and space skipped over by **ORG** will be probably be filled with zeros when the program is loaded (but it would be bad practice to assume that).

instruction Opcodes

Opcodes that change the flow of execution

• EXIT

Stop the emulator. (00FD)

• **JP** address

Jump to specified address. (1aaa)

• JP v0, address

Jump to specified address plus contents of **v0** (indexed jump). (Baaa)

• CALL address

Push PC on the call stack; jump to specified address. The call stack is limited to 12 levels. A thirteenth call forces an error stop. (2aaa)

• RET

Pop top address from call stack into PC: thus, return from subroutine. If the call stack is empty, force an error stop. (00EE)

• **SE** *vx*, *byte*

Skip the following instruction if the contents of vx equal byte. (3xbb)

• **SNE** *vx*, *byte*

Skip the following instruction if the contents of vx do not equal byte. (4xbb)

• **SE** *vx*, *vy*

Skip the following instruction if the contents of vx equal those of vy. (5xy0)

• **SNE** *vx*, *vy*

Skip the following instruction if the contents of vx do not equal those of vy. (9xy0)

• **SKP** *vx*

Skip the following instruction if the key (0-16) specified by vx is down (being pressed). (Ex9E)

• SKNP vx

Skip the following instruction if the key (0-16) specified by vx is not pressed. (EXA1)

Opcodes related to the data registers

• LD vt, byte

Put the value of *byte* in register *vt*. (6tbb)

• LD vt, vs

Put the contents of vs into vt. (8ts0)

• **ADD** vt, byte

Add the value of *byte* to the contents of *vt*. Overflow is not recorded. (7tbb)

• **ADD** vt, vs

Add the contents of vs into vt. If the sum overflows eight bits, register VF is set to 01, else it is set to 00. (8ts4)

• **RND** vt, byte

Generate a random number in 0..255, logically AND it with *byte*, and put the result in *vt*. For example to get a 4-bit random number, specify *byte* as 15 or 0xf. (Ctbb)

• LD vt, K

Wait for a key to be pressed and put that key (0..16) in vt. Note that this ties up the emulator until a key is entered on the display or the Run/Stop button is toggled. (Ft0A)

• **OR** *vt*, *vs*

OR the contents of vs and vt and put the result in vt. (8ts1)

• **AND** *vt*, *vs*

AND the contents of vs and vt and put the result in vt. (8ts2)

• XOR vt, vs

XOR the contents of vs and vt and put the result in vt. (8ts3)

• SUB vt, vs

Subtract the contents of vs from vt and put the result in vt. If $vs \le vt$, **VF** is set to 1. If vs > vt there is underflow and **VF** is set to 0. (8ts5)

• SUBN vt, vs

Subtract the contents of vt from vs and put the result in vt. If vt <= vs, \mathbf{vF} is set to 1. If vt > vs there is underflow and \mathbf{vF} is set to 0. (8ts7)

• SHR vt, vs

Shift the contents of vs right 1. The result is stored in vt (see note below). The shifted-out bit, 0 or 1, is set as the value of **VF**. To effect shifting the contents of a register within that register, specify the same register twice, for example SHR v5, v5; shift v5 right. (8ts6)

• SHL vt, vs

Shift the contents of vs left 1. The shifted-out bit, 0 or 1, is set as the value of **VF**. The result is stored in vt (see note below). (8xyE)

Opcodes related to the Timer and Sound registers

• LD vt, DT

Store the current contents of the **DT** register in *vt*. (Ft07)

• LD DT, vs

Store the contents of vx in the timer register. (Fs15)

• LD ST, vs

Store the contents of vs in the sound timer register. (There is no instruction for loading the contents from ST.) (Fs18)

Opcodes that change or use the I register

• LD I, addr

Put the value of *addr* in **I**. (Aaaa)

• **ADD I**, *vs*

Add the contents of vs to **I**. Note that this can potentially set **I** to an address outside memory (>4095). (Fs1E)

• LDC I, vs

Load **I** with the address of a CHIP-8 (5x4 pixel) character sprite for the number in the low four bits of *vs*. (Fs29)

• LDH I, vs

Load **I** with the address of the SCHIP (10x8 pixel) character sprite for the number in the low four bits of vs. (Fs30)

• **STD** *vs*

Store three bytes for the decimal digits of the contents of vs into memory at the address in **I**. The **I** contents is incremented by three. For example if **V5** contains 0x76 and **I** contains 0x0500, the instruction std v5 places the bytes 0x01, 0x02, 0x06 into memory starting at 0x0500, and **I** will contain 0x0503. (Fs33)

• LDM vt

Load the data registers from **V0** through *vt* from memory beginning at the address in **I**. The **I** value is incremented for each register loaded. A typical use for load-multiple would be to load the three digit values stored by **STD** into registers. (Ft65)

• STM vs

Store the data registers from $\mathbf{V0}$ through vx into memory beginning at the address in \mathbf{I} . The \mathbf{I} value is incremented for each register stored. (Fs55)

Opcodes that affect the display

• CLS

Clear the display to all-black pixels. (00E0)

• LOW

Set the display to CHIP-8 mode, with 32 rows of 64 pixels. The display is cleared. (00FE)

• HIGH

Set the display to SCHIP mode, with 64 rows of 128 pixels. The display is cleared. (00FF)

• **DRAW** vx, vy, len

Draw the sprite currently addressed by **I** on the screen. The X-coordinate is taken from vx, the Y-coordinate from vy. The value of *len* specifies how many bytes are in the sprite. (A character sprite has either 5 or 10 bytes.) If the sprite extends past the right side or bottom row of the display, some pixels "wrap" to the left side or top edge. (Dxyn)

If len is 0, I must address an SCHIP 16x16 sprite, 32 bytes in all.

• SCD rows

Scroll the display down by *rows* (from 0 to 15). Note there is no scroll up instruction. (00Cn)

• SCR

Scroll the display right by four pixels. (00FB)

• SCL

Scroll the display left by four pixels. (00FC)

(*Note on SHL and SHR instructions*: These instructions, along with the XOR and SUBN opcodes, were not documented in the original COSMAC VIP manual and were later discovered by users who disassembled the CHIP-8 interpreter code. Unfortunately information on how SHL and SHR were originally implemented was reported incorrectly. As a result, most extant emulators do these instructions incorrectly: they shift *vs* and put the result in *vs*, ignoring *vt*. This has probably not been noticed because almost all uses of the instructions name the same register for *vs* and *vt*. See the README for this program for a link to more information.)