CHIP-8 Assembler Reference

This is a reference to the legal statements and the syntax rules of the CHIP-8 assembler. It assumes you already understand the features and parts of the CHIP-8 virtual machine, which is explained in detail in the included files: `An_Easy_Programming_System.pdf` and `COSMAC_VIP_Manual.pdf`.

In this document, **bold** means a value to be entered exactly as shown, while *italic* means a value to be chosen by you. For example a register argument might be shown as **V***t* which means it must start with a V, followed by a digit of your choice indicated by *t*.

Input rules

The assembler is case-insensitive. Lowercase character input is treated as uppercase.

An assembler source file consists of lines of ASCII or Latin-1 characters only. When you use File>Open to open a file, CHIP8IDE examines the file. If it consists only of ASCII or Latin-1 characters, it is assumed to be a source file. CHIP8IDE loads it into the Edit window.

If you open a file that contains characters that are not in the Latin-1 set, CHIP8IDE assumes you have opened a CHIP-8 or SCHIP *binary* executable. It automatically disassembles the binary into source statements and loads these lines into the editor window.

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:

```
ONLY_A_LABEL:
; just the comment
EXIT: ; label and 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 listed below are reserved.
- 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**.

These names may not be used as labels. For example, `CALL SUB` would be an error because it is seen as two opcodes (`CALL` , `SUB`) in sequence, not a call to a label `SUB` . Similarly, `ST EQU 5` would be seen as a special register, not a variable `ST` being given the value 5.

Strings

String data is ASCII characters enclosed in single quotation marks, `'LIKE THIS'` (Strings are used only in the **DA** directive). Use a pair of quotation marks to include a single quotation mark in the string: `'THAT''S HOW'` . Lowercase letters are converted to uppercase.

Numeric Values

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.
- A name defined by an `EQU` directive represents its assigned value.

Expressions

A numeric value is an expression. You can combine numeric values into more complex expressions 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 *value* left *count* bits
- *value* **>** *count* shift *value* right *count* bits
- *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)

In the instructions below,

- *nybble* means, any expression with a value between 0 and 15 (`#0` to `#F`)
- *byte* means, any expression with a value between 0 and 255 (`#0` to `#FF`)
- *address* means, any expression with a value between 0 and 4095 (`#0` to `#FFF`)
- *word* means, any expression with a value between 0 and 16535 (`#0` to `FFFF`)

Directives

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

DA *string*
The ASCII bytes representing the characters are assembled.

```
DA 'COPYRIGHT 2017 FLOYD FLOYD`
```

DB *byte* [**,** *byte*...]
One or more bytes are assembled.

```
DB 0, $...11...
```

DW *word* [**,** *word*...]
One or more 16-bit words are assembled (convenient for making SCHIP 16x16 sprites).

```
DW #8080, #4040, #2020
```

DS *address*
Reserve *expression* bytes of space.

```
DS 128
```

name **EQU** *expression*
Give the value of *expression* to *name* so it can be used in other expressions. You may use **=** in place of **EQU**.

```
SCREEN EQU 1 ; make 2 for SCHIP
MAX_X EQU 63*SCREEN
MAX_Y EQU MAX_X/2
```

ORG *expression*
Set the assembly location to *expression*, possibly skipping over undefined space or possibly returning to assemble over bytes assembled previously.

```
ORG BFR+128
```

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

In the following descriptions, the hexadecimal form of each instruction is shown in parentheses. For example the hexadecimal form of `ADD I, Vs` is `#Fs1E`, where *s* is the number of the specified register.

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 the Program Counter; thus, return from subroutine. If the call stack is empty, force an error stop. (00EE)

SE V*s*, *byte*
Skip the following instruction if the contents of **V***s* equal *byte*. (3sbb)

SNE V*s*, *byte*
Skip the following instruction if the contents of **V***s* do not equal *byte*. (4sbb)

SE V*x*, **V***y*
Skip the following instruction if the contents of **V***x* equal those of **V***x*. (5xy0)

SNE V*x*, **V***y*
Skip the following instruction if the contents of **V***x* do not equal those of **V***x*. (9xy0)

SKP V*s*
Skip the following instruction if the key (0-15) specified by **V***s* is down (being pressed). Only the low four bits of **V***s* are used. This only samples the keypad; the pressed key (if any) is not cleared. (Es9E)

SKNP V*s*
Skip the following instruction if the key (0-15) specified by **V***s* is not pressed. Only the low four bits of **V***s* are used. This only samples the keypad; the pressed key (if any) is not cleared. (EsA1)

Opcodes related to the data registers

LD V*t*, *byte*
Put the value of *byte* in register **V***t*. (6tbb)

LD V*t*, **V***s*
Put the contents of **V***s* into **V***t*. (8ts0)

ADD V*t*, *byte*
Add the value of *byte* to the contents of **V***t*. Overflow is not recorded. (7tbb)

ADD V*t*, **V***s*
Add the contents of **V***s* into **V***t*. If the sum overflows eight bits, register **VF** is set to 01, else it is set to 00. (8ts4)

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

LD V*t*, **K**
Stop the program until a key pressed. Put the code for that key (0..15) in **V***t*. Continue to wait until the key is released before returning. (Note that this ties up the emulator until a key is entered and released on the display, or the Run/Stop button is toggled.) (Ft0A)

OR V*t*, **V***s*
OR the contents of **V***s* and **V***t* and put the result in **V***t*. (8ts1)

AND V*t*, **V***s*
AND the contents of **V***s* and **V***t* and put the result in **V***t*. (8ts2)

XOR V*t*, **V***s*
XOR the contents of **V***s* and **V***t* and put the result in **V***t*. (8ts3)

SUB V*t*, **V***s*
Subtract the contents of **V***s* from **V***t* and put the result in **V***t*. If **V***s* <= **V***t*, **VF** is set to 1. If **V***s* > **V***t* there is underflow; **VF** is set to 0 and **V***t* has the two's-complement of the answer. (8ts5)

```
LD V0, 4
LD V1, 6
SUB V0, V1
; VF==0, V0==#FE
```

SUBN V*t*, V*s*
Subtract the contents of **V*t*** from **V*s*** and put the result in **V*t***. (Same as SUB except **V*t*** is the subtrahend and **V*s*** the minuend.) (8ts7)

SHR V*t*, V*s*
Shift the contents of **V*s*** right 1. The result is stored in **V*t*** (*see note below*). The shifted-out bit, 0 or 1, is set as the value of **VF**. To shifting the contents of a register within that same register, specify the same register twice. (8ts6)

```
LD V7, #02
SHR V3, V7 ; V3=#01, V7=#02
SHR V7, V7 ; V7=#01
```

SHL V*t*, V*s*
Shift the contents of **V*s*** left 1. The shifted-out bit, 0 or 1, is set as the value of **VF**. The result is stored in **V*t*** (*see note below*). To gain the effect of shifting the contents of a register within that same register, specify the same register twice. (8tsE)

Opcodes related to the Timer and Sound registers

LD V*t*, DT
Store the current contents of the **DT** register in **V*t***. (Ft07)

LD DT, V*s*
Store the contents of **V*s*** in the timer register. (Fs15)

LD ST, V*s*
Store the contents of **V*s*** 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, *address*
Put the value of *address* in **I**. (Aaaa)

ADD I, V*s*
Add the contents of **V*s*** to **I**. Note that this can potentially set **I** to a value greater than 4095. (Fs1E)

LDC I, V*s*
Load **I** with the address of a CHIP-8 (5x4 pixel) character sprite for the number (`#0` to `#F`) in the low four bits of **V*s***. (Fs29)

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

STD V*s*
Store three bytes for the decimal digits of the contents of **V*s*** into memory at the address in **I**. The **I** contents is incremented by three. If **I** contains a value > 4093, an error stop will occur. (Fs33)

```
    LD V5, #76 ; 118 decimal
    LD I, DIGITS
    STD V5
    RET ; I=DIGITS+3
DIGITS: DB 0, 0, 0 # receives #01, #01, #08
```

LDM V*t*
Load the data registers from **V0** through **V*t*** from memory beginning at the address in **I**. The **I** value is incremented for each register loaded. (Ft65)

```
LD I, DIGITS ; see previous example
LDM V2 ; V0=#01, V1=#01, V2=#08, I=DIGITS+3
```

This is often used to load the single register **V0**: `LDM V0`.

STM V*s*
Store the data registers from **V0** through **V*s*** into memory beginning at the address in **I**. The **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 V*x*, **V*y***, *nybble*
Draw the sprite currently addressed by **I** on the screen. The X-coordinate of the upper left corner of the sprite is taken from **V*x***; the Y-coordinate from **V*y***. The value of *nybble* specifies how many bytes are in the sprite, from 1 to 15. If *nybble* is 0, **I** must address an SCHIP 16x16 sprite, comprising 32 bytes in all. (Dxyn)

If the sprite extends past the right side or bottom row of the display, some pixels "wrap" to the left side or top edge.

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 **V*s*** and put the result in **V*s***, ignoring **V*t***. This has probably not been noticed because almost all uses of the instructions name the same register for *vs* and *vt*. See the CHIP8IDE README for more information.)