# Game Boy Assembler & Emulator/Debugger

Table of Contents

[Game Boy Assembler & Emulator/Debugger 1](#_Toc106305298)

[WebASM Demo 1](#_Toc106305299)

[The Emulator 1](#_Toc106305300)

[The Assembler 1](#_Toc106305301)

[Printing 1](#_Toc106305302)

[Available Prints 1](#_Toc106305303)

[Writing Assertions 1](#_Toc106305304)

[Available Assertions 1](#_Toc106305305)

[Game Boy OpCodes 1](#_Toc106305306)

I've been having some fun writing a Game Boy game with a co-worker, nothing fancy, just for fun to learn more about the Game Boy and teaching my co-worker how to write games for it in assembly. One of the problems we have is that the tools we use to debug the code are a little difficult to use. Basically, you must run it in an emulator/debugger like No$GMB and then hit break points and review the state of the device. I thought it would be cool to just write assertions directly into the assembly code and have it told me when things are wrong. Basically, like unit testing the Game Boy code.

So, I wrote both a non-graphical **emulator** and a custom **assembler** for the Game Boy.

## WebASM Demo

Below is a demo of the C code compiled for WebASM. You can test out its functionality. To know how to use it, check out the descriptions below the demo.

[iframe src="https://brentfarris.github.io/TheLastValkyrieBuilds/gbasmemu/" style="min-width:100%;min-height:375px;"/]

## The Emulator

The emulator is being developed in C for maximum portability. I mainly wrote the emulator for debugging purposes. I wanted an easy way to test subroutines for the Game Boy without having to load up a big graphical debugger to do so. Also, those debuggers often lack the things I would like where are mainly assertions to prove that values are what they should be at a given point in the code/memory. So, this emulator is not a graphical emulator and I have yet to do any of the timing-based parts of the Game Boy to call it a complete emulator. What it is good at right now is being able to run the opcode instructions given to it and manipulate registers and memory.

## The Assembler

The assembler is being developed in C for maximum portability as well. To be able to make assertions directly in the code, it was important for me to write my own Assembler. It currently does not have any support for macros, IF statements or any fancy math, but it does have the ability to assemble the z80 code and labels into code that the emulator can run and test against. All the Game Boy instruction set is supported in the current build of the Assembler and subroutines (with dot labels) are supported as well to be able to test loops, jumps and all that sort of stuff easily. It also strips comments from the code as well.

## Printing

Though you can get a printout of the value of registers or number of clock cycles that have passed when doing assertions, you may not want to kill the program with the wrong assert value. For this reason, I've added the ability to print out any of the registers in 8-bit mode or 16-bit pair mode (including the stack pointer and program counter). You can also print out the number of clock cycles that have passed (great for performance testing). Below is an example of how to print out things.

ld hl, $2021  *; Load a value int HL*

ld [hl], $09  *; Save the value 9 to memory at HL*

print hl      *; Print what the current value of HL is*

print h       *; Print the high byte of HL*

print l       *; Print the low byte of HL*

print [hl]    *; Print the value in memory pointed to by HL*

## Available Prints

Below is a list of all prints that you can do currently.

| **Keywords** | **Description** |
| --- | --- |
| print a | Prints the value in the A register |
| print f | Prints the value in the F register |
| print b | Prints the value in the B register |
| print c | Prints the value in the C register |
| print d | Prints the value in the D register |
| print e | Prints the value in the E register |
| print h | Prints the value in the H register |
| print l | Prints the value in the L register |
| print af | Prints the 16-bit value in the AF register pair |
| print bc | Prints the 16-bit value in the BC register pair |
| print de | Prints the 16-bit value in the DE register pair |
| print hl | Prints the 16-bit value in the HL register pair |
| print [bc] | Prints the value in memory pointed to by [BC] |
| print [de] | Prints the value in memory pointed to by [DE] |
| print [hl] | Prints the value in memory pointed to by [HL] |
| print sp | Prints the value of the stack pointer |
| print pc | Prints the value of the program counter |
| print clocks | Prints the number of clock cycles that have passed |

## Writing Assertions

Here is an example of what the Assembler can do with assertions.

*; Subtract BC from HL and store the result in HL*

HL\_minus\_BC::

    ld hl, $1104    *; Our test LHS*

    ld bc, $1005    *; Our test RHS*

    push af

    ld a, l         *; Get low byte*

    sub c           *; Subtract rhs low byte*

    jr nc, .skip    *; If we didn't go negative, jump to skip*

    dec h           *; Otherwise we decrement high byte*

.skip

    ld l, a         *; Set low byte to new value*

    ld a, h         *; Get high byte*

    sub b           *; Subtract rhs high byte*

    ld h, a         *; Set high byte to new value*

    pop af

    assert eq hl, $00FF

What you will see in the code above we have a line assert eq hl, $00FF. This will test the code immediately after pop af has run to determine if the value in HL is equal to the value $00FF. This will then print out to the console if the assertion has passed or failed. This allows for quickly testing out subroutines to make sure they work as expected.

Since the assertions are available on the state of the machine, you can use assertions to dynamically check code while it is running instead of doing static checking on that specific line. For example, here is some code that uses the e register as a temporary value to use in the assert in each iteration through the loop.

check\_my\_sanity::

    ld a, $09       *; Increment value in memory at address $FF00 9x*

    ld hl, $FF00        *; Address to increment*

    ld e, $00       *; Our assertion checking device*

    ld [hl], e      *; Start our value off as 0*

.loop

    inc [hl]        *; Increment the value at $FF00*

    inc e           *; Increment our sanity checker*

    dec a           *; Decrement our loop counter*

    assert eq [hl], e   *; Assert on our dynamic value*

    jr nz, .loop

The above code uses the e register as a temp value to use in the assertions. Of course, this follows the same rules for non-asserted code, so you'd probably want to push whatever is inside of e to save and restore it if you are going to do something like this.

Also, for those of you who enjoy counting clock cycles to see how fast you can make a piece of code, you can assert on cycles as well. You can check ==, !=, <=, >=, <, > in your clocks assert. Below is an example of checking clock cycles at a given line. You can imagine checking eq clocks might not be as useful as using lt though!

*; Just loading up some stuff, both take 3 clock cycles*

ld hl, $1104    *; Our test LHS*

ld bc, $1005    *; Our test RHS*

assert eq clocks, $06

## Available Assertions

Below are 3 tables, the first table is explaining the syntax used, the second is the comparison options, and the third are the actual assertions (reference the 2 tables above it).

**Keywords**

| **Keyword** | **Description** |
| --- | --- |
| R | Any 8-bit register (a, f, b, c, d, e, h, l) |
| RR | Any 16-bit register pair (af, bc, de, hl) |
| %x | Any 8-bit number (5, $3A) |
| %xx | Any 16-bit number (536, $3A9E) |
| %xxxx | Any 32-bit number (12345678, $F36B3A9E) |
| <=> | Comparison operator (eq, neq, leq, geq, lt, gt) |
| clocks | The number of clock cycles that have passed since start |

**Comparison operators**

| **Keyword** | **Description** |
| --- | --- |
| eq | Are equal |
| neq | Are not equal |
| leq | Left is less than or equal to right |
| geq | Left is greater than or equal to right |
| lt | Left is less than right |
| gt | Left is greater than right |

**Assertion instructions**

| **Format** | **Example** | **Description** |
| --- | --- | --- |
| assert <=> R, %x | assert eq a, $3F | Compares a register to an 8-bit value |
| assert <=> R, R | assert neq b, e | Compares the value of 2 registers |
| assert <=> RR, RR | assert leq bc, de | Compares the values of 2 16-bit register pairs |
| assert <=> RR, %xx | assert geq de, $020F | Compares the values of a 16-bit register to a 16-bit value |
| assert <=> [RR], %x | assert lt [hl], $03 | Compares the value in memory at address held in 16-bit register pair to an 8-bit value |
| assert <=> [RR], R | assert gt [hl], e | Compares the value in memory at address held in 16-bit register pair to a register value |
| assert <=> [%xx], %x | assert eq [$3F9A], $09 | Compares the value in memory at address to an 8-bit value |
| assert <=> [%xx], R | assert eq [$2000], a | Compares the value in memory at address to a register value |
| assert <=> clocks, %xxxx | assert lt clocks, 9 | Compares the number of clock cycles that have passed to the given value |

## Game Boy OpCodes

| **Instruction** | **OpCode** | **Clocks** |
| --- | --- | --- |
| nop | 00 | 1 |
| ld bc, %xx | 01 | 3 |
| ld [bc], a | 02 | 2 |
| inc bc | 03 | 2 |
| inc b | 04 | 1 |
| dec b | 05 | 1 |
| ld b, %x | 06 | 2 |
| rlca | 07 | 1 |
| ld [%xx], sp | 08 | 5 |
| add hl, bc | 09 | 2 |
| ld a, [bc] | 0A | 2 |
| inc bc | 0B | 2 |
| inc c | 0C | 1 |
| dec c | 0D | 1 |
| ld c, %x | 0E | 2 |
| rrca | 0F | 1 |
| ld de, %xx | 11 | 3 |
| ld [de], a | 12 | 2 |
| inc de | 13 | 2 |
| inc d | 14 | 1 |
| dec d | 15 | 1 |
| ld d, %x | 16 | 2 |
| rla | 17 | 1 |
| jr %x | 18 | 2 |
| add hl, de | 19 | 2 |
| ld a, [de] | 1A | 2 |
| inc de | 1B | 2 |
| inc e | 1C | 1 |
| dec e | 1D | 1 |
| ld e, %x | 1E | 2 |
| rra | 1F | 1 |
| jr nz, %x | 20 | 2 |
| ld hl, %xx | 21 | 3 |
| ld [hli], a | 22 | 2 |
| inc hl | 23 | 2 |
| inc h | 24 | 1 |
| dec h | 25 | 1 |
| ld h, %x | 26 | 2 |
| daa | 27 | 1 |
| jr z, %xx | 28 | 2 |
| add hl, hl | 29 | 2 |
| ld a, [hli] | 2A | 2 |
| inc hl | 2B | 2 |
| inc l | 2C | 1 |
| dec l | 2D | 1 |
| ld l, %x | 2E | 2 |
| cpl | 2F | 1 |
| jr nc, %x | 30 | 2 |
| ld sp, %xx | 31 | 3 |
| ld [hld], a | 32 | 2 |
| inc sp | 33 | 2 |
| inc [hl] | 34 | 3 |
| dec [hl] | 35 | 3 |
| ld [hl], %x | 36 | 3 |
| scf | 37 | 1 |
| jr c, %x | 38 | 2 |
| add hl, sp | 39 | 2 |
| ld a, [hld] | 3A | 2 |
| inc sp | 3B | 2 |
| inc a | 3C | 1 |
| dec a | 3D | 1 |
| ld a, %x | 3E | 2 |
| ccf | 3F | 1 |
| ld b, b | 40 | 1 |
| ld b, c | 41 | 1 |
| ld b, d | 42 | 1 |
| ld b, e | 43 | 1 |
| ld b, h | 44 | 1 |
| ld b, l | 45 | 1 |
| ld b, [hl] | 46 | 2 |
| ld b, a | 47 | 1 |
| ld c, b | 48 | 1 |
| ld c, c | 49 | 1 |
| ld c, d | 4A | 1 |
| ld c, e | 4B | 1 |
| ld c, h | 4C | 1 |
| ld c, l | 4D | 1 |
| ld c, [hl] | 4E | 2 |
| ld c, a | 4F | 1 |
| ld d, b | 50 | 1 |
| ld d, c | 51 | 1 |
| ld d, d | 52 | 1 |
| ld d, e | 53 | 1 |
| ld d, h | 54 | 1 |
| ld d, l | 55 | 1 |
| ld d, [hl] | 56 | 2 |
| ld d, a | 57 | 1 |
| ld e, b | 58 | 1 |
| ld e, c | 59 | 1 |
| ld e, d | 5A | 1 |
| ld e, e | 5B | 1 |
| ld e, h | 5C | 1 |
| ld e, l | 5D | 1 |
| ld e, [hl] | 5E | 2 |
| ld e, a | 5F | 1 |
| ld h, b | 60 | 1 |
| ld h, c | 61 | 1 |
| ld h, d | 62 | 1 |
| ld h, e | 63 | 1 |
| ld h, h | 64 | 1 |
| ld h, l | 65 | 1 |
| ld h, [hl] | 66 | 2 |
| ld h, a | 67 | 1 |
| ld l, b | 68 | 1 |
| ld l, c | 69 | 1 |
| ld l, d | 6A | 1 |
| ld l, e | 6B | 1 |
| ld l, h | 6C | 1 |
| ld l, l | 6D | 1 |
| ld l, [hl] | 6E | 2 |
| ld l, a | 6F | 1 |
| ld [hl], b | 70 | 2 |
| ld [hl], c | 71 | 2 |
| ld [hl], d | 72 | 2 |
| ld [hl], e | 73 | 2 |
| ld [hl], h | 74 | 2 |
| ld [hl], l | 75 | 2 |
| halt | 76 | 1 |
| ld [hl], a | 77 | 2 |
| ld a, b | 78 | 1 |
| ld a, c | 79 | 1 |
| ld a, d | 7A | 1 |
| ld a, e | 7B | 1 |
| ld a, h | 7C | 1 |
| ld a, l | 7D | 1 |
| ld a, [hl] | 7E | 2 |
| ld a, a | 7F | 1 |
| add b | 80 | 1 |
| add c | 81 | 1 |
| add d | 82 | 1 |
| add e | 83 | 1 |
| add h | 84 | 1 |
| add l | 85 | 1 |
| add [hl] | 86 | 2 |
| add a | 87 | 1 |
| adc b | 88 | 1 |
| adc c | 89 | 1 |
| adc d | 8A | 1 |
| adc e | 8B | 1 |
| adc h | 8C | 1 |
| adc l | 8D | 1 |
| adc [hl] | 8E | 2 |
| adc a | 8F | 1 |
| sub b | 90 | 1 |
| sub c | 91 | 1 |
| sub d | 92 | 1 |
| sub e | 93 | 1 |
| sub h | 94 | 1 |
| sub l | 95 | 1 |
| sub [hl] | 96 | 2 |
| sub a | 97 | 1 |
| sbc b | 98 | 1 |
| sbc c | 99 | 1 |
| sbc d | 9A | 1 |
| sbc e | 9B | 1 |
| sbc h | 9C | 1 |
| sbc l | 9D | 1 |
| sbc [hl] | 9E | 2 |
| sbc a | 9F | 1 |
| and b | A0 | 1 |
| and c | A1 | 1 |
| and d | A2 | 1 |
| and e | A3 | 1 |
| and h | A4 | 1 |
| and l | A5 | 1 |
| and [hl] | A6 | 2 |
| and a | A7 | 1 |
| xor b | A8 | 1 |
| xor c | A9 | 1 |
| xor d | AA | 1 |
| xor e | AB | 1 |
| xor h | AC | 1 |
| xor l | AD | 1 |
| xor [hl] | AE | 2 |
| xor a | AF | 1 |
| or b | B0 | 1 |
| or c | B1 | 1 |
| or d | B2 | 1 |
| or e | B3 | 1 |
| or h | B4 | 1 |
| or l | B5 | 1 |
| or [hl] | B6 | 2 |
| or a | B7 | 1 |
| cp b | B8 | 1 |
| cp c | B9 | 1 |
| cp d | BA | 1 |
| cp e | BB | 1 |
| cp h | BC | 1 |
| cp l | BD | 1 |
| cp [hl] | BE | 2 |
| cp a | BF | 1 |
| ret nz | C0 | 2 |
| pop bc | C1 | 3 |
| jp nz, %xx | C2 | 3 |
| jp %xx | C3 | 3 |
| call nz, %xx | C4 | 3 |
| push bc | C5 | 4 |
| add a, %x | C6 | 2 |
| rst 00H | C7 | 8 |
| ret z | C8 | 2 |
| ret | C9 | 2 |
| jp z, %xx | CA | 3 |
| call z, %xx | CC | 3 |
| call %xx | CD | 3 |
| adc a, %x | CE | 2 |
| rst 08H | CF | 8 |
| ret nc | D0 | 2 |
| pop de | D1 | 3 |
| jp nc, %xx | D2 | 3 |
| call nc, %xx | D4 | 0 |
| push de | D5 | 0 |
| sub a, %x | D6 | 2 |
| rst 10H | D7 | 8 |
| ret c | D8 | 2 |
| reti | D9 | 2 |
| jp c, %xx | DA | 3 |
| call c, %xx | DC | 3 |
| sbc a, %x | DE | 0 |
| rst 18H | DF | 8 |
| ld [$ff00+c], a | E0 | 3 |
| pop hl | E1 | 3 |
| ld [c], a | E2 | 2 |
| push hl | E5 | 4 |
| and %x | E6 | 2 |
| rst 20H | E7 | 8 |
| add sp, %x | E8 | 4 |
| jp [hl] | E9 | 1 |
| ld [%xx], a | EA | 4 |
| xor %x | EE | 2 |
| rst 28H | EF | 8 |
| ld a, [$ff00+c] | F0 | 3 |
| pop af | F1 | 3 |
| ld a, [c] | F2 | 2 |
| di | F3 | 1 |
| push af | F5 | 4 |
| or %x | F6 | 2 |
| rst 30H | F7 | 8 |
| ldhl sp, %x | F8 | 3 |
| ld sp, hl | F9 | 2 |
| ld a, [%xx] | FA | 4 |
| ei | FB | 1 |
| cp %x | FE | 2 |
| rst 38H | FF | 8 |
| swap a | CB37 | 2 |
| swap b | CB30 | 2 |
| swap c | CB31 | 2 |
| swap d | CB32 | 2 |
| swap e | CB33 | 2 |
| swap h | CB34 | 2 |
| swap l | CB35 | 2 |
| swap [hl] | CB36 | 4 |
| rlc a | CB07 | 2 |
| rlc b | CB00 | 2 |
| rlc c | CB01 | 2 |
| rlc d | CB02 | 2 |
| rlc e | CB03 | 2 |
| rlc h | CB04 | 2 |
| rlc l | CB05 | 2 |
| rlc [hl] | CB06 | 4 |
| rl a | CB17 | 2 |
| rl b | CB10 | 2 |
| rl c | CB11 | 2 |
| rl d | CB12 | 2 |
| rl e | CB13 | 2 |
| rl h | CB14 | 2 |
| rl l | CB15 | 2 |
| rl [hl] | CB16 | 4 |
| rrc a | CB0F | 2 |
| rrc b | CB08 | 2 |
| rrc c | CB09 | 2 |
| rrc d | CB0A | 2 |
| rrc e | CB0B | 2 |
| rrc h | CB0C | 2 |
| rrc l | CB0D | 2 |
| rrc [hl] | CB0E | 4 |
| rr a | CB1F | 2 |
| rr b | CB18 | 2 |
| rr c | CB19 | 2 |
| rr d | CB1A | 2 |
| rr e | CB1B | 2 |
| rr h | CB1C | 2 |
| rr l | CB1D | 2 |
| rr [hl] | CB1E | 4 |
| sla a | CB27 | 2 |
| sla b | CB20 | 2 |
| sla c | CB21 | 2 |
| sla d | CB22 | 2 |
| sla e | CB23 | 2 |
| sla h | CB24 | 2 |
| sla l | CB25 | 2 |
| sla [hl] | CB26 | 4 |
| sra a | CB2F | 2 |
| sra b | CB28 | 2 |
| sra c | CB29 | 2 |
| sra d | CB2A | 2 |
| sra e | CB2B | 2 |
| sra h | CB2C | 2 |
| sra l | CB2D | 2 |
| sra [hl] | CB2E | 4 |
| srl a | CB3F | 2 |
| srl b | CB38 | 2 |
| srl c | CB39 | 2 |
| srl d | CB3A | 2 |
| srl e | CB3B | 2 |
| srl h | CB3C | 2 |
| srl l | CB3D | 2 |
| srl [hl] | CB3E | 4 |
| bit a | CB47 | 2 |
| bit b | CB40 | 2 |
| bit c | CB41 | 2 |
| bit d | CB42 | 2 |
| bit e | CB43 | 2 |
| bit h | CB44 | 2 |
| bit l | CB45 | 2 |
| bit [hl] | CB46 | 4 |
| set a | CBC7 | 2 |
| set b | CBC0 | 2 |
| set c | CBC1 | 2 |
| set d | CBC2 | 2 |
| set e | CBC3 | 2 |
| set h | CBC4 | 2 |
| set l | CBC5 | 2 |
| set [hl] | CBC6 | 4 |
| res a | CB87 | 2 |
| res b | CB80 | 2 |
| res c | CB81 | 2 |
| res d | CB82 | 2 |
| res e | CB83 | 2 |
| res h | CB84 | 2 |
| res l | CB85 | 2 |
| res [hl] | CB86 | 4 |
| stop | 1000 | 1 |