Blazing Trails:

Building the World's Fastest GameBoy Emulator in Modern C++

TOM TESCH

Cppcon
The C++ Conference

2024

September 15 - 20

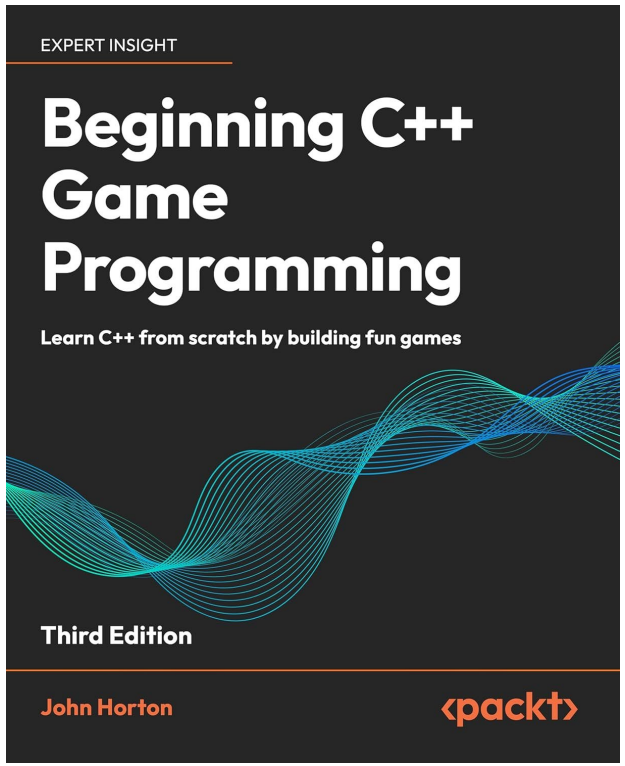
About me

- Teacher @ Howest University of Applied Sciences within “Digital Arts and Entertainment” bachelor,r Game Development major
  - C++
  - Algorithms
  - Computational System Fundamentals
  - ***new*** Retro Console and Emulator Programming

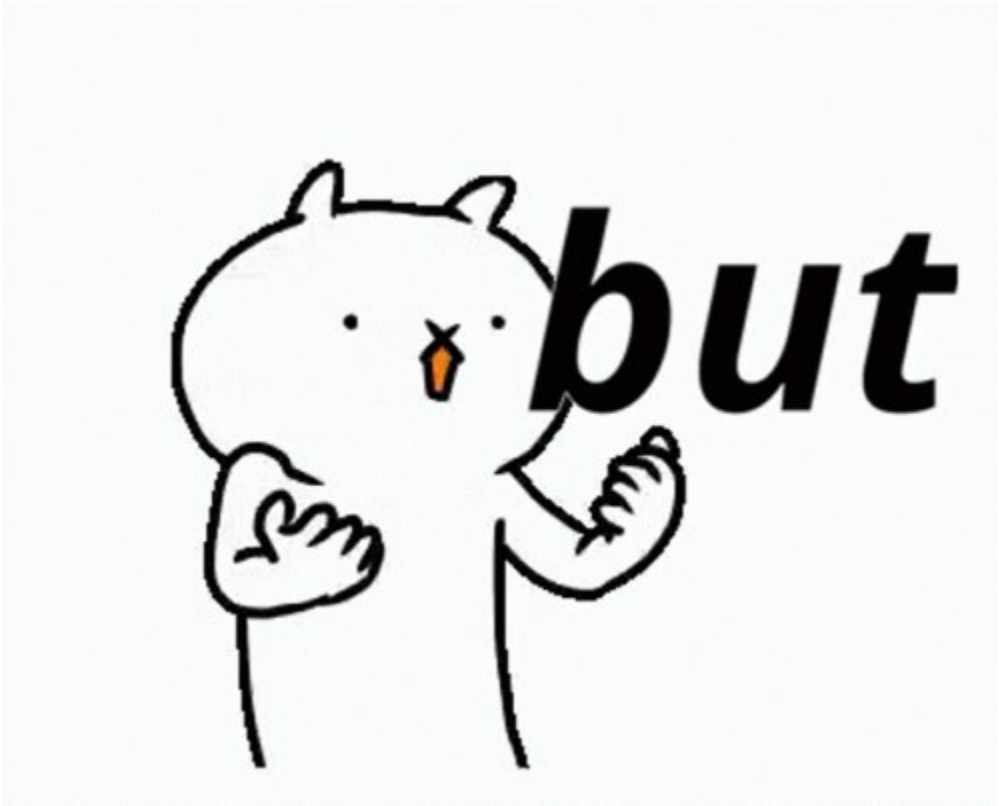
- https://www.howest.be/en
- https://www.digitalartsandentertainment.be/

- tom.tesch@howest.be
- https://www.linkedin.com/in/tom-tesch/

During this talk: some giveaways!



Building the World's Fastest GameBoy Emulator in Modern C++



Building the World's Fastest GameBoy Emulator **in Modern C++**

Building the World's Fastest GameBoy **Emulator** in Modern C++

Deep System Understanding

Emulators require a strong grasp of hardware, low-level operations, and CPU architecture.

Building one helps you understand how hardware like memory, registers, and buses work, and how they interact with machine code.

This is valuable for a C++ programmer, as C++ often deals with system-level programming.

Building the World's Fastest **GameBoy** Emulator in Modern C++

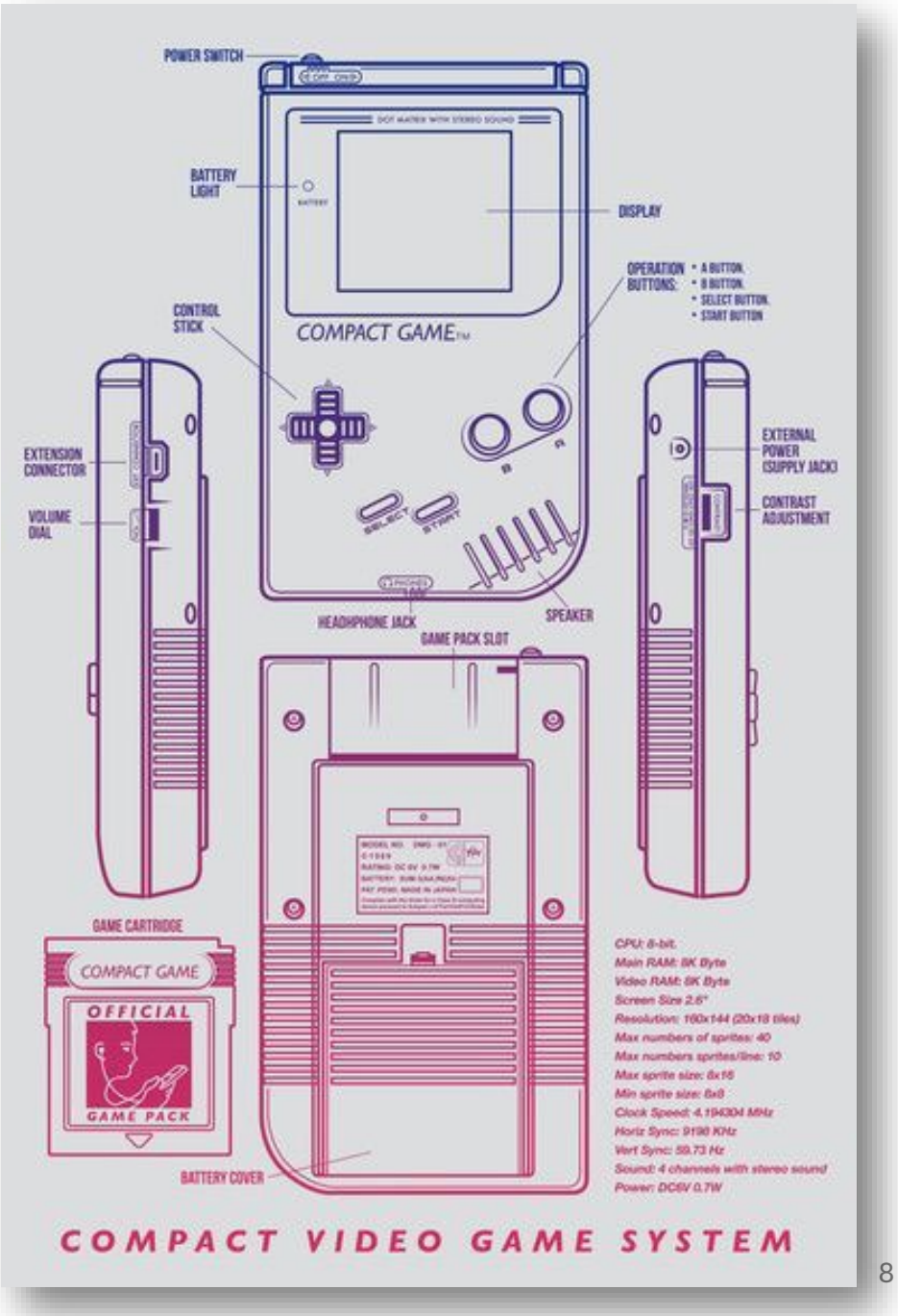
The original Game Boy DMG-01

- Released on
  - 21/04/1989 in Japan
  - 31/07/1989 in America
  - 28/09/1990 in Europe

- The Game Boy can be considered as a portable version of the NES with limited power

- 119 million units sold (197M including GBA)

- (Terrible) Screen 160 x 144 pixels
- 8 buttons (d-pad, start, select, A, B)
- Stereo sound (via headphones)
- Mono Speaker



Emulating a Gameboy is easy(ish)

hhugboy	
SkyEmu	
binjgb	
higan	
BDM	
VGB	
Gamebert	
Pantheon	
UGE	
KiGB	
TGB Dual	[N 9]
GiiBiiAdvance	
MetroBoy	
hdmg	

SameBoy	
BGB	
Gambatte	
BizHawk	
Emulicious	
Mesen2	
GBE+	
MAME	
Gearboy	
GBCC	
GameRoy	
ares	
mGBA	
VBA-M	

Game Boy: Complete Technical Reference

gekkio

https://gekkio.fi

January 16, 2023

Revision 125

Game Boy™ CPU Manual

Sources by: Pan of Anthrox, GABY, Marat Fayzullin, Pascal Felber, Paul Robson, Martin Korth, kOOPa, Bowser

Contents:

Assembly Language Commands, Timings and Opcodes, and everything you always wanted to know about GB but were afraid to ask.

And if it isn't easy: it's testable

https://github.com/c-sp/game-boy-test-roms

Game Boy Test Roms

Have a Game Boy emulator you want to test for accuracy? This collection of compiled Game Boy test roms might help. It includes (in alphabetical order):

- AGE test roms
  by me
- Blargg's test roms
  by Shay Green (a.k.a. Blargg)
- Bully, Scribbltests and Strikethrough
  by Hacktix
- cgb-acid-hell, cgb-acid2, dmg-acid2 and Mealybug Tearoom Tests
  by Matt Currie
- Gambatte test suite (as the original Gambatte repository is not public anymore, we use pokemon-speedrunning/gambatte-core instead)
  by sinamas
- GBMicrotest
  by aappleby
- MBC3 Tester
  by Eric Kirschenmann
- Mooneye Test Suite
  by Joonas Javanainen
- Mooneye Test Suite (wilbertpol)
  by Joonas Javanainen and wilbertpol
- (parts of) little-things-gb
  by Damian Yerrick
- rtc3test
  by ax6
- SameSuite
  by Lior Halphon
- TurtleTests
  by Brian Jia

Building the World's Fastest **GameBoy** Emulator in Modern C++

A huge library:

- GameBoy: 1043
- GameBoy Color: 576

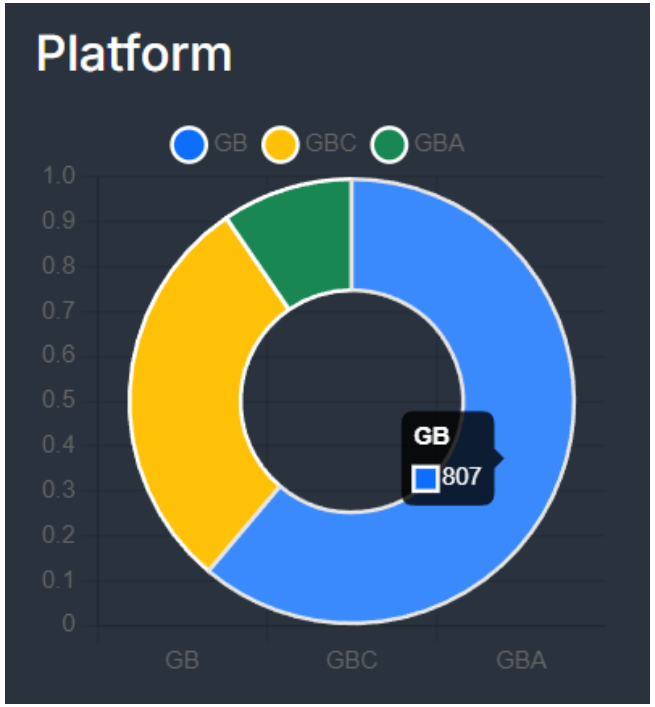
30% are backwards compatible

A tiny library:

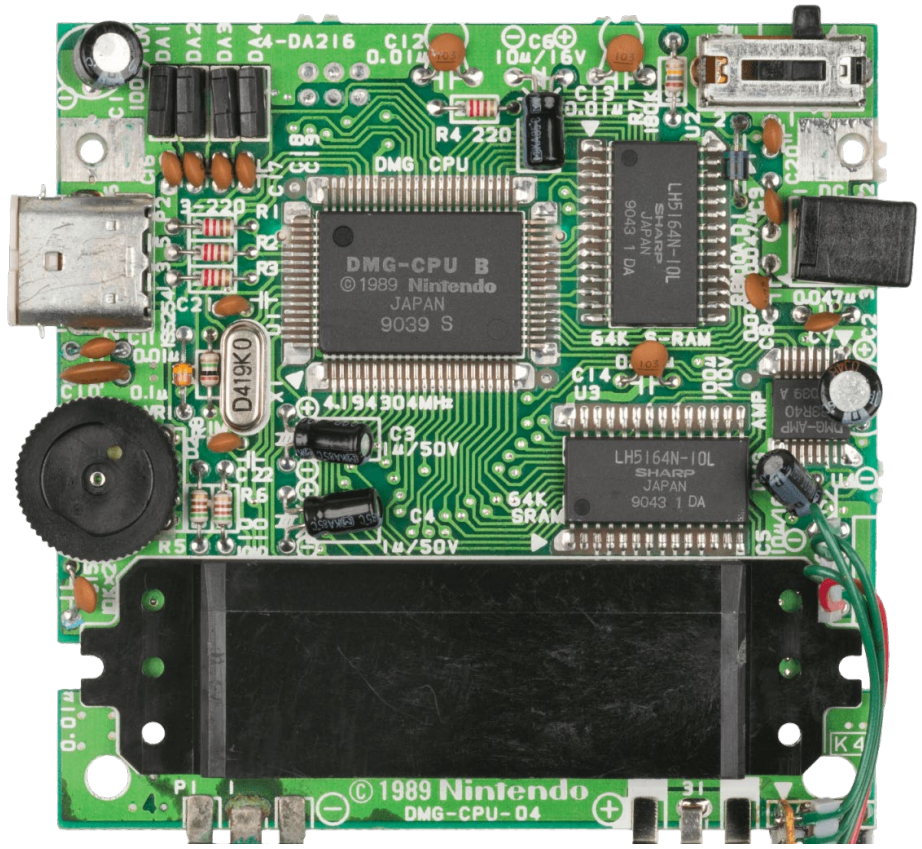
- GameBoy: ± 800MB
- GameBoy Color: ± 1.5GB

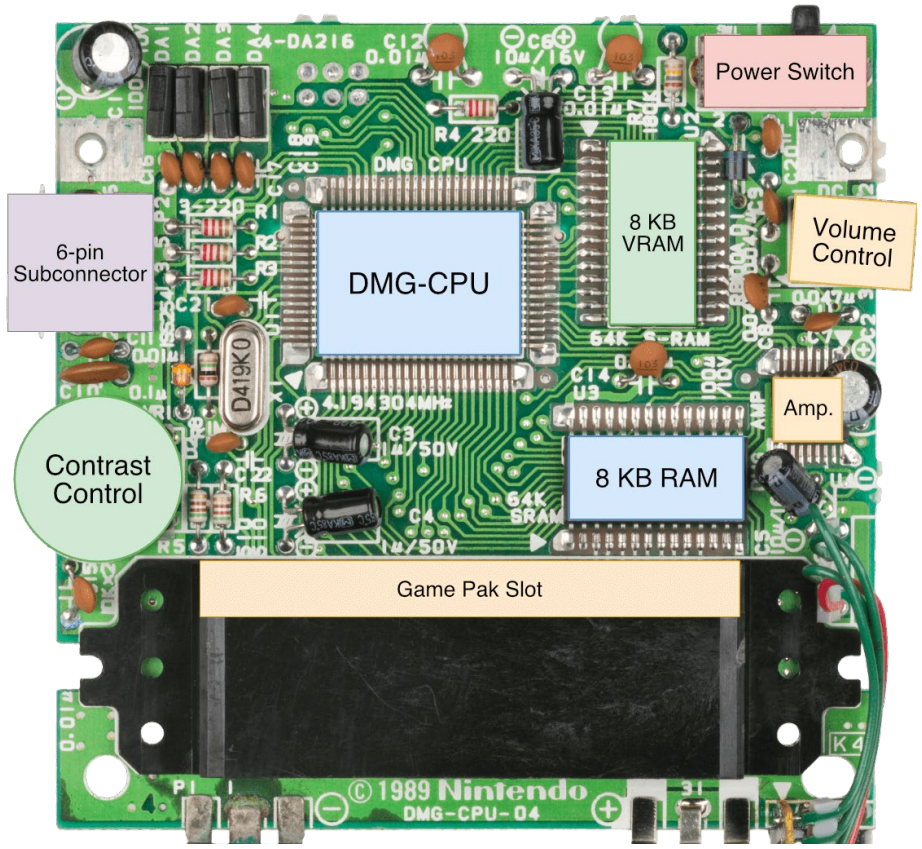
Active Homebrew

https://hh.gbdev.io/



Motherboard (revision 04)



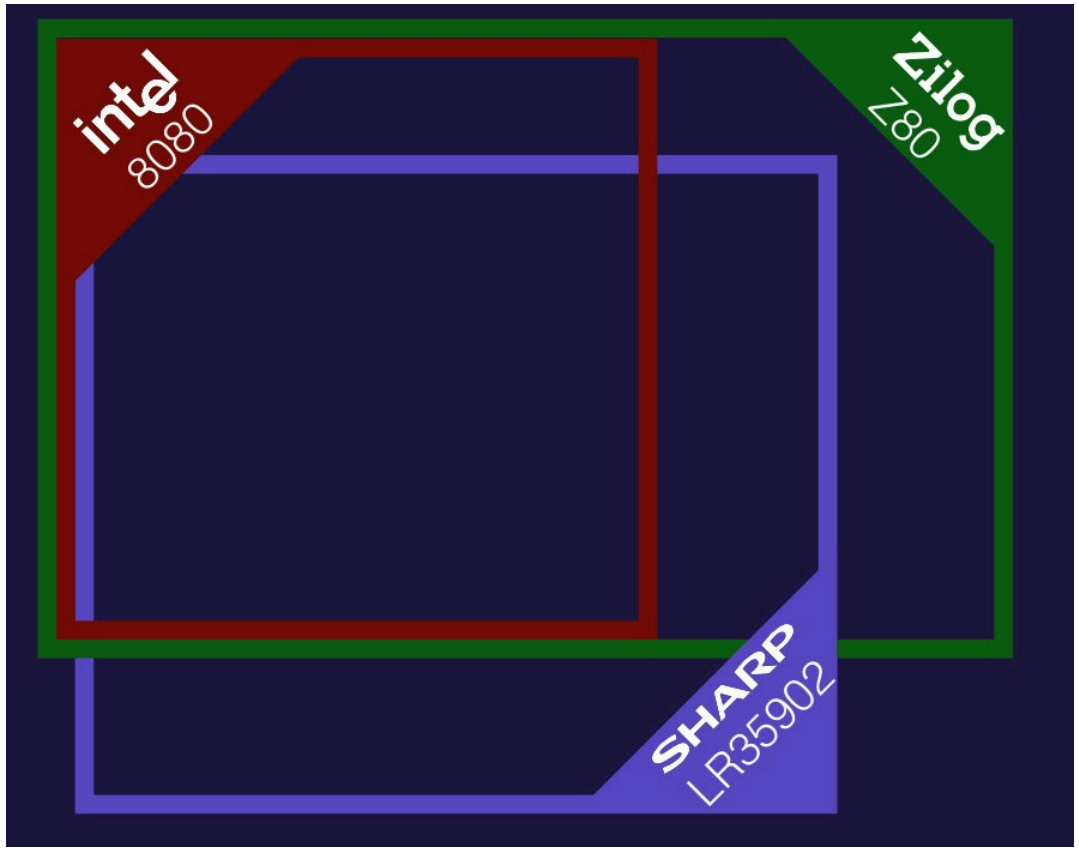


CPU

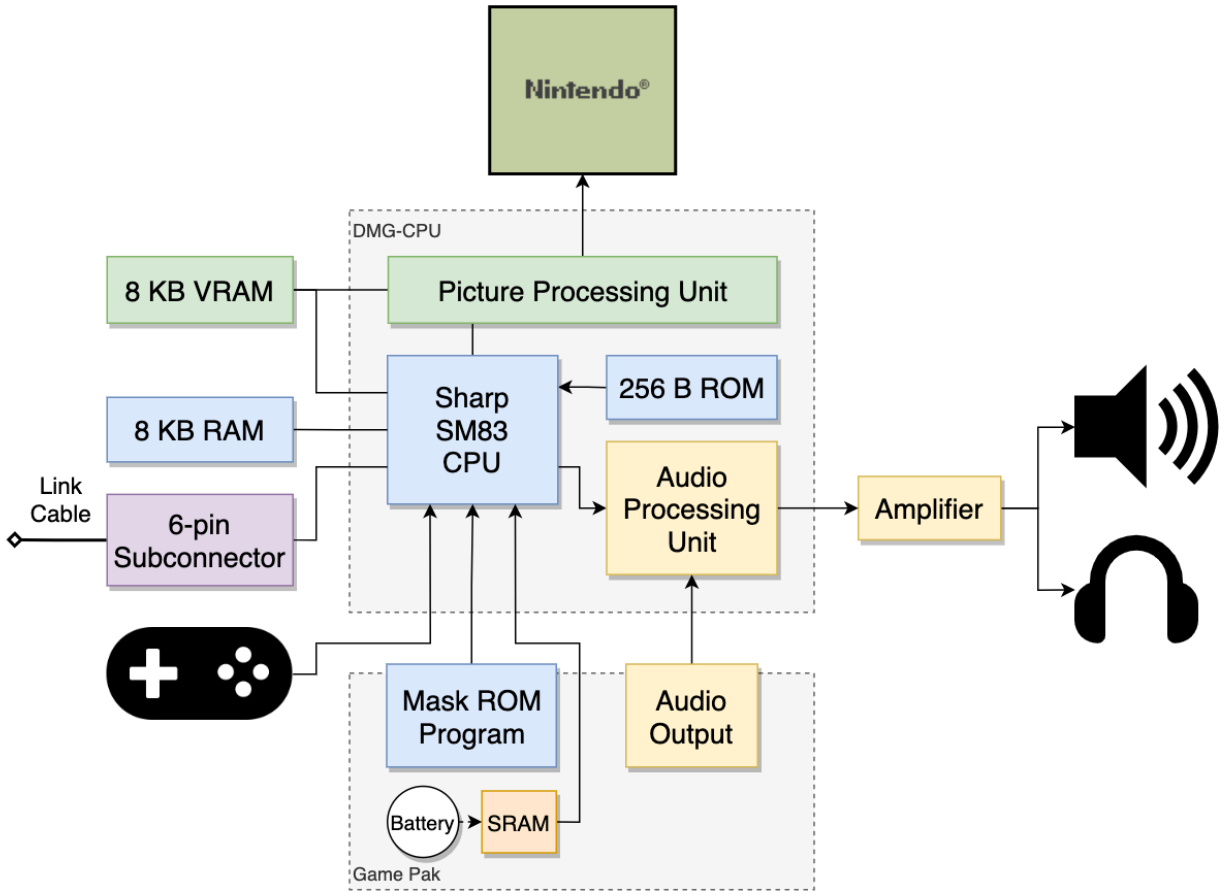
- System On a Chip (SoC)
- DMG-CPU or Sharp LR35902

- Sharp SM83 (mix between Z80 and Intel 8080)
- 4,194,304 Hertz ≈ 4 Mhz

- No IO ports = **completely memory mapped**
- Only 8080's set of registers
- Z80's extended bit manipulation instruction set
- Some aditional new instructions

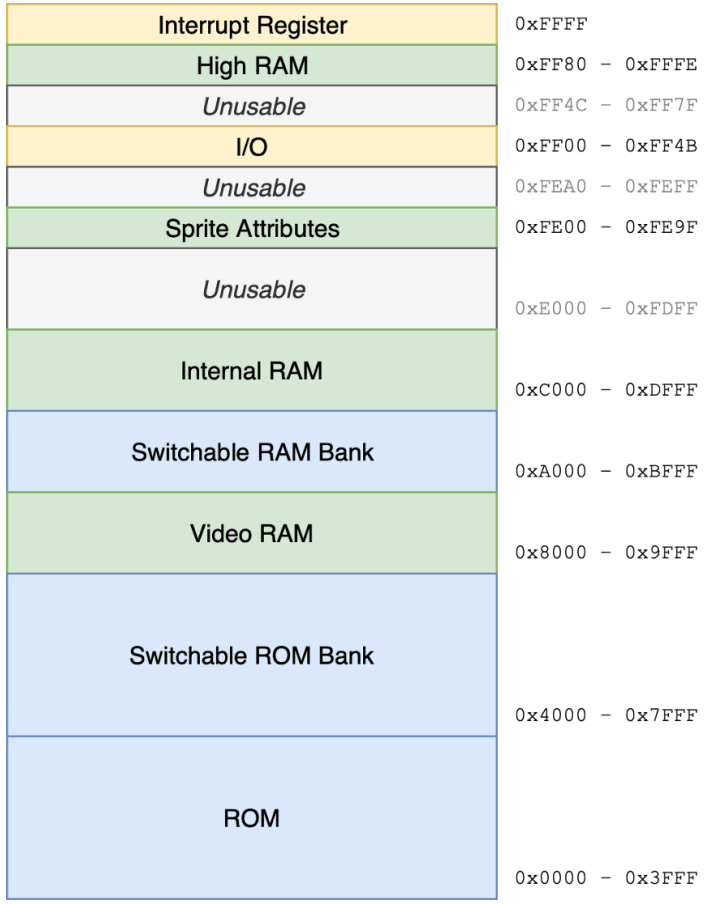


CPU Block Diagram



Memory & Memory mapped access

- 8 KiB of RAM (called WRAM)
  - Four times more than the NES!
- 8 KiB of Video RAM (called VRAM)
- Memory map:
  - Cartridge space.
  - WRAM and Display RAM.
  - I/O (joypad, audio, graphics and LCD)
  - Interrupt controls.



The ultimate Gameboy talk – Michael Steil

https://media.ccc.de/v/33c3-8029-the_ultimate_game_boy_talk

Still too much to fully implement in 1 hour

Game Boy CPU (SM83) instruction set (JSON)

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 4 - - - -	LD BC, n16 3 12 - - - -	LD [BC], A 1 8 - - - -	INC BC 1 8 - - - -	INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B, n8 2 8 - - - -	RLCA 1 4 0 0 0 C	LD [a16], SP 3 20 - - - -	ADD HL, BC 1 8 - 0 H C	LD A, [BC] 1 8 - - - -	DEC BC 1 8 - - - -	INC C 1 4 Z 0 H -	DEC C 1 4 Z 1 H -	LD C, n8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x	STOP n8 2 4 - - - -	LD DE, n16 3 12 - - - -	LD [DE], A 1 8 - - - -	INC DE 1 8 - - - -	INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D, n8 2 8 - - - -	RLA 1 4 0 0 0 C	JR e8 2 12 - - - -	ADD HL, DE 1 8 - 0 H C	LD A, [DE] 1 8 - - - -	DEC DE 1 8 - - - -	INC E 1 4 Z 0 H -	DEC E 1 4 Z 1 H -	LD E, n8 2 8 - - - -	RRA 1 4 0 0 0 C
2x	JR NZ, e8 2 12/8 - - - -	LD HL, n16 3 12 - - - -	LD [HL+], A 1 8 - - - -	INC HL 1 8 - - - -	INC H 1 4 Z 0 H -	DEC H 1 4 Z 1 H -	LD H, n8 2 8 - - - -	DAA 1 4 Z - 0 C	JR Z, e8 2 12/8 - - - -	ADD HL, HL 1 8 - 0 H C	LD A, [HL+] 1 8 - - - -	DEC HL 1 8 - - - -	INC L 1 4 Z 0 H -	DEC L 1 4 Z 1 H -	LD L, n8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	JR NC, e8 2 12/8 - - - -	LD SP, n16 3 12 - - - -	LD [HL-], A 1 8 - - - -	INC SP 1 8 - - - -	INC [HL] 1 12 Z 0 H -	DEC [HL] 1 12 Z 1 H -	LD [HL], n8 2 12 - - - -	SCF 1 4 - 0 0 1	JR C, e8 2 12/8 - - - -	ADD HL, SP 1 8 - 0 H C	LD A, [HL-] 1 8 - - - -	DEC SP 1 8 - - - -	INC A 1 4 Z 0 H -	DEC A 1 4 Z 1 H -	LD A, n8 2 8 - - - -	CCF 1 4 - 0 0 C
4x	LD B, B 1 4 - - - -	LD B, C 1 4 - - - -	LD B, D 1 4 - - - -	LD B, E 1 4 - - - -	LD B, H 1 4 - - - -	LD B, L 1 4 - - - -	LD B, [HL] 1 8 - - - -	LD B, A 1 4 - - - -	LD C, B 1 4 - - - -	LD C, C 1 4 - - - -	LD C, D 1 4 - - - -	LD C, E 1 4 - - - -	LD C, H 1 4 - - - -	LD C, L 1 4 - - - -	LD C, [HL] 1 8 - - - -	LD C, A 1 4 - - - -
5x	LD D, B 1 4 - - - -	LD D, C 1 4 - - - -	LD D, D 1 4 - - - -	LD D, E 1 4 - - - -	LD D, H 1 4 - - - -	LD D, L 1 4 - - - -	LD D, [HL] 1 8 - - - -	LD D, A 1 4 - - - -	LD E, B 1 4 - - - -	LD E, C 1 4 - - - -	LD E, D 1 4 - - - -	LD E, E 1 4 - - - -	LD E, H 1 4 - - - -	LD E, L 1 4 - - - -	LD E, [HL] 1 8 - - - -	LD E, A 1 4 - - - -
6x	LD H, B 1 4 - - - -	LD H, C 1 4 - - - -	LD H, D 1 4 - - - -	LD H, E 1 4 - - - -	LD H, H 1 4 - - - -	LD H, L 1 4 - - - -	LD H, [HL] 1 8 - - - -	LD H, A 1 4 - - - -	LD L, B 1 4 - - - -	LD L, C 1 4 - - - -	LD L, D 1 4 - - - -	LD L, E 1 4 - - - -	LD L, H 1 4 - - - -	LD L, L 1 4 - - - -	LD L, [HL] 1 8 - - - -	LD L, A 1 4 - - - -
7x	LD [HL], B 1 8 - - - -	LD [HL], C 1 8 - - - -	LD [HL], D 1 8 - - - -	LD [HL], E 1 8 - - - -	LD [HL], H 1 8 - - - -	LD [HL], L 1 8 - - - -	HALT 1 4 - - - -	LD [HL], A 1 8 - - - -	LD A, B 1 4 - - - -	LD A, C 1 4 - - - -	LD A, D 1 4 - - - -	LD A, E 1 4 - - - -	LD A, H 1 4 - - - -	LD A, L 1 4 - - - -	LD A, [HL] 1 8 - - - -	LD A, A 1 4 - - - -
8x	ADD A, B 1 4 Z 0 H C	ADD A, C 1 4 Z 0 H C	ADD A, D 1 4 Z 0 H C	ADD A, E 1 4 Z 0 H C	ADD A, H 1 4 Z 0 H C	ADD A, L 1 4 Z 0 H C	ADD A, [HL] 1 8 Z 0 H C	ADD A, A 1 4 Z 0 H C	ADC A, B 1 4 Z 0 H C	ADC A, C 1 4 Z 0 H C	ADC A, D 1 4 Z 0 H C	ADC A, E 1 4 Z 0 H C	ADC A, H 1 4 Z 0 H C	ADC A, L 1 4 Z 0 H C	ADC A, [HL] 1 8 Z 0 H C	ADC A, A 1 4 Z 0 H C
9x	SUB A, B 1 4 Z 1 H C	SUB A, C 1 4 Z 1 H C	SUB A, D 1 4 Z 1 H C	SUB A, E 1 4 Z 1 H C	SUB A, H 1 4 Z 1 H C	SUB A, L 1 4 Z 1 H C	SUB A, [HL] 1 8 Z 1 H C	SUB A, A 1 4 1 1 0 0	SBC A, B 1 4 Z 1 H C	SBC A, C 1 4 Z 1 H C	SBC A, D 1 4 Z 1 H C	SBC A, E 1 4 Z 1 H C	SBC A, H 1 4 Z 1 H C	SBC A, L 1 4 Z 1 H C	SBC A, [HL] 1 8 Z 1 H C	SBC A, A 1 4 Z 1 H -
Ax	AND A, B 1 4 Z 0 1 0	AND A, C 1 4 Z 0 1 0	AND A, D 1 4 Z 0 1 0	AND A, E 1 4 Z 0 1 0	AND A, H 1 4 Z 0 1 0	AND A, L 1 4 Z 0 1 0	AND A, [HL] 1 8 Z 0 1 0	AND A, A 1 4 Z 0 1 0	XOR A, B 1 4 Z 0 0 0	XOR A, C 1 4 Z 0 0 0	XOR A, D 1 4 Z 0 0 0	XOR A, E 1 4 Z 0 0 0	XOR A, H 1 4 Z 0 0 0	XOR A, L 1 4 Z 0 0 0	XOR A, [HL] 1 8 Z 0 0 0	XOR A, A 1 4 1 0 0 0
Bx	OR A, B 1 4 Z 0 0 0	OR A, C 1 4 Z 0 0 0	OR A, D 1 4 Z 0 0 0	OR A, E 1 4 Z 0 0 0	OR A, H 1 4 Z 0 0 0	OR A, L 1 4 Z 0 0 0	OR A, [HL] 1 8 Z 0 0 0	OR A, A 1 4 Z 0 0 0	CP A, B 1 4 Z 1 H C	CP A, C 1 4 Z 1 H C	CP A, D 1 4 Z 1 H C	CP A, E 1 4 Z 1 H C	CP A, H 1 4 Z 1 H C	CP A, L 1 4 Z 1 H C	CP A, [HL] 1 8 Z 1 H C	CP A, A 1 4 1 1 0 0
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ, a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ, a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A, n8 2 8 Z 0 H C	RST $00 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z, a16 3 16/12 - - - -	PREFIX 1 4 - - - -	CALL Z, a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A, n8 2 8 Z 0 H C	RST $08 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC, a16 3 16/12 - - - -	—	CALL NC, a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB A, n8 2 8 Z 1 H C	RST $10 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C, a16 3 16/12 - - - -	—	CALL C, a16 3 24/12 - - - -	—	SBC A, n8 2 8 Z 1 H C	RST $18 1 16 - - - -
Ex	LDH [a8], A 2 12 - - - -	POP HL 1 12 - - - -	LD [C], A 1 8 - - - -	—	—	PUSH HL 1 16 - - - -	AND A, n8 2 8 Z 0 1 0	RST $20 1 16 - - - -	ADD SP, e8 2 16 0 0 H C	JP HL 1 4 - - - -	LD [a16], A 3 16 - - - -	—	—	—	XOR A, n8 2 8 Z 0 0 0	RST $28 1 16 - - - -
Fx	LDH A, [a8] 2 12 - - - -	POP AF 1 12 Z N H C	LD A, [C] 1 8 - - - -	DI 1 4 - - - -	—	PUSH AF 1 16 - - - -	OR A, n8 2 8 Z 0 0 0	RST $30 1 16 - - - -	LD HL, SP + e8 2 12 0 0 H C	LD SP, HL 1 8 - - - -	LD A, [a16] 3 16 - - - -	EI 1 4 - - - -	—	—	CP A, n8 2 8 Z 1 H C	RST $38 1 16 - - - -

(SM83) instruction set (JSON)

- Misc / control instructions
- Jumps / calls
- 8-bit load instructions
- 16-bit load instructions
- 8-bit arithmetic / logical instructions
- 16-bit arithmetic / logical instructions
- 8-bit shift, rotate and bit instructions

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x					INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B, n8 2 8 - - - -	RLCA 1 4 0 0 0 C	LD [a16], SP 3 20 - - - -	ADD HL, BC 1 8 - 0 H C	LD A, [BC] 1 8 - - - -	DEC BC 1 8 - - - -	INC C 1 4 Z 0 H -	DEC C 1 4 Z 1 H -	LD C, n8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x					INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D, n8 2 8 - - - -	RLA 1 4 0 0 0 C	JR e8 2 12 - - - -	ADD HL, DE 1 8 - 0 H C	LD A, [DE] 1 8 - - - -	DEC DE 1 8 - - - -	INC E 1 4 Z 0 H -	DEC E 1 4 Z 1 H -	LD E, n8 2 8 - - - -	RRA 1 4 0 0 0 C
2x					INC H 1 4 Z 0 H -	DEC H 1 4 Z 1 H -	LD H, n8 2 8 - - - -	DAA 1 4 Z - 0 C	JR Z, e8 2 12/8 - - - -	ADD HL, HL 1 8 - 0 H C	LD A, [HL+] 1 8 - - - -	DEC HL 1 8 - - - -	INC L 1 4 Z 0 H -	DEC L 1 4 Z 1 H -	LD L, n8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	2 12/8 - - - -	3 12 - - - -	1 8 - - - -	1 8 - - - -	INC [HL] 1 12 Z 0 H -	DEC [HL] 1 12 Z 1 H -	LD [HL], n8 2 12 - - - -	SCF 1 4 - 0 0 1	JR C, e8 2 12/8 - - - -	ADD HL, SP 1 8 - 0 H C	LD A, [HL-] 1 8 - - - -	DEC SP 1 8 - - - -	INC A 1 4 Z 0 H -	DEC A 1 4 Z 1 H -	LD A, n8 2 8 - - - -	CCF 1 4 - 0 0 C
4x	LD B, B 1 4 - - - -	LD B, C 1 4 - - - -	LD B, D 1 4 - - - -	LD B, E 1 4 - - - -	LD B, H 1 4 - - - -	LD B, L 1 4 - - - -	LD B, [HL] 1 8 - - - -	LD B, A 1 4 - - - -	LD C, B 1 4 - - - -	LD C, C 1 4 - - - -	LD C, D 1 4 - - - -	LD C, E 1 4 - - - -	LD C, H 1 4 - - - -	LD C, L 1 4 - - - -	LD C, [HL] 1 8 - - - -	LD C, A 1 4 - - - -
5x	LD D, B 1 4 - - - -	LD D, C 1 4 - - - -	LD D, D 1 4 - - - -	LD D, E 1 4 - - - -	LD D, H 1 4 - - - -	LD D, L 1 4 - - - -	LD D, [HL] 1 8 - - - -	LD D, A 1 4 - - - -	LD E, B 1 4 - - - -	LD E, C 1 4 - - - -	LD E, D 1 4 - - - -	LD E, E 1 4 - - - -	LD E, H 1 4 - - - -	LD E, L 1 4 - - - -	LD E, [HL] 1 8 - - - -	LD E, A 1 4 - - - -
6x	LD H, B 1 4 - - - -	LD H, C 1 4 - - - -	LD H, D 1 4 - - - -	LD H, E 1 4 - - - -	LD H, H 1 4 - - - -	LD H, L 1 4 - - - -	LD H, [HL] 1 8 - - - -	LD H, A 1 4 - - - -	LD L, B 1 4 - - - -	LD L, C 1 4 - - - -	LD L, D 1 4 - - - -	LD L, E 1 4 - - - -	LD L, H 1 4 - - - -	LD L, L 1 4 - - - -	LD L, [HL] 1 8 - - - -	LD L, A 1 4 - - - -
7x	LD [HL], B 1 8 - - - -	LD [HL], C 1 8 - - - -	LD [HL], D 1 8 - - - -	LD [HL], E 1 8 - - - -	LD [HL], H 1 8 - - - -	LD [HL], L 1 8 - - - -	HALT 1 4 - - - -	LD [HL], A 1 8 - - - -	LD A, B 1 4 - - - -	LD A, C 1 4 - - - -	LD A, D 1 4 - - - -	LD A, E 1 4 - - - -	LD A, H 1 4 - - - -	LD A, L 1 4 - - - -	LD A, [HL] 1 8 - - - -	LD A, A 1 4 - - - -
8x	ADD A, B 1 4 Z 0 H C	ADD A, C 1 4 Z 0 H C	ADD A, D 1 4 Z 0 H C	ADD A, E 1 4 Z 0 H C	ADD A, H 1 4 Z 0 H C	ADD A, L 1 4 Z 0 H C	ADD A, [HL] 1 8 Z 0 H C	ADD A, A 1 4 Z 0 H C	ADC A, B 1 4 Z 0 H C	ADC A, C 1 4 Z 0 H C	ADC A, D 1 4 Z 0 H C	ADC A, E 1 4 Z 0 H C	ADC A, H 1 4 Z 0 H C	ADC A, L 1 4 Z 0 H C	ADC A, [HL] 1 8 Z 0 H C	ADC A, A 1 4 Z 0 H C
9x	SUB A, B 1 4 Z 1 H C	SUB A, C 1 4 Z 1 H C	SUB A, D 1 4 Z 1 H C	SUB A, E 1 4 Z 1 H C	SUB A, H 1 4 Z 1 H C	SUB A, L 1 4 Z 1 H C	SUB A, [HL] 1 8 Z 1 H C	SUB A, A 1 4 1 1 0 0	SBC A, B 1 4 Z 1 H C	SBC A, C 1 4 Z 1 H C	SBC A, D 1 4 Z 1 H C	SBC A, E 1 4 Z 1 H C	SBC A, H 1 4 Z 1 H C	SBC A, L 1 4 Z 1 H C	SBC A, [HL] 1 8 Z 1 H C	SBC A, A 1 4 Z 1 H -
Ax	AND A, B 1 4 Z 0 1 0	AND A, C 1 4 Z 0 1 0	AND A, D 1 4 Z 0 1 0	AND A, E 1 4 Z 0 1 0	AND A, H 1 4 Z 0 1 0	AND A, L 1 4 Z 0 1 0	AND A, [HL] 1 8 Z 0 1 0	AND A, A 1 4 Z 0 1 0	XOR A, B 1 4 Z 0 0 0	XOR A, C 1 4 Z 0 0 0	XOR A, D 1 4 Z 0 0 0	XOR A, E 1 4 Z 0 0 0	XOR A, H 1 4 Z 0 0 0	XOR A, L 1 4 Z 0 0 0	XOR A, [HL] 1 8 Z 0 0 0	XOR A, A 1 4 1 0 0 0
Bx	OR A, B 1 4 Z 0 0 0	OR A, C 1 4 Z 0 0 0	OR A, D 1 4 Z 0 0 0	OR A, E 1 4 Z 0 0 0	OR A, H 1 4 Z 0 0 0	OR A, L 1 4 Z 0 0 0	OR A, [HL] 1 8 Z 0 0 0	OR A, A 1 4 Z 0 0 0	CP A, B 1 4 Z 1 H C	CP A, C 1 4 Z 1 H C	CP A, D 1 4 Z 1 H C	CP A, E 1 4 Z 1 H C	CP A, H 1 4 Z 1 H C	CP A, L 1 4 Z 1 H C	CP A, [HL] 1 8 Z 1 H C	CP A, A 1 4 1 1 0 0
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ, a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ, a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A, n8 2 8 Z 0 H C	RST $00 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z, a16 3 16/12 - - - -	PREFIX 1 4 - - - -	CALL Z, a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A, n8 2 8 Z 0 H C	RST $08 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC, a16 3 16/12 - - - -	—	CALL NC, a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB A, n8 2 8 Z 1 H C	RST $10 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C, a16 3 16/12 - - - -	—	CALL C, a16 3 24/12 - - - -	—	SBC A, n8 2 8 Z 1 H C	RST $18 1 16 - - - -
Ex	LDH [a8], A 2 12 - - - -	POP HL 1 12 - - - -	LD [C], A 1 8 - - - -	—	—	PUSH HL 1 16 - - - -	AND A, n8 2 8 Z 0 1 0	RST $20 1 16 - - - -	ADD SP, e8 2 16 0 0 H C	JP HL 1 4 - - - -	LD [a16], A 3 16 - - - -	—	—	—	XOR A, n8 2 8 Z 0 0 0	RST $28 1 16 - - - -
Fx	LDH A, [a8] 2 12 - - - -	POP AF 1 12 Z N H C	LD A, [C] 1 8 - - - -	DI 1 4 - - - -	—	PUSH AF 1 16 - - - -	OR A, n8 2 8 Z 0 0 0	RST $30 1 16 - - - -	LD HL, SP + e8 2 12 0 0 H C	LD SP, HL 1 8 - - - -	LD A, [a16] 3 16 - - - -	EI 1 4 - - - -	—	—	CP A, n8 2 8 Z 1 H C	RST $38 1 16 - - - -

Game Boy CPU (SM83) instruction set (JSON)

Length in bytes → INS reg ← Instruction mnemonic
2 8 ← Duration in T-states*
Z N H C ← Flags affected

*) Often instruction durations are given in "M-cycles" (machine cycles) instead of "T-states" (system clock ticks) because each instruction takes a multiple of four T-states to complete, thus a NOP takes one M-cycle or four T-states to complete.

	x0	x1	x2	x3	x4	x5	x6								
0x	NOP 1 4 - - - -	LD BC, n16 3 12 - - - -	LD [BC], A 1 8 - - - -	INC BC 1 8 - - - -	INC B 1 4 Z 0 H -	DEC B 1 4 Z 1 H -	LD B, n8 2 8 - - - -								
1x	STOP n8 2 4 - - - -	LD DE, n16 3 12 - - - -	LD [DE], A 1 8 - - - -	INC DE 1 8 - - - -	INC D 1 4 Z 0 H -	DEC D 1 4 Z 1 H -	LD D, n8 2 8 - - - -								
2x	JR NZ, e8 2 12/8 - - - -	LD HL, n16 3 12 - - - -	LD [HL+], A 1 8 - - - -	INC HL 1 8 - - - -	INC H 1 4 Z 0 H -	DEC H 1 4 Z 1 H -	LD H, n8 2 8 - - - -								
3x	JR NC, e8 2 12/8 - - - -	LD SP, n16 3 12 - - - -	LD [HL-], A 1 8 - - - -	INC SP 1 8 - - - -	INC [HL] 1 12 Z 0 H -	DEC [HL] 1 12 Z 1 H -	LD [HL], n8 2 12 - - - -								
4x	LD B, B 1 4 - - - -	LD B, C 1 4 - - - -	LD B, D 1 4 - - - -	LD B, E 1 4 - - - -	LD B, H 1 4 - - - -	LD B, L 1 4 - - - -	LD B, [HL] 1 8 - - - -	LD B, A 1 4 - - - -	LD C, B 1 4 - - - -	LD C, C 1 4 - - - -	LD C, D 1 4 - - - -	LD C, E 1 4 - - - -	LD C, H 1 4 - - - -	LD C, L 1 4 - - - -	LD C, [HL] 1 8 - - - -	LD C, A 1 4 - - - -
5x	LD D, B 1 4 - - - -	LD D, C 1 4 - - - -	LD D, D 1 4 - - - -	LD D, E 1 4 - - - -	LD D, H 1 4 - - - -	LD D, L 1 4 - - - -	LD D, [HL] 1 8 - - - -	LD D, A 1 4 - - - -	LD E, B 1 4 - - - -	LD E, C 1 4 - - - -	LD E, D 1 4 - - - -	LD E, E 1 4 - - - -	LD E, H 1 4 - - - -	LD E, L 1 4 - - - -	LD E, [HL] 1 8 - - - -	LD E, A 1 4 - - - -
6x	LD H, B 1 4 - - - -	LD H, C 1 4 - - - -	LD H, D 1 4 - - - -	LD H, E 1 4 - - - -	LD H, H 1 4 - - - -	LD H, L 1 4 - - - -	LD H, [HL] 1 8 - - - -	LD H, A 1 4 - - - -	LD L, B 1 4 - - - -	LD L, C 1 4 - - - -	LD L, D 1 4 - - - -	LD L, E 1 4 - - - -	LD L, H 1 4 - - - -	LD L, L 1 4 - - - -	LD L, [HL] 1 8 - - - -	LD L, A 1 4 - - - -
7x	LD [HL], B 1 8 - - - -	LD [HL], C 1 8 - - - -	LD [HL], D 1 8 - - - -	LD [HL], E 1 8 - - - -	LD [HL], H 1 8 - - - -	LD [HL], L 1 8 - - - -	HALT 1 4 - - - -	LD [HL], A 1 8 - - - -	LD A, B 1 4 - - - -	LD A, C 1 4 - - - -	LD A, D 1 4 - - - -	LD A, E 1 4 - - - -	LD A, H 1 4 - - - -	LD A, L 1 4 - - - -	LD A, [HL] 1 8 - - - -	LD A, A 1 4 - - - -
8x	ADD A, B 1 4 Z 0 H C	ADD A, C 1 4 Z 0 H C	ADD A, D 1 4 Z 0 H C	ADD A, E 1 4 Z 0 H C	ADD A, H 1 4 Z 0 H C	ADD A, L 1 4 Z 0 H C	ADD A, [HL] 1 8 Z 0 H C	ADD A, A 1 4 Z 0 H C	ADC A, B 1 4 Z 0 H C	ADC A, C 1 4 Z 0 H C	ADC A, D 1 4 Z 0 H C	ADC A, E 1 4 Z 0 H C	ADC A, H 1 4 Z 0 H C	ADC A, L 1 4 Z 0 H C	ADC A, [HL] 1 8 Z 0 H C	ADC A, A 1 4 Z 0 H C
9x	SUB A, B 1 4 Z 1 H C	SUB A, C 1 4 Z 1 H C	SUB A, D 1 4 Z 1 H C	SUB A, E 1 4 Z 1 H C	SUB A, H 1 4 Z 1 H C	SUB A, L 1 4 Z 1 H C	SUB A, [HL] 1 8 Z 1 H C	SUB A, A 1 4 1 1 0 0	SBC A, B 1 4 Z 1 H C	SBC A, C 1 4 Z 1 H C	SBC A, D 1 4 Z 1 H C	SBC A, E 1 4 Z 1 H C	SBC A, H 1 4 Z 1 H C	SBC A, L 1 4 Z 1 H C	SBC A, [HL] 1 8 Z 1 H C	SBC A, A 1 4 Z 1 H -
Ax	AND A, B 1 4 Z 0 1 0	AND A, C 1 4 Z 0 1 0	AND A, D 1 4 Z 0 1 0	AND A, E 1 4 Z 0 1 0	AND A, H 1 4 Z 0 1 0	AND A, L 1 4 Z 0 1 0	AND A, [HL] 1 8 Z 0 1 0	AND A, A 1 4 Z 0 1 0	XOR A, B 1 4 Z 0 0 0	XOR A, C 1 4 Z 0 0 0	XOR A, D 1 4 Z 0 0 0	XOR A, E 1 4 Z 0 0 0	XOR A, H 1 4 Z 0 0 0	XOR A, L 1 4 Z 0 0 0	XOR A, [HL] 1 8 Z 0 0 0	XOR A, A 1 4 1 0 0 0
Bx	OR A, B 1 4 Z 0 0 0	OR A, C 1 4 Z 0 0 0	OR A, D 1 4 Z 0 0 0	OR A, E 1 4 Z 0 0 0	OR A, H 1 4 Z 0 0 0	OR A, L 1 4 Z 0 0 0	OR A, [HL] 1 8 Z 0 0 0	OR A, A 1 4 Z 0 0 0	CP A, B 1 4 Z 1 H C	CP A, C 1 4 Z 1 H C	CP A, D 1 4 Z 1 H C	CP A, E 1 4 Z 1 H C	CP A, H 1 4 Z 1 H C	CP A, L 1 4 Z 1 H C	CP A, [HL] 1 8 Z 1 H C	CP A, A 1 4 1 1 0 0
Cx	RET NZ 1 20/8 - - - -	POP BC 1 12 - - - -	JP NZ, a16 3 16/12 - - - -	JP a16 3 16 - - - -	CALL NZ, a16 3 24/12 - - - -	PUSH BC 1 16 - - - -	ADD A, n8 2 8 Z 0 H C	RST $00 1 16 - - - -	RET Z 1 20/8 - - - -	RET 1 16 - - - -	JP Z, a16 3 16/12 - - - -	PREFIX 1 4 - - - -	CALL Z, a16 3 24/12 - - - -	CALL a16 3 24 - - - -	ADC A, n8 2 8 Z 0 H C	RST $08 1 16 - - - -
Dx	RET NC 1 20/8 - - - -	POP DE 1 12 - - - -	JP NC, a16 3 16/12 - - - -	—	CALL NC, a16 3 24/12 - - - -	PUSH DE 1 16 - - - -	SUB A, n8 2 8 Z 1 H C	RST $10 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C, a16 3 16/12 - - - -	—	CALL C, a16 3 24/12 - - - -	—	SBC A, n8 2 8 Z 1 H C	RST $18 1 16 - - - -
Ex	LDH [a8], A 2 12 - - - -	POP HL 1 12 - - - -	LD [C], A 1 8 - - - -	—	—	PUSH HL 1 16 - - - -	AND A, n8 2 8 Z 0 1 0	RST $20 1 16 - - - -	ADD SP, e8 2 16 0 0 H C	JP HL 1 4 - - - -	LD [a16], A 3 16 - - - -	—	—	—	XOR A, n8 2 8 Z 0 0 0	RST $28 1 16 - - - -
Fx	LDH A, [a8] 2 12 - - - -	POP AF 1 12 Z N H C	LD A, [C] 1 8 - - - -	DI 1 4 - - - -	—	PUSH AF 1 16 - - - -	OR A, n8 2 8 Z 0 0 0	RST $30 1 16 - - - -	LD HL, SP + e8 2 12 0 0 H C	LD SP, HL 1 8 - - - -	LD A, [a16] 3 16 - - - -	EI 1 4 - - - -	—	—	CP A, n8 2 8 Z 1 H C	RST $38 1 16 - - - -

T-Cycle vs M-Cycle

- **T-Cycle**: Tick Cycle (also called Machine Cycle in some contexts). A T-cycle represents the smallest unit of time in the Game Boy's CPU, typically corresponding to one clock tick of the CPU. Emulators with T-cycle accuracy simulate the exact number of clock ticks for every instruction, providing the highest level of timing precision.

- **M-Cycle**: Memory Cycle. An M-cycle represents a higher-level unit of time used by the Game Boy's CPU for executing instructions. Each instruction takes a specific number of M-cycles, with each M-cycle typically equating to 4 T-cycles.

T-Cycle vs M-Cycle

- **T-Cycle**: Tick Cycle (also called Machine Cycle in som
  A T-cycle represents the smallest unit of time in the Ga
  CPU, typically corresponding to one clock tick of the C
  Emulators with T-cycle accuracy simulate the exact number of clock ticks for every instruction, providing the highest level of timing precision.

- **M-Cycle**: Memory Cycle. An M-cycle represents a higher-level unit of time used by the Game Boy's CPU for executing instructions. Each instruction takes a specific number of M-cycles, with each M-cycle typically equating to 4 T-cycles.

Prefixed ($CB $xx)

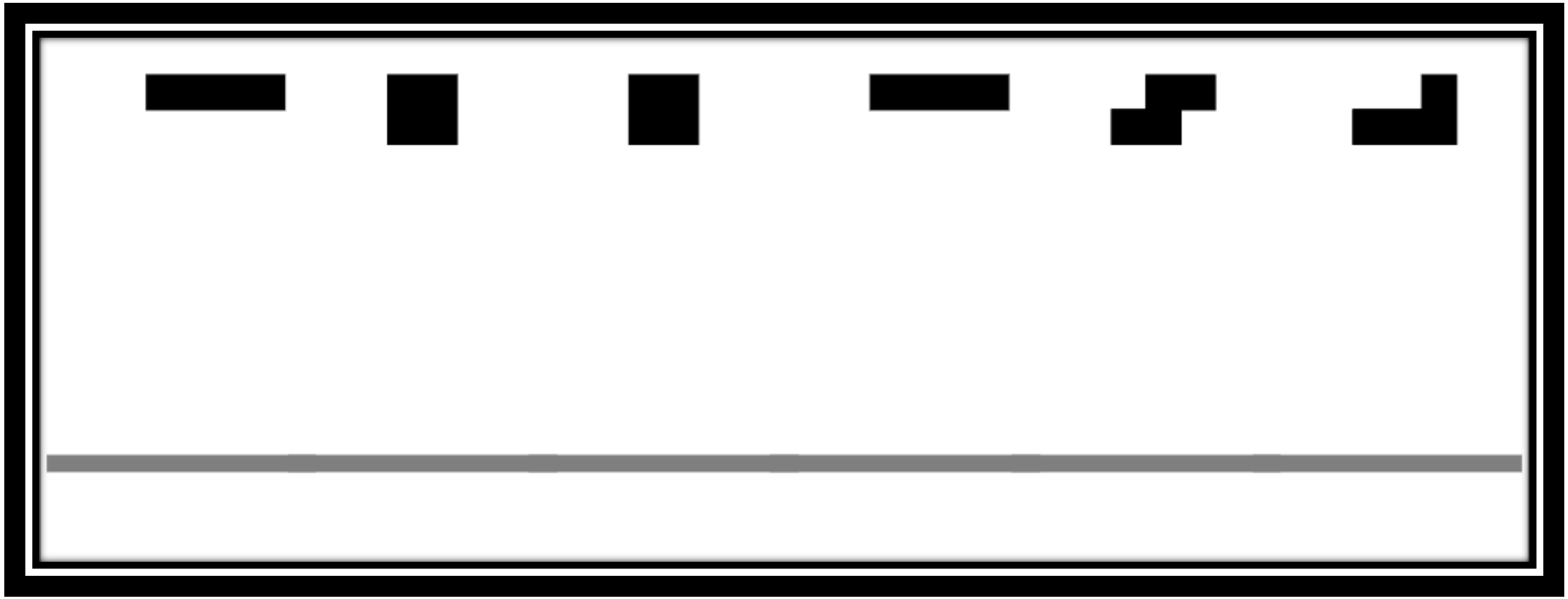
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 8 Z 0 0 C	RLC C 2 8 Z 0 0 C	RLC D 2 8 Z 0 0 C	RLC E 2 8 Z 0 0 C	RLC H 2 8 Z 0 0 C	RLC L 2 8 Z 0 0 C	RLC [HL] 2 16 Z 0 0 C	RLC A 2 8 Z 0 0 C	RRC B 2 8 Z 0 0 C	RRC C 2 8 Z 0 0 C	RRC D 2 8 Z 0 0 C	RRC E 2 8 Z 0 0 C	RRC H 2 8 Z 0 0 C	RRC L 2 8 Z 0 0 C	RRC [HL] 2 16 Z 0 0 C	RRC A 2 8 Z 0 0 C
1x	RL B 2 8 Z 0 0 C	RL C 2 8 Z 0 0 C	RL D 2 8 Z 0 0 C	RL E 2 8 Z 0 0 C	RL H 2 8 Z 0 0 C	RL L 2 8 Z 0 0 C	RL [HL] 2 16 Z 0 0 C	RL A 2 8 Z 0 0 C	RR B 2 8 Z 0 0 C	RR C 2 8 Z 0 0 C	RR D 2 8 Z 0 0 C	RR E 2 8 Z 0 0 C	RR H 2 8 Z 0 0 C	RR L 2 8 Z 0 0 C	RR [HL] 2 16 Z 0 0 C	RR A 2 8 Z 0 0 C
2x	SLA B 2 8 Z 0 0 C	SLA C 2 8 Z 0 0 C	SLA D 2 8 Z 0 0 C	SLA E 2 8 Z 0 0 C	SLA H 2 8 Z 0 0 C	SLA L 2 8 Z 0 0 C	SLA [HL] 2 16 Z 0 0 C	SLA A 2 8 Z 0 0 C	SRA B 2 8 Z 0 0 C	SRA C 2 8 Z 0 0 C	SRA D 2 8 Z 0 0 C	SRA E 2 8 Z 0 0 C	SRA H 2 8 Z 0 0 C	SRA L 2 8 Z 0 0 C	SRA [HL] 2 16 Z 0 0 C	SRA A 2 8 Z 0 0 C
3x	SWAP B 2 8 Z 0 0 0	SWAP C 2 8 Z 0 0 0	SWAP D 2 8 Z 0 0 0	SWAP E 2 8 Z 0 0 0	SWAP H 2 8 Z 0 0 0	SWAP L 2 8 Z 0 0 0	SWAP [HL] 2 16 Z 0 0 0	SWAP A 2 8 Z 0 0 0	SRL B 2 8 Z 0 0 C	SRL C 2 8 Z 0 0 C	SRL D 2 8 Z 0 0 C	SRL E 2 8 Z 0 0 C	SRL H 2 8 Z 0 0 C	SRL L 2 8 Z 0 0 C	SRL [HL] 2 16 Z 0 0 C	SRL A 2 8 Z 0 0 C
4x	BIT 0, B 2 8 Z 0 1 -	BIT 0, C 2 8 Z 0 1 -	BIT 0, D 2 8 Z 0 1 -	BIT 0, E 2 8 Z 0 1 -	BIT 0, H 2 8 Z 0 1 -	BIT 0, L 2 8 Z 0 1 -	BIT 0, [HL] 2 12 Z 0 1 -	BIT 0, A 2 8 Z 0 1 -	BIT 1, B 2 8 Z 0 1 -	BIT 1, C 2 8 Z 0 1 -	BIT 1, D 2 8 Z 0 1 -	BIT 1, E 2 8 Z 0 1 -	BIT 1, H 2 8 Z 0 1 -	BIT 1, L 2 8 Z 0 1 -	BIT 1, [HL] 2 12 Z 0 1 -	BIT 1, A 2 8 Z 0 1 -
5x	BIT 2, B 2 8 Z 0 1 -	BIT 2, C 2 8 Z 0 1 -	BIT 2, D 2 8 Z 0 1 -	BIT 2, E 2 8 Z 0 1 -	BIT 2, H 2 8 Z 0 1 -	BIT 2, L 2 8 Z 0 1 -	BIT 2, [HL] 2 12 Z 0 1 -	BIT 2, A 2 8 Z 0 1 -	BIT 3, B 2 8 Z 0 1 -	BIT 3, C 2 8 Z 0 1 -	BIT 3, D 2 8 Z 0 1 -	BIT 3, E 2 8 Z 0 1 -	BIT 3, H 2 8 Z 0 1 -	BIT 3, L 2 8 Z 0 1 -	BIT 3, [HL] 2 12 Z 0 1 -	BIT 3, A 2 8 Z 0 1 -
6x	BIT 4, B 2 8 Z 0 1 -	BIT 4, C 2 8 Z 0 1 -	BIT 4, D 2 8 Z 0 1 -	BIT 4, E 2 8 Z 0 1 -	BIT 4, H 2 8 Z 0 1 -	BIT 4, L 2 8 Z 0 1 -	BIT 4, [HL] 2 12 Z 0 1 -	BIT 4, A 2 8 Z 0 1 -	BIT 5, B 2 8 Z 0 1 -	BIT 5, C 2 8 Z 0 1 -	BIT 5, D 2 8 Z 0 1 -	BIT 5, E 2 8 Z 0 1 -	BIT 5, H 2 8 Z 0 1 -	BIT 5, L 2 8 Z 0 1 -	BIT 5, [HL] 2 12 Z 0 1 -	BIT 5, A 2 8 Z 0 1 -
7x	BIT 6, B 2 8 Z 0 1 -	BIT 6, C 2 8 Z 0 1 -	BIT 6, D 2 8 Z 0 1 -	BIT 6, E 2 8 Z 0 1 -	BIT 6, H 2 8 Z 0 1 -	BIT 6, L 2 8 Z 0 1 -	BIT 6, [HL] 2 12 Z 0 1 -	BIT 6, A 2 8 Z 0 1 -	BIT 7, B 2 8 Z 0 1 -	BIT 7, C 2 8 Z 0 1 -	BIT 7, D 2 8 Z 0 1 -	BIT 7, E 2 8 Z 0 1 -	BIT 7, H 2 8 Z 0 1 -	BIT 7, L 2 8 Z 0 1 -	BIT 7, [HL] 2 12 Z 0 1 -	BIT 7, A 2 8 Z 0 1 -
8x	RES 0, B 2 8 - - - -	RES 0, C 2 8 - - - -	RES 0, D 2 8 - - - -	RES 0, E 2 8 - - - -	RES 0, H 2 8 - - - -	RES 0, L 2 8 - - - -	RES 0, [HL] 2 16 - - - -	RES 0, A 2 8 - - - -	RES 1, B 2 8 - - - -	RES 1, C 2 8 - - - -	RES 1, D 2 8 - - - -	RES 1, E 2 8 - - - -	RES 1, H 2 8 - - - -	RES 1, L 2 8 - - - -	RES 1, [HL] 2 16 - - - -	RES 1, A 2 8 - - - -
9x	RES 2, B 2 8 - - - -	RES 2, C 2 8 - - - -	RES 2, D 2 8 - - - -	RES 2, E 2 8 - - - -	RES 2, H 2 8 - - - -	RES 2, L 2 8 - - - -	RES 2, [HL] 2 16 - - - -	RES 2, A 2 8 - - - -	RES 3, B 2 8 - - - -	RES 3, C 2 8 - - - -	RES 3, D 2 8 - - - -	RES 3, E 2 8 - - - -	RES 3, H 2 8 - - - -	RES 3, L 2 8 - - - -	RES 3, [HL] 2 16 - - - -	RES 3, A 2 8 - - - -
Ax	RES 4, B 2 8 - - - -	RES 4, C 2 8 - - - -	RES 4, D 2 8 - - - -	RES 4, E 2 8 - - - -	RES 4, H 2 8 - - - -	RES 4, L 2 8 - - - -	RES 4, [HL] 2 16 - - - -	RES 4, A 2 8 - - - -	RES 5, B 2 8 - - - -	RES 5, C 2 8 - - - -	RES 5, D 2 8 - - - -	RES 5, E 2 8 - - - -	RES 5, H 2 8 - - - -	RES 5, L 2 8 - - - -	RES 5, [HL] 2 16 - - - -	RES 5, A 2 8 - - - -
Bx	RES 6, B 2 8 - - - -	RES 6, C 2 8 - - - -	RES 6, D 2 8 - - - -	RES 6, E 2 8 - - - -	RES 6, H 2 8 - - - -	RES 6, L 2 8 - - - -	RES 6, [HL] 2 16 - - - -	RES 6, A 2 8 - - - -	RES 7, B 2 8 - - - -	RES 7, C 2 8 - - - -	RES 7, D 2 8 - - - -	RES 7, E 2 8 - - - -	RES 7, H 2 8 - - - -	RES 7, L 2 8 - - - -	RES 7, [HL] 2 16 - - - -	RES 7, A 2 8 - - - -
Cx	SET 0, B 2 8 - - - -	SET 0, C 2 8 - - - -	SET 0, D 2 8 - - - -	SET 0, E 2 8 - - - -	SET 0, H 2 8 - - - -	SET 0, L 2 8 - - - -	SET 0, [HL] 2 16 - - - -	SET 0, A 2 8 - - - -	SET 1, B 2 8 - - - -	SET 1, C 2 8 - - - -	SET 1, D 2 8 - - - -	SET 1, E 2 8 - - - -	SET 1, H 2 8 - - - -	SET 1, L 2 8 - - - -	SET 1, [HL] 2 16 - - - -	SET 1, A 2 8 - - - -
Dx	SET 2, B 2 8 - - - -	SET 2, C 2 8 - - - -	SET 2, D 2 8 - - - -	SET 2, E 2 8 - - - -	SET 2, H 2 8 - - - -	SET 2, L 2 8 - - - -	SET 2, [HL] 2 16 - - - -	SET 2, A 2 8 - - - -	SET 3, B 2 8 - - - -	SET 3, C 2 8 - - - -	SET 3, D 2 8 - - - -	SET 3, E 2 8 - - - -	SET 3, H 2 8 - - - -	SET 3, L 2 8 - - - -	SET 3, [HL] 2 16 - - - -	SET 3, A 2 8 - - - -
Ex	SET 4, B 2 8 - - - -	SET 4, C 2 8 - - - -	SET 4, D 2 8 - - - -	SET 4, E 2 8 - - - -	SET 4, H 2 8 - - - -	SET 4, L 2 8 - - - -	SET 4, [HL] 2 16 - - - -	SET 4, A 2 8 - - - -	SET 5, B 2 8 - - - -	SET 5, C 2 8 - - - -	SET 5, D 2 8 - - - -	SET 5, E 2 8 - - - -	SET 5, H 2 8 - - - -	SET 5, L 2 8 - - - -	SET 5, [HL] 2 16 - - - -	SET 5, A 2 8 - - - -
Fx	SET 6, B 2 8 - - - -	SET 6, C 2 8 - - - -	SET 6, D 2 8 - - - -	SET 6, E 2 8 - - - -	SET 6, H 2 8 - - - -	SET 6, L 2 8 - - - -	SET 6, [HL] 2 16 - - - -	SET 6, A 2 8 - - - -	SET 7, B 2 8 - - - -	SET 7, C 2 8 - - - -	SET 7, D 2 8 - - - -	SET 7, E 2 8 - - - -	SET 7, H 2 8 - - - -	SET 7, L 2 8 - - - -	SET 7, [HL] 2 16 - - - -	SET 7, A 2 8 - - - -

A toy example…

Let’s pick a simple game and just implement that…

But which one?

What game should we pick?



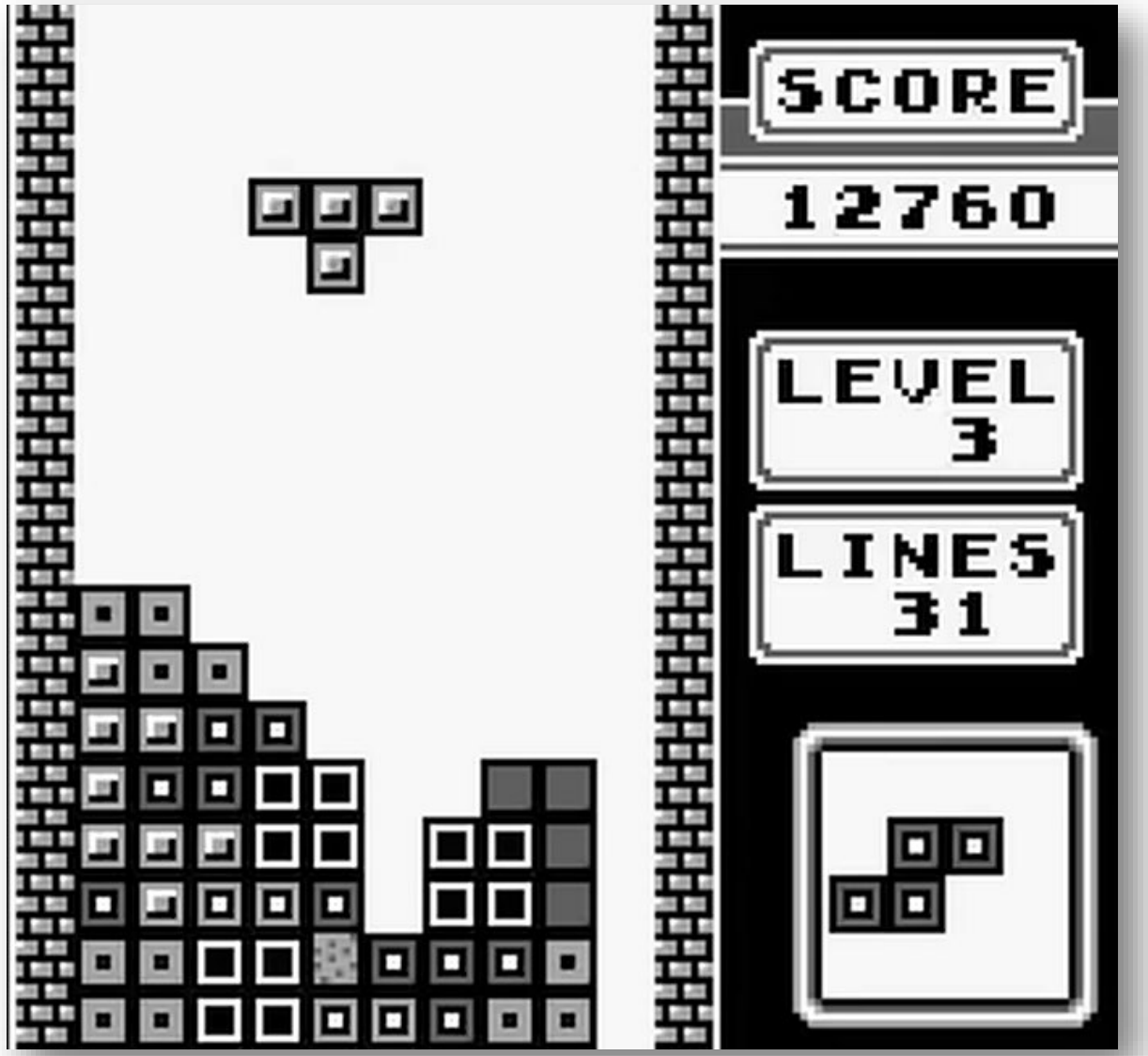
Why choose Tetris?

Besides the fact that it was a launch title and bundled with the original GameBoy?

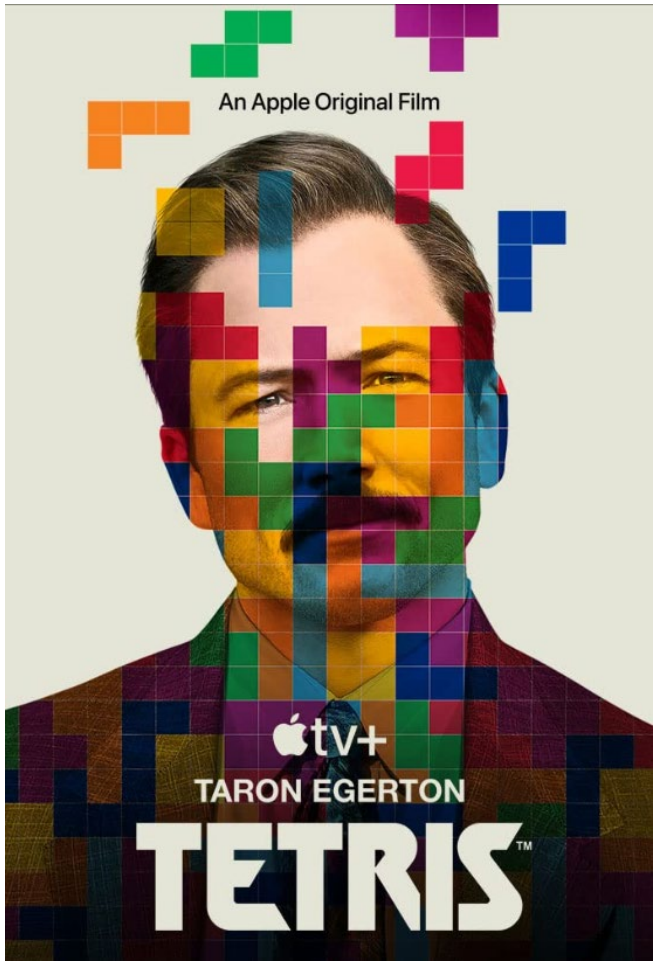
And the fact that I own several copies?

Tetris?

A falling-blocks puzzle **video game** created by Soviet game designer Alexey Pajitnov in **1984**. Tetris variants were later commercialized and released on a vast spectrum of platforms, from video game consoles and computers to mobile devices and calculators, with the version bundled with the Game Boy selling over 35 million units, while various mobile games had seen over 425 million paid downloads by 2014. It is the most successful video game franchise to originate from Russia and the former Soviet Union, the best-selling puzzle video game series and the best-selling video game franchise not owned by Nintendo.



Tetris is still hot!



Tetris Included in Launch Lineup of Game Boy Nintendo Switch Online

Posted February 8, 2023 2:00 PM, Nintendo
In #Feature

Game Boy Games are being added to Nintendo Switch Online. Launch lineup includes Tetris.

Game Boy and Game Boy Advance Games: Select Game Boy games are being added to Nintendo Switch Online! On Nintendo Switch, you can play these games anytime, anywhere – just like you could back in the day. You can use Game Boy, Game Boy Pocket or Game Boy Color screen filters to help customize your play style. And, up to two players can play compatible games together locally or online for the first time. Here's the full list of classic games that will be available at launch, with more games being added in the future:

Launch Lineup of Game Boy – Nintendo Switch Online

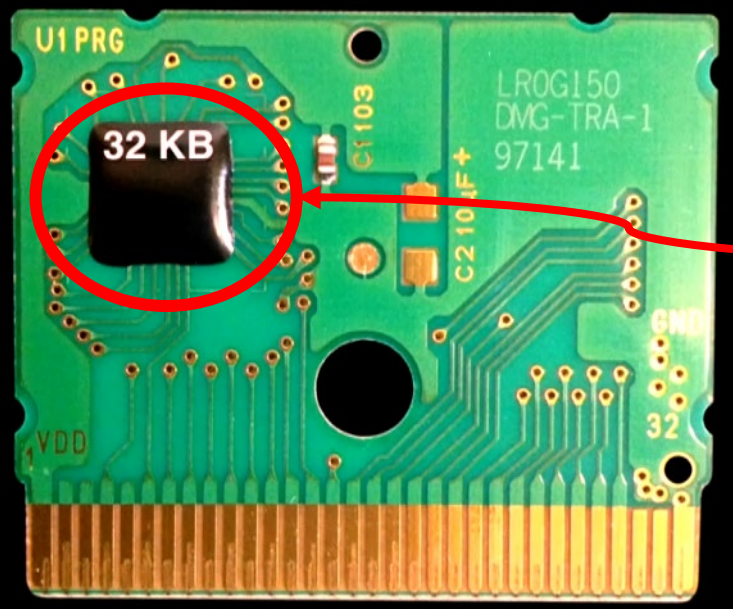
- *Tetris®*
- *Super Mario Land 2 – 6 Golden Coins*
- *The Legend of Zelda: Link's Awakening DX*
- *GARGOYLE'S QUEST*
- *Game & Watch Gallery 3*
- *Alone in the Dark: The New Nightmare*
- *Metroid II – Return of Samus*
- *Wario Land 3*
- *Kirby's Dream Land*

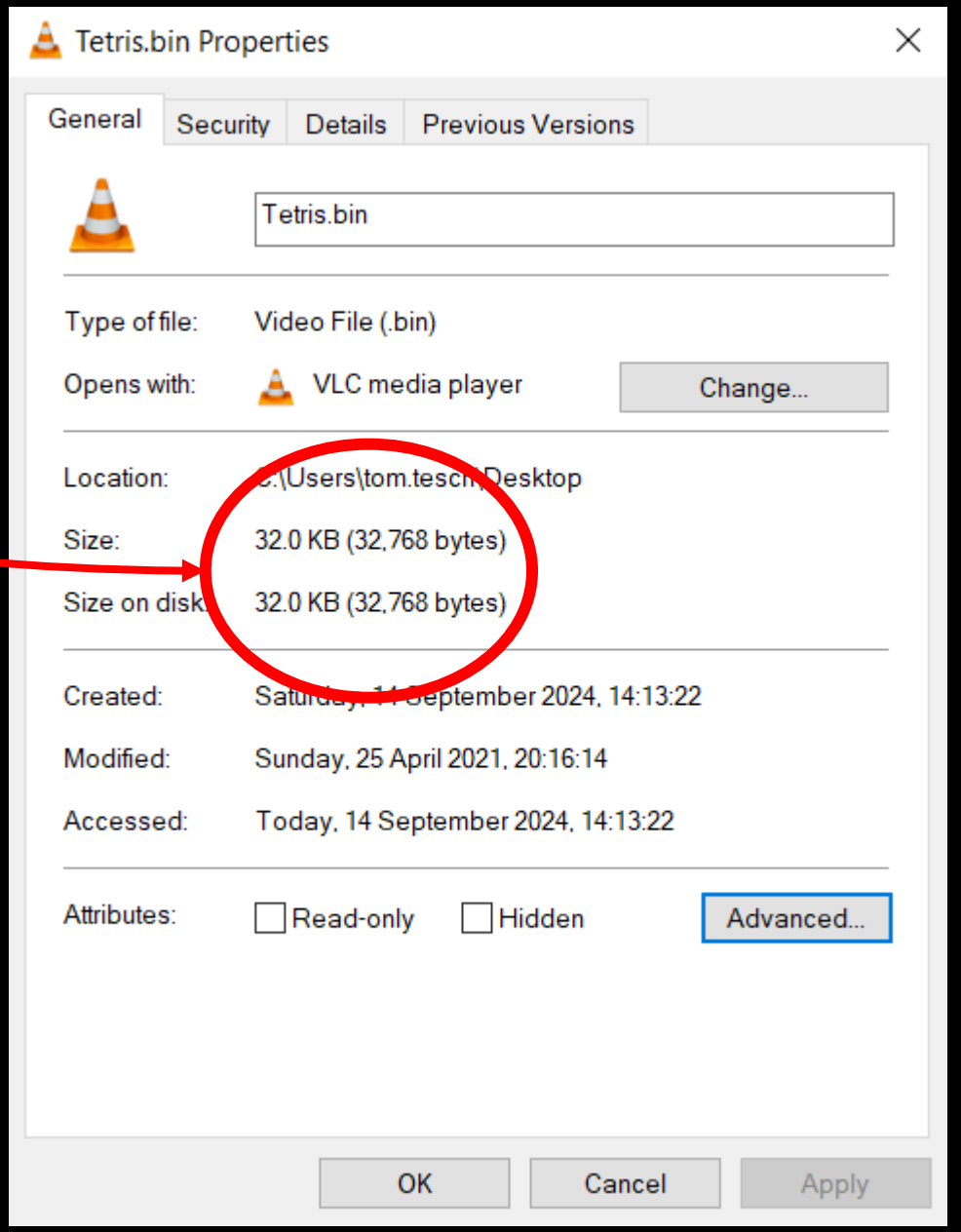
But really: why tetris?

- M-cycle precision emulation is sufficient
- Only 284 of the 501 opcodes are used (and need to be implemented)

- Limited size means no memory banking – no MBC needed

No MBC needed!





Possible interesting extension

Possible interesting extension

As soon as networking is added to the C++ standard :-P

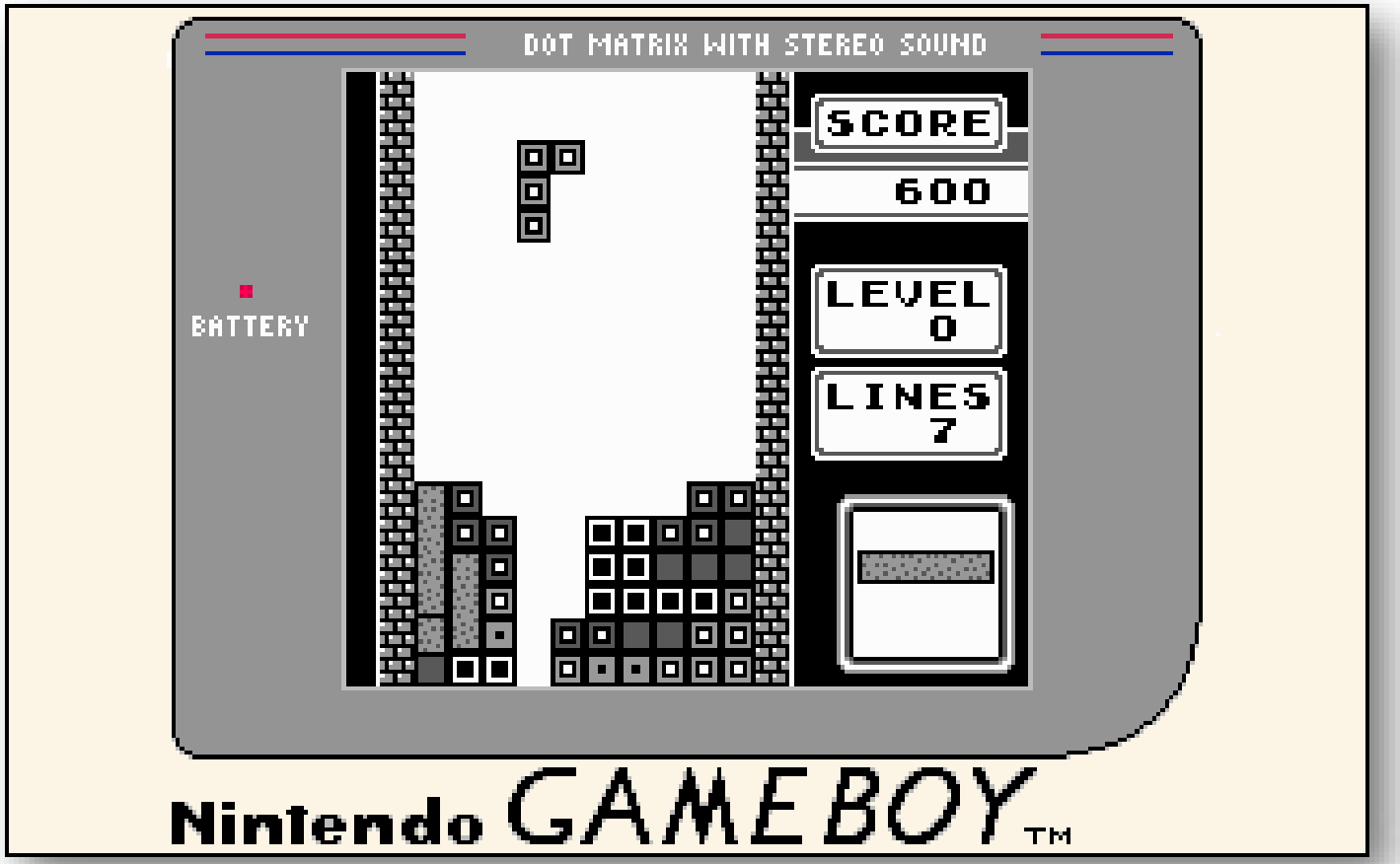
Why does it have to be fast?

- Bragging rights?
- To run on low end hardware
- To use for machine learning
- TEGLON

TEGLON?

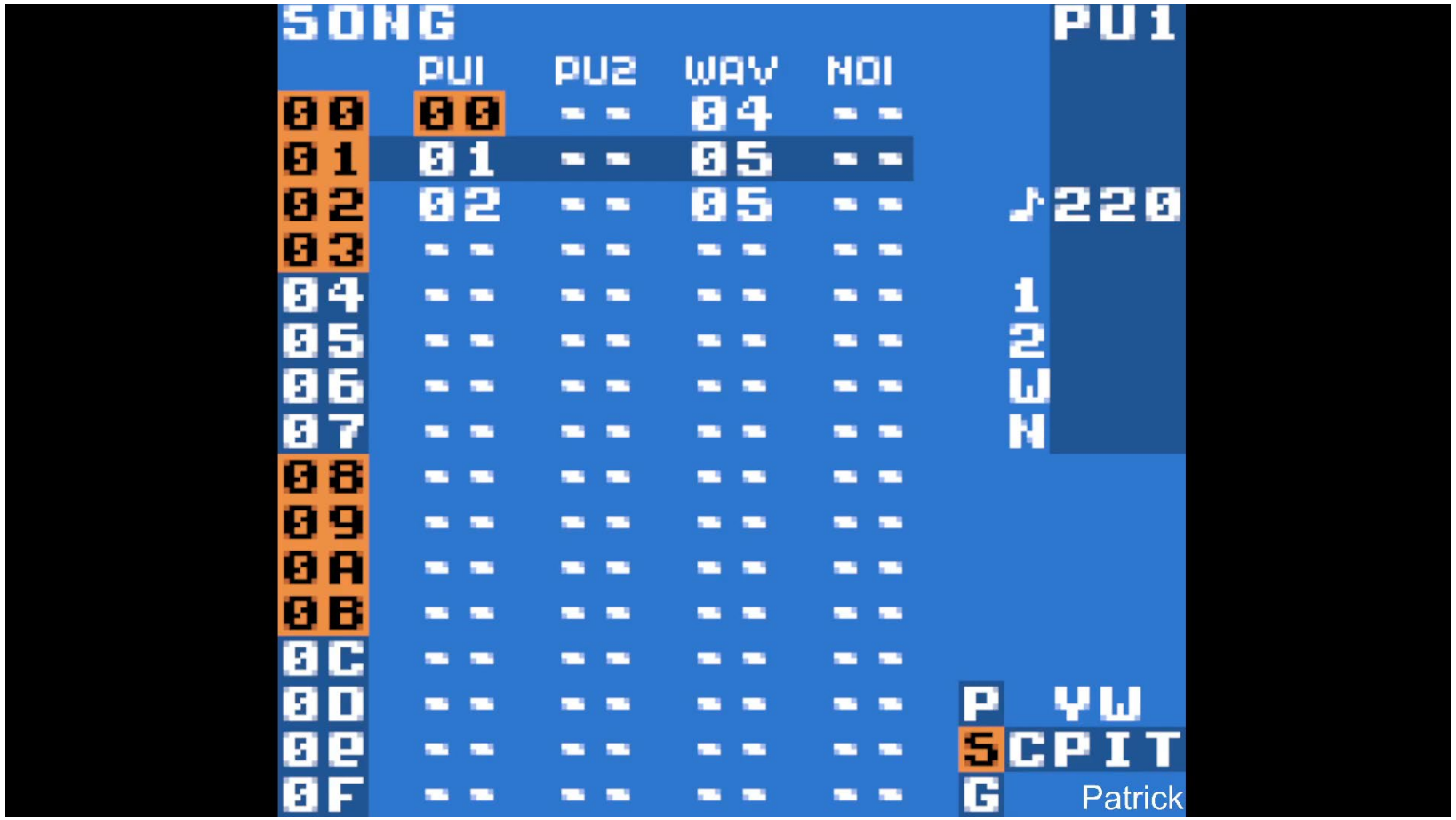
- **T**etris for
- **G**ameboy
- **LO**gic
- i**N**nversion

Where does the term originate from?



Why no sound?

Little Sound DJ



The Legend of Zelda: Link's Awakening

olcPixelGameEngine?

https://github.com/OneLoneCoder/olcPixelGameEngine

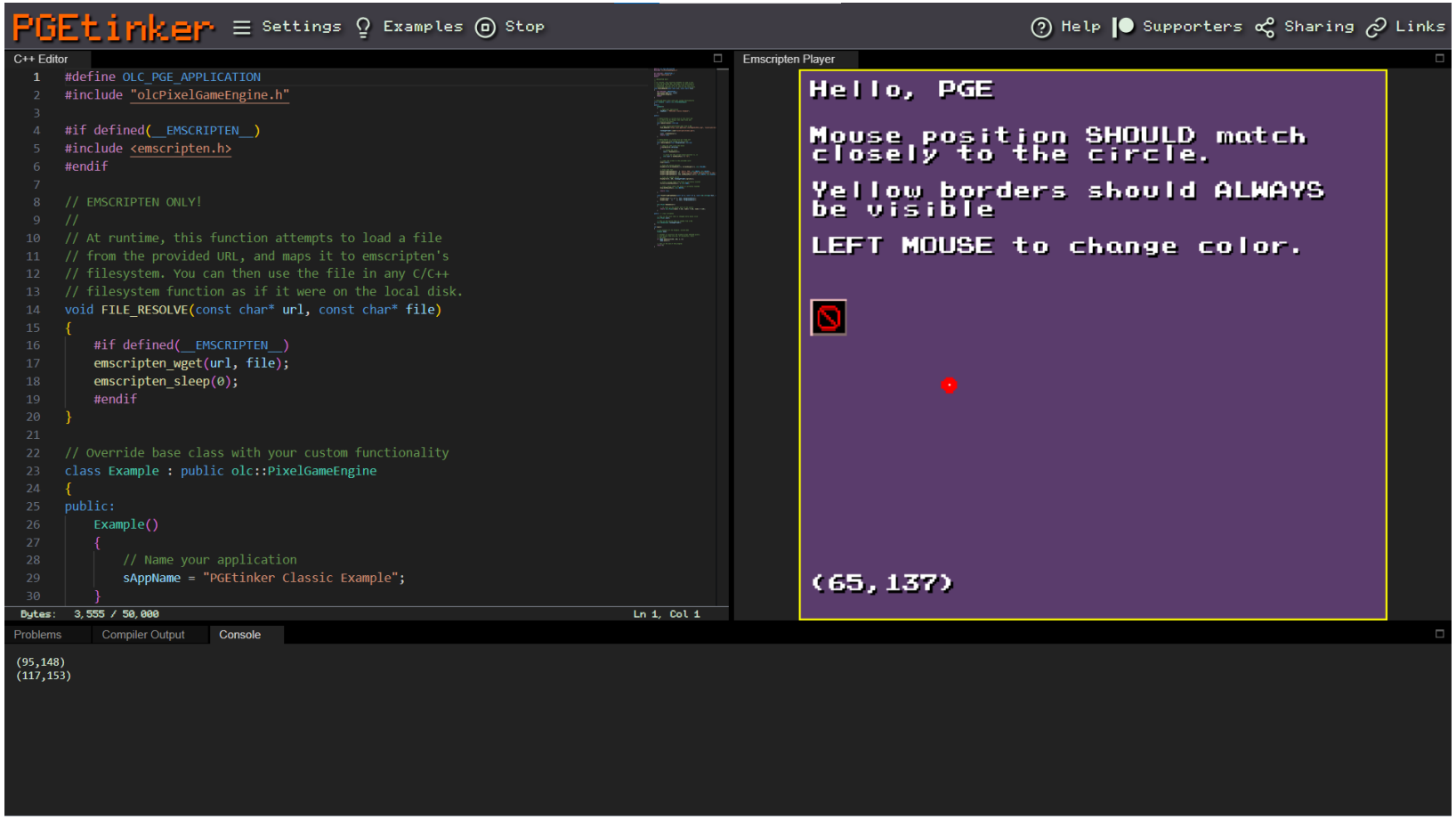
The official distribution of olcPixelGameEngine, a tool used in javidx9's YouTube videos and projects (https://www.youtube.com/@javidx9)

Single header include(!!!)

Compiles to windows, mac, linux and webassembly

Online playground

https://pgetinker.com/



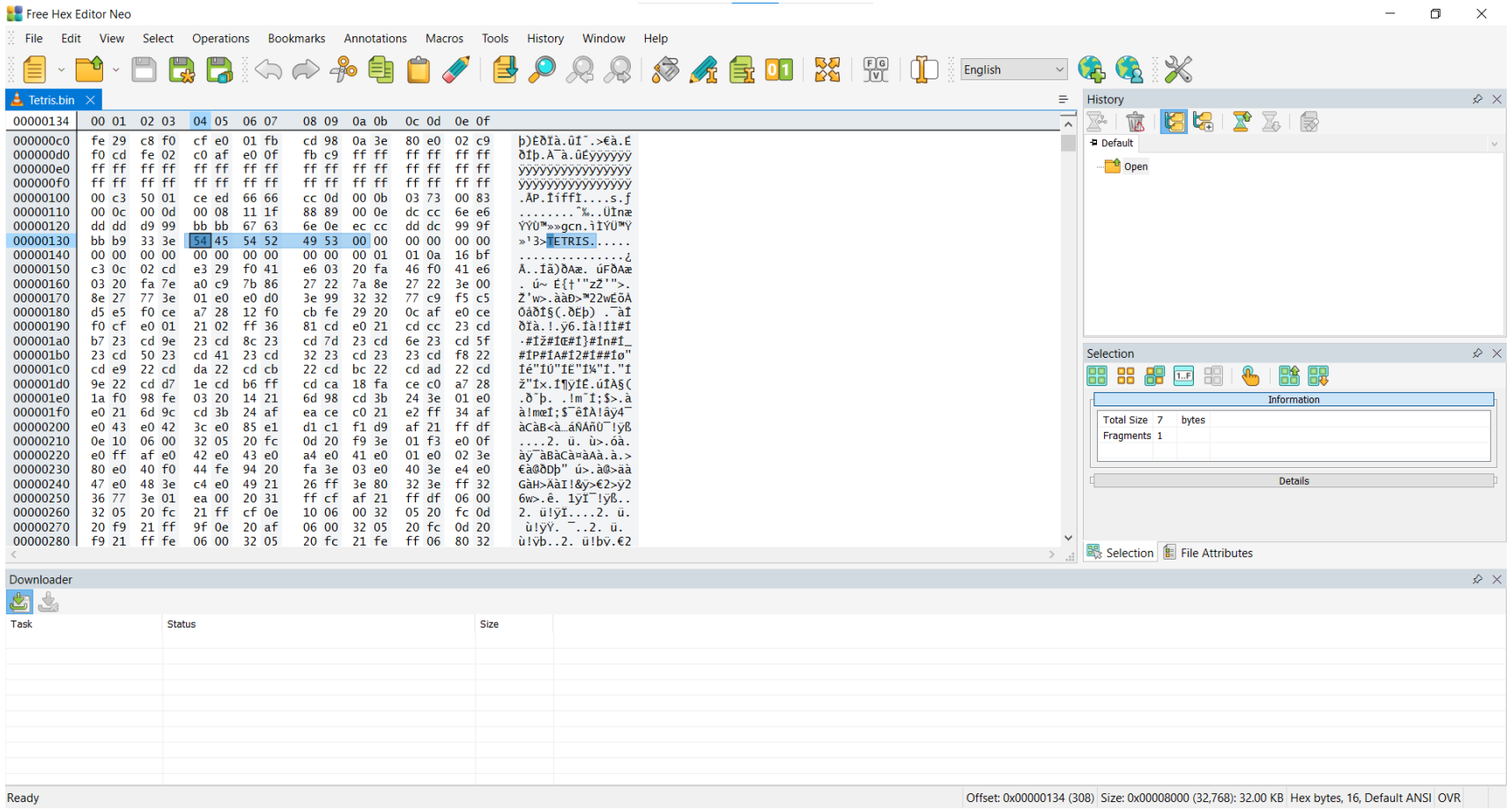
An empty application

```
1   #define OLC_PGE_APPLICATION
2   #include "olcPixelGameEngine.h"
3
4   class Example : public olc::PixelGameEngine {
5    public:
6     Example() { sAppName = "Bare PGE"; }
7
8    public:
9     bool OnUserCreate() override { return true; }
10
11    bool OnUserUpdate(float fElapsedTime) override { return true; }
12  };
13
14  int main() {
15    Example demo;
16    if (demo.Construct(256, 240, 2, 2)) demo.Start();
17    return 0;
18  }
```

Minimal API

```
1   #ifndef PG_EBOY_H
2   #define PG_EBOY_H
3
4   class Emulator {
5    private:
6     const char* romName;
7
8    public:
9     Emulator(const char* romFile);
10    void Start();
11    const char* getName() const;
12    void getDisplay(uint8_t* screen, const bool* buttons, int frames);
13    static constexpr size_t nButtons = 8;
14    static constexpr int xSizeScreen = 160;
15    static constexpr int ySizeScreen = 144;
16  };
17
18  #endif
19
```

FYI: The name is stored in the ROM



```cpp
#include "PGEboy.h"

#define OLC_PGE_APPLICATION
#include "olcPixelGameEngine.h"

Emulator emu{"Tetris.bin"};

class Example : public olc::PixelGameEngine {
 public:
  Example() { sAppName = emu.getName(); }

 public:
  bool OnUserCreate() override {
    emu.Start();
    return true;
  }
```

```cpp
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PWSTR pCmdLine, int nCmdShow) {
  Example demo;
  if (demo.Construct(Emulator::xSizeScreen, Emulator::ySizeScreen, 4, 4, false, true))
    demo.Start();

  return 0;
}
```

```cpp
#include "PGEboy.h"

#define OLC_PGE_APPLICATION
#include "olcPixelGameEngine.h"

Emulator emu{"Tetris.bin"};

class Example : public olc::PixelGameEngine {
public:
  Example() { sAppName = emu.getName(); }

public:
  bool OnUserCreate() override {
    emu.Start();
    return true;
  }
```

```cpp
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE
                    PWSTR pCmdLine, int nCmdShow) {
  Example demo;
  if (demo.Construct(Emulator::xSizeScreen, Emulato
    demo.Start();

  return 0;
}
```

```cpp
bool OnUserUpdate(float fElapsedTime) override {
  bool buttons[Emulator::nButtons]{};

  buttons[0] = GetKey(olc::Key::UP).bHeld;
  buttons[1] = GetKey(olc::Key::DOWN).bHeld;
  buttons[2] = GetKey(olc::Key::LEFT).bHeld;
  buttons[3] = GetKey(olc::Key::RIGHT).bHeld;
  buttons[4] = GetKey(olc::Key::ENTER).bHeld;
  buttons[5] = GetKey(olc::Key::SPACE).bHeld;
  buttons[6] = GetKey(olc::Key::A).bHeld;
  buttons[7] = GetKey(olc::Key::D).bHeld;

  static float totalElapsedTime{};
  totalElapsedTime += fElapsedTime;
  if (totalElapsedTime > 0.0597f) {
    totalElapsedTime -= 0.0597f;
    uint8_t screen[Emulator::xSizeScreen * Emulator::ySizeScreen];
    emu.getDisplay(screen, buttons, 1);
    for (int x = 0; x < ScreenWidth(); x++)
      for (int y = 0; y < ScreenHeight(); y++) {
        auto index = (y * ScreenWidth() + x);
        olc::Pixel color{olc::Pixel(15, 56, 15)};  // darkest green
        if (screen[index] == 1) {
          color = olc::Pixel(155, 188, 15);  // lightest green
        } else if (screen[index] == 2) {
          color = olc::Pixel(139, 172, 15);  // light green
        } else if (screen[index] == 3) {
          color = olc::Pixel(48, 98, 48);  // dark green
        }
        Draw(x, y, color);
      }
  }

  return true;
```

Darkest Green
Hex: #0f380f
RGB: 15, 56, 15

Dark Green
Hex: #306230
RGB: 48, 98, 48

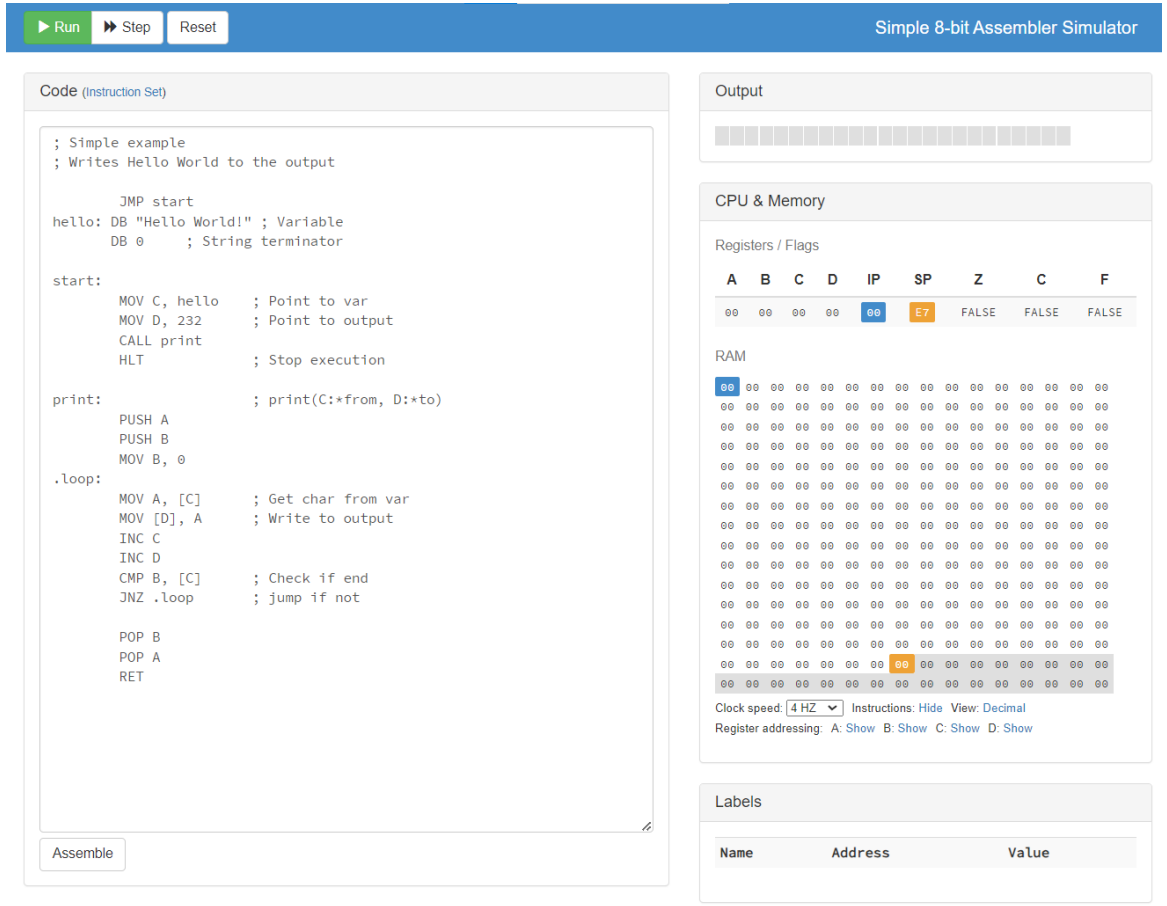
Light Green
Hex: #8bac0f
RGB: 139, 172, 15

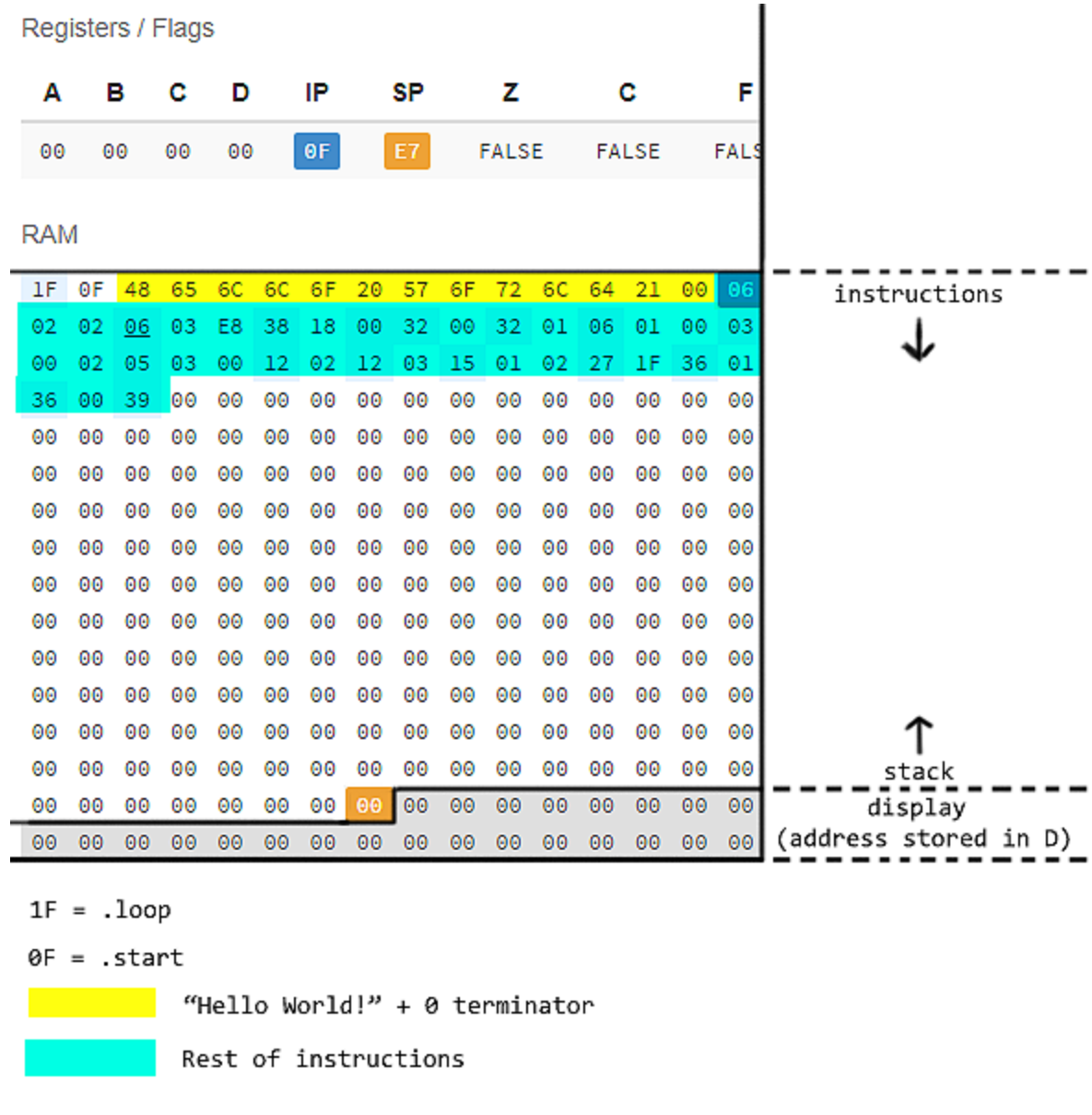
Lightest Green
Hex: #9bbc0f
RGB: 155, 188, 15

The reality is...

What about us...

https://schweigi.github.io/assembler-simulator/




Registers / Flags
A B C D IP SP Z C F
00 00 00 00 0F E7 FALSE FALSE FALS
RAM
1F 0F 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00 06
02 02 06 03 E8 38 18 00 32 00 32 01 06 01 00 03
00 02 05 03 00 12 02 12 03 15 01 02 27 1F 36 01
36 00 39 00 00 00 00 00 00 00 00 00 00 00 00 00
instructions
stack
display
(address stored in D)
1F = .loop
0F = .start
"Hello World!" + 0 terminator
Rest of instructions

“Hardware”

- Eight bit controller
- 256 bytes of RAM – 24 bytes used for character display
- 4 general purpose registers (**A**, **B**, **C** & **D**)
- Stack pointer (**SP**)
- Instruction pointer (**IP**)
- Three flags (**Z**ero, **C**arry and **F**ault)
- Simulator runs up to a blazing 16Hz

Operators (part 1)

Copying a value

MOV : move

Defining a variable

DB : define bytes

Mathematical operations

ADD : add or subtract

INC : Increment

DEC : Decrement

MUL : Multiply

DIV : Divide

Stack operations

PUSH: push on stack

POP : pop from stack

Operators (part 2)

Logical Operations

- **AND** : Binary and
- **OR** : binary or
- **XOR** : binary xor
- **NOT** : binary not

Shift operations

- **SHL** : shift left
- **SHR** : shift right

Function calling

- **CALL** : function call
- **RET** : return from function

Halting the processor

- **HLT** : stop program

Operators (part 3)

Unconditional jump

JMP : jump

Compare

CMP : compare

Conditional jump

JC : jump if carry
JNC : jump if no carry
JZ : jump if zero
JNZ : jump if not zero
JE : jump if equal
JNE : jump if not equal
JA : jump if above (<)
JNA : jump if not above
JAE : jump if above or equal (>=)
JB : jump if below (<)
JNB : jump if not below
JNBE : jump if not below or equal (<=)

For math (ADD, SUB, INC, DEC)

ADD reg, reg

ADD reg, address

ADD reg, constant

~~ADD address, reg~~

~~ADD address, constant~~

INC reg

DEC reg

Target operand always a register!

SUB reg, reg

SUB reg, address

SUB reg, constant

Watch out for math instructions MUL and DIV!

MUL reg
MUL address
MUL constant

DIV reg
DIV address
DIV constant

Target of MUL and DIV is always register A!

Why?

Types of operands for MOV

MOV reg, reg

MOV reg, address

MOV reg, constant

MOV address, reg

MOV address, constant

Types of operands for MOV - aside

MOV A, 128
MOV A, B
MOV [A], 128
MOV A, [128]
MOV [A], [128]
MOV 128, A
MOV 128, [A]
MOV [128], A
MOV [128], [A]

Types of operands for MOV - aside - opcodes

MOV A, 128
MOV A, B
MOV [A], 128
MOV A, [128]
MOV [A], [128]
~~**MOV** 128, A~~ Makes no sense!
~~**MOV** 128, [A]~~ Makes no sense!
MOV [128], A
MOV [128], [A]

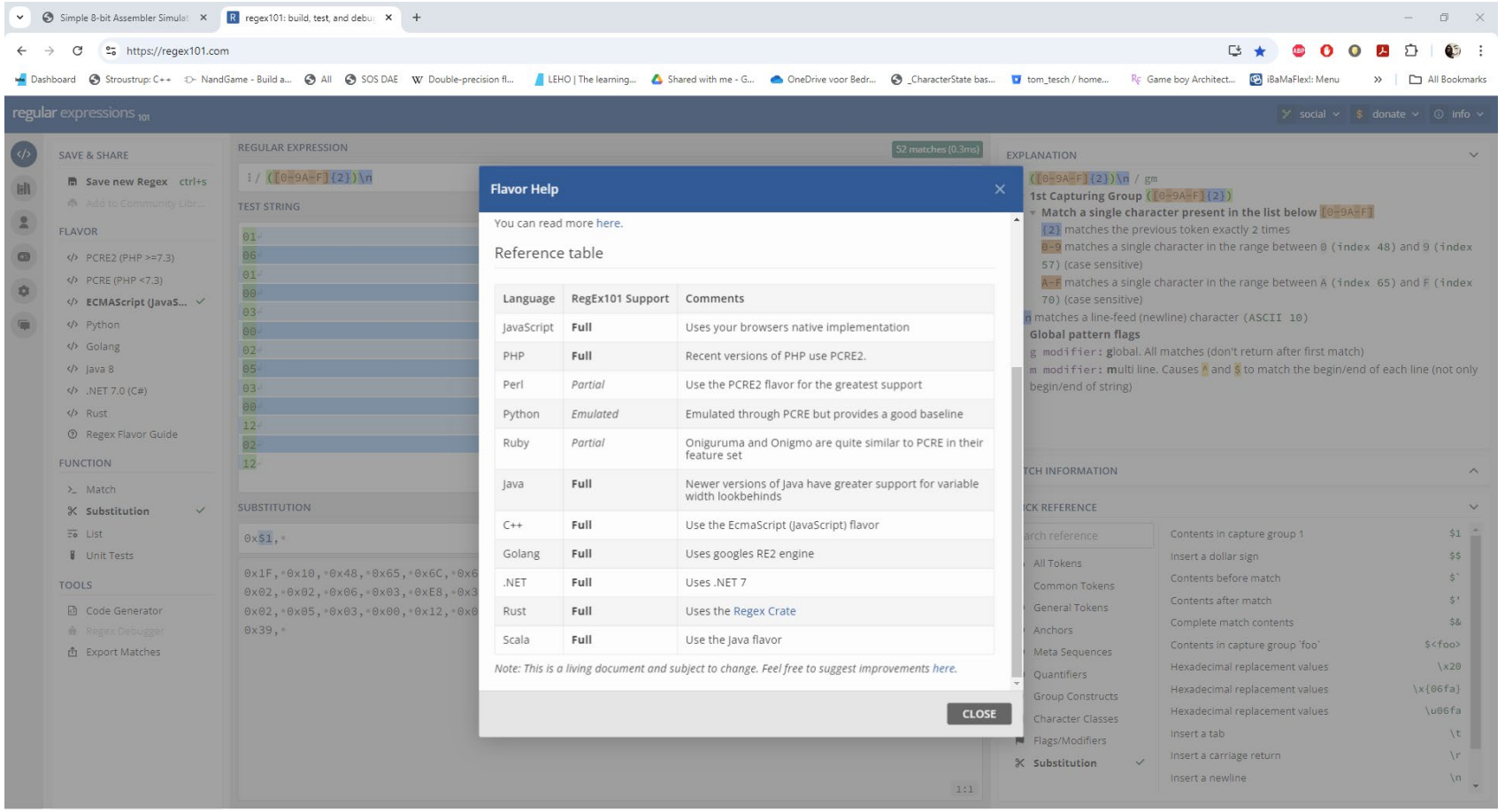
Types of operands for MOV – aside -

MOV A, 128
MOV A, B
MOV [A], 128
MOV A, [128]
~~**MOV** [A], [128]~~ Moving from address to address not supported!
~~**MOV** 128, A~~ Makes no sense!
~~**MOV** 128, [A]~~ Makes no sense!
MOV [128], A
~~**MOV** [128], [A]~~ Moving from address to address not supported!

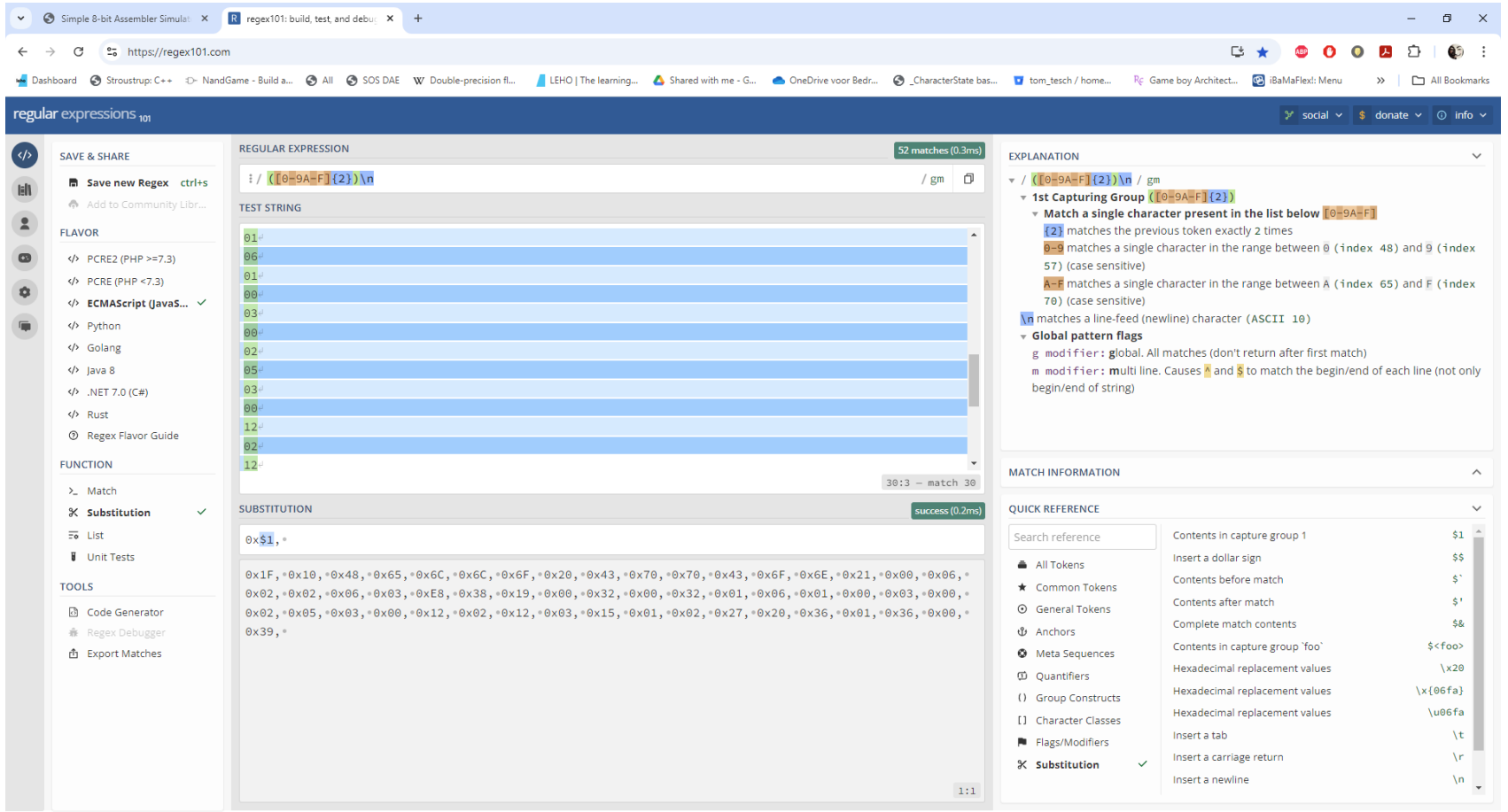
Types of operands for MOV – aside - opcodes

MOV A, 128	**0x06 0x00 0x80**
MOV A, B	**0x01 0x00 0x01**
MOV [A], 128	**0x08 0x00 0x80**
MOV A, [128]	**0x02 0x00 0x80**
~~**MOV** [A], [128]~~	Moving from address to address not supported!
~~**MOV** 128, A~~	Makes no sense!
~~**MOV** 128, [A]~~	Makes no sense!
MOV [128], A	**0x04 0x80 0x00**
~~**MOV** [128], [A]~~	Moving from address to address not supported!

REGEX 101

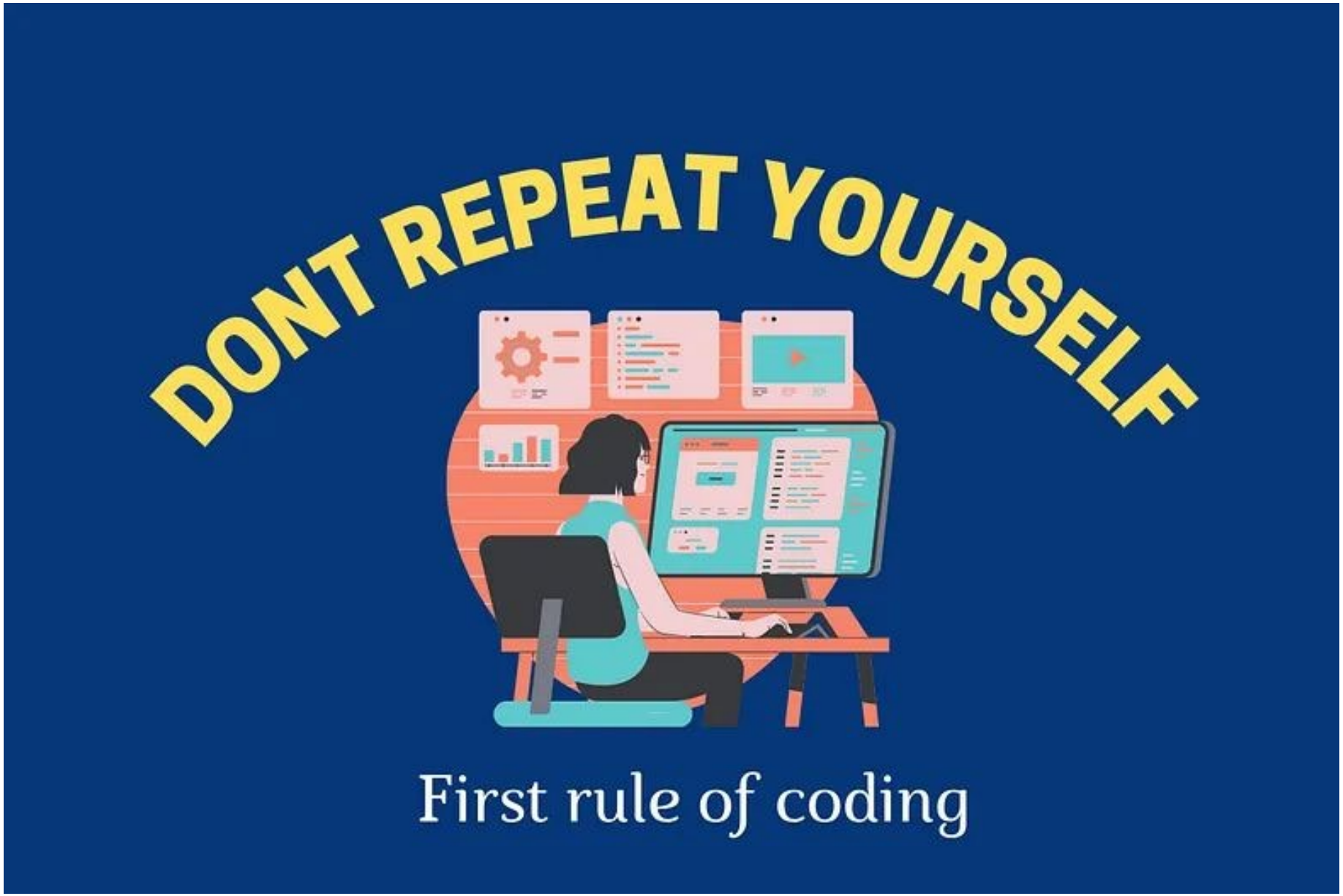

regex101: build, test, and debu
https://regex101.com
regular expressions 101
SAVE & SHARE
Save new Regex ctrl+s
FLAVOR
PCRE2 (PHP >=7.3)
PCRE (PHP <7.3)
ECMAScript (JavaS...
Python
Golang
Java 8
.NET 7.0 (C#)
Rust
Regex Flavor Guide
FUNCTION
Match
Substitution
List
Unit Tests
TOOLS
Code Generator
Export Matches
REGULAR EXPRESSION
52 matches (0.3ms)
TEST STRING
SUBSTITUTION
EXPLANATION
Flavor Help
You can read more here.
Reference table
Language
RegEx101 Support
Comments
JavaScript
Full
Uses your browsers native implementation
PHP
Full
Recent versions of PHP use PCRE2.
Perl
Partial
Use the PCRE2 flavor for the greatest support
Python
Emulated
Emulated through PCRE but provides a good baseline
Ruby
Partial
Oniguruma and Onigmo are quite similar to PCRE in their feature set
Java
Full
Newer versions of Java have greater support for variable width lookbehinds
C++
Full
Use the EcmaScript (JavaScript) flavor
Golang
Full
Uses googles RE2 engine
.NET
Full
Uses .NET 7
Rust
Full
Uses the Regex Crate
Scala
Full
Use the Java flavor
Note: This is a living document and subject to change. Feel free to suggest improvements here.
CLOSE

REGEX 101


regex101: build, test, and debu
https://regex101.com
regular expressions 101
social
donate
info
SAVE & SHARE
Save new Regex ctrl+s
Add to Community Libr...
FLAVOR
PCRE2 (PHP >=7.3)
PCRE (PHP <7.3)
ECMAScript (JavaS...
Python
Golang
Java 8
.NET 7.0 (C#)
Rust
Regex Flavor Guide
FUNCTION
Match
Substitution
List
Unit Tests
TOOLS
Code Generator
Regex Debugger
Export Matches
REGULAR EXPRESSION
52 matches (0.3ms)
/ ([0-9A-F]{2})\n / gm
TEST STRING
01
06
01
00
03
00
02
05
03
00
12
02
12
30:3 — match 30
SUBSTITUTION
success (0.2ms)
0x$1,
0x1F, 0x10, 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x20, 0x43, 0x70, 0x70, 0x43, 0x6F, 0x6E, 0x21, 0x00, 0x06, 0x02, 0x02, 0x06, 0x03, 0xE8, 0x38, 0x19, 0x00, 0x32, 0x00, 0x32, 0x01, 0x06, 0x01, 0x00, 0x03, 0x00, 0x02, 0x05, 0x03, 0x00, 0x12, 0x02, 0x12, 0x03, 0x15, 0x01, 0x02, 0x27, 0x20, 0x36, 0x01, 0x36, 0x00, 0x39,
1:1
EXPLANATION
/ ([0-9A-F]{2})\n / gm
1st Capturing Group ([0-9A-F]{2})
Match a single character present in the list below [0-9A-F]
{2} matches the previous token exactly 2 times
0-9 matches a single character in the range between 0 (index 48) and 9 (index 57) (case sensitive)
A-F matches a single character in the range between A (index 65) and F (index 70) (case sensitive)
\n matches a line-feed (newline) character (ASCII 10)
Global pattern flags
g modifier: global. All matches (don't return after first match)
m modifier: multi line. Causes ^ and $ to match the begin/end of each line (not only begin/end of string)
MATCH INFORMATION
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
Quantifiers
Group Constructs
Character Classes
Flags/Modifiers
Substitution
Contents in capture group 1 $1
Insert a dollar sign $$
Contents before match $`
Contents after match $'
Complete match contents $&
Contents in capture group `foo` $<foo>
Hexadecimal replacement values \x20
Hexadecimal replacement values \x{06fa}
Hexadecimal replacement values \u06fa
Insert a tab \t
Insert a carriage return \r
Insert a newline \n

A first naïve implementation

https://godbolt.org/z/4vobzbvYT


DONT REPEAT YOURSELF
First rule of coding


WET
Write Everything Twice

Write every time!

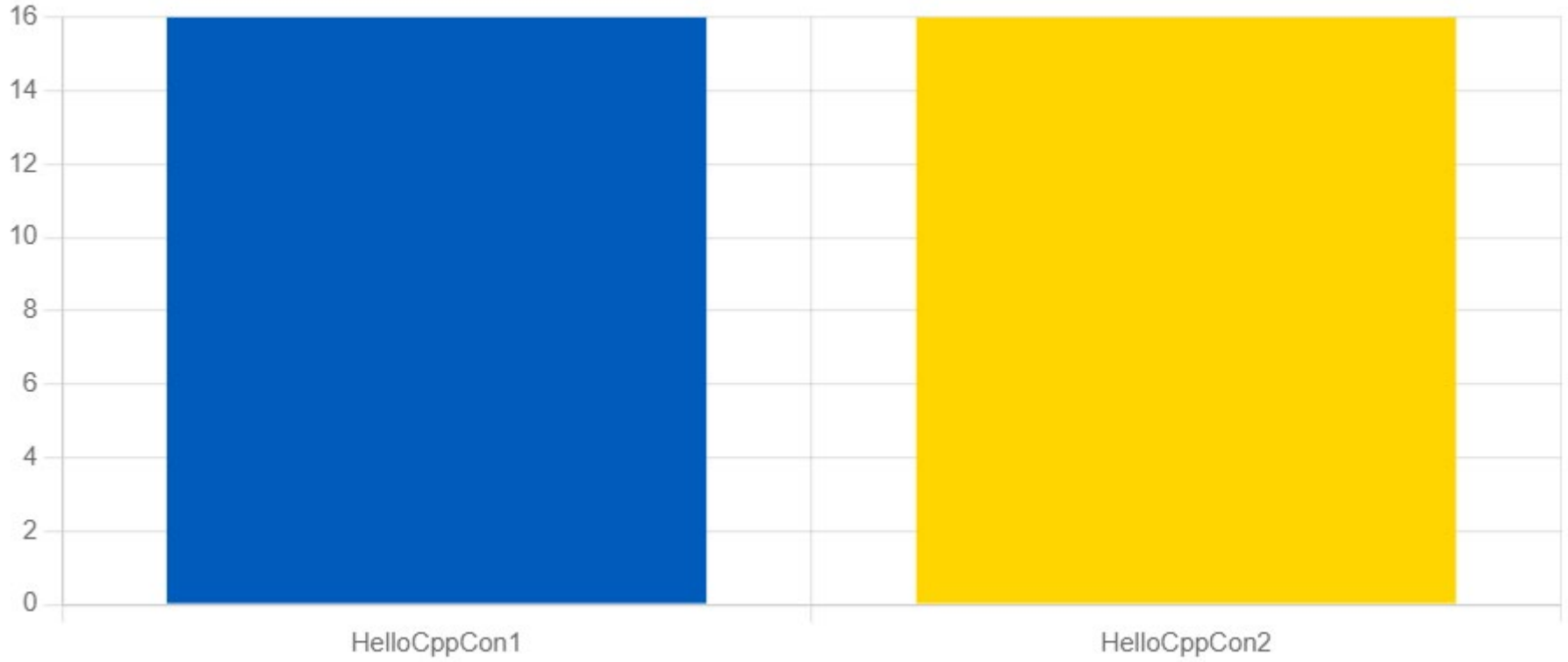
We Enjoy Typing!

Waste Everyones Time

DRY and exceptions

https://godbolt.org/z/WadGrWPKx

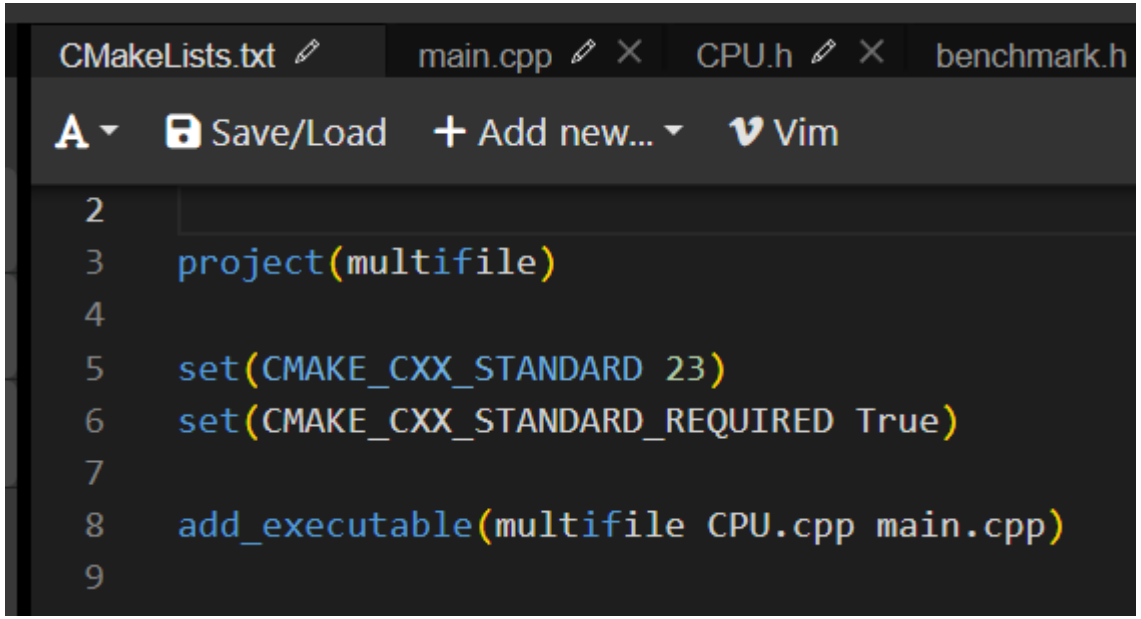
https://quick-bench.com/q/dso6fz4WN2NzBZvv3p_CRxk_iRY



Making a class

- Hurray for Godbolt supporting cmake!
  - This allows for multifile projects

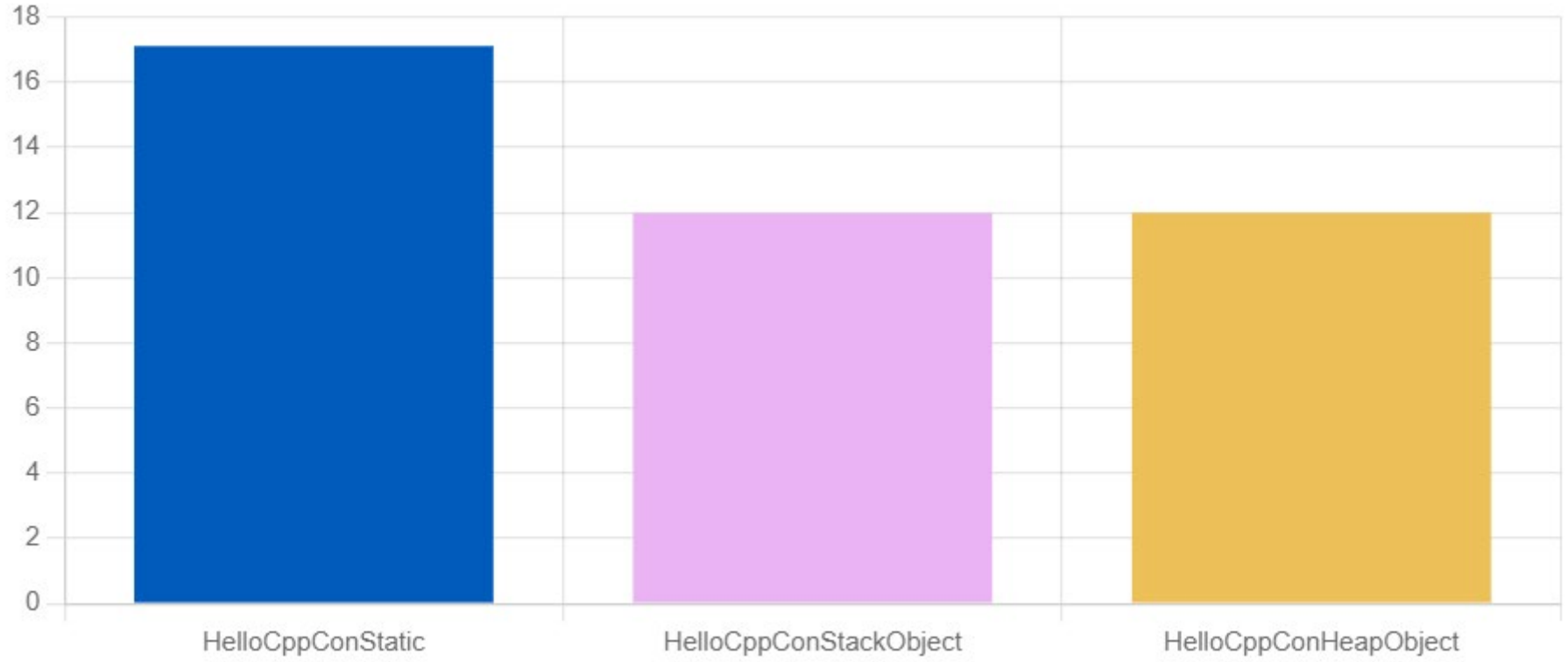
https://godbolt.org/z/Mzdd6n1fW



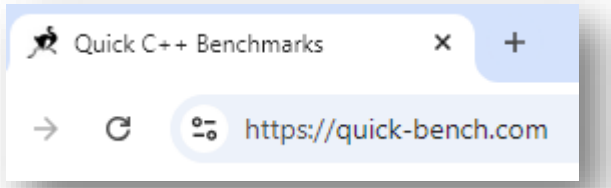
Aside: avoiding naked for's (C++23)

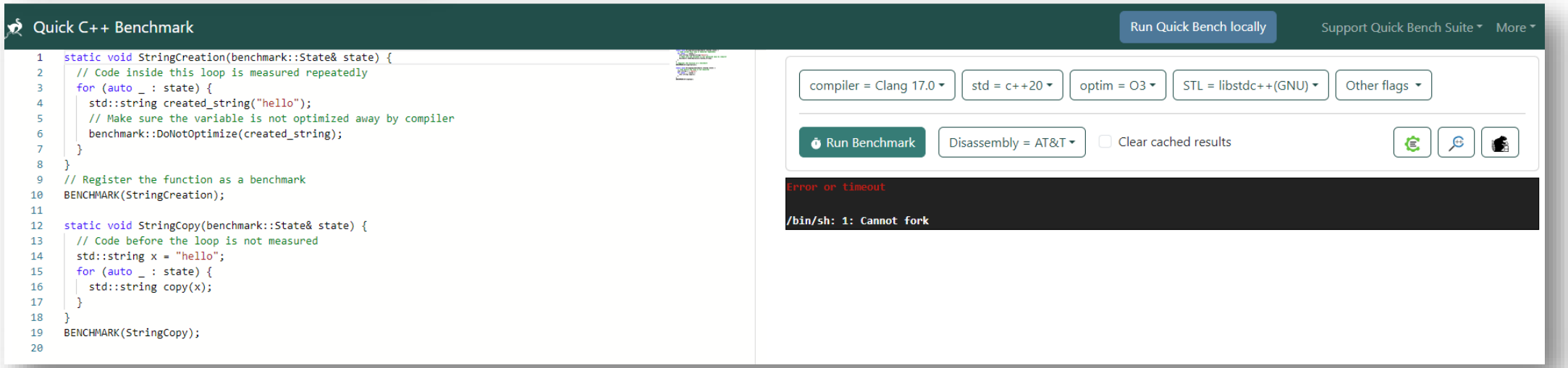
```
void CPU::load_program(std::span<uint8_t const> program) {
    for (auto const [idx, c] : std::views::enumerate(program)) {
        m_memory[idx] = c;
    }
}
```

https://quick-bench.com/q/4lgc8EFV-7bBkZoORTS1hXWM8Vo



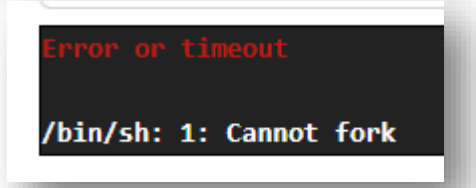
Disaster strikes






```cpp
static void StringCreation(benchmark::State& state) {
  // Code inside this loop is measured repeatedly
  for (auto _ : state) {
    std::string created_string("hello");
    // Make sure the variable is not optimized away by compiler
    benchmark::DoNotOptimize(created_string);
  }
}
// Register the function as a benchmark
BENCHMARK(StringCreation);

static void StringCopy(benchmark::State& state) {
  // Code before the loop is not measured
  std::string x = "hello";
  for (auto _ : state) {
    std::string copy(x);
  }
}
BENCHMARK(StringCopy);
```



Benchmark.h to the rescue

https://godbolt.org/z/WarhrqGva

benchmark.h to the rescue

```cpp
#ifndef benchmark_h
#define benchmark_h

#include <algorithm>
#include <chrono>
#include <cmath>
#include <iostream>
#include <numeric>
#include <utility>
#include <vector>

template <typename Func>
  requires std::invocable<Func>
auto measure_runtime(Func func, size_t runs = 10,
                     int numberOfRepetitions = 100) {
  using namespace std::chrono;
  std::vector<nanoseconds> times(runs);

  std::generate(times.begin(), times.end(), [&]() {
    auto start = high_resolution_clock::now();
    for (int i{}; i < numberOfRepetitions; ++i) func();
    auto end = high_resolution_clock::now();
    return duration_cast<nanoseconds>(end - start);
  });
```

```cpp
  std::sort(times.begin(), times.end());

  auto discard_count = static_cast<int>(std::sqrt(runs) / 2);

  auto sum = std::accumulate(times.begin() + discard_count,
                             times.end() - discard_count, nanoseconds{0});
  auto average = sum / (times.size() - discard_count*2);

  auto square_sum = std::accumulate(times.begin() + discard_count,
                                    times.end() - discard_count, int64_t{0},
                                    [average](int64_t acc, nanoseconds time) {
                                      int64_t diff = (time - average).count();
                                      return acc + diff * diff;
                                    });

  auto standard_deviation = nanoseconds{static_cast<long long>(std::sqrt(
      square_sum / static_cast<double>(times.size() - discard_count * 2)))};

  return std::pair{average, standard_deviation};
}

#endif
```

Compile Time Analysis to a map

Function pointers are constexpr!

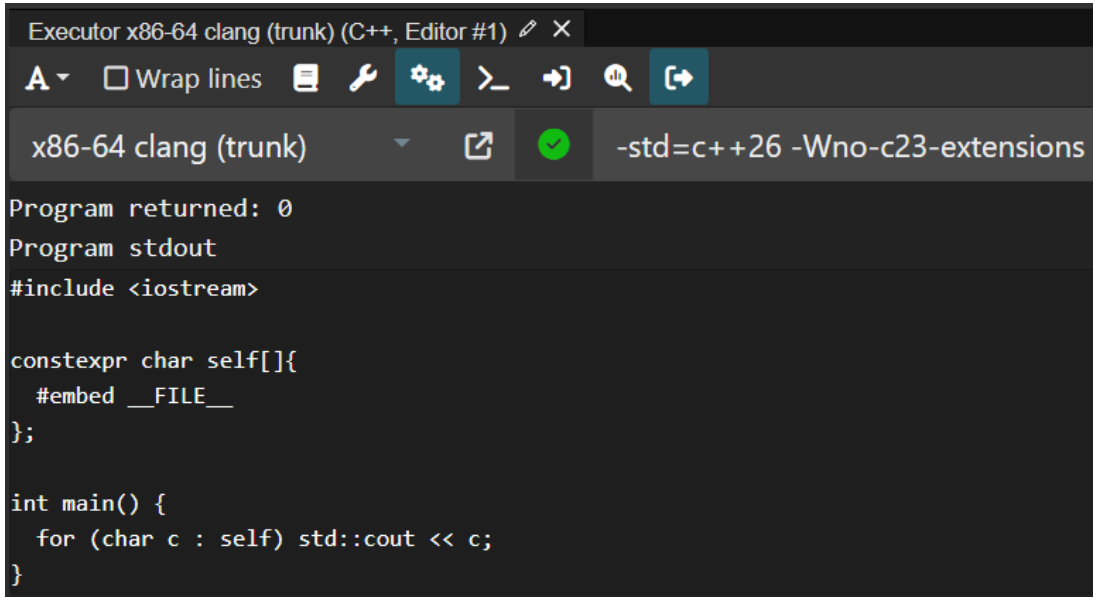
https://godbolt.org/z/YYKdYTGP6

Asside #embed – a Quine

```cpp
#include <iostream>

constexpr char self[]{
  #embed __FILE__
};

int main() {
  for (char c : self) std::cout << c;
}
```



This means: We can load a file at compile time!

Analyse to map

```
template <std::size_t n_instructions>
consteval static std::array<Instruction, n_instructions> analyse_to_map(
    const std::span<const uint8_t> bytecode, size_t skipStart,
    size_t skipEnd) {
  std::array<Instruction, n_instructions> instructions{};
  size_t location{};
  for (auto& instruction : instructions) {
    const int instruction_size{get_instruction_size(bytecode, location)};
    instruction = Instruction{
        static_cast<uint8_t>(location),
        {get_right_method(static_cast<Opcode>(bytecode[location])),
         bytecode[std::min(location + 1, bytecode.size() - 1)],
         bytecode[std::min(location + 2, bytecode.size() - 1)]}};
    location += instruction_size;
    if (location >= skipStart and location <= skipEnd) {
      location = skipEnd + 1;
    }
  }
  return instructions;
}
```

Need the number of instructions at compile time

```
static consteval int count_instructions(std::span<const uint8_t> bytecode,
                                        size_t skipStart, size_t skipEnd) {
  int count{};
  size_t location{};
  while (location < bytecode.size()) {
    const auto instruction_size{get_instruction_size(bytecode, location)};
    ++count;
    location += instruction_size;
    if (location >= skipStart and location <= skipEnd) {
      location = skipEnd + 1;
    }
  }
  return count;
}
```

```cpp
static consteval uint8_t get_instruction_size(
    std::span<const uint8_t> bytecode, size_t location) {
  CPU::Opcode opcode = static_cast<CPU::Opcode>(bytecode[location]);
  return get_opcode_size(opcode);
}
```

```cpp
static constexpr uint8_t get_opcode_size(CPU::Opcode opcode) {
  switch (opcode) {
    case CPU::Opcode::mov_reg_to_reg:
    case CPU::Opcode::mov_reg_to_address:
    case CPU::Opcode::mov_regaddress_to_reg:
    case CPU::Opcode::mov_reg_to_regaddress:
    case CPU::Opcode::mov_number_to_reg:
    case CPU::Opcode::cmp_reg_with_reg:
    case CPU::Opcode::cmp_regaddress_with_reg:
      return 3;
    case CPU::Opcode::inc_reg:
    case CPU::Opcode::jmp_address:
    case CPU::Opcode::jnz_address:
    case CPU::Opcode::push_reg:
    case CPU::Opcode::pop_reg:
    case CPU::Opcode::call_address:
      return 2;
    case CPU::Opcode::ret:
    case CPU::Opcode::hlt:
      return 1;
    case CPU::Opcode::last_opcode:
    default:
      throw "stop everything! This should not happen :-(";
  }
}
```

Runtime initialization

```
static inline auto initialize_map(
    std::span<const CPU::Instruction> instructions) {
  using KeyType = typename std::tuple_element<0, Instruction>::type;
  using ValueType = typename std::tuple_element<1, Instruction>::type;

  std::map<KeyType, ValueType> instruction_map;

  for (const auto& [opcode, name] : instructions) {
    instruction_map[opcode] = name;
  }

  return instruction_map;
}
```

```
Opcode execute_instruction(
    const std::map<std::tuple_element<0, Instruction>::type,
                   std::tuple_element<1, Instruction>::type>&
        instruction_map);
```

```
CPU::Opcode CPU::execute_instruction(
    const std::map<std::tuple_element<0, Instruction>::type,
                   std::tuple_element<1, Instruction>::type>& instruction_map) {
  auto const it = instruction_map.find(m_IP);
  auto const& [func, param1, param2] = it->second;
  if (it != instruction_map.end()) {
    auto const old_IP{m_IP};
    (this->*func)(param1, param2);
    m_IP += get_opcode_size(static_cast<CPU::Opcode>(m_memory[old_IP]));
    return static_cast<CPU::Opcode>(m_memory[old_IP]);
  }
  throw InvalidAddressAndOpcode{m_IP, m_memory[m_IP]};
}
```

```
using coreInstruction =
    std::tuple<void (CPU::*)(uint8_t, uint8_t), uint8_t, uint8_t>;

using Instruction = std::pair<uint8_t, coreInstruction>;
```

```
Opcode execute_instruction(
    const std::map<std::tuple_element<0, Instruction>::type,
                   std::tuple_element<1, Instruction>::type>&
        instruction_map);
```

```
CPU::Opcode CPU::execute_instruction(
    const std::map<std::tuple_element<0, Instruction>::type,
                   std::tuple_element<1, Instruction>::type>& instruction_map) {
  auto const it = instruction_map.find(m_IP);
  auto const& [func, param1, param2] = it->second;
  if (it != instruction_map.end()) {
    auto const old_IP{m_IP};
    (this->*func)(param1, param2);
    m_IP += get_opcode_size(static_cast<CPU::Opcode>(m_memory[old_IP]));
    return static_cast<CPU::Opcode>(m_memory[old_IP]);
  }
  throw InvalidAddressAndOpcode{m_IP, m_memory[m_IP]};
}
```

Compile Time Analysis to a array

```
constexpr auto instruction_array =
    CPU::initialize_array<cmd_map.size(), cmd_map[cmd_map.size() - 1].first>(
        cmd_map);
```

```
template <std::size_t n_instructions, std::size_t last_instruction_location>
static constexpr auto initialize_array(
    const std::array<Instruction, n_instructions>& cmd_array) {
  std::array<coreInstruction, last_instruction_location + 1> all_locations;
  for (const auto& instr : cmd_array) {
    all_locations[instr.first] = instr.second;
  }
  return all_locations;
}
```

Execution is simple

```
template <std::size_t arraySize>
Opcode execute_instruction(
    const std::array<coreInstruction, arraySize>& instruction_array) {
  auto const& [func, param1, param2] = instruction_array[m_IP];
  auto const old_IP{m_IP};
  if (func)
    (this->*func)(param1, param2);
  else
    throw InvalidAddressAndOpcode{old_IP, 0};
  m_IP += get_opcode_size(static_cast<CPU::Opcode>(m_memory[old_IP]));
  return static_cast<CPU::Opcode>(m_memory[old_IP]);
}
```

Aside: CTAD for std::array size (since C++20)

```
constexpr auto Program{std::to_array<uint8_t>({
    0x06, 0x00, 0x00, 0x06, 0x01, 0x21, 0x06, 0x02, 0x00, 0x06, 0x03, 0x00,
    0x12, 0x00, 0x12, 0x02, 0x12, 0x03, 0x15, 0x03, 0x01, 0x27, 0x10, 0x15,
    0x02, 0x01, 0x27, 0x0E, 0x15, 0x00, 0x01, 0x27, 0x0C, 0x00,
})};
```

Performance Measurement

The experiment was a failure ☹

Or wasn't it?

Device specifications

Device name	5CD125YX3X
Full device name	5CD125YX3X.hogeschool-wvl.be
Processor	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
Installed RAM	16.0 GB (15.3 GB usable)
Device ID	3C3A4882-A44A-4632-858C-C31D9D7DE87C
Product ID	00329-00000-00003-AA855
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

```
Some calculated value to avoid compiling things away 230000 [should be identical for all runs]
Average runtime for runtime: 6ms, standard deviation: 430us.
Some calculated value to avoid compiling things away 230000 [should be identical for all runs]
Average runtime for compiletime (to map): 12ms, standard deviation: 398us.
Some calculated value to avoid compiling things away 230000 [should be identical for all runs]
Average runtime for compiletime (to array): 9ms, standard deviation: 189us.
```

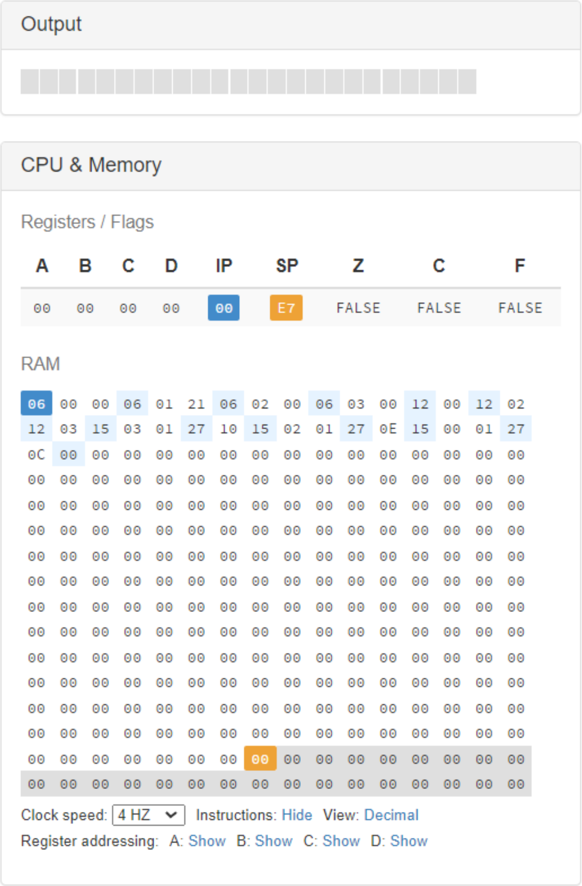
Last version we discussed

https://godbolt.org/z/7Tvb9xfbe

Three nested loops for long execution

Code (Instruction Set)

```
MOV A, 0
MOV B, stop
MOV C, 0
MOV D, 0
loop1: INC A
loop2: INC C
loop3: INC D
CMP D, [B]
JNZ loop3
CMP C, [B]
JNZ loop2
CMP A, [B]
JNZ loop1
stop: HLT
```



On my 3.2 GHz machine:
50529029 in 200ms
-> 250MIPS

Thanks!!!

- **You –** both here at CppCon and watching this online!
- **The CppCon team –** always helpful and friendly
- **My colleagues –** always encourage me
- **Students and former students –** you’re a daily inspiration and source of motivation
- **My wife and son –** for always believing in me and encouraging me to push further


DIGITAL ARTS & ENTERTAINMENT
howest
university of applied sciences

Questions?

1: say lasagna in hopes making them laugh.
-> Recommend!
2: run and cry in the corner/ dark area. -
-> In case option 3 is a no
3: come home in the comfort of Jax, me and mom.
-> In case it hit deep
4: don't give a F* and say idc to stand up for yourself.**
-> Good to prove yourself