



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

Tom Tesch

About me

- Teacher @ Howest University of Applied Sciences within “Digital Arts and Entertainment” bachelor Game Development major
 - C++
 - Algorithms
 - Computational System Fundamentals
 - ***new*** Retro Console Programming
- <https://www.howest.be/en>
- <https://www.digitalartsandentertainment.be/>
- tom.tesch@howest.be
- <https://www.linkedin.com/in/tom-tesch/>



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++

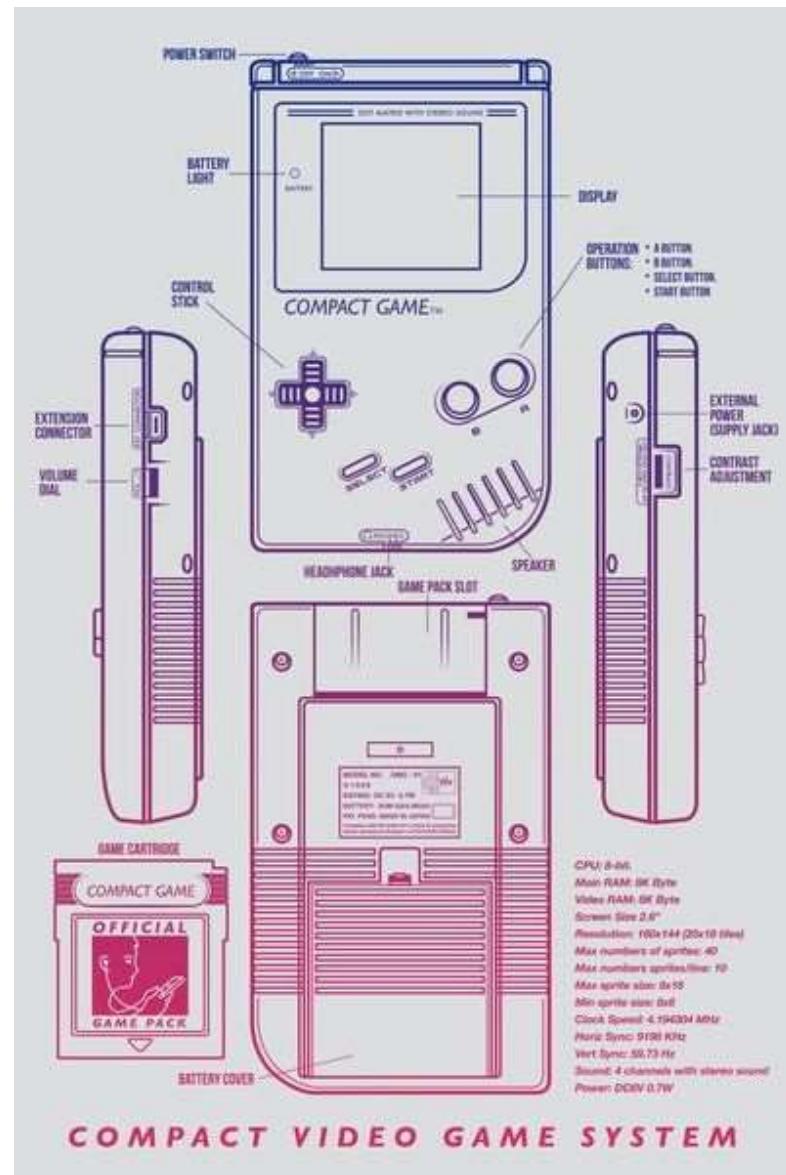
The original Game Boy DMG-01

- 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	Windows, Linux, macOS, iOS, Android
SkyEmu	Windows, Linux, macOS, iOS, Android
binjgb	Windows, Linux, macOS, iOS, Android
higan	Windows, Linux, macOS, iOS, Android
BDM	Windows, Linux, macOS, iOS, Android
VGB	Windows, Linux, macOS, iOS, Android
Gamebert	Windows, Linux, macOS, iOS, Android
Pantheon	Windows, Linux, macOS, iOS, Android
UGE	Windows, Linux, macOS, iOS, Android
KiGB	Windows, Linux, macOS, iOS, Android
TGB Dual	Windows, Linux, macOS, iOS, Android
GiiBiiAdvance	Windows, Linux, macOS, iOS, Android
MetroBoy	Windows, Linux, macOS, iOS, Android
hdmg	Windows, Linux, macOS, iOS, Android

SameBoy	Windows, Linux, macOS, iOS, Android
BGB	Windows
Gambatte	Windows, Linux, macOS, iOS, Android
BizHawk	Windows
Emulicious	Windows, Linux, macOS, iOS, Android
Mesen2	Windows
GBE+	Windows, Linux, macOS, iOS, Android
MAME	Windows, Linux, macOS, iOS, Android
Gearboy	Windows, Linux, macOS, iOS, Android
GBCC	Windows, Linux, macOS, iOS, Android
GameRoy	Windows, Linux, macOS, iOS
ares	Windows, Linux, macOS, iOS
mGBA	Windows, Linux, macOS, iOS, Android
VBA-M	Windows, Linux, macOS, iOS, Android

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, Scribbletests](#) 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++

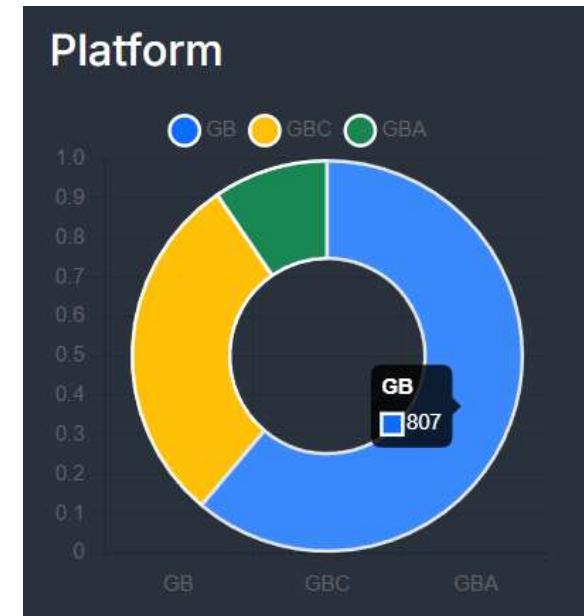
A huge library:

- GameBoy: 1043
- GameBoy Color: 576

30% are backwards compatible

Active Homebrew

<https://hh.gbdev.io/>

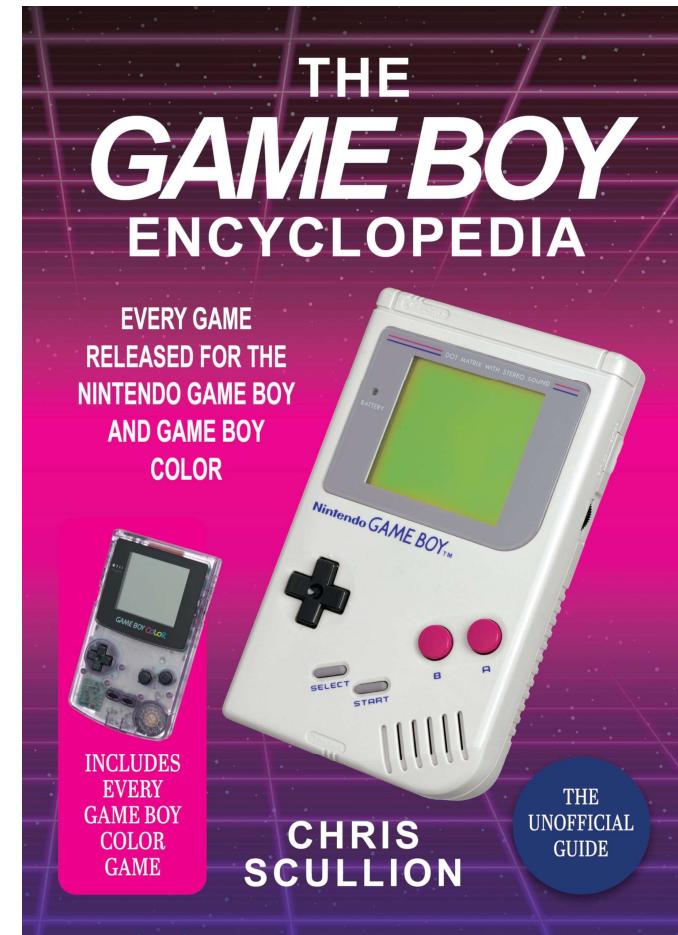


Building the World's Fastest GameBoy Emulator in Modern C++

A huge library:

- GameBoy: 1043
- GameBoy Color: 576

30% are backwards compatible



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

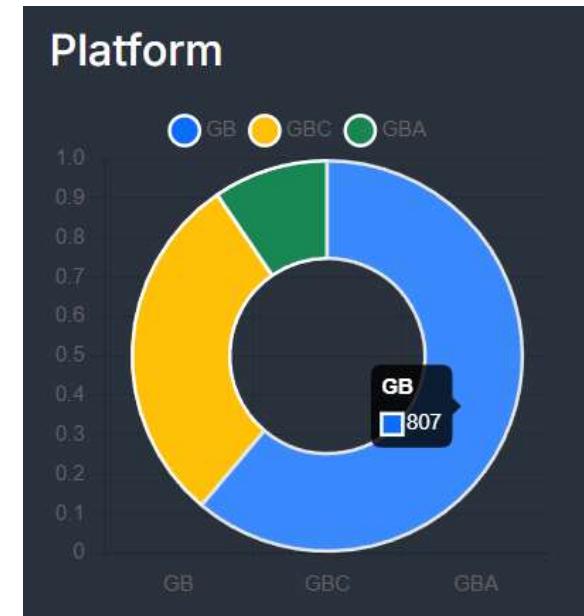
A huge library:

- GameBoy: 1043
 - GameBoy Color: 576
- 30% are backwards compatible

A tiny library:

Active Homebrew

<https://hh.gbdev.io/>



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

A huge library:

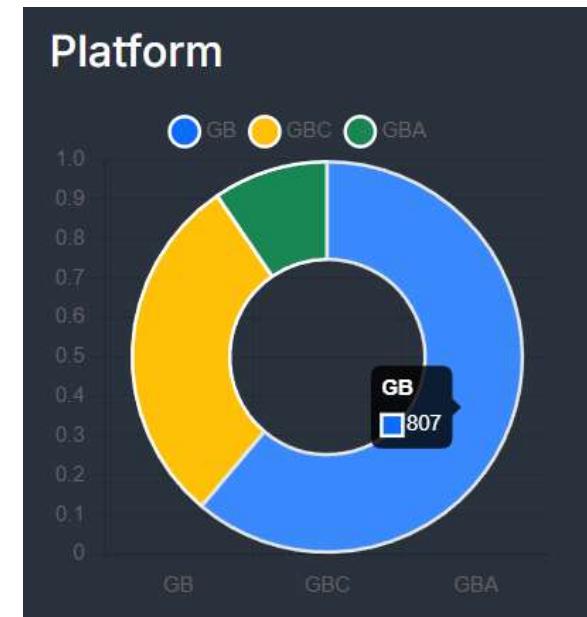
- GameBoy: 1043
 - GameBoy Color: 576
- 30% are backwards compatible

A tiny library:

- GameBoy: \pm 800MB
- GameBoy Color: \pm 1.5GB

Active Homebrew

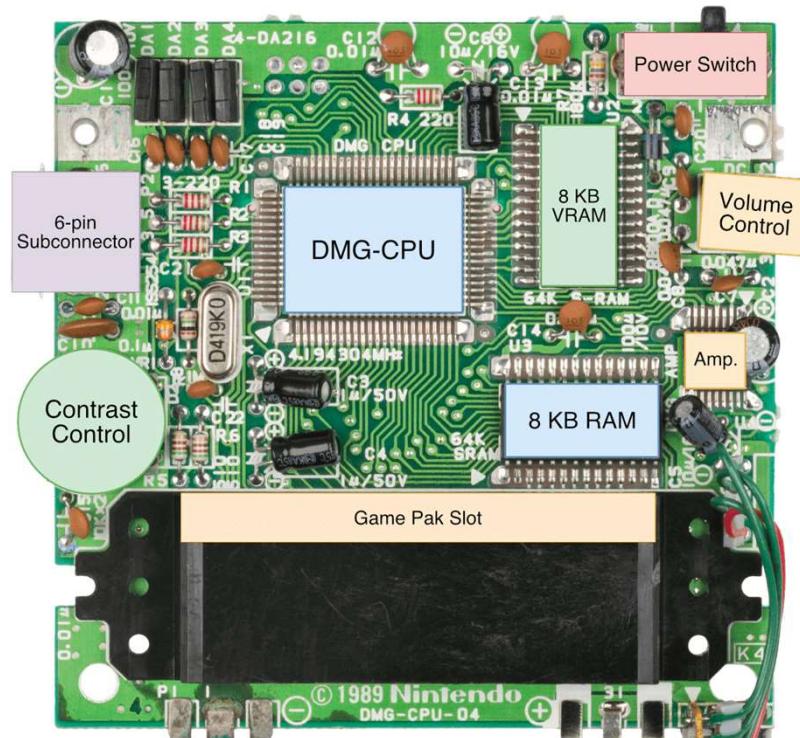
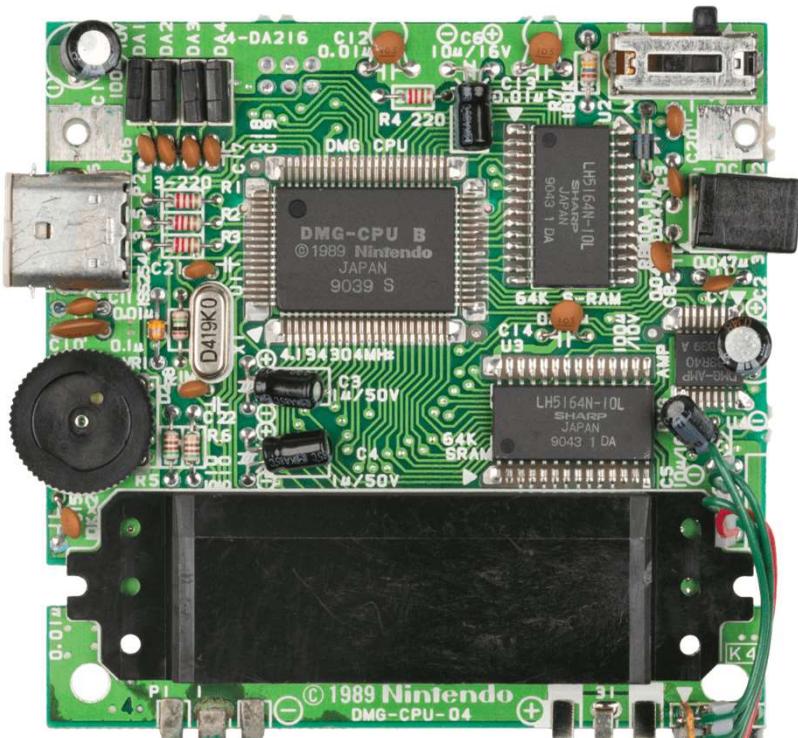
<https://hh.gbdev.io/>



Compatible clones/parts are still being made!

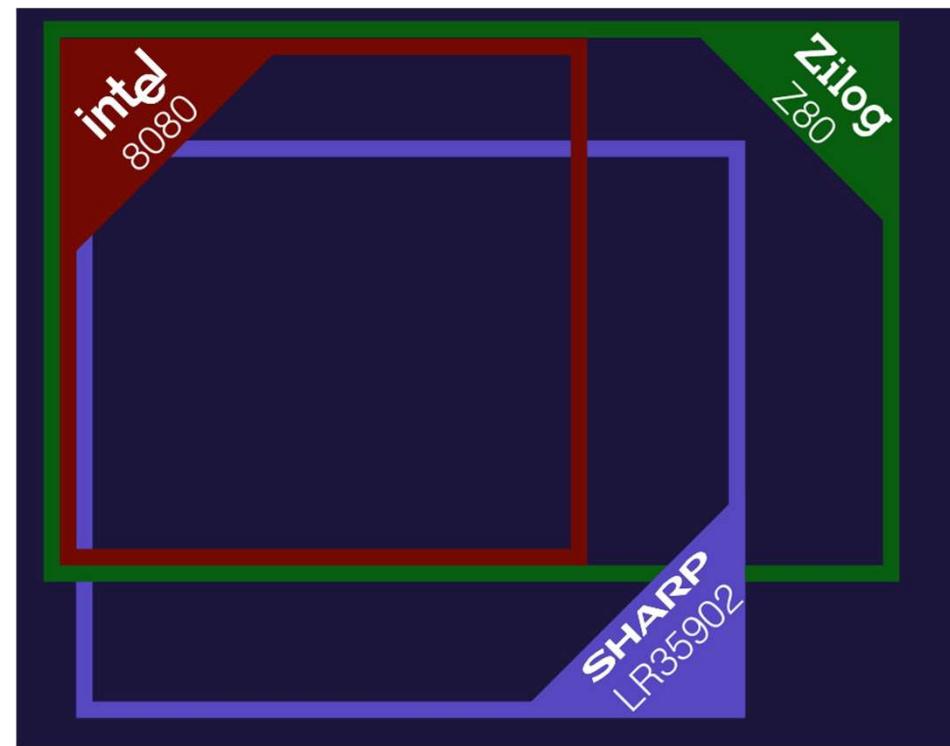


Motherboard (revision 04)

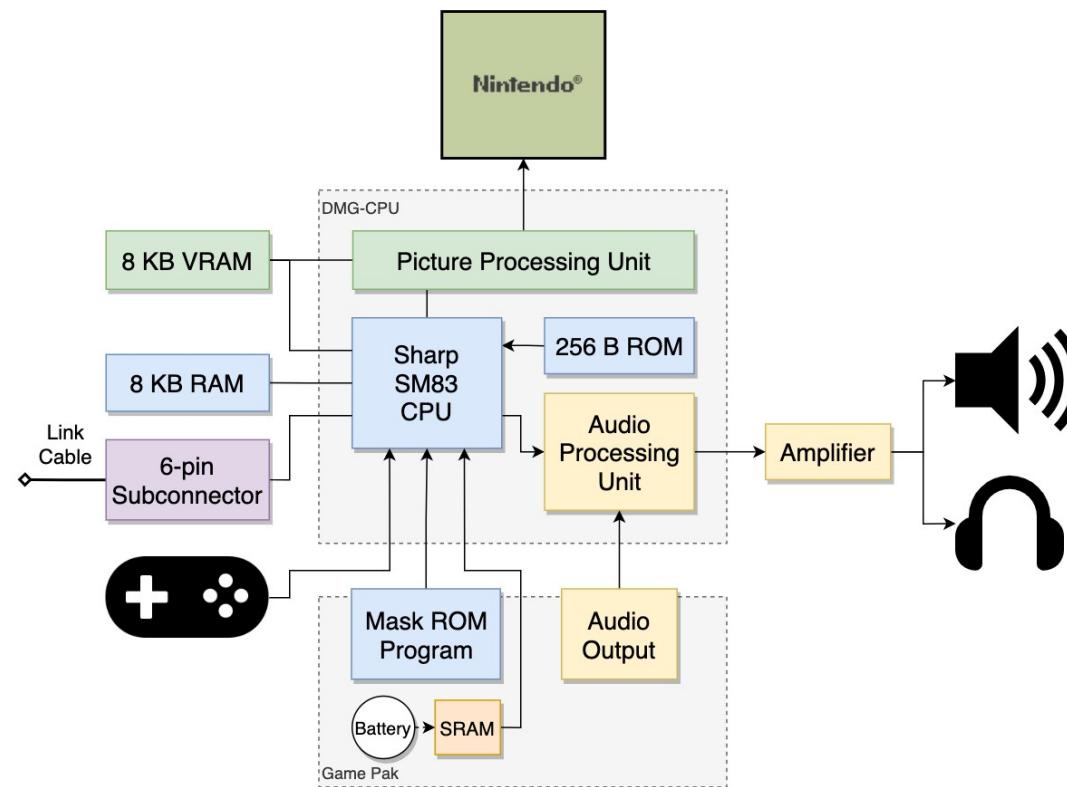


CPU

- System On a Chip (SoC)
- DMG-CPU or Sharp LR35902
- Sharp SM83 (mix between Z80 and Intel 8080)
- 4,194,304 Hertz \approx 4 Mhz
- No IO ports = completely memory mapped
- Only 8080's set of registers
- Z80's extended bit manipulation instruction set
- Some additional 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.

Interrupt Register	0xFFFF
High RAM	0xFF80 – 0xFFFFE
Unusable	0xFF4C – 0xFF7F
I/O	0xFF00 – 0xFF4B
Unusable	0xFEAO – 0xFEFF
Sprite Attributes	0xFE00 – 0xFE9F
Unusable	0xE000 – 0xFDFF
Internal RAM	0xC000 – 0xDFFF
Switchable RAM Bank	0xA000 – 0xBFFF
Video RAM	0x8000 – 0x9FFF
Switchable ROM Bank	0x4000 – 0x7FFF
ROM	0x0000 – 0x3FFF

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 Z0H -	DEC B 1 4 Z1H -	LD B, n8 2 8 -----	RLCA 1 4 000C	LD [a16], SP 3 20	ADD HL, BC 1 8 -0HC	LD A, [BC] 1 8 -----	DEC BC 1 8 -----	INC C 1 4 Z0H -	DEC C 1 4 Z1H -	LD C, n8 2 8 -----	RRCA 1 4 000C
1x	STOP n8 2 4 -----	LD DE, n16 3 12	LD [DE]. A 1 8 -----	INC DE 1 8 -----	INC D 1 4 Z0H -	DEC D 1 4 Z1H -	LD D, n8 2 8 -----	RLA 1 4 000C	JR e8 2 12 -----	ADD HL, DE 1 8 -0HC	LD A, [DE] 1 8 -----	DEC DE 1 8 -----	INC E 1 4 Z0H -	DEC E 1 4 Z1H -	LD E, n8 2 8 -----	RRA 1 4 000C
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 Z0H -	DEC H 1 4 Z1H -	LD H, n8 2 8 -----	DAA 1 4 Z-0C	JR Z, e8 2 12/8 -----	ADD HL, HL 1 8 -0HC	LD A, [HL+] 1 8 -----	DEC HL 1 8 -----	INC L 1 4 Z0H -	DEC L 1 4 Z1H -	LD L, n8 2 8 -----	CPL 1 4 -11-
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 Z0H -	DEC [HL] 1 12 Z1H -	LD [HL], n8 2 12 -----	SCF 1 4 -001	JR C, e8 2 12/8 -----	ADD HL, SP 1 8 -0HC	LD A, [HL-] 1 8 -----	DEC SP 1 8 -----	INCA 1 4 Z0H -	DECA 1 4 Z1H -	LD A, n8 2 8 -----	CCF 1 4 -00C
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 -----	HALT -----	LD [HL]. A 1 4 -----	LD A, B 1 8 -----	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 Z0H C	ADD A, C 1 4 Z0H C	ADD A, D 1 4 Z0H C	ADD A, E 1 4 Z0H C	ADD A, H 1 4 Z0H C	ADD A, L 1 4 Z0H C	ADD A, [HL] 1 8 Z0H C	ADD A, A 1 4 Z0H C	ADC A, B 1 4 Z0H C	ADC A, C 1 4 Z0H C	ADC A, D 1 4 Z0H C	ADC A, E 1 4 Z0H C	ADC A, H 1 4 Z0H C	ADC A, L 1 4 Z0H C	ADC A, [HL] 1 8 Z0H C	ADC A, A 1 4 Z0H C
9x	SUB A, B 1 4 Z1H C	SUB A, C 1 4 Z1H C	SUB A, D 1 4 Z1H C	SUB A, E 1 4 Z1H C	SUB A, H 1 4 Z1H C	SUB A, L 1 4 Z1H C	SUB A, [HL] 1 8 Z1H C	SUB A, A 1 4 Z1H C	SBC A, B 1 4 Z1H C	SBC A, C 1 4 Z1H C	SBC A, D 1 4 Z1H C	SBC A, E 1 4 Z1H C	SBC A, H 1 4 Z1H C	SBC A, L 1 4 Z1H C	SBC A, [HL] 1 8 Z1H C	SBC A, A 1 4 Z1H C
Ax	AND A, B 1 4 Z010	AND A, C 1 4 Z010	AND A, D 1 4 Z010	AND A, E 1 4 Z010	AND A, H 1 4 Z010	AND A, L 1 4 Z010	AND A, [HL] 1 8 Z010	AND A, A 1 4 Z010	XOR A, B 1 4 Z000	XOR A, C 1 4 Z000	XOR A, D 1 4 Z000	XOR A, E 1 4 Z000	XOR A, H 1 4 Z000	XOR A, L 1 4 Z000	XOR A, [HL] 1 8 Z000	XOR A, A 1 4 1000
Bx	ORA, B 1 4 Z000	ORA, C 1 4 Z000	ORA, D 1 4 Z000	ORA, E 1 4 Z000	ORA, H 1 4 Z000	ORA, L 1 4 Z000	ORA, [HL] 1 8 Z000	ORA, A 1 4 Z000	CPO, B 1 4 Z1H C	CPO, C 1 4 Z1H C	CPO, D 1 4 Z1H C	CPO, E 1 4 Z1H C	CPO, H 1 4 Z1H C	CPO, L 1 4 Z1H C	CPO, [HL] 1 8 Z1H C	CPO, A 1 4 1100
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 Z0H 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	ADCA, n8 2 8 Z0H 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 Z1H 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 Z1H 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 Z010	RST \$20 1 16	ADD SP, e8 2 16 00HC	JP HL 1 4	LD [a16]. A 3 16	—	—	—	XORA, n8 2 8 Z000	RST \$28 1 16
Fx	LDH A, [a8] 2 12	POP AF 1 12	LD A, [C] 1 8	DI 1 4	—	PUSH AF 1 16	OR A, n8 2 8 Z000	RST \$30 1 16	LD HL, SP + e8 2 12 00HC	LD SP, HL 1 8	LD A, [a16] 3 16	EI 1 4	—	—	CPO, n8 2 8 Z1H C	RST \$38 1 16

J (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

	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF				
	INC B 1 4 Z0H-	DEC B 1 4 Z1H-	LD B, n8 2 8 ----	RLCA 1 4 000C	LD [a16], SP 3 20 ----	ADD HL, BC 1 8 -0HC	LD A, [BC] 1 8 ----	DEC BC 1 8 ----	INC C 1 4 Z0H-	DEC C 1 4 Z1H-	LD C, n8 2 8 ----	RRCA 1 4 000C				
	INC D 1 4 Z0H-	DEC D 1 4 Z1H-	LD D, n8 2 8 ----	RLA 1 4 000C	JR e8 2 12 ----	ADD HL, DE 1 8 -0HC	LD A, [DE] 1 8 ----	DEC DE 1 8 ----	INC E 1 4 Z0H-	DEC E 1 4 Z1H-	LD E, n8 2 8 ----	RRA 1 4 000C				
	INC H 1 4 Z0H-	DEC H 1 4 Z1H-	LD H, n8 2 8 ----	DAA 1 4 Z-0C	JR Z, e8 2 12/8 ----	ADD HL, HL 1 8 -0HC	LD A, [HL+] 1 8 ----	DEC HL 1 8 ----	INCL 1 4 Z0H-	DECL 1 4 Z1H-	LD L, n8 2 8 ----	CPL 1 4 -11-				
3x	INC [HL] 2 12/8 ----	DEC [HL] 3 12 ----	LD [HL], n8 1 12 Z0H-	SCF 2 12 ----	JR C, e8 2 12/8 ----	ADD HL, SP 1 8 -0HC	LD A, [HL-] 1 8 ----	DEC SP 1 8 ----	INCA 1 4 Z0H-	DECA 1 4 Z1H-	LD A, n8 2 8 ----	CCF 1 4 -00C				
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, [HL] 1 8 ----	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, [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, [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 ----	HALT 1 4 ----	LD [HL], A 1 8 ----	LD A, B 1 8 ----	LD A, C 1 4 ----	LD A, D 1 4 ----	LD A, E 1 4 ----	LD A, H 1 4 ----	LD A, L 1 8 ----	LD A, [HL] 1 8 ----	LD A, A 1 4 ----	
8x	ADD A, B 1 4 Z0H-	ADD A, C 1 4 Z0H-	ADD A, D 1 4 Z0H-	ADD A, E 1 4 Z0H-	ADD A, H 1 4 Z0H-	ADD A, L 1 4 Z0H-	ADD A, [HL] 1 8 Z0H-	ADD A, A 1 4 Z0H-	ADC A, B 1 4 Z0H-	ADC A, C 1 4 Z0H-	ADC A, D 1 4 Z0H-	ADC A, E 1 4 Z0H-	ADC A, H 1 4 Z0H-	ADC A, L 1 4 Z0H-	ADC A, [HL] 1 8 Z0H-	ADC A, A 1 4 Z0H-
9x	SUB A, B 1 4 Z1H-	SUB A, C 1 4 Z1H-	SUB A, D 1 4 Z1H-	SUB A, E 1 4 Z1H-	SUB A, H 1 4 Z1H-	SUB A, L 1 4 Z1H-	SUB A, [HL] 1 8 Z1H-	SUB A, A 1 4 1100	SBC A, B 1 4 Z1H-	SBC A, C 1 4 Z1H-	SBC A, D 1 4 Z1H-	SBC A, E 1 4 Z1H-	SBC A, H 1 4 Z1H-	SBC A, L 1 4 Z1H-	SBC A, [HL] 1 8 Z1H-	SBC A, A 1 4 Z1H-
Ax	AND A, B 1 4 Z010	AND A, C 1 4 Z010	AND A, D 1 4 Z010	AND A, E 1 4 Z010	AND A, H 1 4 Z010	AND A, L 1 4 Z010	AND A, [HL] 1 8 Z010	AND A, A 1 4 Z010	XOR A, B 1 4 Z000	XOR A, C 1 4 Z000	XOR A, D 1 4 Z000	XOR A, E 1 4 Z000	XOR A, H 1 4 Z000	XOR A, L 1 8 Z000	XOR A, [HL] 1 8 Z000	XOR A, A 1 4 1000
Bx	ORA, B 1 4 Z000	ORA, C 1 4 Z000	ORA, D 1 4 Z000	ORA, E 1 4 Z000	ORA, H 1 4 Z000	ORA, L 1 4 Z000	ORA, [HL] 1 8 Z000	ORA, A 1 4 Z000	COPA, B 1 4 Z1H-	COPA, C 1 4 Z1H-	COPA, D 1 4 Z1H-	COPA, E 1 4 Z1H-	COPA, H 1 4 Z1H-	COPA, L 1 8 Z1H-	COPA, [HL] 1 8 Z1H-	COPA, A 1 4 1100
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 Z0H-	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 Z0H-	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 Z1H-	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 Z1H-	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 Z010	RST \$20 1 16 ----	ADD SP, e8 2 16 00HC	JP HL 1 4	LD [a16], A 3 16 ----	—	—	XORA, n8 2 8 Z000	RST \$28 1 16	
Fx	LDH A, [a8] 2 12	POP AF 1 12	LD A, [C] 1 8	DI 1 4	—	PUSH AF 1 16	OR A, n8 2 8 Z000	RST \$30 1 16 ----	LD HL, SP + e8 2 12 00HC	LD SP, HL 1 8 ----	LD A, [a16] 3 16 ----	EI 1 4	—	CPA, n8 2 8 Z1H-	RST \$38 1 16	

Game Boy CPU (SM83) instruction set (ISON)

	x0	x1	x2	x3	x4	x5	x6	INS reg	— Instruction mnemonic
0x	NOP 1 4 -----	LD BC, n16 3 12	LD [BC]. A 1 8 -----	INC BC 1 8 -----	INC B 1 4 Z0H -	DEC B 1 4 Z1H -	LD B, n8 2 8 -----	2 8	— Duration in T-states*
1x	STOP n8 2 4 -----	LD DE, n16 3 12	LD [DE]. A 1 8 -----	INC DE 1 8 -----	INC D 1 4 Z0H -	DEC D 1 4 Z1H -	LD D, n8 2 8 -----	Z N H C	— Flags affected
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 Z0H -	DEC H 1 4 Z1H -	LD H, n8 2 8 -----	-001	Length in bytes →
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 Z0H -	DEC [HL] 1 12 Z1H -	LD [HL], n8 2 12 -----	-0 HC) 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.
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 -----
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 -----
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 -----
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 -----	HALT 1 4 -----	LD [HL]. A 1 8 -----	LD A, B 1 4 -----	LD A, C 1 4 -----
8x	ADD A, B 1 4 Z0H C	ADD A, C 1 4 Z0H C	ADD A, D 1 4 Z0H C	ADD A, E 1 4 Z0H C	ADD A, H 1 4 Z0H C	ADD A, L 1 4 Z0H C	ADD A, [HL] 1 8 Z0H C	ADD A, A 1 4 Z0H C	ADC A, B 1 4 Z0H C
9x	SUB A, B 1 4 Z1H C	SUB A, C 1 4 Z1H C	SUB A, D 1 4 Z1H C	SUB A, E 1 4 Z1H C	SUB A, H 1 4 Z1H C	SUB A, L 1 4 Z1H C	SUB A, [HL] 1 8 Z1H C	SUB A, A 1 4 Z1H C	SBC A, B 1 4 Z1H C
Ax	AND A, B 1 4 Z010	AND A, C 1 4 Z010	AND A, D 1 4 Z010	AND A, E 1 4 Z010	AND A, H 1 4 Z010	AND A, L 1 4 Z010	AND A, [HL] 1 8 Z010	AND A, A 1 4 Z010	XOR A, B 1 4 Z000
Bx	ORA, B 1 4 Z000	ORA, C 1 4 Z000	ORA, D 1 4 Z000	ORA, E 1 4 Z000	ORA, H 1 4 Z000	ORA, L 1 4 Z000	ORA, [HL] 1 8 Z000	ORA, A 1 4 Z000	CPO, B 1 4 Z1H C
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 Z0H C	RST \$00 1 16 -----	RET Z 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 Z1H C	RST \$10 1 16 -----	RETC 1 20/8 -----
Ex	LDH [a8]. A 2 12	POP HL 1 12	LD [C]. A 1 8	—	—	PUSH HL 1 16	AND A, n8 2 8 Z010	RST \$20 1 16 -----	ADD SP, e8 2 16 00H C
Fx	LDH A, [a8] 2 12	POP AF 1 12	LD A, [C] 1 8	DI 1 4	—	PUSH AF 1 16	OR A, n8 2 8 Z000	RST \$30 1 16 -----	LD HL, SP + e8 2 12 00H C
									LD SP, HL 1 8 -----
									LD A, [a16] 3 16 -----
									EI 1 4 -----
									—
									CPA, n8 2 8 Z1H 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 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.

⌚ Prefixed (\$CB \$xx)

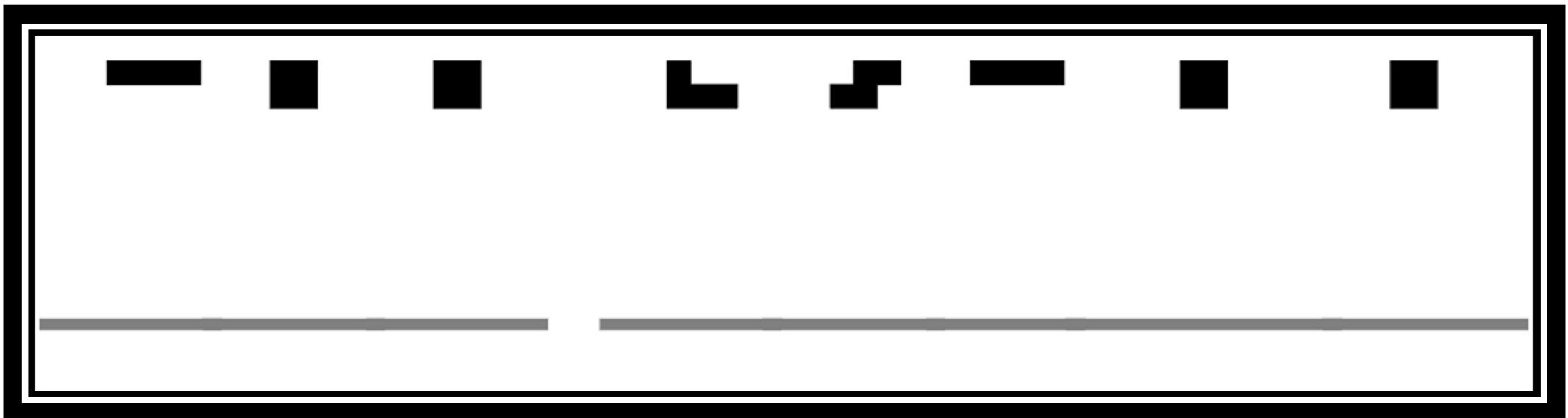
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 8 Z00 C	RLC C 2 8 Z00 C	RLC D 2 8 Z00 C	RLC E 2 8 Z00 C	RLC H 2 8 Z00 C	RLC L 2 8 Z00 C	RLC [HL] 2 16 Z00 C	RLC A 2 8 Z00 C	RRB B 2 8 Z00 C	RRC C 2 8 Z00 C	RRD D 2 8 Z00 C	RRR E 2 8 Z00 C	RRH F 2 8 Z00 C	RRCL G 2 16 Z00 C	RRCH H 2 16 Z00 C	RRCA I 2 8 Z00 C
1x	RLB B 2 8 Z00 C	RLC J 2 8 Z00 C	RLD K 2 8 Z00 C	RLE L 2 8 Z00 C	RLH M 2 8 Z00 C	RLL N 2 16 Z00 C	RL[H] O 2 16 Z00 C	RLA P 2 8 Z00 C	RRB Q 2 8 Z00 C	RRC R 2 8 Z00 C	RRD S 2 8 Z00 C	RRR T 2 8 Z00 C	RRH U 2 8 Z00 C	RRL V 2 16 Z00 C	RR[H] W 2 16 Z00 C	RR A X 2 8 Z00 C
2x	SLA B 2 8 Z00 C	SLA C 2 8 Z00 C	SLA D 2 8 Z00 C	SLA E 2 8 Z00 C	SLA H 2 8 Z00 C	SLA L 2 8 Z00 C	SLA [HL] 2 16 Z00 C	SLAA M 2 8 Z00 C	SRA B 2 8 Z00 C	SRA C 2 8 Z00 C	SRA D 2 8 Z00 C	SRA E 2 8 Z00 C	SRA H 2 8 Z00 C	SRA L 2 16 Z00 C	SRA [HL] 2 8 Z00 C	SRA A X 2 8 Z00 C
3x	SWAP B 2 8 Z000	SWAP C 2 8 Z000	SWAP D 2 8 Z000	SWAP E 2 8 Z000	SWAP H 2 8 Z000	SWAP L 2 8 Z000	SWAP [HL] 2 16 Z000	SWAP A 2 8 Z000	SRL B 2 8 Z00 C	SRL C 2 8 Z00 C	SRL D 2 8 Z00 C	SRL E 2 8 Z00 C	SRL H 2 8 Z00 C	SRL L 2 16 Z00 C	SRL [HL] 2 8 Z00 C	SRL A X 2 8 Z00 C
4x	BIT0.B 2 8 Z01 -	BIT0.C 2 8 Z01 -	BIT0.D 2 8 Z01 -	BIT0.E 2 8 Z01 -	BIT0.H 2 8 Z01 -	BIT0.L 2 8 Z01 -	BIT0,[HL] 2 12 Z01 -	BIT0.A 2 8 Z01 -	BIT1.B 2 8 Z01 -	BIT1.C 2 8 Z01 -	BIT1.D 2 8 Z01 -	BIT1.E 2 8 Z01 -	BIT1.H 2 8 Z01 -	BIT1,[HL] 2 12 Z01 -	BIT1.A X 2 8 Z01 -	
5x	BIT2.B 2 8 Z01 -	BIT2.C 2 8 Z01 -	BIT2.D 2 8 Z01 -	BIT2.E 2 8 Z01 -	BIT2.H 2 8 Z01 -	BIT2.L 2 8 Z01 -	BIT2,[HL] 2 12 Z01 -	BIT2.A 2 8 Z01 -	BIT3.B 2 8 Z01 -	BIT3.C 2 8 Z01 -	BIT3.D 2 8 Z01 -	BIT3.E 2 8 Z01 -	BIT3.H 2 8 Z01 -	BIT3,[HL] 2 12 Z01 -	BIT3.A X 2 8 Z01 -	
6x	BIT4.B 2 8 Z01 -	BIT4.C 2 8 Z01 -	BIT4.D 2 8 Z01 -	BIT4.E 2 8 Z01 -	BIT4.H 2 8 Z01 -	BIT4.L 2 8 Z01 -	BIT4,[HL] 2 12 Z01 -	BIT4.A 2 8 Z01 -	BIT5.B 2 8 Z01 -	BIT5.C 2 8 Z01 -	BIT5.D 2 8 Z01 -	BIT5.E 2 8 Z01 -	BIT5.H 2 8 Z01 -	BIT5,[HL] 2 12 Z01 -	BIT5.A X 2 8 Z01 -	
7x	BIT6.B 2 8 Z01 -	BIT6.C 2 8 Z01 -	BIT6.D 2 8 Z01 -	BIT6.E 2 8 Z01 -	BIT6.H 2 8 Z01 -	BIT6.L 2 8 Z01 -	BIT6,[HL] 2 12 Z01 -	BIT6.A 2 8 Z01 -	BIT7.B 2 8 Z01 -	BIT7.C 2 8 Z01 -	BIT7.D 2 8 Z01 -	BIT7.E 2 8 Z01 -	BIT7.H 2 8 Z01 -	BIT7,[HL] 2 12 Z01 -	BIT7.A X 2 8 Z01 -	
8x	RES0.B 2 8 ----	RES0.C 2 8 ----	RES0.D 2 8 ----	RES0.E 2 8 ----	RES0.H 2 8 ----	RES0.L 2 8 ----	RES0,[HL] 2 16 ----	RES0.A 2 8 ----	RES1.B 2 8 ----	RES1.C 2 8 ----	RES1.D 2 8 ----	RES1.E 2 8 ----	RES1.H 2 8 ----	RES1,[HL] 2 16 ----		
9x	RES2.B 2 8 ----	RES2.C 2 8 ----	RES2.D 2 8 ----	RES2.E 2 8 ----	RES2.H 2 8 ----	RES2.L 2 8 ----	RES2,[HL] 2 16 ----	RES2.A 2 8 ----	RES3.B 2 8 ----	RES3.C 2 8 ----	RES3.D 2 8 ----	RES3.E 2 8 ----	RES3.H 2 8 ----	RES3,[HL] 2 16 ----		
Ax	RES4.B 2 8 ----	RES4.C 2 8 ----	RES4.D 2 8 ----	RES4.E 2 8 ----	RES4.H 2 8 ----	RES4.L 2 8 ----	RES4,[HL] 2 16 ----	RES4.A 2 8 ----	RES5.B 2 8 ----	RES5.C 2 8 ----	RES5.D 2 8 ----	RES5.E 2 8 ----	RES5.H 2 8 ----	RES5,[HL] 2 16 ----		
Bx	RES6.B 2 8 ----	RES6.C 2 8 ----	RES6.D 2 8 ----	RES6.E 2 8 ----	RES6.H 2 8 ----	RES6.L 2 8 ----	RES6,[HL] 2 16 ----	RES6.A 2 8 ----	RES7.B 2 8 ----	RES7.C 2 8 ----	RES7.D 2 8 ----	RES7.E 2 8 ----	RES7.H 2 8 ----	RES7,[HL] 2 16 ----		
Cx	SET0.B 2 8 ----	SET0.C 2 8 ----	SET0.D 2 8 ----	SET0.E 2 8 ----	SET0.H 2 8 ----	SET0.L 2 8 ----	SET0,[HL] 2 16 ----	SET0.A 2 8 ----	SET1.B 2 8 ----	SET1.C 2 8 ----	SET1.D 2 8 ----	SET1.E 2 8 ----	SET1.H 2 8 ----	SET1,[HL] 2 16 ----		
Dx	SET2.B 2 8 ----	SET2.C 2 8 ----	SET2.D 2 8 ----	SET2.E 2 8 ----	SET2.H 2 8 ----	SET2.L 2 8 ----	SET2,[HL] 2 16 ----	SET2.A 2 8 ----	SET3.B 2 8 ----	SET3.C 2 8 ----	SET3.D 2 8 ----	SET3.E 2 8 ----	SET3.H 2 8 ----	SET3,[HL] 2 16 ----		
Ex	SET4.B 2 8 ----	SET4.C 2 8 ----	SET4.D 2 8 ----	SET4.E 2 8 ----	SET4.H 2 8 ----	SET4.L 2 8 ----	SET4,[HL] 2 16 ----	SET4.A 2 8 ----	SET5.B 2 8 ----	SET5.C 2 8 ----	SET5.D 2 8 ----	SET5.E 2 8 ----	SET5.H 2 8 ----	SET5,[HL] 2 16 ----		
Fx	SET6.B 2 8 ----	SET6.C 2 8 ----	SET6.D 2 8 ----	SET6.E 2 8 ----	SET6.H 2 8 ----	SET6.L 2 8 ----	SET6,[HL] 2 16 ----	SET6.A 2 8 ----	SET7.B 2 8 ----	SET7.C 2 8 ----	SET7.D 2 8 ----	SET7.E 2 8 ----	SET7.H 2 8 ----	SET7,[HL] 2 16 ----		

A toy example...

Let's pick a simple game and just implement that...

But which one?

What game should we pick?



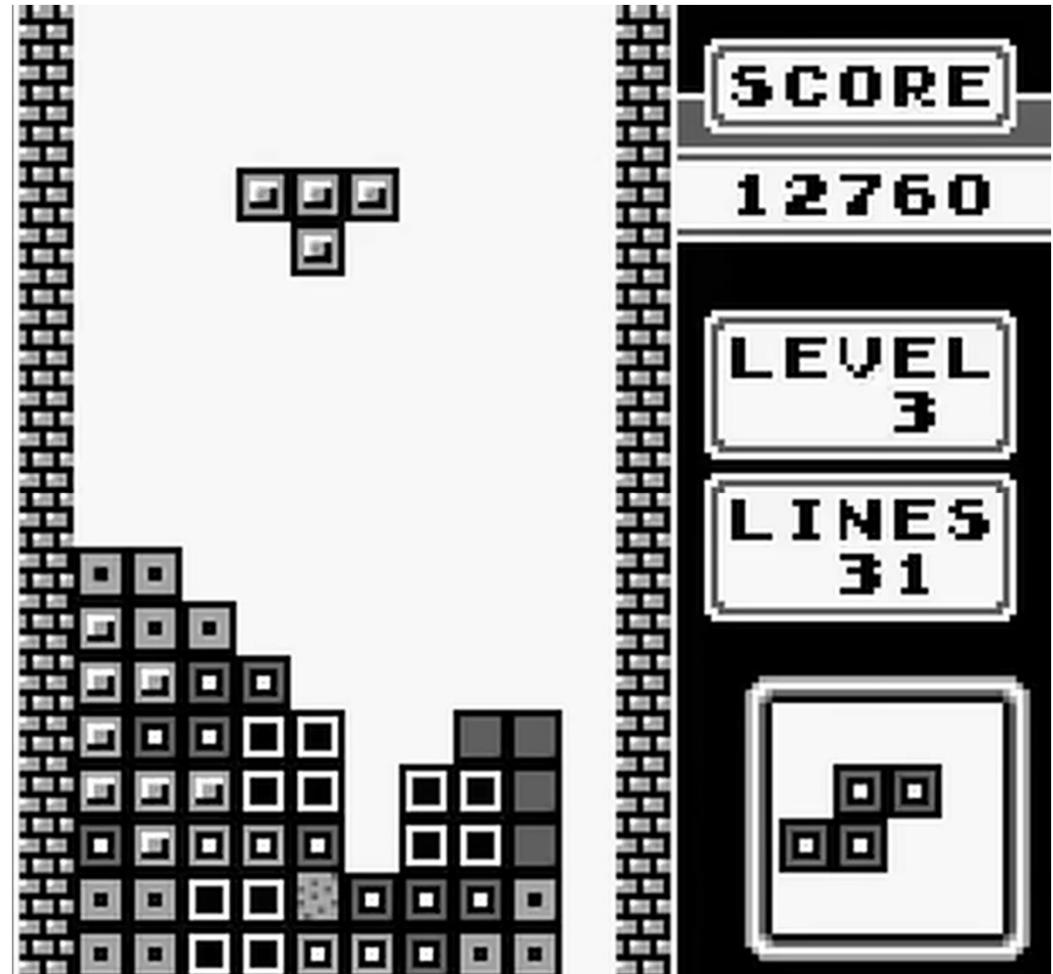
Why choose Tetris?

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

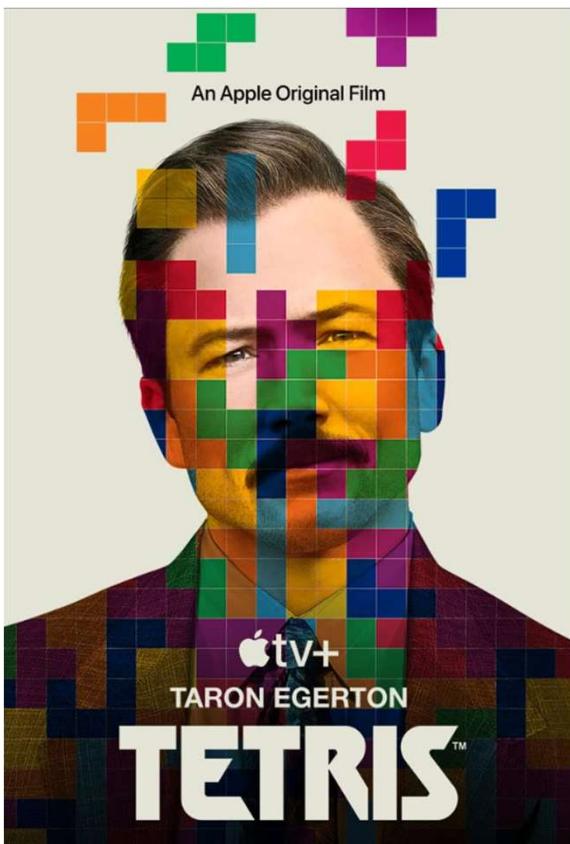
And the fact that I own several copies?

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:

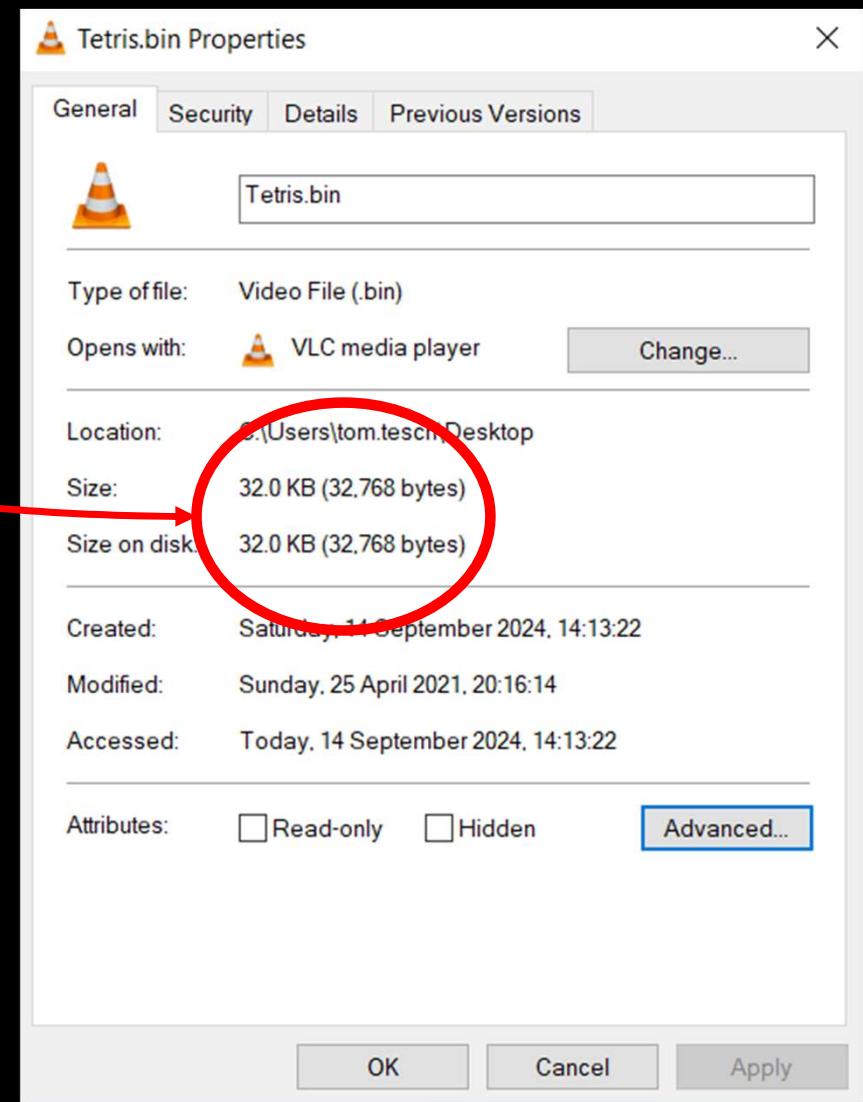
[Launch Lineup of Game Boy – Nintendo Switch Online](#)

- *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*

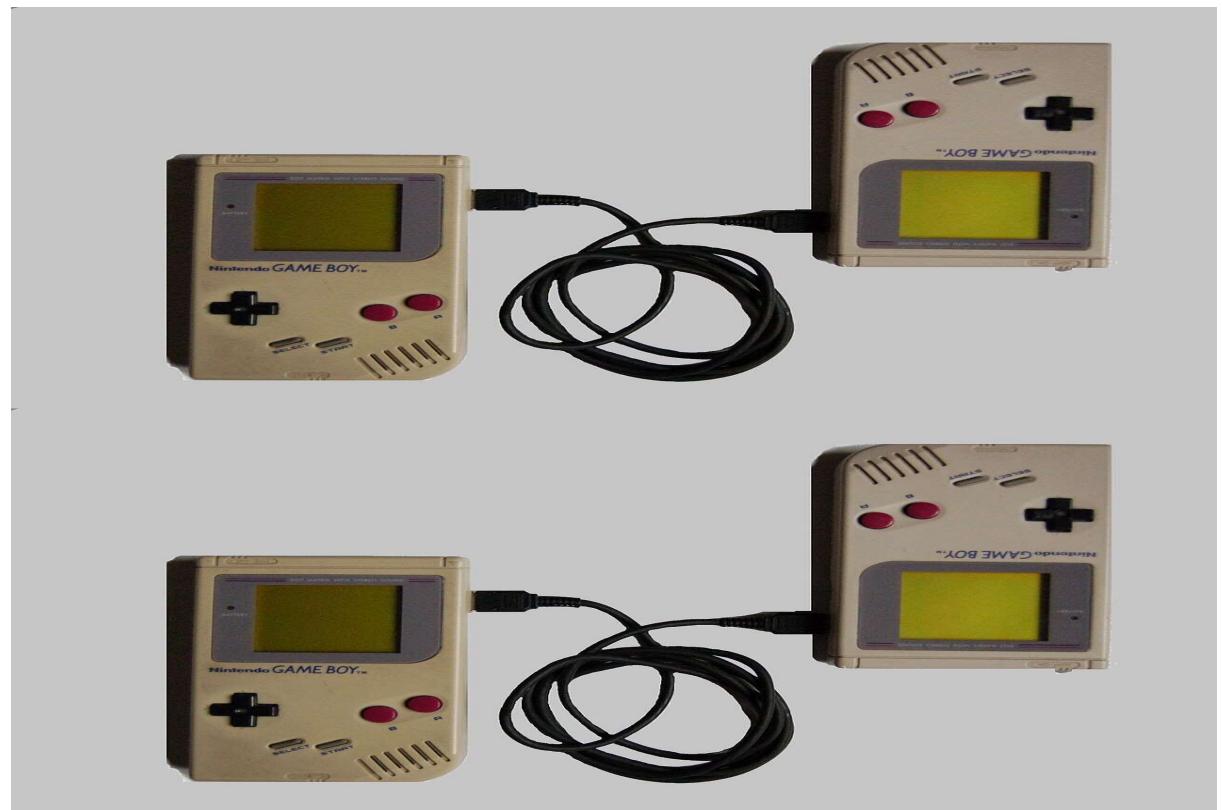
But really: why Tetris?

- M-cycle precision emulation is sufficient
- Only 284 of the 501 opcodes are used (and thus need to be implemented)
- Limited size means no memory banking – no MBC needed

No MBC needed!

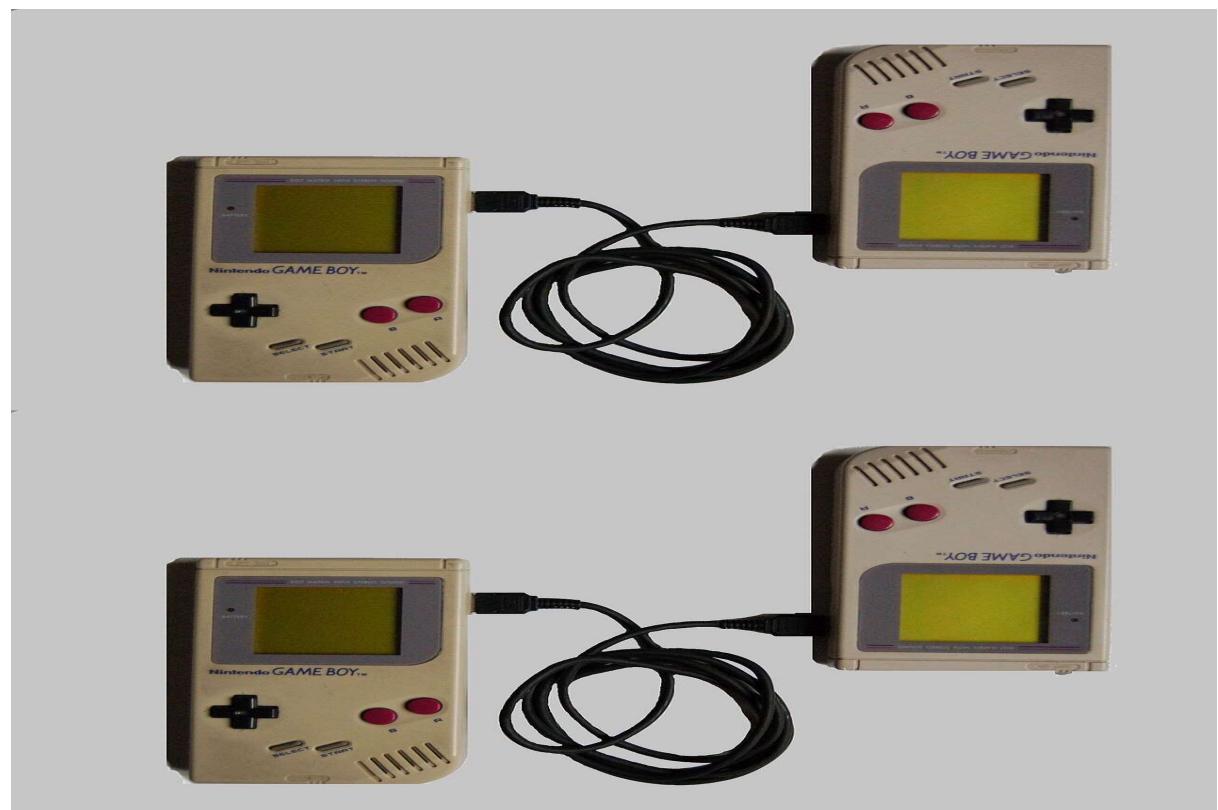


Possible interesting extension



Possible interesting extension

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



How do you make sure to win a race?

How do you make sure to win a race?



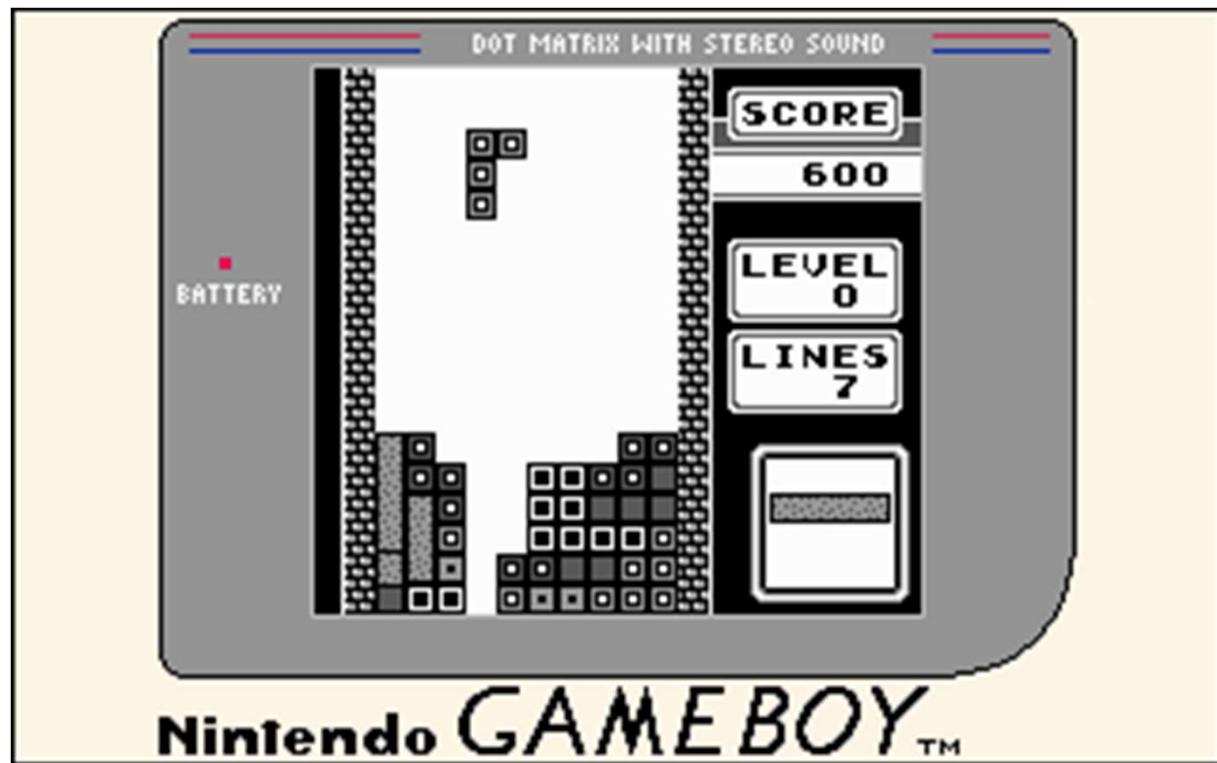
Building **the World's Fastest** GameBoy Emulator in Modern C++

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

TEGLON?

- Tetris for
- Gameboy
- Logic
- iNversion

Where does the term
originate from?



Why no sound?

Aside: Little Sound DJ

SONG	PU1	PU2	WAV	NDI	PU1
00	00	--	04	--	
01	01	--	05	--	
02	02	--	05	--	J220
03	--	--	--	--	
04	--	--	--	--	1
05	--	--	--	--	2
06	--	--	--	--	W
07	--	--	--	--	N
08	--	--	--	--	
09	--	--	--	--	
0A	--	--	--	--	
0B	--	--	--	--	
0C	--	--	--	--	
0D	--	--	--	--	
0E	--	--	--	--	
0F	--	--	--	--	



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>)

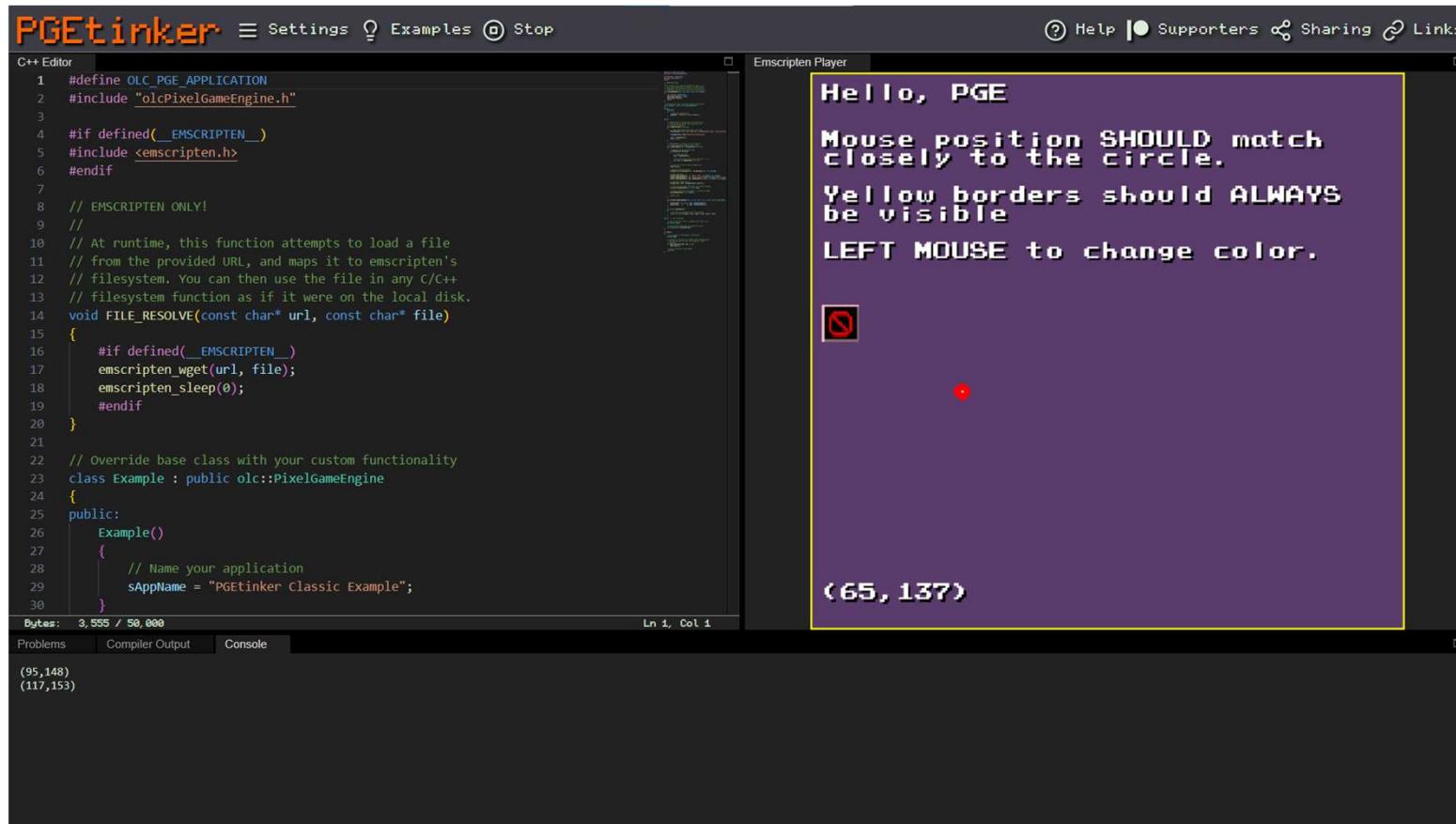
Single header include!!!!

Compiles to windows, mac, linux and webassembly

Online playground



<https://pgetinker.com/>



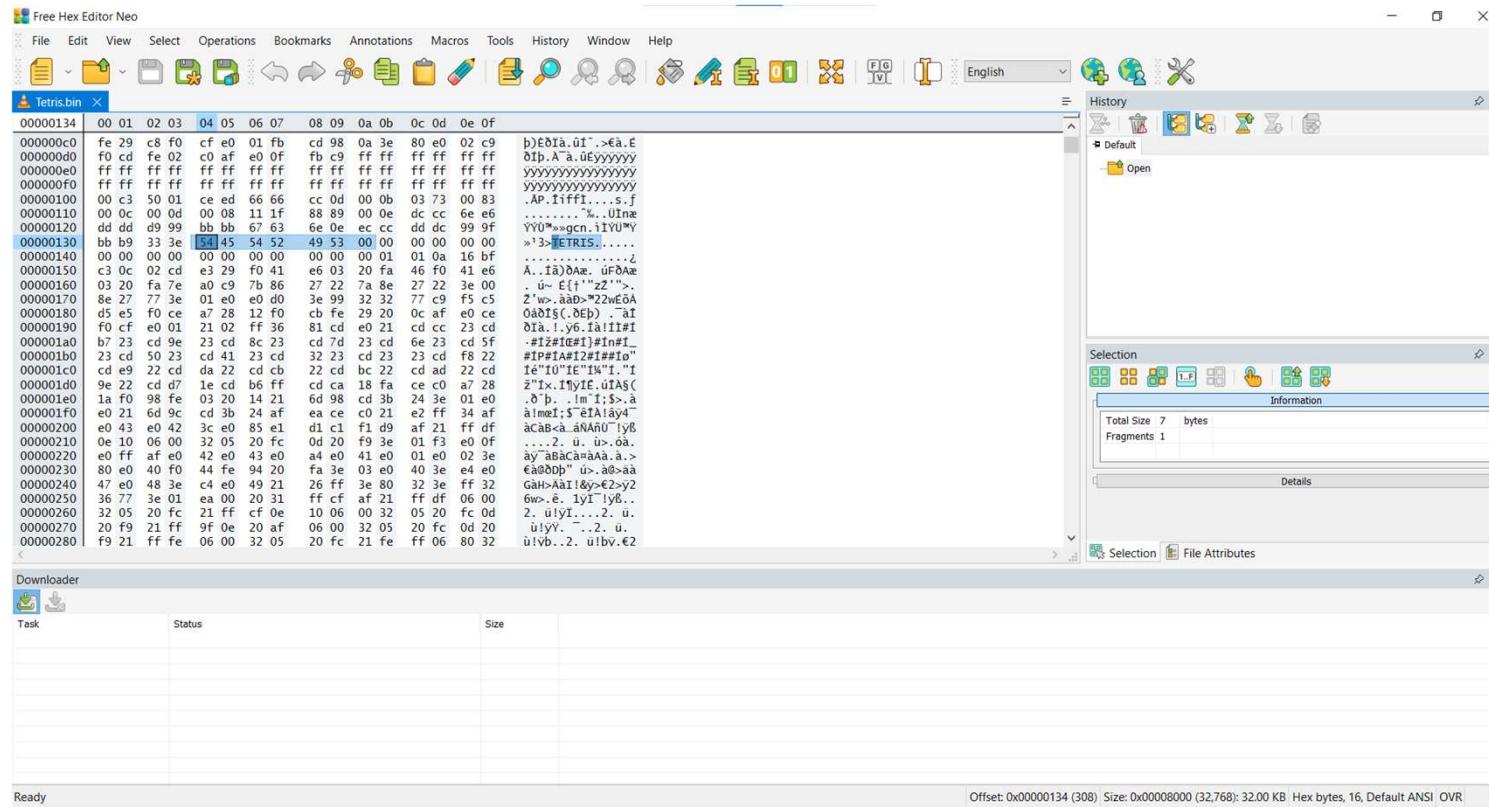
An empty application

```
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



```
1 #include "PGEboy.h"
2
3 #define OLC_PGE_APPLICATION
4 #include "olcPixelGameEngine.h"
5
6 Emulator emu{"Tetris.bin"};
7
8 class Example : public olc::PixelGameEngine {
9 public:
10    Example() { sAppName = emu.getName(); }
11
12 public:
13    bool OnUserCreate() override {
14        emu.Start();
15        return true;
16    }
17 }
```

```
55 int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
56                         PWSTR pCmdLine, int nCmdShow) {
57     Example demo;
58     if (demo.Construct(Emulator::xSizeScreen, Emulator::ySizeScreen, 4, 4, false, true))
59         demo.Start();
60
61     return 0;
62 }
```

```
1 #include "PGEboy.h"
2
3 #define OLC_PGE_APPLICATION
4 #include "olcPixelGameEngine.h"
5
6 Emulator emu{"Tetris.bin"};
7
8 class Example : public olc::PixelGameEngine {
9 public:
10     Example() { sAppName = emu.getName(); }
11
12 public:
13     bool OnUserCreate() override {
14         emu.Start();
15         return true;
16     }
17 }
```

```
18    bool OnUserUpdate(float fElapsed) override {
19        bool buttons[Emulator::nButtons]{};  
20  
21        buttons[0] = GetKey(olc::Key::UP).bHeld;  
22        buttons[1] = GetKey(olc::Key::DOWN).bHeld;  
23        buttons[2] = GetKey(olc::Key::LEFT).bHeld;  
24        buttons[3] = GetKey(olc::Key::RIGHT).bHeld;  
25        buttons[4] = GetKey(olc::Key::ENTER).bHeld;  
26        buttons[5] = GetKey(olc::Key::SPACE).bHeld;  
27        buttons[6] = GetKey(olc::Key::A).bHeld;  
28        buttons[7] = GetKey(olc::Key::D).bHeld;  
29  
30        static float totalElapsed{};  
31        totalElapsed += fElapsed;  
32        if (totalElapsed > 0.0597f) {  
33            totalElapsed -= 0.0597f;  
34            uint8_t screen[Emulator::xSizeScreen * Emulator::ySizeScreen];  
35            emu.getDisplay(screen, buttons, 1);  
36            for (int x = 0; x < ScreenWidth(); x++)  
37                for (int y = 0; y < ScreenHeight(); y++) {  
38                    auto index = (y * ScreenWidth() + x);  
39                    olc::Pixel color{olc::Pixel(15, 56, 15)}; // darkest green  
40                    if (screen[index] == 1) {  
41                        color = olc::Pixel(155, 188, 15); // lightest green  
42                    } else if (screen[index] == 2) {  
43                        color = olc::Pixel(139, 172, 15); // light green  
44                    } else if (screen[index] == 3) {  
45                        color = olc::Pixel(48, 98, 48); // dark green  
46                    }  
47                    Draw(x, y, color);  
48                }  
49            }  
50        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...



https://www.youtube.com/watch?v=4lliFwe5_yg

+ 24

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

TOM TESCH



Cppcon
The C++ Conference

20
24



September 15 - 20

What about us...



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

Simple 8-bit Assembler Simulator

Code (Instruction Set)

```
; Simple example
; Writes Hello World to the output

JMP start
hello: DB "Hello World!" ; Variable
      DB 0      ; String terminator

start:
      MOV C, hello    ; Point to var
      MOV D, 232     ; Point to output
      CALL print
      HLT          ; Stop execution

print:   ; print(C:<from, D:<to)
      PUSH A
      PUSH B
      MOV B, 0

.loop:
      MOV A, [C]    ; Get char from var
      MOV [D], A    ; Write to output
      INC C
      INC D
      CMP B, [C]    ; Check if end
      JNZ .loop    ; jump if not

      POP B
      POP A
      RET
```

Output

CPU & Memory

Registers / Flags

A	B	C	D	IP	SP	Z	C	F
00	00	00	00	00	E7	FALSE	FALSE	FALSE

RAM

Clock speed: 4 Hz Instructions: Hide View: Decimal
Register addressing: A: Show B: Show C: Show D: Show

Labels

Name	Address	Value
------	---------	-------

Assemble

Registers / Flags									
A	B	C	D	IP	SP	Z	C	F	
00	00	00	00	0F	E7	FALSE	FALSE	FALSE	

RAM

instructions



↑

display
ess 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 (**Zero**, **Carry** and **Fault**)
- 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

Function calling

CALL : function call

RET : return from function

Shift operations

SHL : shift left

SHR : shift right

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 the A register!

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

The screenshot shows the regex101.com interface. The URL in the address bar is `https://regex101.com`. The main area displays a regular expression pattern: `/([0-9A-F]{2})\n`. The results section indicates 52 matches found in 0.3ms. A modal window titled "Flavor Help" provides detailed information about the pattern. It includes a "Reference table" comparing support across various languages and tools, and an "EXPLANATION" section with a breakdown of the regex components. The "EXPLANATION" section highlights the first capturing group `([0-9A-F]{2})`, which matches a single character present in the list below `[0-9A-F]`. The "FUNCTION" sidebar on the left lists Match, Substitution, List, and Unit Tests.

REGULAR EXPRESSION

`/([0-9A-F]{2})\n`

TEST STRING

01
06
01
08
03
09
02
05
03
09
12
02
12

SUBSTITUTION

0x\$1,

0x1F, 0x10, 0x48, 0x65, 0x6C, 0x66, 0x02, 0x02, 0x06, 0x03, 0xE8, 0x30, 0x02, 0x05, 0x03, 0x00, 0x12, 0x00, 0x39,

Flavor Help

You can read more here.

Reference table

Language	RegEx101 Support	Comments
JavaScript	Full	Uses your browser's 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 Google's 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.

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)

REPLACEMENT

0x\$1,

CHARACTER INFORMATION

BACK REFERENCES

- 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{00fa}
- Hexadecimal replacement values: \u00fa
- Insert a tab: \t
- Insert a carriage return: \r
- Insert a newline: \n

Flags/Modifiers

- Substitution

REGEX 101

The screenshot shows the regex101.com interface. The URL in the address bar is `https://regex101.com`. The main area displays a regular expression `/([0-9A-F]{2})\n` with the flavor set to ECMAScript (JavaScript). The test string consists of 52 matches (0.3ms) containing pairs of hex digits followed by a newline character. The substitution field contains `0x$1,*`, resulting in a success (0.2ms) output of a long string of hex values separated by commas. The explanation panel details the regex components: `([0-9A-F]{2})` matches exactly two characters from the sets 0-9 and A-F, and `\n` matches a line-feed character. Global pattern flags `g` and `m` are also explained. The match information and quick reference panels provide additional details on regex syntax.

<https://godbolt.org/z/MKbadKde6>

```
57 ~ int main() {
58     const std::regex hex_line{R"(([0-9A-F]{2})\n)"};
59     const auto transformed = std::regex_replace(input, hex_line, "0x$1, ");
60     std::cout << transformed << '\n';
61 }
```

```
1 #include <iostream>
2 #include <regex>
3
4 ~ constexpr auto input{
5     R"(1F
6     0F
7     48
8     65
9     6C
10    6C
11    6F
12    20
13    57
```

A first naïve implementation

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

```
int main() {
    CPU::load_program(Program);
    while (CPU::execute_instruction() != CPU::Opcode::hlt) {
    }
    CPU::display_state();
    return 1;
}
```

DONT REPEAT YOURSELF



First rule of coding

WETT

← → *Write Everything Twice* ← →

Write every time!

We Enjoy Typing!

WETT

← → *Write Everything Twice* → →

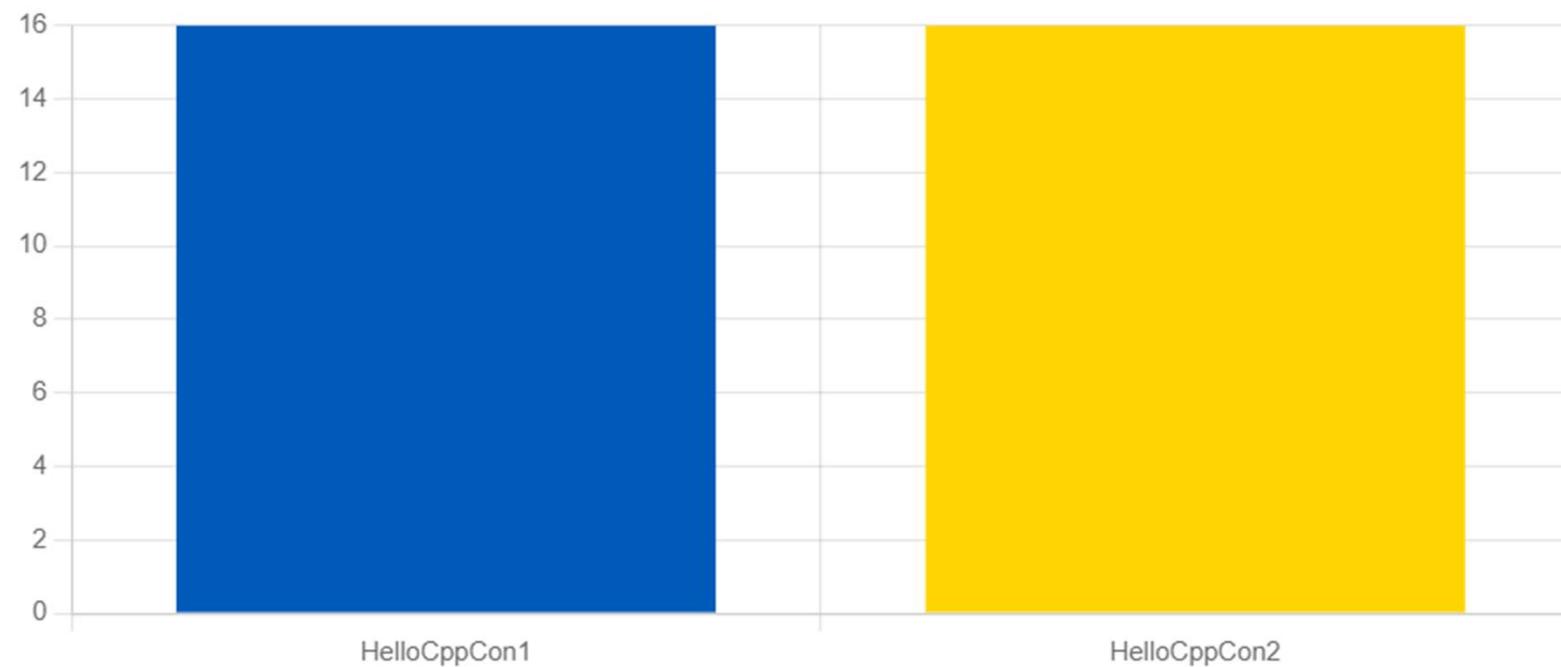
Waste Everyones Time

DRY and exceptions

<https://godbolt.org/z/WadGrWPKx>

```
int main() {
    CPU::load_program(Program);
    try {
        while (CPU::execute_instruction() != CPU::Opcode::hlt) {
        }
    } catch (CPU::InvalidOpcodeException ioe) {
        std::cout << "Invalid Opcode: " << ioe.which() << "\n";
    } catch (CPU::InvalidRegisterException ire) {
        std::cout << "Invalid Register: " << ire.which() << "\n";
    }
    CPU::display_state();
    return 1;
}
```

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



<https://www.youtube.com/watch?v=bY2FlayomlE>

A dark blue slide featuring a presentation title and logos. In the top right corner, there is a green graphic element consisting of a white plus sign and the number '24'. The main title 'C++ Exceptions for Smaller Firmware' is centered in large yellow text. Below the title, the speaker's name 'KHALIL ESTELL' is displayed in smaller yellow text. At the bottom left, the Cppcon logo (a white circle with a stylized '++' symbol) and the text 'Cppcon The C++ Conference' are shown. At the bottom right, the year '2024' is displayed vertically next to a white mountain icon with a yellow peak, and the text 'September 15 - 20'.

C++ Exceptions for Smaller Firmware

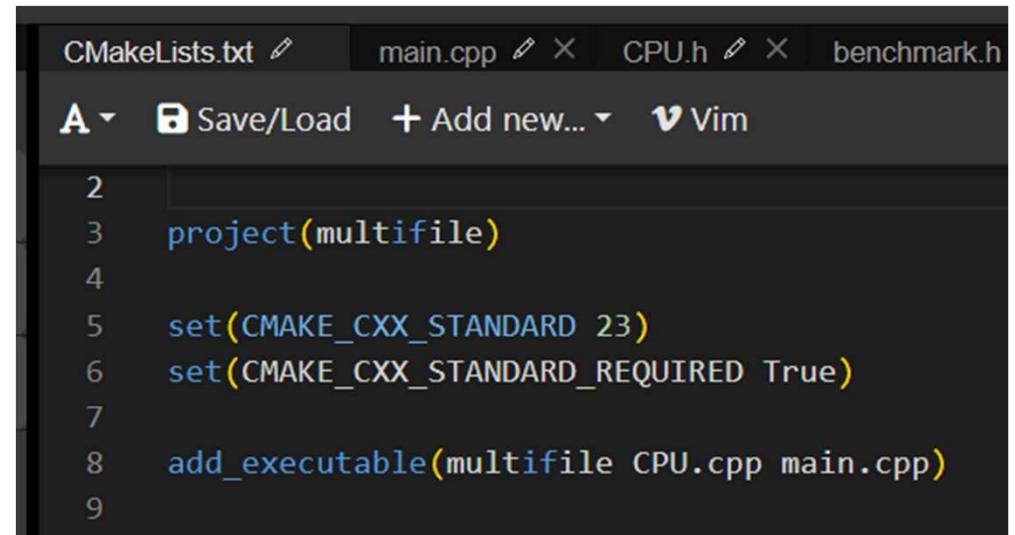
KHALIL ESTELL

2024 | September 15 - 20

Making a class

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

<https://godbolt.org/z/Mzdd6n1fW>



The screenshot shows the Godbolt CMake interface with four tabs at the top: CMakeLists.txt, main.cpp, CPU.h, and benchmark.h. Below the tabs is a toolbar with icons for Save/Load, Add new..., and Vim. The code editor displays the CMakeLists.txt file with the following content:

```
2
3 project(multifile)
4
5 set(CMAKE_CXX_STANDARD 23)
6 set(CMAKE_CXX_STANDARD_REQUIRED True)
7
8 add_executable(multifile CPU.cpp main.cpp)
9
```

Aside: avoiding naked for statements

<https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es71-prefer-a-range-for-statement-to-a-for-statement-when-there-is-a-choice>

ES.71: Prefer a range- for -statement to a for -statement when there is a choice

Reason Readability. Error prevention. Efficiency.

<https://www.youtube.com/watch?v=W2tWOdgzXHA>



"If you want to improve code quality in your organization, I would say, take all your coding guidelines and replace them with the one goal. That's how important I think this one goal is:

No Raw Loops.

This will make the biggest change in code quality within your organization."

(Sean Parent, C++ Seasoning, Going Native 2013)

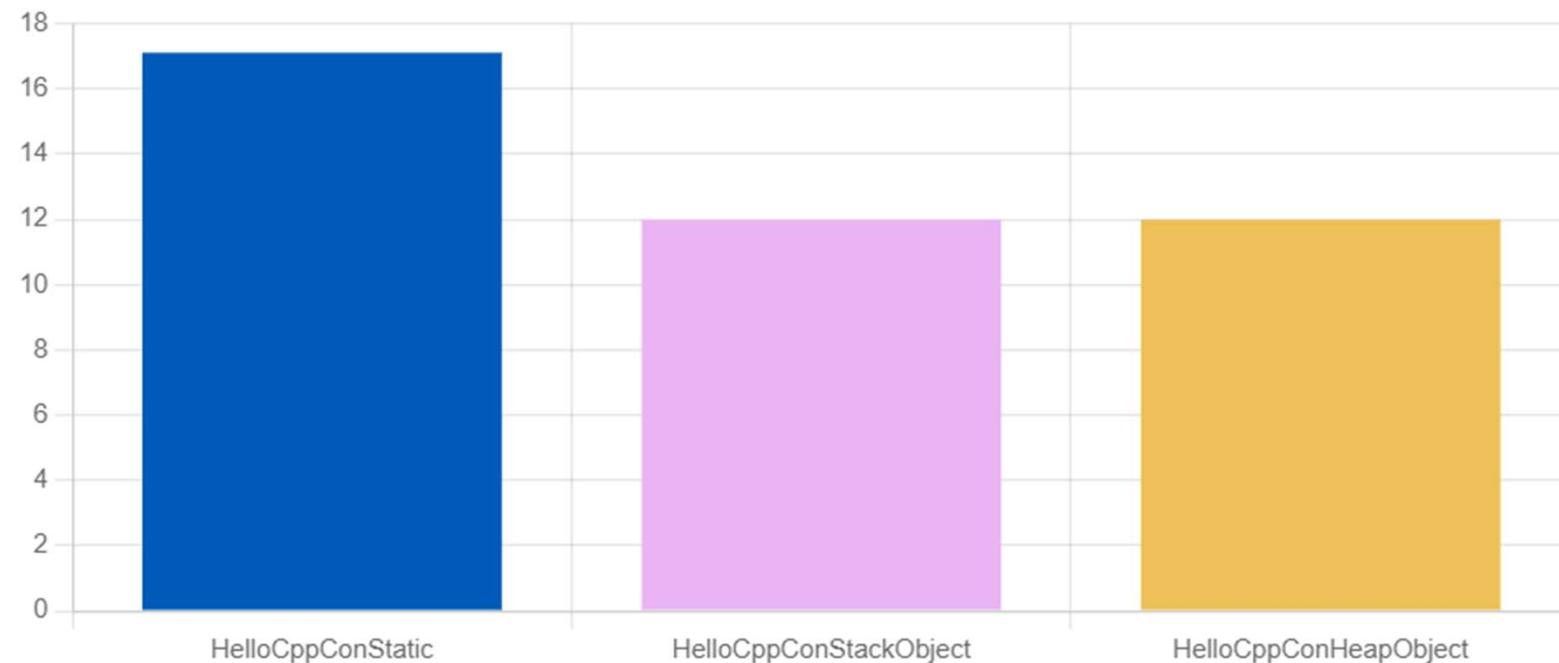
Aside: avoiding naked for statements

```
void CPU::load_program(std::span<uint8_t const> program) {
    for (size_t idx{}; idx < program.size(); ++idx) {
        m_memory[idx] = program[idx];
    }
}
```

Aside: avoiding naked for statements (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/4Igc8EFV-7bBkZoORTS1hXWM8Vo>



Disaster strikes

The screenshot shows a browser window with the title "Quick C++ Benchmarks". The address bar displays the URL <https://quick-bench.com>. The main content area is titled "Quick C++ Benchmark". On the left, there is a code editor containing C++ code for benchmarking string operations. On the right, there are configuration options for the compiler (Clang 17.0), standard (c++20), optimization level (O3), STL (libstdc++(GNU)), and other flags. Below these are buttons for "Run Benchmark", "Disassembly = AT&T", and "Clear cached results". A "Run Quick Bench locally" button is also present. The output section shows an error message: "Error or timeout" followed by "/bin/sh: 1: Cannot fork". This error is repeated in a larger box below.

```
1 static void StringCreation(benchmark::State& state) {
2     // Code inside this loop is measured repeatedly
3     for (auto _ : state) {
4         std::string created_string("hello");
5         // Make sure the variable is not optimized away by compiler
6         benchmark::DoNotOptimize(created_string);
7     }
8 }
9 // Register the function as a benchmark
10 BENCHMARK(StringCreation);
11
12 static void StringCopy(benchmark::State& state) {
13     // Code before the loop is not measured
14     std::string x = "hello";
15     for (auto _ : state) {
16         std::string copy(x);
17     }
18 }
19 BENCHMARK(StringCopy);
20
```

Run Quick Bench locally Support Quick Bench Suite ▾ More ▾

compiler = Clang 17.0 ▾ std = c++20 ▾ optim = O3 ▾ STL = libstdc++(GNU) ▾ Other flags ▾

Run Benchmark Disassembly = AT&T ▾ Clear cached results

Error or timeout
/bin/sh: 1: Cannot fork

Error or timeout
/bin/sh: 1: Cannot fork



benchmark.h to the rescue

```
1  #ifndef benchmark_h
2  #define benchmark_h
3
4  #include <algorithm>
5  #include <chrono>
6  #include <cmath>
7  #include <iostream>
8  #include <numeric>
9  #include <utility>
10 #include <vector>
11
12 template <typename Func>
13     requires std::invocable<Func>
14 auto measure_runtime(Func func, size_t runs = 10,
15                     int number_of_repetitions = 100) {
16     using namespace std::chrono;
17     std::vector<nanoseconds> times(runs);
18
19     std::generate(times.begin(), times.end(), [&]() {
20         auto start = high_resolution_clock::now();
21         for (int i{}; i < number_of_repetitions; ++i) func();
22         auto end = high_resolution_clock::now();
23         return duration_cast<nanoseconds>(end - start);
24     });
25 }
```

<https://godbolt.org/z/WarhrqGva>

```
26     std::sort(times.begin(), times.end());
27
28     auto discard_count = static_cast<int>(std::sqrt(runs) / 2);
29
30     auto sum = std::accumulate(times.begin() + discard_count,
31                               times.end() - discard_count, nanoseconds{0});
32     auto average = sum / (times.size() - discard_count * 2);
33
34     auto square_sum = std::accumulate(times.begin() + discard_count,
35                                      times.end() - discard_count, int64_t{0},
36                                      [average](int64_t acc, nanoseconds time) {
37             int64_t diff = (time - average).count();
38             return acc + diff * diff;
39         });
40
41     auto standard_deviation = nanoseconds{static_cast<long long>(std::sqrt(
42         square_sum / static_cast<double>(times.size() - discard_count * 2)))};
43
44     return std::pair{average, standard_deviation};
45 }
46
47 #endif
```

Compile Time Analysis...

Function pointers are `constexpr`! <https://godbolt.org/z/WE7Kz4Y4v>

```
1 #include <iostream>
2
3 int add(int a, int b) { return a + b; }
4
5 int main() {
6     // constexpr int (*funcPtr)(int, int) = add;
7     constexpr auto funcPtr{add};
8
9     std::cout << "Result: " << funcPtr(2, 3) << '\n';
10
11    return 0;
12 }
```

const, constexpr, consteval and constinit

Keyword	Introduced	Can be used with	Meaning
const	C++98	Variables, methods, pointers	Value cannot be changed after initialization.
constexpr	C++11	Variables, functions, constructors	Compile-time constant (if possible).
consteval	C++20	Functions only	Must be evaluated at compile-time.
constinit	C++20	Variables (usually static or global)	Ensures compile-time initialization, but value can change later.

<https://godbolt.org/z/fqY7Y1Wof>

Why Can a Function Pointer Be `constexpr`?

- Function identity is known at compile time

Even if the address isn't, the compiler knows which function is being referenced.
- `constexpr` means fixed target, not fixed address

The function being pointed to is constant — not its memory location.
- Linker fills in addresses later

The compiler emits relocation info; actual addresses are assigned at link/load time.
- Analogy: Reserved seat

You know you're in seat A3 — even if you don't know where that seat ends up in the venue.

Aside #embed – a Quine

```
1 #include <iostream>
2
3 constexpr char self[] {
4     #embed __FILE__
5 };
6
7 int main() {
8     for (char c : self) std::cout << c;
9 }
```

<https://godbolt.org/z/b4x3147xz>

```
Executor x86-64 clang (trunk) (C++, Editor #1) ✘ X
A - Wrap lines ⌂ 🔒 ⚙️ ➔ 🔍 ⌂
x86-64 clang (trunk) ⌂ ✓ -std=c++26 -Wno-c23-extensions
Program returned: 0
Program stdout
#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!

Compile Time Analysis to a map

Goal: make a map containing the location in memory, a pointer to the function to execute, and the parameters to that function

<https://godbolt.org/z/YYKdYTGP6>

Analyse to map

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

```
template <std::size_t n_instructions>
constexpr 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;
}
```

```
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);
}

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 :-(";
    }
}
```

```
using cpu_method_ptr = void (CPU::*)(uint8_t, uint8_t);

consteval static cpu_method_ptr get_right_method(Opcode opcode) {
    switch (opcode) {
        case CPU::Opcode::mov_reg_to_reg: {
            return &CPU::mov_reg_to_reg;
        }
        case CPU::Opcode::mov_reg_to_address: {
            return &CPU::mov_reg_to_address;
        }
        case CPU::Opcode::mov_regaddress_to_reg: {
            return &CPU::mov_regaddress_to_reg;
        }
        case CPU::Opcode::mov_reg_to_regaddress: {
            return &CPU::mov_reg_to_regaddress;
        }
        case CPU::Opcode::mov_number_to_reg: {
            return &CPU::mov_number_to_reg;
        }
        case CPU::Opcode::inc_reg: {
            return &CPU::inc_reg;
        }
        case CPU::Opcode::cmp_reg_with_reg: {
            return &CPU::cmp_reg_with_reg;
        }
        case CPU::Opcode::cmp_regaddress_with_reg: {
            return &CPU::cmp_regaddress_with_reg;
        }
        case CPU::Opcode::jmp_address: {
            return &CPU::jmp_address;
        }
        case CPU::Opcode::jnz_address: {
            return &CPU::jnz_address;
        }
        case CPU::Opcode::push_reg: {
            return &CPU::push_reg;
        }
        case CPU::Opcode::pop_reg: {
            return &CPU::pop_reg;
        }
        case CPU::Opcode::call_address: {
            return &CPU::call_address;
        }
        case CPU::Opcode::ret: {
            return &CPU::ret;
        }
        case CPU::Opcode::hlt: {
            return &CPU::hlt;
        }
        default:
            throw "stop compilation! 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;
}
```

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

```
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]};  
}
```

Failure 😞

```
Average runtime for myCPU: 6.291ms, standard deviation: 892ns.  
Average runtime for otherCPU: 8.573ms, standard deviation: 1288ns.
```

Compile Time Analysis to an 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,  
})};
```

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,  
})};
```

```
constexpr auto Program{std::to_array<uint8_t>({  
#embed "/code/hello.bin"  
})};
```

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

```
Some calculated value to avoid compiling things away 230 [should be identical for all
Average runtime for runtime: 1ms, standard deviation: 271us.
```

```
Some calculated value to avoid compiling things away 230 [should be identical for all
Average runtime for compiletime (to map): 4ms, standard deviation: 148us.
```

```
Some calculated value to avoid compiling things away 230 [should be identical for all
Average runtime for compiletime (to array): 2ms, standard deviation: 1077us.
```

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.
```

Three nested loops for long execution

The screenshot shows a CPU simulation interface with the following sections:

- Code (Instruction Set):** Displays the assembly code:

```
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
```
- Output:** Shows a 16x8 grid of gray squares.
- CPU & Memory:**
 - Registers / Flags:** Shows the state of registers A, B, C, D, IP, SP, Z, C, and F. IP is at address E6, SP is at E7, and all other flags are FALSE.
 - RAM:** Displays memory starting at address 00. The first few bytes are 06, 00, 00, 06, 01, 21, 06, 02, 00, 06, 03, 00, 12, 00, 12, 02, followed by many zeros. Address E6 contains the value 06, and address E7 contains the value 07.

On my 3.2 GHz machine:

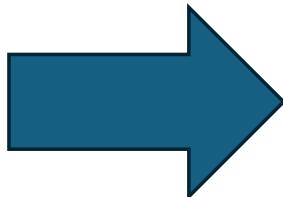
50529029 instructions
in 200milliseconds

→ 250MIPS

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
```



iBeC++

```
__asm {
    mov al, 0
    mov ebx, offset zero
    mov cl, 0
    mov dl, 0
loop1:
    inc al
loop2:
    inc cl
loop3:
    inc dl
    cmp dl, [ebx]
    jne loop3
    cmp cl, [ebx]
    jne loop2
    cmp al, [ebx]
    jne loop1
}
```

On my 3.2 GHz machine:

50529029 instructions
in 200milliseconds

→ 250MIPS

```
__asm {
    mov al, 0
    mov ebx, offset zero
    mov cl, 0
    mov dl, 0
loop1:
    inc al
loop2:
    inc cl
loop3:
    inc dl
    cmp dl, [ebx]
    jne loop3
    cmp cl, [ebx]
    jne loop2
    cmp al, [ebx]
    jne loop1
}
```

```
#include <chrono>
#include <iostream>

int main() {
    // Must be static so MSVC __asm can use 'offset zero'
    static unsigned char zero{};

    auto start{std::chrono::high_resolution_clock::now()};
    __asm { ... }
    auto end{std::chrono::high_resolution_clock::now()};

    auto elapsed_us{
        std::chrono::duration_cast<std::chrono::microseconds>(end - start)
            .count()};

    constexpr auto instruction_count{50'529'029.0};
    auto elapsed_seconds{static_cast<double>(elapsed_us) / 1'000'000.0};
    auto mips{instruction_count / (elapsed_seconds * 1e6)};

    std::cout << "Executed " << instruction_count << " instructions in "
        << elapsed_us << " microseconds.\n";
    std::cout << "MIPS: " << mips << "\n";
}

return 0;
}
```

```
__asm {
    mov al, 0
    mov ebx, offset zero
    mov cl, 0
    mov dl, 0
loop1:
    inc al
loop2:
    inc cl
loop3:
    inc dl
    cmp dl, [ebx]
    jne loop3
    cmp cl, [ebx]
    jne loop2
    cmp al, [ebx]
    jne loop1
}
```

**“Only” 40x
slower**

```
|Executed 5.0529e+07 instructions in 4872 microseconds.
|MIPS: 10371.3
```

```
#include <chrono>
#include <iostream>

int main() {
    // Must be static so MSVC __asm can use 'offset zero'
    static unsigned char zero{};

    auto start{std::chrono::high_resolution_clock::now()};
    __asm { ... }
    auto end{std::chrono::high_resolution_clock::now()};

    auto elapsed_us{
        std::chrono::duration_cast<std::chrono::microseconds>(end - start)
            .count()};

    constexpr auto instruction_count{50'529'029.0};
    auto elapsed_seconds{static_cast<double>(elapsed_us) / 1'000'000.0};
    auto mips{instruction_count / (elapsed_seconds * 1e6)};

    std::cout << "Executed " << instruction_count << " instructions in "
        << elapsed_us << " microseconds.\n";
    std::cout << "MIPS: " << mips << "\n";
}

return 0;
}
```

Thanks!!!

- **You** – both here at CppNorth and watching this online!
- **The CppNorth 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



DNE
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

