Building a GameBoy Emulator in Rust

by Baptiste Jacquemot

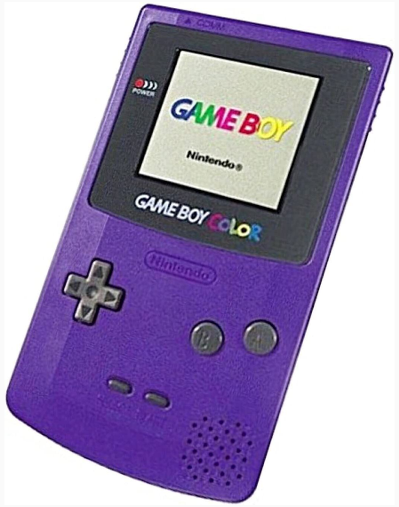
About Myself

- French
- Arrived in Switzerland in 2017
- Finished my master at EPFL/ETH in cyber security
- Competitive programmer and CTF player

The GameBoy

GameBoy 1989

GameBoy Color 1998



Why creating a gameboy emulator?

Answer: It’s fun!

First year project

- Took 3 months
- No sound, no color, part of the code is already given

This project

- (re)Learn rust
- Do it faster, Go further (run it on bare metal)

State of the project

Done

- MMU, CPU, Joypad, Lcd screen, most common cartridges types

To Do:

- Sound, gameboy color features, other cartridges types

Where to begin

Memory management unit (MMU)

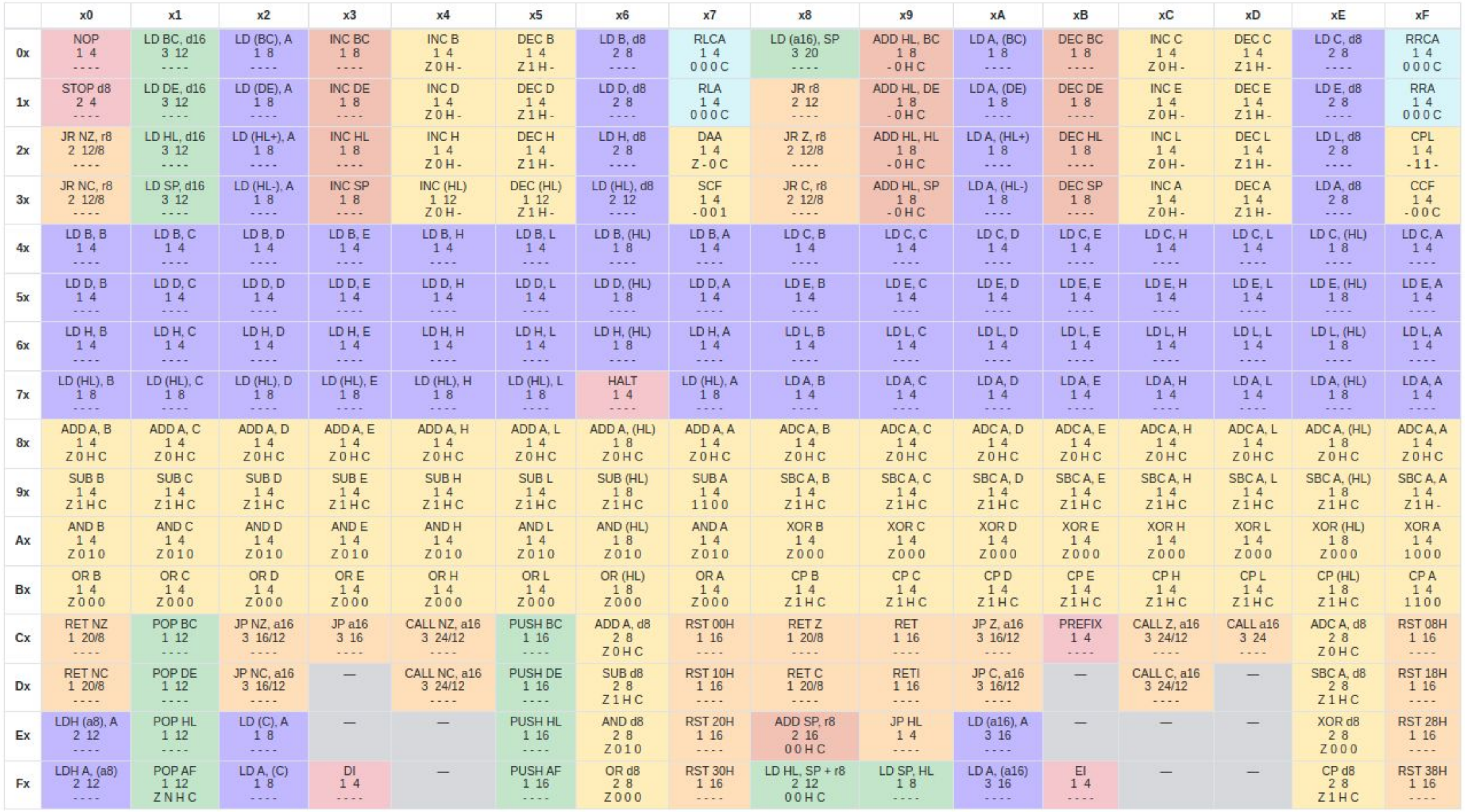
- 16 bits address, 8 bit value

CPU: similar to Zilog z80 and intel 8080.

- 8 8bits registers: A, B, C, D, E, F, H, L
- 2 16bits registers: SP, PC

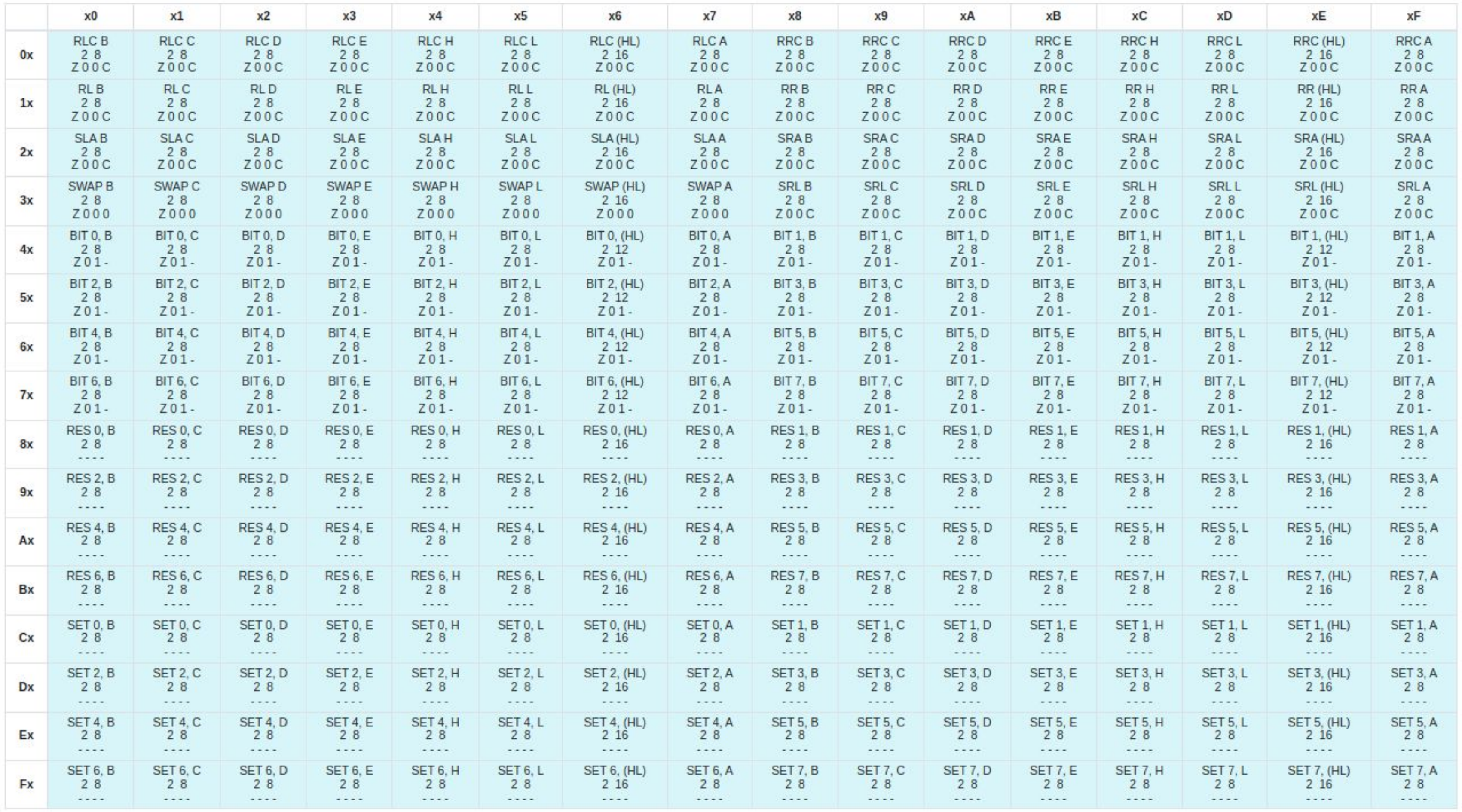
	Clock frequency	Release year
Sharp LR35902 core	4.19MHz	1989
Intel 8080	3.125 MHz	1974

https://www.nintendo.co.uk/Support/Game-Boy-Pocket-Color/Product-information/Technical-data/Technical-data-619585.html



	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NOP 1 4 - - - -	LD BC, d16 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, d8 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, d8 2 8 - - - -	RRCA 1 4 0 0 0 C
1x	STOP d8 2 4 - - - -	LD DE, d16 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, d8 2 8 - - - -	RLA 1 4 0 0 0 C	JR r8 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, d8 2 8 - - - -	RRA 1 4 0 0 0 C
2x	JR NZ, r8 2 12/8 - - - -	LD HL, d16 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, d8 2 8 - - - -	DAA 1 4 Z - 0 C	JR Z, r8 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, d8 2 8 - - - -	CPL 1 4 - 1 1 -
3x	JR NC, r8 2 12/8 - - - -	LD SP, d16 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), d8 2 12 - - - -	SCF 1 4 - 0 0 1	JR C, r8 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, d8 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 B 1 4 Z 1 H C	SUB C 1 4 Z 1 H C	SUB D 1 4 Z 1 H C	SUB E 1 4 Z 1 H C	SUB H 1 4 Z 1 H C	SUB L 1 4 Z 1 H C	SUB (HL) 1 8 Z 1 H C	SUB 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 B 1 4 Z 0 1 0	AND C 1 4 Z 0 1 0	AND D 1 4 Z 0 1 0	AND E 1 4 Z 0 1 0	AND H 1 4 Z 0 1 0	AND L 1 4 Z 0 1 0	AND (HL) 1 8 Z 0 1 0	AND A 1 4 Z 0 1 0	XOR B 1 4 Z 0 0 0	XOR C 1 4 Z 0 0 0	XOR D 1 4 Z 0 0 0	XOR E 1 4 Z 0 0 0	XOR H 1 4 Z 0 0 0	XOR L 1 4 Z 0 0 0	XOR (HL) 1 8 Z 0 0 0	XOR A 1 4 1 0 0 0
Bx	OR B 1 4 Z 0 0 0	OR C 1 4 Z 0 0 0	OR D 1 4 Z 0 0 0	OR E 1 4 Z 0 0 0	OR H 1 4 Z 0 0 0	OR L 1 4 Z 0 0 0	OR (HL) 1 8 Z 0 0 0	OR A 1 4 Z 0 0 0	CP B 1 4 Z 1 H C	CP C 1 4 Z 1 H C	CP D 1 4 Z 1 H C	CP E 1 4 Z 1 H C	CP H 1 4 Z 1 H C	CP L 1 4 Z 1 H C	CP (HL) 1 8 Z 1 H C	CP 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, d8 2 8 Z 0 H C	RST 00H 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, d8 2 8 Z 0 H C	RST 08H 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 d8 2 8 Z 1 H C	RST 10H 1 16 - - - -	RET C 1 20/8 - - - -	RETI 1 16 - - - -	JP C, a16 3 16/12 - - - -	—	CALL C, a16 3 24/12 - - - -	—	SBC A, d8 2 8 Z 1 H C	RST 18H 1 16 - - - -
Ex	LDH (a8), A 2 12 - - - -	POP HL 1 12 - - - -	LD (C), A 1 8 - - - -	—	—	PUSH HL 1 16 - - - -	AND d8 2 8 Z 0 1 0	RST 20H 1 16 - - - -	ADD SP, r8 2 16 0 0 H C	JP HL 1 4 - - - -	LD (a16), A 3 16 - - - -	—	—	—	XOR d8 2 8 Z 0 0 0	RST 28H 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 d8 2 8 Z 0 0 0	RST 30H 1 16 - - - -	LD HL, SP + r8 2 12 0 0 H C	LD SP, HL 1 8 - - - -	LD A, (a16) 3 16 - - - -	EI 1 4 - - - -	—	—	CP d8 2 8 Z 1 H C	RST 38H 1 16 - - - -

opcodes (https://gbdev.io/gb-opcodes/optables/)

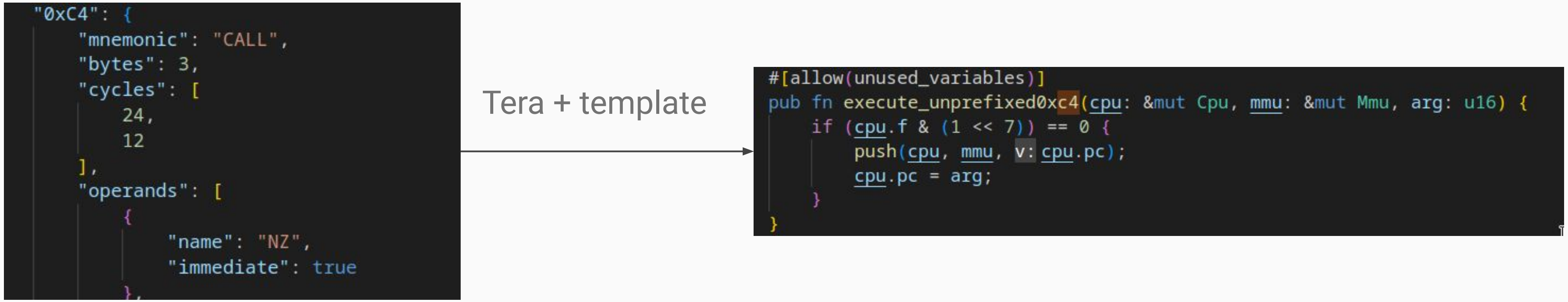


	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	RLC B 2 8 Z 0 0 C	RLC C 2 8 Z 0 0 C	RLC D 2 8 Z 0 0 C	RLC E 2 8 Z 0 0 C	RLC H 2 8 Z 0 0 C	RLC L 2 8 Z 0 0 C	RLC (HL) 2 16 Z 0 0 C	RLC A 2 8 Z 0 0 C	RRC B 2 8 Z 0 0 C	RRC C 2 8 Z 0 0 C	RRC D 2 8 Z 0 0 C	RRC E 2 8 Z 0 0 C	RRC H 2 8 Z 0 0 C	RRC L 2 8 Z 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 - - - -

Prefixed opcode (https://gbdev.io/gb-opcodes/optables/)

How to implement the CPU?

- Implement each opcode
- Add a layer of abstraction in the code
- Generate the cpu code



```
"0xC4": {
    "mnemonic": "CALL",
    "bytes": 3,
    "cycles": [
        24,
        12
    ],
    "operands": [
        {
            "name": "NZ",
            "immediate": true
        },
```

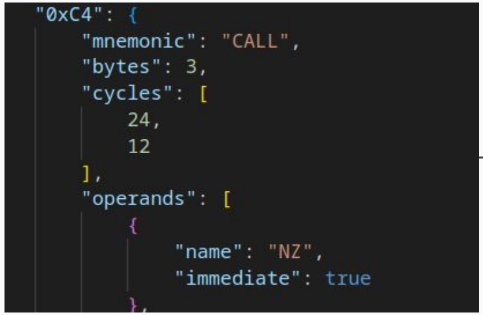
```rust
#[allow(unused_variables)]
pub fn execute_unprefixed0xc4(cpu: &mut Cpu, mmu: &mut Mmu, arg: u16) {
    if (cpu.f & (1 << 7)) == 0 {
        push(cpu, mmu, v: cpu.pc);
        cpu.pc = arg;
    }
}
```

Debugging

Does the cpu work? 2 steps

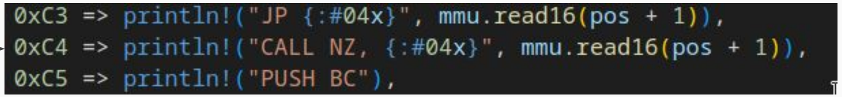
- Create some Test / or use tested ROM (blargg test)
- Realise it does not work

Create a debugging tool “gdb” like. Create a disassembler



```
"0xC4": {
    "mnemonic": "CALL",
    "bytes": 3,
    "cycles": [
        24,
        12
    ],
    "operands": [
        {
            "name": "NZ",
            "immediate": true
        },
```

Tera + template



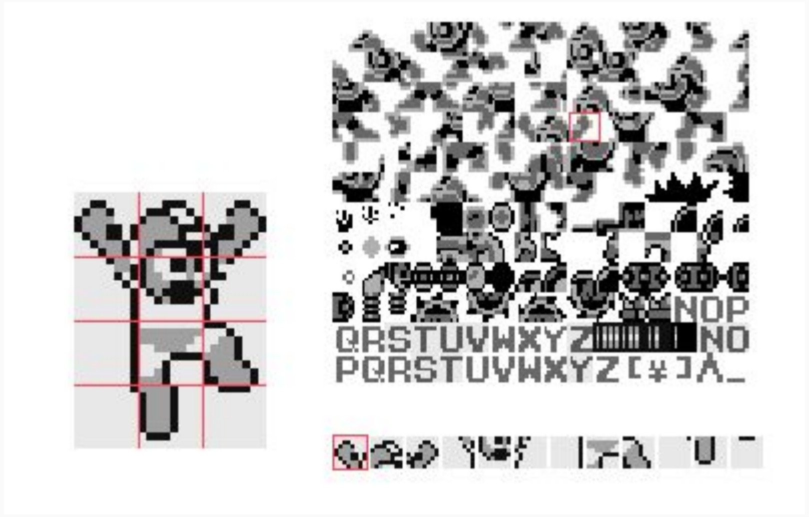
```
0xC3 => println!("JP {:#04x}", mmu.read16(pos + 1)),
0xC4 => println!("CALL NZ, {:#04x}", mmu.read16(pos + 1)),
0xC5 => println!("PUSH BC"),
```

Direct Memory Access (DMA)

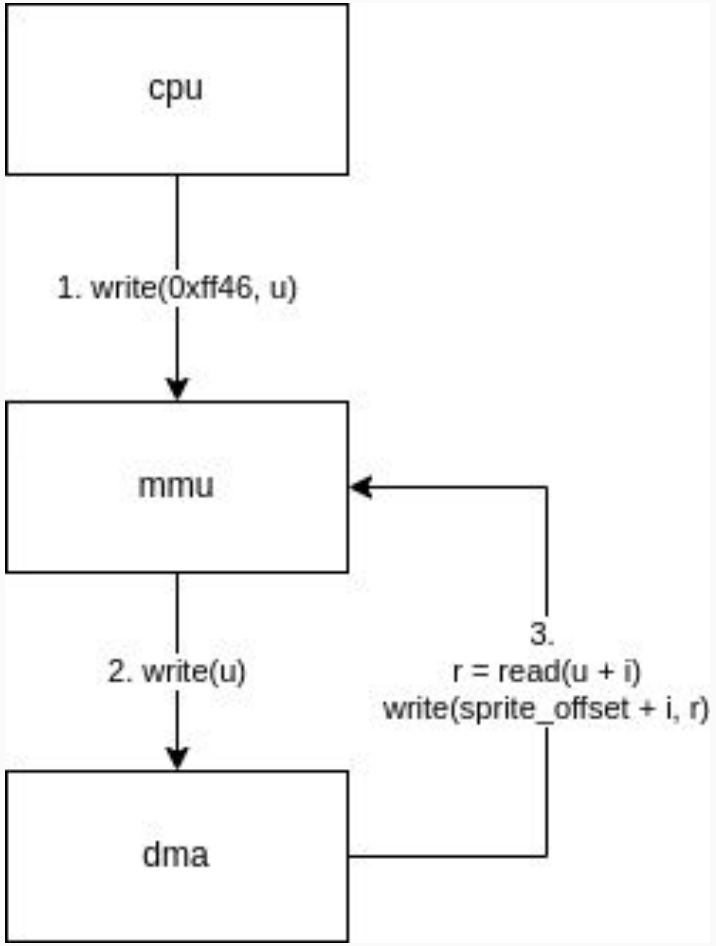
Part of the LCD controller.

Fast copy 160 bytes from anywhere in memory to the sprite memory.

Is triggered by writing to address 0xff46



https://gbdev.gg8.se/wiki/articles/File:Z99zL9R.png

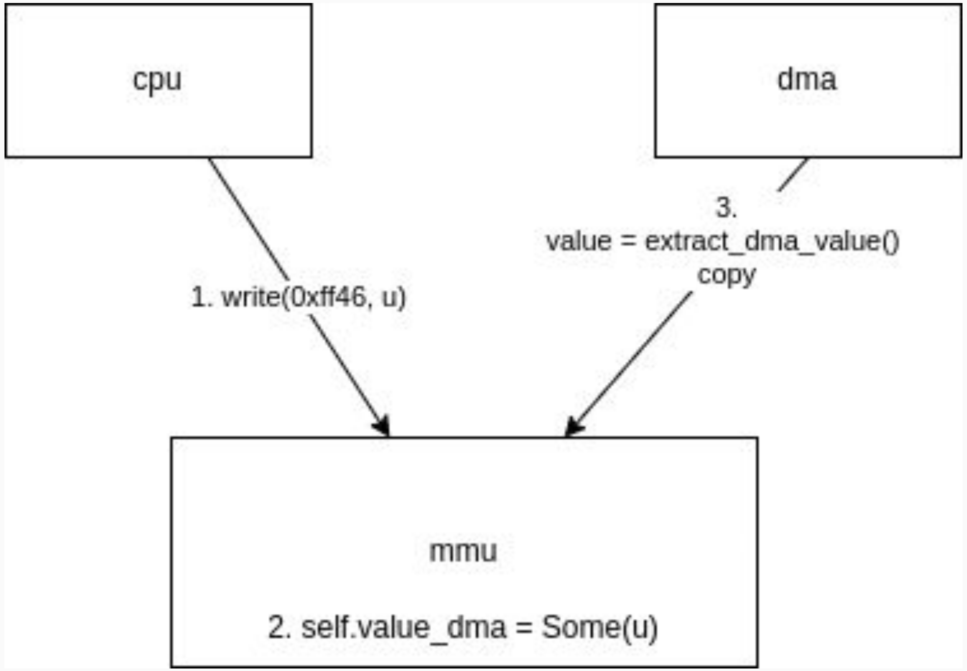


My intuitive implementation

"At any given time, you can have either one mutable reference or any number of immutable references."

Rust Rules of References

https://doc.rust-lang.org/stable/book/ch04-02-references-and-borrowing.html?highlight=mutable#mutable-references



My rust implementation

Demo Time!

Links

check my github:
https://github.com/bajacc/rust-gameboy

My website: bajac.xyz

- The Ultimate GameBoy talk
- Game Boy unofficial wiki
- Game Boy CPU manual
- cs108: The EPFL project in java

Thanks!

Any Questions?