SM83 instruction encoding table (unprefixed)

7	6	5	4	3	2	1	0	instruction
0	0	0	0	0	0	0	0	NOP
0	0	0	0	1	0	0	0	LD (u16), SP
0	0	0	1	0	0	0	0	STOP
0	0	0	1	1	0	0	0	JR (unconditional)
0	0	1	condition		0	0	0	JR (conditional)
0	0	r16 (group 1)		0	0	0	1	LD r16, u16
0	0	r16 (group 1)		1	0	0	1	ADD HL, r16
0	0	r16 (group 2)		0	0	1	0	LD (r16), A
0	0	r16 (group 2)		1	0	1	0	LD A, (r16)
0	0	r16 (group 1)		0	0	1	1	INC r16
0	0	r16 (group 1)		1	0	1	1	DEC r16
0	0	r8			1	0	0	INC r8
0	0	r8			1	0	1	DEC r8
0	0	r8			1	1	0	LD r8, u8
0	0	opcode (group 1)			1	1	1	(see opcode group 1 below)
0	1	1	1	0	1	1	0	HALT
0	1	r8 (destination)			r8 (source)			LD r8, r8 (see notes: 1)
1	0	opcode (group 2)			r8 (operand 2)			ALU A, r8
1	1	0	condition		0	0	0	RET condition
1	1	1	0	0	0	0	0	LD (FF00 + u8), A
1	1	1	0	1	0	0	0	ADD SP, i8
1	1	1	1	0	0	0	0	LD A, (FF00 + u8)
1	1	1	1	1	0	0	0	LD HL, SP + i8
1	1	r16 (group 3)		0	0	0	1	POP r16
1	1	opcode (see segment to the right)		1	0	0	1	0: RET, 1: RETI, 2: JP HL, 3: LD SP, HL

1	1	0	condition		0	1	0	JP condition
1	1	1	0	0	0	1	0	LD (FF00+C), A
1	1	1	0	1	0	1	0	LD (u16), A
1	1	1	1	0	0	1	0	LD A, (0xFF00+C)
1	1	1	1	1	0	1	0	LD A, (u16)
1	1	opcode (see segment to the right)			0	1	1	0: JP u16, 1: (CB prefix), 6: DI, 7: EI. ALL OTHER OPCODES ARE ILLEGAL
1	1	0	condition		1	0	0	CALL condition
1	1	r16 (group 3)		0	1	0	1	PUSH r16
1	1	0	0	1	1	0	1	CALL u16
1	1	opcode (group 2)			1	1	0	ALU a, u8
1	1	EXP			1	1	1	RST (Call to 00EXP000)

SM83 instruction encoding table (CB-prefixed opcodes)

7	6	5	4	3	2	1	0	instruction
0	0	opcode (group 3)			r8			Shifts/rotates
0	1	bit			r8			BIT bit, r8
1	0	bit			r8			RES bit, r8
1	1	bit			r8			SET bit, r8

r8: 8-bit register
r16: 16-bit register
u8: 8-bit unsigned immediate
u16: 16-bit unsigned immediate
i8: 8-bit signed immediate

r8 table

number	corresponding register
0	B
1	C
2	D
3	E
4	H
5	L
6	(HL)
7	A

r16 table (group 1)

number	corresponding register
0	BC
1	DE
2	HL
3	SP

r16 table (group 2)

number	corresponding register
0	BC
1	DE
2	HL+
3	HL-

r16 table (group 3)

number	corresponding register
0	BC
1	DE
2	HL
3	AF

condition code table

number	corresponding condition
0	NZ
1	Z
2	NC
3	C

opcode table group 1 (accumulator/flag operations)

number	corresponding opcode
0	RLCA
1	RRCA
2	RLA
3	RRA
4	DAA
5	CPL
6	SCF
7	CCF

opcode table group 2 (ALU operations)

number	corresponding opcode
0	ADD
1	ADC
2	SUB
3	SBC
4	AND
5	XOR
6	OR
7	CP

opcode table group 3 (CB shift/rotate operations)

number	corresponding opcode
0	RLC
1	RRC
2	RL
3	RR
4	SLA
5	SRA
6	SWAP
7	SRL

NOTES

1) LD (HL), (HL) is not a valid opcode as its encoding (if it existed) would overlap with HALT.

2) On an illegal opcode, the system has been proven to hang. If you’re making an emulator and encounter one, your CPU prolly has a bug that derails program execution.

3) If an opcode encoding isn’t in this page, it’s either illegal or a mistake of the author (Contact me on my discord: guccirodakino#1457 if you want to offer feedback or if you’ve found a mistake in this document. You can also contact me on Nintendo Switch Online: SW-8356-6970-6111)

4) If you want to see how these instructions are laid in an opcode table, visit https://izik1.github.io/gbops/index.html

5) Despite popular belief, **the Gameboy CPU is not a Zilog Z80** even though they share a lot of similarities. As such, opcode encoding isn’t the same as on the Z80.

6) I lobs u all <3

Use Cases

This document could potentially prove helpful in a variety of contexts, such as making a Gameboy emulator that’s less than 10.000 lines, or making an assembler. Or you could just be curious about how instructions are encoded :p