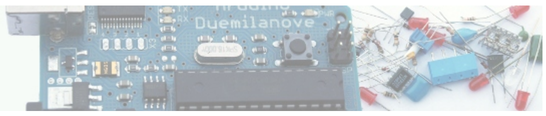


8085 instruction set: the octal table

The instruction set of the 8085 microprocessor has an underlying structure that becomes much clearer if expressed in an octal-based table, rather than usual hexadecimal-based table:

	\0_0	\0_1	\0_2	\0_3	\0_4	\0_5	\0_6	\0_7	\1_0	\1_1	\1_2	\1_3	\1_4	\1_5	\1_6	\1_7
\00_	NOP	LXI B,d16	STAX B	INX B	INR B	DCR B	MVI B,d8	RLC	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A
\01_	dsub	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,d8	RRC	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A
\02_	arhl	LXI D,d16	STAX D	INX D	INR D	DCR D	MVI D,d8	RAL	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A
\03_	rdel	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,d8	RAR	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A
\04_	RIM	LXI H,d16	SHLD a16	INX H	INR H	DCR H	MVI H,d8	DAA	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A
\05_	ldhi r8	DAD H	LHLD a16	DCX H	INR L	DCR L	MVI L,d8	CMA	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A
\06_	SIM	LXI SP,d16	STA a16	INX SP	INR M	DCR M	MVI M,d8	STC	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A
\07_	ldsi r8	DAD SP	LDA a16	DCX SP	INR A	DCR A	MVI A,d8	CMC	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A
\20_	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	RNZ	POP B	JNZ a16	JMP a16	CNZ a16	PUSH B	ADI d8	RST 0
\21_	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	RZ	RET	JZ a16	rstv	CZ a16	CALL a16	ACI d8	RST 1
\22_	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	RNC	POP D	JNC a16	OUT d8	CNC a16	PUSH D	SUI d8	RST 2
\23_	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	RC	shlx	JC a16	IN d8	CC a16	jnk a16	SBI d8	RST 3
\24_	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	RPO	POP H	JPO a16	XTHL	CPO a16	PUSH H	ANI d8	RST 4
\25_	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	RPE	PCHL	JPE a16	XCHG	CPE a16	lhlx	XRI d8	RST 5
\26_	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	RP	POP PSW	JP a16	DI	CP a16	PUSH PSW	ORI d8	RST 6
\27_	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	RM	SPHL	JM a16	EI	CM a16	jk a16	CPI d8	RST 7

The large-scale structure of the instruction set is by quadrant (i.e. the top two bits): MOV instructions in the pink quadrant, arithmetic instructions in the cyan quadrant, increment, decrement, rotates in the yellow quadrant, and control flow (jump, call, return, push, pop, rst) in the purple quadrant. It's not totally regular, of course. Some instructions are wedged in where they can fit, for example the spot where memory-to-memory move (MOV M, M) would go is replaced by HLT.

Note how registers are controlled by an octal digit in the sequence B, C, D, E, H, L, M, and A. This is especially notable for the MOV instructions and arithmetic instructions. For instructions acting on register pairs, the structure is similar: BC, BC, DE, DE, HL, HL, SP, SP.

Although octal is unpopular now, early microprocessors were designed with octal in mind, using groups of three bits to select registers and operations. Now hexadecimal is popular, but when the opcodes are displayed in a hex-based table, the underlying structure of the instructions is obscured.

Note that the four blocks have been arranged for ease of display - strictly speaking they should be stacked vertically rather than a 2x2 grid. The table includes undocumented instructions, which are shown in lower case. Mouse over a cell to see the hex value of the instruction. Credits: original data from pastraiser.com 8085 instruction table.

How the 8085 decodes instructions internally

The 8085 uses a set of PLAs to decode and process instructions. In the first step of processing an instruction the instruction decode ROM (details) decodes the instruction into one of 48 different instruction groups. The grid below is colored according to the instruction group (0 through 47).

NOP	LXI B,d16 42	STAX B 40	INX B 36	INR B 38	DCR B 38	MVI B,d8 14	RLC 25	MOV B,B 45	MOV B,C 45	MOV B,D 45	MOV B,E 45	MOV B,H 45	MOV B,L 45	MOV B,M 44	MOV B,A 45
dsub 21	DAD B 20	LDAX B 41	DCX B 37	INR C 38	DCR C 38	MVI C,d8 14	RRC 25	MOV C,B 45	MOV C,C 45	MOV C,D 45	MOV C,E 45	MOV C,H 45	MOV C,L 45	MOV C,M 44	MOV C,A 45
arhl 24	LXI D,d16 42	STAX D 40	INX D 36	INR D 38	DCR D 38	MVI D,d8 14	RAL 25	MOV D,B 45	MOV D,C 45	MOV D,D 45	MOV D,E 45	MOV D,H 45	MOV D,L 45	MOV D,M 44	MOV D,A 45
rdel 22	DAD D 20	LDAX D 41	DCX D 37	INR E 38	DCR E 38	MVI E,d8 14	RAR 25	MOV E,B 45	MOV E,C 45	MOV E,D 45	MOV E,E 45	MOV E,H 45	MOV E,L 45	MOV E,M 44	MOV E,A 45
RIM 3	LXI H,d16 42	SHLD a16 12	INX H 36	INR H 38	DCR H 38	MVI H,d8 14	DAA 6	MOV H,B 45	MOV H,C 45	MOV H,D 45	MOV H,E 45	MOV H,H 45	MOV H,L 45	MOV H,M 44	MOV H,A 45
ldhi r8 23	DAD H 20	LHLD a16 13	DCX H 37	INR L 38	DCR L 38	MVI L,d8 14	CMA 6	MOV L,B 45	MOV L,C 45	MOV L,D 45	MOV L,E 45	MOV L,H 45	MOV L,L 45	MOV L,M 44	MOV L,A 45
SIM 3	LXI SP,d16 42	STA a16 8	INX SP 36	INR M 39	DCR M 39	MVI M,d8 16	STC 6	MOV M,B 43	MOV M,C 43	MOV M,D 43	MOV M,E 43	MOV M,H 43	MOV M,L 43	HLT 47	MOV M,A 43
ldsi r8 23	DAD SP 20	LDA a16 9	DCX SP 37	INR A 38	DCR A 38	MVI A,d8 14	CMC 6	MOV A,B 45	MOV A,C 45	MOV A,D 45	MOV A,E 45	MOV A,H 45	MOV A,L 45	MOV A,M 44	MOV A,A 45
ADD B 1	ADD C 1	ADD D 1	ADD E 1	ADD H 1	ADD L 1	ADD M 4	ADD A 1	RNZ 19	POP B 27	JNZ a16 29	JMP a16 30	CNZ a16 33	PUSH B 26	ADI d8 2	RST 0 5
ADC B 1	ADC C 1	ADC D 1	ADC E 1	ADC H 1	ADC L 1	ADC M 4	ADC A 1	RZ 19	RET 18	JZ a16 29	rstv 7	CZ a16 33	CALL a16 34	ACI d8 2	RST 1 5
SUB B 1	SUB C 1	SUB D 1	SUB E 1	SUB H 1	SUB L 1	SUB M 4	SUB A 1	RNC 19	POP D 27	JNC a16 29	OUT d8 17	CNC a16 33	PUSH D 26	SUI d8 2	RST 2 5
SBB B 1	SBB C 1	SBB D 1	SBB E 1	SBB H 1	SBB L 1	SBB M 4	SBB A 1	RC 19	shlx 10	JC a16 29	IN d8 15	CC a16 33	jnk a16 31	SBI d8 2	RST 3 5
ANA B 1	ANA C 1	ANA D 1	ANA E 1	ANA H 1	ANA L 1	ANA M 4	ANA A 1	RPO 19	POP H 27	JPO a16 29	XTHL 35	CPO a16 33	PUSH H 26	ANI d8 2	RST 4 5
XRA B 1	XRA C 1	XRA D 1	XRA E 1	XRA H 1	XRA L 1	XRA M 4	XRA A 1	RPE 19	PCHL 32	JPE a16 29	XCHG 46	CPE a16 33	lhlx 11	XRI d8 2	RST 5 5
ORA B 1	ORA C 1	ORA D 1	ORA E 1	ORA H 1	ORA L 1	ORA M 4	ORA A 1	RP 19	POP PSW 27	JP a16 29	DI 0	CP a16 33	PUSH PSW 26	ORI d8 2	RST 6 5
CMP B 1	CMP C 1	CMP D 1	CMP E 1	CMP H 1	CMP L 1	CMP M 4	CMP A 1	RM 19	SPHL 28	JM a16 29	EI 0	CM a16 33	jk a16 31	CPI d8 2	RST 7 5

Colors by iWantHue

The internal decoding shown above reveals a few interesting things. The NOP instruction is literally no operation - it doesn't get decoded into any instruction group. The MOV instructions are all decoded together, except for the memory operations. Similarly, the arithmetic instructions are all grouped together, except for the memory instructions. There are other smaller groups (e.g. INR/DCR, conditional jumps, conditional calls, returns), and 21 instructions that are handled uniquely(e.g. CALL, PCHL, XCHG, HALT, and 6 undocumented instructions). Surprisingly, DAA, CMA, STC, and CMC are handled together at this stage, despite having very different actions.

Labels: 8085

Get new posts by email:

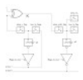
Enter your email

Subscribe

About the site

Contact info and site index

Popular Posts



Silicon reverse engineering: The 8085's undocumented flags

Inside a vintage aerospace navigation computer of uncertain purpose

A Multi-Protocol Infrared Remote Library for the Arduino

Apple iPhone charger teardown: quality in a tiny expensive package

Teardown and exploration of Apple's Magsafe connector

Macbook charger teardown: The surprising complexity inside Apple's power adapter

Mining Bitcoin with pencil and paper: 0.67 hashes per day

A dozen USB chargers in the lab: Apple is very good, but not quite the best

Search This Blog

Search

Labels

386 6502 8008 8085 8086 8087 8088 aerospace alto analog Apollo apple arc arduino arm beaglebone bitcoin c# cadc calculator chips css datapoint dx7 electronics f# fpga fractals genome globus haskell HP html5 ibm ibm1401 ibm360 intel ipv6 ir java javascript math microcode oscilloscope photo power supply random reverse-engineering

6 comments:

Mr Z said...

First, a typo of sorts: Shouldn't JK in your first table be lower case?

Second: I remember one of my EE profs (Dr. Donald Schertz) pointing out to me the octal nature of the 8085 ISA, and how what would be "MOV M, M" ends up being HLT. We had multiple EE labs at the time that required us to hand-assemble and hex-key code into SDK-85 boards, which is probably why Dr. Schertz was so keenly aware of the ISA's encoding properties.

(Our SDK-85s were fancier than most, with a bank of LEDs and toggles on the left side of the board; but, I digress.)

Anyway, that observation has caused me to look for octalness in other ISAs from the era. Indeed, the CP-1600 ISA is /very/ octal. I don't want to spam your 8085 entry with a dissertation on another processor. If anyone's interested in hearing more about the CP-1600, comment on this entry and it'll email me.

February 23, 2013 at 10:00 PM

Ken Shirriff said...

Thanks, Mr Z. I've fixed the typo. Please feel free to post a long comment about the CP-1600 (which Wikipedia tells me is the processor in the Intellivision game).

February 24, 2013 at 9:17 AM

Mr Z said...

Ken,

I'd be happy to. Blogger apparently doesn't like my post too much (it's too long, contains HTML markup it doesn't like), so I've just moved it here: http://spatula-city.org/~im14u2c/intv/comment_for_ken_shirrifs_blog.html

Enjoy!

--Joe

February 24, 2013 at 10:40 AM

Mr Z said...

I put together a Google Docs spreadsheet with a similar breakdown of CP1600/CP1610 opcodes, with a complete expansion of the opcode space here:

https://docs.google.com/spreadsheet/ccc?key=0Ar_02usomyeqdDlESlZLZ0NKcGhzT0xYdmxYb29BTVE&usp=sharing

February 24, 2013 at 3:41 PM

Anonymous said...

Sorry if my question is completely irrelevant, but I had to know.
In the Jump Statements of 8085, whenever the condition is satisfied, there are 3 Machine Cycles and 2 Machine Cycles if the condition isn't satisfied. I want to know what are these machine cycles? First one is Opcode Fetch, I'm confused about the rest. Please, do help.

September 4, 2013 at 11:21 PM

Mr Z said...

@Anonymous: Each M cycle corresponds to a single memory fetch, if I recall correctly. Jumps are 3 bytes. You need all 3 bytes to take the jump, as the second and third bytes are the branch target address.

So then the question is "Why is it 2 M cycles when the jump isn't taken?" I imagine they're getting the first byte of the jump destination in the same M cycle as they're evaluating the jump condition, although I haven't traced through the PLAs to see if that's actually the case.

September 5, 2013 at 6:38 AM

sheevaplug snark space spanish synth teardown theory unicode Z-80

Blog Archive

- ► 2024 (10)
- ► 2023 (35)
- ► 2022 (18)
- ► 2021 (26)
- ► 2020 (33)
- ► 2019 (18)
- ► 2018 (17)
- ► 2017 (21)
- ► 2016 (34)
- ► 2015 (12)
- ► 2014 (13)
- ▼ 2013 (24)
  - ► November (2)
  - ► September (4)
  - ► August (4)
  - ► July (4)
  - ► June (2)
  - ► April (1)
  - ► March (2)
  - ▼ February (2)
    - 8085 instruction set: the octal table
    - Silicon reverse engineering: The 8085's undocument...
  - ► January (3)
- ► 2012 (10)
- ► 2011 (11)
- ► 2010 (22)
- ► 2009 (22)
- ► 2008 (27)