

Digitális technika 2. jegyzet

Sebők Bence

2017. tavasz

Contents

1	Bevezetés	2
2	Alapismeretek	3
2.1	Intel 8085 processzor	3
3	Assembly	3
3.1	Tanszéki szimulátor használata	3
3.1.1	1. lépés: kód beillesztése	4
3.1.2	2. lépés: szimuláció beállítása	5
3.1.3	3. lépés: kód követése	7

1 Bevezetés

Ez a jegyzet a BME Digitális technika 2. tantárgyhoz szeretne segítséget nyújtani. Ezen kezdeményezés célja, hogy segítsen a hallgatóknak megérteni a tananyagot. A tantárgy nehéz, erősen ajánlott előadásra, gyakorlatra járni. Ez a jegyzet csak az előadáson, gyakorlaton készült jegyzet mellé jelent segítséget, nem tanít meg a nulláról a tantárgy minden apróságára.

Ez egy hallgatói jegyzet, nincs lektorálva, se egyetemi oktató által felügyelve. Amit itt olvasol, azt csak saját felelősségre használd, hivatalos helyeken nem hivatkozási alap ez a jegyzet.

2 Alapismeretek

2.1 Intel 8085 processzor

- PC (Program Counter, program számláló): regisztárpár, ami az aktuálisan használt memóriacímet tartalmazza.
- SP (Stack Pointer, stack mutató): a stack tetejére mutató pointer (regiszterpárban tárolja ezt a 2 bájtos címet)

3 Assembly

Digitális technika 2. tantárgy során az Intel 8085 processzor programozásához használt Assembly programozási nyelvet tanítják. Ez a fejezet erről az Assembly-ről szól.

3.1 Tanszéki szimulátor használata

Van egy az IIT-n fejlesztett Intel 8085 szimulátor. Ennek segítségével tetszőleges kódot tudunk futtatni egy virtuális Intel 8085 processzoron. Tetszőleges kód alatt értem, hogy bármit, ami a processzor utasításkészlete és fordítói direktívái lehetővé tesznek.

A tanszéki szimulátor a következő linken érhető el:

<http://topcat.iit.bme.hu/tools/i8085sim/i8085sim.cgi>

A szimulátor a következő módon néz ki:

i8085 Simulator

Import file into i8085 Simulator:

Acceptable file types: intel hex, i8085 assembly source ([beta version](#)). There is also a small [help](#) in hungarian.

Direct assembly

Type in assembly source.

Online examples

Study Examples

[Gyak1a source](#)[Gyak1b source](#)[Gyak1c source](#)[Gyak6 I source](#)[Gyak6 II source](#)[Gyak7 I source](#)[Gyak7 II source](#)

Module Tests

[Hetszegmens source](#)[Johnson source](#)[KapcsLed source](#)[Kepernyo source](#)[Menu source](#)

Miscellaneous

[instr source](#)

Course Prezi

[c01 source](#)[c02 source](#)[c03 source](#)[c04 source](#)[c05 source](#)[c06 source](#)

© Doxence, 2007-2014

A következő kódot szeretnénk a szimulátorban futtatni:

```
ORG 2000h ; 2000h-tol helyezze el a kódot a fordító  
LXI H, 2100h ; HL ← 2100h  
MVI M, 11h ; [2100h] = 11h  
HLT ; processzor → HALT állapot  
END ; eddig fordítson
```

3.1.1 1. lépés: kód beillesztése

A szimulálandó kódot a Direct assembly ablakba kell írni:

```

Direct assembly
Type in assembly source.

ORG 2000h ; 2000h-tól helyezze el a kódot a fordító
LXI H, 2100h ; HL <-- 2100h
MVI M, 11h ; [2100h] = 11h
HLT ; processzor --> HALT állapot
END ; eddig fordítson

```

Send

Ezután a Send gombra kattintva menjünk tovább, ahol ez fogad minket:

i8085 Simulator

Reset Run Stop Step Config Elapsed time: 0 phases, 0 usec

0000	[00]	NOP
0001	[00]	NOP
0002	[00]	NOP
0003	[00]	NOP
0004	[00]	NOP
0005	[00]	NOP
0006	[00]	NOP
0007	[00]	NOP
0008	[00]	NOP
0009	[00]	NOP
000A	[00]	NOP
000B	[00]	NOP
000C	[00]	NOP
000D	[00]	NOP
000E	[00]	NOP
000F	[00]	NOP
0010	[00]	NOP

Registers: A 00, B 00, C 00, D 00, H 00, L 00, SP 0000, PC 0000, INTE 0, CY 0, Z 0, P 0, S 0, AC 0, RST5.5 0, RST6.5 0, RST7.5 0, ... mask 0, SOD 0, SID 0, Output regs: 00-00, 01-00, 02-00, 03-00, 04-00, 05-00, 06-00

Memory: 0000 - 00 00 00 00 00 00 00 00, 0008 - 00 00 00 00 00 00 00 00, 0010 - 00 00 00 00 00 00 00 00, 0018 - 00 00 00 00 00 00 00 00, 0020 - 00 00 00 00 00 00 00 00, 0028 - 00 00 00 00 00 00 00 00, 0030 - 00 00 00 00 00 00 00 00

Stack: 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000

Input: 00-00, 01-00, 02-00, 03-00, 04-00, 05-00, 06-00

. Processing PASS1...
 . Processing PASS2...
 = No errors.
 = No warnings.

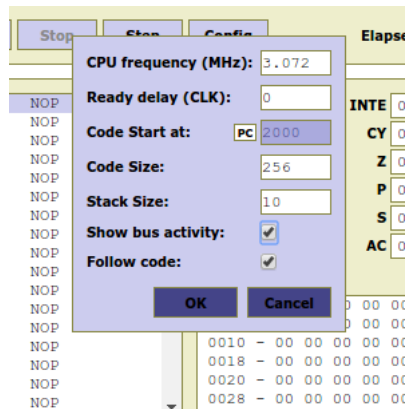
© Vajda Ferenc, 2007-2014

3.1.2 2. lépés: szimuláció beállítása

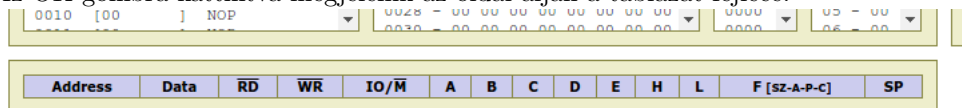
Néhány beállítást el kell végeznünk, hogy a megfelelő módon tudjuk nyomon követni a futó kódot.

1. Coda Start at: cím, ahova elhelyezzük a kódot (és kattintsunk bele a fehér téglalapba, hogy megjelenjen ott a PC felirat)
2. Show bus activity: legyen bepipálva
3. Follow code: legyen bepipálva

Az utolsó 2 opció ahhoz kell, hogy egy szép táblázatos formában jelenítse meg a kód futását és utasításonként haladjon a kód végrehajtása során. Ezek elvégzése után ezt kell látni:



Az OK gombra kattintva megjelenik az oldal alján a táblázat fejléce:



© Vajda Ferenc, 2007-2014

A táblázat fejléce:

1. Address: milyen címre mutat éppen a PC
2. Data: milyen adat van ezen a memóriacímen
3. \overline{RD} : olvasás-e a jelenlegi művelet
4. \overline{WR} : írás-e a jelenlegi művelet
5. IO/\overline{M} : a mostani utasítás memória vagy periféria műveletet hajt végre
6. regiszterek aktuális értékei
7. F [SZ-A-P-C]: flag-ek aktuális értékei
 - (a) Sign flag: előjel flag
 - (b) Zero flag: zérus flag
 - (c) Auxillary carry flag: fél-átvitel flag
 - (d) Parity flag: paritás flag
 - (e) Carry flag: átvitel flag
8. SP: Stack Pointer aktuális értéke

A PC mezőbe írjuk bele a címet, ahova helyeztük a kódot, jelen példában a 2000h-t:

A	00		
B	00	C	00
D	00	E	00
H	00	L	00
SP	0000		
PC	2000		

Ha be kell állítani a regiszterek kezdőértéket, akkor a megfelelő regiszterek mezőjébe írjuk bele a szükséges értéket.

3.1.3 3. lépés: kód követése

A bal oldali ablakban látható, hogy a PC a 2000h-ra mutat, ezt a kék háttérrel jelöli a rendszer (soron következő utasítás).

Ha rámegyünk a Step gombra az oldal tetején, akkor lefuttatja a PC által jelölt utasítást és az oldal alján elhelyezkedő táblázatban megjelenik néhány adat:

The screenshot shows a debugger interface with several components:

- Register Window:** Shows values for A (00), B (00), C (00), D (00), E (00), H (21), L (00), SP (0000), and PC (2003). Status flags include INTE (0), RST5.5 (0), RST6.5 (0), RST7.5 (0), CY (0), Z (0), S (0), AC (0), SOD (0), and SID (0).
- Memory Window:** Shows memory addresses from 0000 to 0030 with their corresponding data values.
- Stack Window:** Shows stack addresses from 0000 to 0006 with their corresponding data values.
- Input Window:** Shows input values from 00 to 06.
- Output Window:** Shows output values from 00 to 06.
- Instruction List:** A list of instructions with addresses: 2003 [36 11] MVI M, 11; 2005 [76] HLT; 2006 [00] NOP; 2007 [00] NOP; 2008 [00] NOP; 2009 [00] NOP; 200A [00] NOP; 200B [00] NOP; 200C [00] NOP; 200D [00] NOP; 200E [00] NOP; 200F [00] NOP; 2010 [00] NOP; 2011 [00] NOP; 2012 [00] NOP; 2013 [00] NOP; 2014 [00] NOP.
- Table:** A table with columns: Address, Data, RD, WR, IO/M, A, B, C, D, E, H, L, F [sz-A-P-C], and SP.

Address	Data	RD	WR	IO/M	A	B	C	D	E	H	L	F [sz-A-P-C]	SP
2000	21	0	1	0	00	00	00	00	00	00	00	00000000	0000
2001	00	0	1	0	00	00	00	00	00	00	00	00000000	0000
2002	21	0	1	0	00	00	00	00	00	00	00	00000000	0000

Vegyük sorra, hogy mi minden történt:

1. A bal oldali fehér ablakban a kék háttér most már a soron következő utasításra mutat (MVI M, 11), hiszen az előző utasítást már lefuttattuk és tovább lépett.
2. A táblázatban megjelentek az LXI H, 2100h utasítás során történt dolgok.
 - (a) Mivel a 2000h-ra helyeztük el a kódot, a PC onnan olvassa fel az első adatot. Ott egy LXI utasítás van, ezért az első adat, amit felolvass az az utasítás opkódja.
 - (b) Ökölszabály: minden utasítás első bájta az opkódja.
 - (c) Opkód: utasítás egyedi azonosító kódja

- (d) Az LXI utasításhoz 2 dolog tartozik még: a regiszterpár, ahova adatot mozgatunk és maga az adat, amit mozgatunk. A regiszterpár az opkódba van kódolva, ezt a segédletből ki lehet olvasni. A másik paraméter, hogy milyen adatot akarunk elhelyezni a regiszterpárba. 2 bájtos adatot tudunk egy regiszterpárba tenni, ezért ez a 2 bájt az opkódot követő következő 2 címen helyezkedik el.
 - (e) Little-endian: kisebb címen kisebb helyiérték
 - (f) a 2100h számot szeretnénk a HL regiszterekbe tenni, szóval először felolvassuk a 2100h alsó bájtját (00h) a 2001h címről, majd a felső bájtját (21h) a 2002h címről.
 - (g) Egy utasítás opkódja és a paraméterei címfolytonosan helyezkednek el a memóriában
 - (h) Címfolytonos: egymást követő memóriacímeken.
 - (i) Az opkód és az utasítás paramétereinek beolvasása mind olvasás művelet.
3. A Step-re kattintva lefuttatja a jelenlegi utasítást, ami az MVI M, 11h, majd megjelennek a táblázatban az eközben történtek:
- (a) Címfolytosan helyezkedik el a kód, szóval a következő címen van az MVI M utasítás opkódja, vagyis a 2003h-n.
 - (b) Az MVI M utasításhoz tartozik egy paraméter: milyen adatot akarunk elhelyezni az M által mutatott memóriacímre. Ezt az adatot az opkódot követő címről olvassa fel, tehát a 2004h-ről.
 - (c) Felolvastunk az MVI M, 11h-hoz tartozó minden adatot, szóval el tudjuk ténylegesen végezni az utasítást fizikailag is: elhelyezzük az M által mutatott memóriacímre a 11h-t.
 - (d) Az M pointer a HL regiszterpár tartalmával jelölt memóriacímre mutat, a HL-ben jelenleg a 2100h van, hiszen az imént tettük bele. Így a 2100h címre írjuk a 11h-t. Ez látszik a táblázatban is.
 - (e) Figyeljük meg, hogy a $\overline{WR} = 0$, mert ez egy adat írása egy adott memóriacímre. Az Address oszlopban a 2100h szerepel, hiszen erre a címre írunk adatot. A Data oszlopban a 11h érték szerepel, mert ezt az adatot írjuk oda.
 - (f) Látható, hogy a H és L oszlopokban a regiszterpár tartalma az előbb odahelyezett 21h és 00h.
4. Ismételten a Step-re nyomva lefut a HLT utasítás is. A HLT címfolytosan helyezkedik el (hiszen nem volt például ORG direktíva vagy hasonló), tehát a 2005h-n. Az adat csupán az utasítás opkódja, mert ehhez az utasításhoz nem tartozik semmilyen paraméter sem.
5. Végeztünk, elvileg most megtanultuk használni a szimulátort. Ezek után sok gyakorlással ezekkel az alapokkal már ügyesen fogunk tudni bánni ezzel a hasznos kis segédeszközzel.