

Priprema za prvu laboratorijsku vežbu

- Arhitektura i organizacija računara 2 -

1. Potrebno je promeniti OPCODE instrukcija:

Instrukcija	Stari OPCODE							Nov OPCODE											
	IR ₃₁	IR ₃₀	IR ₂₉	IR ₂₈	IR ₂₇	IR ₂₆	IR ₂₅	IR ₃₁	IR ₃₀	IR ₂₉	IR ₂₈	IR ₂₇	IR ₂₆	IR ₂₅	IR ₂₄	IR ₂₃	IR ₂₂	IR ₂₁	
POPB	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
POPW	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1
PUSHB	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0
PUSHW	0	0	1	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	1

2. Potrebno je dodati novu instrukciju:

PUSHALL – bezadresna instrukcija, koja treba da na stek stavi registre AB, AW, PSW, GPR_{0..31}. Kod operacije je 00001000100.

Za vežbu:

POPALL – bezadresna instrukcija, koja treba da skida sa steka registre GPR_{31..0}, PSW, AW, AB. Kod operacije je 00001000101.

NEG – bezadresna instrukcija, koja treba da izračuna komplement akumulatora AB i rezultat smesti u akumulator AB. Kod operacije je 00001000110.

MULLU – adresna instrukcija, koja treba da pomnoži dva broja bez znaka (akumulator AB i adresirana vrednost) i rezultat da smesti u akumulator AB. Kod operacije je 00101110.

MULLS – adresna instrukcija, koja treba da pomnoži dva broja sa znakom (akumulator AB i adresirana vrednost) i rezultat da smesti u akumulator AB. Kod operacije je 00101111.

Na laboratorijskoj vežbi mogu se očekivati implementacije neke od gorepomenutih instrukcija, kao i npr. DIV, MOD, RSB (obrnuto oduzimanje : akumulator <= argument – akumulator), CMP, TST...

Rešenje:

1. Zadatak

Za realiziju promene OC, proćićemo kroz sledeće faze:

- I. Dodavanje dekodera
- II. Razrešavanje greške u kodu
- III. Dodavanje instrukcije u program
- IV. Učitavanje 2 bajta za instrukciju
- V. Promena mikroprograma
- VI. Dodavanje uslovnih skokova u konfiguraciji

Napomena oko simulatora:

Da bi se učitao program i stanje memorije, klikom na File>Load memory se otvara dialog sa kojim treba selektovati fajl. Format fajla može se videti u Code.txt (u nastavku zadatka koristićemo taj fajl).

Podešavanja se menjaju u fajlu konfiguracija.txt. Da bi simulator video promene u tom fajlu, potrebno je da se restartuje sam simulator.

Greške na koje simulator reaguje zbog pogrešne konfiguracije, mogu se videti u fajlu error.txt.

Mikroprogram se piše u fajlu microProgram.txt.

I) Dodavanje dekodera

Da bi se promenio OC instrukcije, potrebno je obratiti pažnju na izgled novog OC. U našem slučaju, nov OC ima dužinu 2B, dok je stari imao dužinu 1B. U simulatoru u odeljku Fetch2, možemo da vidimo da će na izlazu dekodera DC4 biti aktivran izlaz G0_PG1_0 (na osnovu novih OC). Analizom bitova drugog bajta instrukcije (IR23,IR22,IR21), zaključujemo da treba da dodamo nov DC na šemu i kao izlaze postaviti instrukcije. Taj nov DC treba da bude aktivran samo ako nam je aktivran signal G0_PG1_0.

Da bi realizovali gore navedeno u fajlu konfiguracija.txt treba pronaći odeljak gde se podešava Fetch2. Radi preglednosti, promenićemo dekoder koji je zadužen za adresiranje, tj. promenićemo mu samo naziv (DC11=>DC12), a dodaćemo liniju:

```
"FETCH2", "DC", "DC11", "3", "G0_PG1_0", "IR23", "IR22", "IR21", "POPB1", "POPW1", "PUSHB1",  
"PUSHW1", "G0_PG1_0_PIN4", "G0_PG1_0_PIN5", "G0_PG1_0_PIN6", "G0_PG1_0_PIN7"
```

Radi testiranja stavićemo nazive signala za operacije sa indexom 1.

II) Razrešavanje greške u kodu

Do sada je procesor signal G0_PG1_0 shvatao kao grešku u kodu. Pošto taj signal više nije greška, moramo to da sredimo. U odeljku Fetch3, vidimo kako se generiše signal gropr, tj. možemo da primetimo da imamo G0_PG1_0 u OR kolu. Potrebno je otkačiti taj signal.

To možemo da uradimo tako što u fajlu konfiguracija.txt pronađemo odeljak gde se podešava Fetch3 i primetimo OR kolo sa nazivom ORgopr. Na tom kolu imamo 10 priključaka, pošto treba da skinemo jedan signal, prepravićemo taj broj na 9. Zatim u nastavku linije nalazi se lista signala koji su prikačeni na to kolo, tu je potrebno pronaći signal Fetch2.GO_PG1_0 i obrisati ga (sve zajedno sa navodnicima).

III) Dodavanje instrukcije u program

Pošto hoćemo da testiramo nove instrukcije, moraćemo da ih dodamo u program (fajl: Code.txt). Dodaćemo pre HALT-a, instrukciju PUSHB (ali sa novim OC), a HALT ćemo staviti odmah posle.

117, 08// PUSHB

118, 40

119, 01// HALT

Simulator treba da resetujemo na dugme RESET (donji levi ugao programa), a zatim učitamo novo stanje memorije (Code.txt).

Naša instrukcija se nalazi na PC=117, zato ćemo u simulatoru da pritiskamo dugme (gornji desni ugao simulatora) INSTRUCTION +, sve dok ne dođemo do PC=117 (trenutni PC može da se pročita iznad dugmića u gornjem desnom uglu). Nakon toga treba da pritiskamo CLK+.

U trenutku CPU clock=358, možemo videti da mikroprogram (Control unit info – desna strana simulatora) proverava signal greške u kodu. U Fetch3 jedinici možemo videti da je signal gropr neaktivovan, što znači da smo dobro uradili predhodnu tačku.

U trenutku CPU clock=359, upravljačka jedinica (na osnovu mikroprograma) proverava da li je dužina trenutne instrukcije 1 bajt. Signal Fetch3.l1 nije aktivovan, što znači da dužina nije 1 bajt, pa onda procesor učitava sledeći bajt instrukcije.

U trenutku CPU clock=370, upravljačka jedinica (na osnovu mikroprograma) proverava da li je dužina trenutne instrukcije 2 bajta, a da se pri tome radi o instrukciji skoka. Signal Fetch3.l2_brnch nije aktiviran, što znači da taj uslov nije zadovoljen.

U trenutku CPU clock=371, upravljačka jedinica (na osnovu mikroprograma) proverava da li je dužina trenutne instrukcije 2 bajta, a da se pri tome radi o aritmetičko logičkoj instrukciji. Signal Fetch3.l2_arlog nije aktiviran, što znači da taj uslov nije zadovoljen. Procesor nakon ovih uslova kreće da učitava treći bajt instrukcije. Ovo nije dobro, jer instrukcije koje treba da se dodaju su dužine 2 bajta.

IV) Učitavanje 2 bajta za instrukciju

Naš procesor je do sada ove instrukcije čitao sa jednim bajtom, sada će morati da ih čita kao dva bajta. Moramo da dodamo neki upravljački signal koji će biti aktiviran ukoliko se izvršava dvobajtna bezadresna instrukcija. Najbolje mesto za pravljenje takvog signala je jedinica Fetch3. U konfiguracionom fajlu dodaćemo jedno ILI kolo i na njega nakačiti signale koji predstavljaju takve instrukcije (PUSHB1, PUSHW1,

POPB1, POPW1). Dodavanje kola može da se uradi, tako što ćemo pronaći Fetch3 odeljak u konfiguracionom fajl, a zatim dodamo sledeću liniju:

```
"FETCH3", "OR", "ORL2", "4", "Fetch2.PUSHB1", "Fetch2.PUSHW1", "Fetch2.POPB1", "Fetch2.POPW1",  
"I2"
```

Na ovaj način smo izlaz ILI kola nazvali I2. Ako bi restartovali simulator, u odeljku Fetch3, trebali bi da vidimo novu komponentu koju smo napravili. Kada smo napravili ovaj upravljački signal, potrebno je izmeniti mikrokod kako bi procesor obradio ovaj signal.

V) Promena mikroprograma

U fajlu mikroProgram.txt, na koraku madrOE i madrOF proverava se da li je dužina instrukcije dva bajta (instrukcija skoka I2_brnch i aritmetičko logička instrukcija I2_arlog). Na ovom mestu bi trebalo da se umetne ispitvanje signala I2. Problem bi bio da se samo doda red, jer bi onda morali sve korake mikroprograma da šiftujemo za jedno mesto. Da to ne bi radili, možemo da umesto mikroinstrukcije "madrOE br (if I2_brnch then madr3D);“ stavimo bezuslovni skok ("madrOE br madrCD;“) koji nas vodi na kraj mikroprograma – madrCD i onda da prebacimo staru mikroinstrukciju sa madrOE na novoformirano mikroinstrukciju "madrCD br (if I2_brnch then madr3D);“. Posle ove mikroinstrukcije možemo da stavimo naš uslovni skok za obradu I2 signala. Ako je trenutna instrukcija dvobajtna bezadresna instrukcija, procesor bi posle čitanja iste, trebalo da ode u fazu za njeno izvršavanje (preskače se deo mikroprograma vezan za adresiranje). Faza izvršavanja počinje u koraku madr3D. Kako bi prethodni postupak realizovali, dodaćemo mikroinstrukciju "madrCE br (if I2 then madr3D);“. Nakon ovih ispitivanja treba da se vratimo na mikroprogram odakle smo skočili, to ćemo uraditi tako što ćemo dodati na karju mikroprograma "madrCF br madrOF;“.

Prethodno objašnjениm postupkom smo zapravo umetnili deo mikroprograma u već postojeći mikroprogram.

Obzirom da smo koristili I2 kao uslovni signal (br (if then)) mikroprograma, moramo taj signal da dodamo u odgovarajući deo upravljačke jedinice, koji će proveravati da li postoji skok u mikroprogramu.

VI) Dodavanje uslovnih skokova u konfiguraciji

U jedinici "Signali upravljačke jedinice" nalazi se dekoder u kome se nalaze svi signali koji služe za skokove unutar mikroprograma. Da bi dodali naš uslovni skok, potrebno je da se u konfiguracionom fajlu pronađe odeljak gde stavke počinju sa "CONTRODC", na kraju tog odeljka treba dodati liniju

```
"CONTRODC", "24", "Fetch3.I2", ""
```

Faza izvršavanja instrukcije počinje sa višestrukim uslovnim skokom, gde na osnovu OC mikroprogram zna kako treba trenutna instrukcija da se izvrši. Pošto smo dodali nov OC, u ovoj fazi naš mikroprogram ne zna šta da radi. Moraćemo da preusmerimo naš mikroprogram da kada se pročita PUSHB1 treba da uradi isti posao koji je radio i PUSHB (isto važi i za ostale instrukcije).

Preusmeravanje ćemo implemetirati tako što ćemo u konfiguracionom fajlu u odeljku gde stavke kreću sa "KMOPR1" staviti nove linije:

```
"KMOPR1", "Fetch2.POPB1", "86"  
"KMOPR1", "Fetch2.POPW1", "91"  
"KMOPR1", "Fetch2.PUSHB1", "99"  
"KMOPR1", "Fetch2.PUSHW1", "104"
```

Vrednosti 86, 91, 99, 104 su decimalne vrednosti heksidecimalnih brojeva 56, 5B, 63, 68. Ove vrednosti koje predstavljaju broj koraka u mikroprogramu na kome se izvršavaju same instrukcije. Treba pogledati madr56, madr5B, madr63, madr68.

Prehodnim koracima smo realizovali promenu OC. Preostaje još da se u Fetch2 promene imena starih OC, da se novim OC skine "1" u imenima i da se prijavljuje greška u kodu ako bi neko koristio stare OC. Ove funkcije bi bilo dobro samostalno provežbati.

2. Zadatak

Za dodavanje PUSHALL, proćićemo kroz sledeće faze:

- I. Dodavanje instrukcije u Fetch2 dekoder i dodavanje signala instrukcije u OR kolo u Fetch3 potrebno za generisanje signala I2
- II. Promena mikroprograma
- III. Dodavanje uslovnih skokova u KMOPR i CONTRODC
- IV. Testiranje

I) Dodavanje instrukcije u Fetch2 dekoder i dodavanje signala instrukcije u OR kolo u Fetch3 potrebno za generisanje signala I2

Ovaj postupak je objašnjen u zadatku 1.

Potrebno izmeniti odgovarajuće linije tako da budu:

```
"FETCH2", "DC", "DC11", "3", "G0_PG1_0", "IR23", "IR22", "IR21", "POPB1", "POPW1", "PUSHB1",  
"PUSHW1", "PUSHALL", "G0_PG1_0_PIN5", "G0_PG1_0_PIN6", "G0_PG1_0_PIN7"
```

```
"FETCH3", "OR", "ORL2", "5", "Fetch2.PUSHB1", "Fetch2.PUSHW1", "Fetch2.POPB1", "Fetch2.POPW1",  
"Fetch2.PUSHALL", "I2"
```

II) Promena mikroprograma

PUSHALL instrukciju ćemo da napravimo u fazama: PUSHAB, PUSHAW, PUSHPSW, PUSHGPR.

Potrebno je da implementiramo samo izvršavanje instrukcije. To možemo da uradimo tako što ćemo da pišemo nove mikroinstrukcije na kraju trenutnog mikroprograma (posle madrCF).

- PUSHAB

Ovu implemntaciju već imamo urađenu, jer je to zapravo instrukcija PUSHB. Tako da možemo da iskopiramo taj deo mikroprograma, ali potrebno je samo promeniti odgovarajuće adrese koraka. Na kraju mikroprograma treba dodati:

Mikroinstrukcija	Komentar
madrD0 incSP;	Povećavamo stek na slobodnu lokaciju
madrD1 SPout2, IdMAR, ABout3, mxMDR, IdMDR;	Učitavamo u adresu steka u MAR i vrednost akumulatora AB u MDR
madrD2 br (if hack then madrD2);	Provera da li je memorija slobodna
madrD3 eMAR, eMDR, wrCPU, br (if !fcCPU then madrD3);	Memoriji se šalje zahtev za upis i čeka se dok memorija ne završi operaciju

- PUSHAW

Ovu implemntaciju već imamo urađenu, jer je to zapravo instrukcija PUSHW. Tako da možemo da iskopiramo taj deo mikroprograma, ali potrebno je samo promeniti odgovarajuće adrese koraka. Na kraju mikroprograma treba dodati:

Mikroinstrukcija	Komentar
madrD4 incSP;	Povećavamo stek na slobodnu lokaciju
madrD5 SPout2, IdMAR, AWOut3, mxMDR, IdMDR;	Učitavamo u adresu steka u MAR i vrednost višeg bajta akumulatora AW u MDR
madrD6 br (if hack then madrD6);	Provera da li je memorija slobodna
madrD7 eMAR, eMDR, wrCPU, br (if !fcCPU then madrD7);	Memoriji se šalje zahtev za upis i čeka se dok memorija ne završi operaciju
madrD8 incSP;	Povećavamo stek na slobodnu lokaciju
madrD9 SPout2, IdMAR, AWout3, mxMDR, IdMDR;	Učitavamo u adresu steka u MAR i vrednost nižeg bajta akumulatora AW u MDR
madr9A br (if hack then madr9A);	Provera da li je memorija slobodna
madr9B eMAR, eMDR, wrCPU, br (if !fcCPU then madr9B);	Memoriji se šalje zahtev za upis i čeka se dok memorija ne završi operaciju

- PUSHPSW

Ova instrukcija nije nigde implementirana, ali je dosta slična kao prethodna. Na kraju mikroprograma treba dodati:

Mikroinstrukcija	Komentar
madrDC incSP;	Povećavamo stek na slobodnu lokaciju
madrDD SPout2, IdMAR, PSWHout3, mxMDR, IdMDR;	Učitavamo u adresu steka u MAR i vrednost višeg bajta PSW-a u MDR
madrDE br (if hack then madrDE);	Provera da li je memorija slobodna
madrDF eMAR, eMDR, wrCPU, br (if !fcCPU then madrDF);	Memoriji se šalje zahtev za upis i čeka se dok memorija ne završi operaciju
madrE0 incSP;	Povećavamo stek na slobodnu lokaciju
madrE1 SPout2, IdMAR, PSWLout3, mxMDR, IdMDR;	Učitavamo u adresu steka u MAR i vrednost nižeg bajta PSW-a u MDR
madrE2 br (if hack then madrE2);	Provera da li je memorija slobodna
madrE3 eMAR, eMDR, wrCPU, br (if !fcCPU then madrE3);	Memoriji se šalje zahtev za upis i čeka se dok memorija ne završi operaciju

- PUSHGPR

Da bi snimili kompletan GPR, moraćemo da iz GPRAR registra izgenerišemo brojeve od 0 do 31 (jer postoji 32 registara opšte namene). Pošto postoji signal incGPRAR, dovoljno je da jednom učitamo vrednost 0, a nakon toga inkrementiranjem dobijamo i ostale vrednosti do 31. Gledano teoretski mogli bi da napravimo deo mikrokoda koji će da prenese samo GPR[0] na stek i recimo da se to realizuje pomoću dvadesetak mikroinstrukcija. Posle toga isto sa dvadesetak mikroinstrukcija prenese GPR[1] i tako dalje za ostale registre. Možemo da zaključimo da je potrebno oko $20 \times 32 = 640$ mikroinstrukcija, to je veliki broj mikroinstrukcija. Drugo rešenje je da napravimo petlju u mikroprogramu. Zbog petlje potrebno je da napravimo neki uslovni skok. Možemo da zaključimo da bi to moglo da se reši tako što ćemo u akumulator AB postaviti broj 32, a zatim kroz ALU jedinicu da dekrementiramo tu vrednost svaki put kada prenesemo jedan registar. Uslov za prekid petlje je signal Z, tj. kada vrednost u AB bude 0. Pošto instrukcija PUSHALL ne bi trebao da menja akumulator AB, moraćemo da snimimo tu vrednost pre nego što učitamo vrednost 32. Da bi učitali broj 32 u AB, jedan od načina je da se u AB formira 0, zatim da se inkrementira AB (AB=1), a nakon toga 5 puta se AB shiftuje u levo (realijacija 2^5).

Drugi potencijalni problem koji moramo da rešimo jeste to što je MDR širine 8 bita, a veličina jednog registra je 16bita. Na osnovu cele šeme možemo da vidimo da samo registar AW može da na internu magistralu postavi zasebno i svojih 8 viših i 8 nižih bita (pomoću trostatičkih bafera i signala AWHout3 i AWLout3). To znači da ćemo morati uvek iz GPR-a prebacivati vrednosti u AW. Pošto instrukcija PUSHALL ne bi trebao da menja akumulator AW, moraćemo da snimimo tu vrednost pre nego što nešto učitamo.

U svakoj iteraciji ćemo prenositi vrednost iz AW (dva pristupa memoriji), a nakon toga ćemo da inkrementiramo vrednost GPRAR, a dekrementiraćemo vrednost AB.

Kada se završi prenos 32 registra, potrebno je samo vratiti originalne vrednosti u AB i AW.

U nastavku je tablica mikroinstrukcija koje realizuju gore pomenuti postupak.

Mikroinstrukcija	Komentar
madrE4 IdBB,ABout3,MOST3_2,MOST2_1;	Čuvamo AB => BB
madrE5 IdCW, BWout2;	Upisivanje vrednosti 0 u GPRAR i AB
madrE6 CWout3, BWout2, IdGPRAR, IdAB, xor, ALUout1;	
madrE7 AWout3,IdCW, MOST3_2;	Čuvamo AW => CW
madrE8 ABout3,inc,ALUout1,IdAB;	
madrE9 shl;	
madrEA shl;	
madrEB shl;	
madrEC shl;	
madrED shl;	
madrEE incSP,GPRout1,IdAW, MOST1_2;	
madrEF SPout2, IdMAR, AWHout3, mxMDR, IdMDR;	
madrF0 br (if hack then madrF0);	
madrF1 eMAR, eMDR, wrCPU, br (if !fcCPU then madrF1);	
madrF2 incSP;	
madrF3 SPout2, IdMAR, AWout3, mxMDR, IdMDR;	
madrF4 br (if hack then madrF4);	
madrF5 eMAR, eMDR, wrCPU, br (if !fcCPU then madrF5);	
madrF6 incGPRAR,ABout3,dec,ALUout1,IdAB;	Uvećavamo indeks u GPRAR i smanjujemo vrednost u AB
madrF7 br (if Z then madrF9);	Ako je u AB=0 (aktivan signal Z), onda izlazimo iz petlje
madrF8 br madrEE;	Ako AB nije 0, vraćamo se na slanje sledećeg registra
madrF9 BBout2,MOST2_1,IdAB;	Vraćamo podatke BB=>AB
madrFA CWout3,MOST3_2,IdAW, br madrA9;	Vraćamo podatke CW=>AW i skok na fazu opsluživanje prekida

Na kraju su potrebne još dve stvari da se urade. Prvo, pošto koristimo signal Z, moramo samo još jednom da vidimo kako se dobija taj signal. U fazi Exec3 vidimo da bi signal Z bio aktivan mora da su svi bitovi AB registra neaktivni i moramo da se izvršava instrukcija koja koristi/menja Z signal. Na osnovu načina rada mikroprograma moraćemo da dodamo signal instrukcije na ILI kolo. To ćemo uraditi tako što ćemo u konfiguracionom fajlu pronaći liniju koja počinje sa "EXEC3" i izmenićemo je tako što ćemo dodati PUSHALL.

"EXEC3", "OR", "NZOR", "19", "Fetch2.PUSHALL", "Fetch2.LDB", "Fetch2.POPB", "Fetch2.ADD", "Fetch2.SUB", "Fetch2.INC", "Fetch2.DEC", "Fetch2.AND", "Fetch2.OR", "Fetch2.XOR", "Fetch2.NOT", "Fetch2.ASR", "Fetch2.LSR", "Fetch2.ROR", "Fetch2.RORC", "Fetch2.ASL", "Fetch2.LSL", "Fetch2.ROL", "Fetch2.ROLC", "NZOR"

Drugo što je potrebno jeste da sredimo uslovne skokove mikroprograma.

III) Dodavanje uslovnih skokova u KMOPR i CONTRODC

Pošto koristimo uslovni skok "br (if Z then madrF9);", moramo (kao u zadatku 1) da signal Z stavimo u dekoder CONTRODC.

"CONTRODC", "25", "Exec3.Z", ""

Moramo još da razrešimo višestruki skok kod razrešavanja operacije (kao u zadatku 1). Potrebno je dodati liniju u konfiguracioni fajl:

"KMOPR1", "Fetch2.PUSHALL", "228"

Decimalni broj 228 je heksidecimalni broj E4, što predstavlja broj koraka (madrE4) prve mikroinstrukcije izvršavanja PUSHALL.

Sa ovim smo završili implementaciju instrukcije.

Treba primeti da je moguće smanjiti mikrokod kada bi premeštali signal incSP da se ranije izvršava (kada je to moguće).

IV) Testiranje

U nastavku je program koji bi trebao da testira instrukciju PUSHALL. Na kraju programa postavljamo u GPR[0]=0, GPR[1]=1h, GPR[2]=2h, GPR[3]=3h, GPR[4]=4h, GPR[31]=1Fh. Pokretanjem programa na kraju bi trebalo na memorijskim lokacima počev od 1001 (vrednost SP pre početka instrukcije PUSHALL) da stoje podaci AB, AW, PSW, GPR_{31..0}.

100, 20// LDB r1	112, 22// ST r4	124, 02//
101, 01	113, 04	125, 32// INC
102, 30// ADD #0x55	114, 37// NOT	126, 22// STB r3
103, E0	115, 38// ASR	127, 03//
104, 55	116, 39// LSR	128, 32// INC
105, 22// ST CC00h	117, 26// PUSHB	129, 22// STB r4
106, 40	118, 08// PUSHB1	12a, 04//
107, CC	119, 40//	12b, 20// LDB #31
108, 00	11a, 20// LDB #0	12c, E0//
109, 20// LDB (r2)	11b, E0//	12d, 1F//
10A, 22	11c, 00//	12e, 22// STB r31
10B, 31// SUB (r3)0x02	11d, 22// STB r0	12f, 1F//
10C, 83	11e, 00//	130, 08//PUSHALL
10D, 00	11f, 32// INC	131, 20
10E, 02	120, 22// STB r1	132, 01// HALT
10F, 34// AND #0xAC	121, 01//	0, 64
110, E0	122, 32// INC	2, 15
111, AC	123, 22// STB r2	

Kada se završi program pritiskom na dugme MEMORY otvorice se prozor sa prikazom memorije. U polje address treba uneti vrednost 1000 i zatim kliknuti na Read. Trebalo bi da su vidljive gore pomenute vrednosti.