Logisim mikroračunarski sistem

Set instrukcija, uputstva za korišćenje i primeri programa

Kontrolna jedinica prihvata, dekoduje i izvršava jednu po jednu instrukciju, na koju pokazuje programski brojač (PC). Sve tri faze sastoje se od određenog broja mikrooperacija (elementarnih operacija): prihvat dve, dekovodanje jedne, a izvršavanje od jedne do čak sedam (zavisi od instrukcije).
Mikrooperacije prihvata instrukcije (svake):

MA ← PC
MD ← [MA], PC←PC+1

Mikrooperacija dekodovanja instrukcije (svake):

IR ← MD

Izvršavanje svake instrukcije se, kao što smo rekli, razlikuje, a u tabeli ispod su date mikrooperacije (Register Transfer Language – RTL notacija) za svaku instrukciju u okviru seta instrukcija našeg edukativnog mikroprocesora implementiranog u Logisim-u.

Instrukcija	Kod operacije (binarno)	Veličina (u bajtovima)	Mikrooperacije	Primer	Objašnjenje
MOV Rn, Rm	0000NNMM	1	Rn ← Rm	MOV R0, R3	NN je binarni zapis indeksa odredišnog registra, a MM izvorišnog, za instrukciju iz primera biće kod operacije 0000 00 11 jer 00 stoji za R0, a 11 stoji za R3
MOV Rn, Const	0001NN00	2	MA ← PC MD ← [MA], PC←PC+1 Rn ← MD	MOV R1,5	NN je binarni zapis indeksa odredišnog registra, za ovaj primer biće kod operacije 0001 01 00 a 00000101 (5) će biti upisano u memoriji na narednoj memorijskoj lokaciji
LD Rn, var	0010NN00	2	MA ← PC MD ← [MA], PC←PC+1 MA ← MD MD ← [MA]	LD R1, sabirak	Vrednost promeljive sabirak iz memorije se upisuje u registar R1

			Rn ← MD		
STR var, Rn	001100NN	2	MA ← PC MD ← [MA], PC←PC+1 MA ← MD MD ← Rn [MA] ← MD	STR suma, R2	U promeljivu suma u memoriji se upisuje vrednost registra R2
MOV R0, &var	01000000	2	MA ← PC MD ← [MA], PC←PC+1 Rn ← MD	MOV R0, &suma	u R0 se upisuje adresa promenljive suma
LD Rn, @R0	0101NN00	1	MA ← R0 MD←[MA] Rn←MD	LD R1, @R0	Učitaj u R1 sadržaj iz memorije sa adrese sadržane u registru R0 (npr adrese 3 ako je R0=3)
STR @R0, Rn	011000NN	1	MA←R0 MD←Rn [MA] ←Rn	STR @R0, R1	Na adresu na koju pokazuje R0 u memoriju upiši sadržaj registra R1
ADD Rn, Rm	0111NNMM	1	Temp←Rn Res←Temp + Rm Rn←Res	ADD R1, R3	Saberi R1 i R3 i upiši rezultat u R1
SUB Rn,Rm	1000NNMM	1	Temp←Rn Res←Temp - Rm Rn←Res	SUB R1, R3	Oduzmi od R1 registar R3 i upiši rezultat u R1
AND Rn,Rm	1001NNMM	1	Temp←Rn Res←Temp AND Rm Rn←Res	AND R1, R3	Logičko I registara R1 i R3 upiši u R1
OR Rn,Rm	1010NNMM	1	Temp←Rn Res←Temp OR Rm Rn←Res	OR R1, R3	Logičko ILI registara R1 i R3 upiši u R1
Call procname	10110000	2	MA←PC MD←[MA], PC←PC+1 Temp←MD MA←SP MD←PC, SP←SP-1 [MA] ←MD PC←Temp	Call pomnozi	Poziva proceduru pomnozi gde ime procedure zapravo sadrži adresu na kojoj je procedura smeštena u memoriji

RET	11000000	1	SP←SP+1 MA←SP MD←[MA] PC←MD		Označava kraj procedure i povratak u glavni program na mesto na kom je bila pozvana procedura
PUSH Rn	1101NN00	1	MA←SP MD←Rn [MA] ←MD SP←SP-1	PUSH R2	Sadržaj registra R2 se stavlja na stek
POP Rn	1110NN00	1	SP←SP+1 MA←SP MD←[MA] Rn←MD	POP R3	Sa steka se uzima poslednji upisan podatak i upisuje u registar R3
Jmp label	11110000	2	MA←PC MD←[MA], PC←PC+1 PC←MD	L1:MOV R1, R0 … JMP L1	Skoči na instrukciju označenu labelom L1 (MOV R1, R0 instrukcija)
JZ label	11110100	2	MA←PC MD←[MA], PC←PC+1 C :PC←MD	L1:SUB R1, R0 … JZ L1	Skoči na instrukciju označenu labelom L1 samo ako je indicator nule (Zero flag) postavljen na 1
JC label	11111000	2	MA←PC MD←[MA], PC←PC+1 Z: PC←MD		Skoči na instrukciju označenu labelom L1 samo ako je indikator prenosa (Carry flag)postavljen na 1
END	11111100	1	-		Ne radi ništa, nemoj nikad označiti da je instrukcija završena, tako da se nikad neće preći na narednu

Pravila pisanja programa

1. Program počinje direktivom DATA nakon čega sledi spisak svih promenljivih koji se koriste u programu
2. Svaka promenljiva se novodi sa imenom, nakon čega sledi DB pa vrednost promenljive u decimalnom formatu ili heksadecimalnom formatu (sa prefiksom 0x)
3. Ukoliko je promenljiva niz navodi se više vrednosti odvojenih zarezom (npr za niz od 3 elementa: niz DB 1, 2, 3)
4. Direktivom ENDDATA se završava segment sa podacima. Čak i u slučaju da se u programu ne koriste nikakve promenljive, neophodno je da postoje ove dve direktive na početku (DATA ispod koje odmah sledi ENDDATA)
5. Direktiva PROGRAM označava početak programa - iza nje sledi prva instrukcija programa (koja će biti smeštena na adresu 0 u memoriji)
6. Instrukcije se navode jedna ispod druge sve do poslednje instrukcije END koja označava kraj programa
7. Ako se koriste procedure, nakon instrukcije END se navodi direktiva PROCEDURE iza koje sledi ime procedure (npr PROCEDURE POMNOZI)
8. U telu procedure instrukcije se navode na isti način kao i u glavnom programu, osim što se procedura završava instrukcijom RET, dok se program završava instrukcijom END
9. Nakon instrukcije RET obavezna je direktiva ENDPROCEDURE koja označava kraj procedure
10. Ako se koristi više procedura, za svaku se moraju dodati direktive PROCEDURE imeprocedure i ENDPROCEDURE (nakon završetka prve odmah počinje druga, nakon druge treća, …
11. Nakon poslednje procedure direktiva ENDPROGRAM označava kraj programa
12. Svaka instrukcija programa može biti označena labelom koja počinje velikim slovom L nakon koje sledi redni broj labele (L1, L2, …). Labele se stavljaju ukoliko u programu postoje instrukcije grananja (JMP, JZ ili JC) da označe mesto u programu od kojeg će se nastaviti izvršavanje instrukcija (kod JMP bezuslovni skok, kod JZ i JC skok ukoliko je odgovarajući indikator-fleg postavljen na 1 – *zero* ili *carry*). Ako se koristi labela, nakon labele ide dvotačka **:** a za njom sledi instrukcija (npr L1: MOV R0,3)
13. U kodu se mogu unositi komentari koji počinju tačka zarezom **;** (npr ;naredna instrukcija sabira registre R1 i R2)
14. Kompajler ne prevodi ni jednu od direktiva DATA, ENDDATA, PROGRAM, ENDPROGRAM, PROCEDURE, ENDPROCEDURE. Prevode se samo instrukcije i smeštaju u memoriju, a direktive samo pomažu kompajleru da prepozna različite segmente u program (deo sa podacima, glavni program i procedure)

Sadržaj memorije (šta radi kompajler)

1. Memorija se popunjava tako da u nju prvo ide program (glavni program i procedure ako se koriste u programu), a tek nakon programa sledi segment sa podacima
2. Stek se smešta u vrh memorijskog prostora – SP inicijalno ima vrednost 0xFF
3. Prva instrukcija programa se smešta na adresu 0 u memoriji
4. Jedna po jedna instrukcija se upisuje u memoriju tako što se prvo upisuje ***kod operacije*** (za svaku instrukciju kod operacije je dat u tabeli na početku dokumenta), a kod instrukcija koje se sadrže od dva bajta, drugi bajt zauzima narednu adresu u memoriji (ako je kod operacije na adresi *N*, drugi bajt, tzv operand se nalazi na adresi *N*+1)
5. Drugi bajt (operand) je nekada podatak koji se odmah može upisati u memoriju (npr kod instrukcije MOV R1, 3 gde će drugi bajt biti 3), ali nekada u momentu obrade trenutne instrukcije, kao posledica tačke 1 od gore, nije moguće upisati drugi bajt (npr kod instrukcije LD R1, BR2 gde će drugi bajt biti adresa promenljive BR2 koja nije poznata u ovom trenutku)
6. Ako drugi bajt instrukcije nije poznat u trenutku kompajliranja instrukcije, kompajler će samo zapamtiti mesto koje je morao da preskoči, a odgovarajući podatak će upisati kasnije kada on bude dostupan. U primerima koji se nalaze u nastavku, ove situacije su označene bojama – npr u prvom programu na adresi 3 se nalazi žutom bojom upisana vrednost 0x14 što je adresa promenljive BR2 koja se nalazi na adresi 0x14, ali u trenutku obrađivanja instrukcije LD R1, BR2 ta adresa koja predstavlja drugi bajt instrukcije nije bila poznata pa je uneta kasnije
7. Slično kao kod tačke 6, u slučaju instrukcija grananja koje koriste labele, ukoliko adresa na koju se odnosi labela nije poznata u trenutku kompajliranja instrukcije, mesto u memoriji na koje ta adresa treba da bude upisana, kao drugi bajt instrukcije, će biti preskočeno, a biće popunjeno kasnije kada ta adresa bude poznata
8. Nakon instrukcije END, memorija će biti popunjavana procedurama, ako se one koriste u programu, na sledeći način: prva instrukcija prve procedure navedene u programu nakon instrukcije END biće upisana na prvu memorijsku lokaciju nakon instrukcije END. Ako u programu ima više procedura, prva instrukcija svake naredne procedure upisivaće se u memoriju neposredno iza RET instrukcije prethodne procedure
9. Adresa procedure se odnosi na adresu na kojoj je smeštena prva instrukcija procedure u memoriji i ona će se upisivati na mesto drugog bajta instrukcije poziva procedura (npr kod instrukcije *CALL saberi* na mesto drugog bajta u memoriji biće upisana adresa na kojoj se nalazi prva instrukcija procedure saberi, naravno na mestu prvog bajta će biti upisan kod operacije instrukcije *CALL procname* – 0xB0)
10. Nakon RET instrukcije poslednje procedure (po redosledu kako su napisane) počinju podaci, opet u redosledu u kojem su navođeni u DATA segmentu (između direktiva DATA i ENDDATA). Ako je neka od promenljivih niz, svi bajtovi niza će biti smeštani u memoriju na sukcesivne memorijske lokacije, jedan za drugim u redosledu kojim su navedeni sa leva na desno

Simulacija i izvršavanje programa

1. Da biste simulirali program u Logisimu, potrebno je samo da promenite sadržaj memorije, nakon čega ćete kliktanjem na CLK ulazni signal izvršavati jednu po jednu mikrooperaciju, jedne po jedne instrukcije iz vašeg programa, sve dok ne dođete do instrukcije END, nakon čega se sadržaj registara više neće menjati
2. U Logisim-u, sadržaj memorije možete menjati tako što se prebacite u mod simulacije (klikom na ikonicu ), nakon čega kliknete na RAM memoriju i menjate jednu po jednu adresu koristeći isključivo heksadecimalne brojeve
3. U svakom trenutku sadržaj memorije možete sačuvati, kako bi kasnije mogli da ga učitate. Da biste sačuvali sadržaj memorije, kliknete na memoriju desnim klikom miša i odaberete opciju *Save image...*
4. Učitavanje prethodno sačuvanog sadržaja memorije vršite tako što kliknete desnim dugmetom miša na memoriju i odaberete opciju *Load image...*
5. Sadržaj memorije za sve primere navedene u ovom dokumentu dajem vam zajedno sa *Microcomputer.circ* kako biste mogli da simulirate i testirate programe iz primera
6. Kada želite da izvršavate svoj program, popunite sadržaj memorije sa heksadecimalnim podacima koji odgovaraju vašem programu i kao u slučaju izvršavanja primera programa koje sam vam ja dao, izvršavate instrukciju po instrukciju – sve dok ne dobijate željene rezultate menjate sadržaj memorije i pokrećete program ponovo
7. Da biste ponovo pokrenuli program iz početka, potrebno je resetovati sve registre, a to ćete uraditi tako što signal *Reset* (ulaz za Reset kolo) postavite na logičko „1“, držite ga aktivnim jedan ceo period takt signala (jedna rastuća i jedna opadajuća ivica CLK signala), nakon čega ga vratite na logičko „0“. Posle toga će svaka sledeća promena na CLK ulazu da rezultuje izvršavanjem instrukcija smeštenih u memoriji počevši od adrese 0
8. Da bi ceo dizajn stao u vidljivu oblast ekrana i da bi bio pregledan, šematik je organizovan po blokovima (registri, kontrolna jedinica, ALU, ...)
9. ***Top_level*** je blok koji se nalazi na najvišem nivou hijerarhije (uključuje ostale blokove) i simulaciju krećete od tog bloka
10. Tokom simulacije možete pogledati interne signale u svakom od pod-blokova tako što kliknete na željeni blok u šematiku, nakon čega će se pojaviti na tom bloku kružić sa lupom, pa vas naredni dvoklik na lupu vodi u „unutrašnjost“ bloka gde možete pogledati trenutne vrednosti signala na nižem nivou hijerarhije
11. Kada ste zadovoljni kako vaš program radi sačuvajte sadržaj memorije (kao u tački 3 od gore)

Primeri programa

Primer 1: Napisati program koji sabira dva 8-bitna broja, pri čemu zbir ne mora biti manji ili jednak sa 255 (rezultat ne staje u 8 bita)

```
DATA
BR1 DB 200
BR2 DB 200
SUMA_L DB 0
SUMA_H DB 0
ENDDATA
PROGRAM
LD R0,BR1
LD R1,BR2
ADD R0, R1
STR SUMA_L, R0
JC L1
JMP L2
L1: LD R3,SUMA_H
MOV R2, 1
ADD R3, R2
STR SUMA_H,R3
L2:END
ENDPROGRAM
```

Adresa(labela)	Sadržaj
0x0	0x20
0x1	0x13
0x2	0x24
0x3	0x14
0x4	0x71
0x5	0x30
0x6	0x15
0x7	0xF8
0x8	0x0b

0x9	0xF0
0x0a	0x12
0x0b L1	0x2c
0x0c	0x16
0x0d	0x18
0x0e	0x1
0x0f	0x7e
0x10	0x33
0x11	0x16
0x12 L2	0xFC
0x13 BR1	0xC8
0x14 BR2	0xC8
0x15 SUMA_L	0x0
0x16 SUMA_H	0x0

Primer 2: Napisati program koji koristi stek da menja sadržaj registara R0<->R3, R1<->R2.

```
DATA
ENDDATA
PROGRAM
MOV R0,1
MOV R1,2
MOV R2,3
MOV R3,4
PUSH R0
PUSH R1
PUSH R2
PUSH R3
POP R0
POP R1
POP R2
POP R3
```

```
END
ENDPROGRAM
```

Adresa(labela)	Sadržaj
0x0	0x10
0x1	0x01
0x2	0x14
0x3	0x2
0x4	0x18
0x5	0x3
0x6	0x1c
0x7	0x4
0x8	0xd0
0x9	0xd4
0x0a	0xd8
0x0b	0xdc
0x0c	0xe0
0x0d	0xe4
0x0e	0xe8
0x0f	0xec
0x10	0xfc

Primer 3: Napisati program koji sabira četiri elementa niza „niz“ i rezultat čuva u promenljivoj suma.

```
DATA
niz DB 1,2,3,4
suma DB 0
ENDDATA
PROGRAM
MOV R0,&niz
```

```
MOV R1,4
MOV R3,0
L1: LD R2,@R0
ADD R3,R2
MOV R2,1
ADD R0,R2
SUB R1,R2
JZ L2
JMP L1
L2: STR Suma,R3
END
ENDPROGRAM
```

Adresa(labela)	Sadržaj
0x0	0x40
0x1	0x13
0x2	0x14
0x3	0x04
0x4	0x1C
0x5	0x00
0x6 L1	0x58
0x7	0x7E
0x8	0x18
0x9	0x01
0x0a	0x72
0x0b	0x86
0x0c	0xF4
0x0d	0x10
0x0e	0xF0
0x0f	0x06
0x10 L2	0x33
0x11	0x17
0x12	0xFC
0x13 niz	0x01

0x14	0x02
0x15	0x03
0x16	0x04
0x17 suma	0

Primer 4: Napisati program koji koristi proceduru za množenje dva broja, pri čemu oba broja moraju biti manji od 16, tako da proizvod staje u jedan bajt (manji je od 256).

```
DATA
REZ1 DB 0
REZ2 DB 0
REZ3 DB 0
ENDDATA
PROGRAM
MOV R0,2
MOV R1,3
CALL POMNOZI
STR REZ1, R2
MOV R0,4
MOV R1,5
CALL POMNOZI
STR REZ2, R2
MOV R0,7
MOV R1,8
CALL POMNOZI
STR REZ3, R2
END
PROCEDURE POMNOZI
MOV R2,0
MOV R3, 1
L2: ADD R2, R1
SUB R0, R3
JZ L1
```

```
JMP L2
L1:RET
ENDPROCEDURE
ENDPROGRAM
```

Adresa(labela)	Sadržaj
0x0	0x10
0x1	0x02
0x2	0x14
0x3	0x03
0x4	0xB0
0x5	0x19
0x6	0x32
0x7	0x24
0x8	0x10
0x9	0x04
0x0a	0x14
0x0b	0x05
0x0c	0xB0
0x0d	0x19
0x0e	0x32
0x0f	0x25
0x10	0x10
0x11	0x07
0x12	0x14
0x13	0x08
0x14	0xB0
0x15	0x19
0x16	0x32
0x17	0x26
0x18	0xfc
0x19 pomnozi	0x18
0x1a	0x00

0x1b	0x1c
0x1c	0x01
0x1d L2	0x79
0x1e	0x83
0x1f	0xf8
0x20	0x23
0x21	0xf0
0x22	0x1d
0x23 L1	0xc0
0x24 REZ1	0x00
0x25 REZ2	0x00
0x26 REZ3	0x00