UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA ELEKTRONIKU

MIKROPROCESORSKI SISTEMI

simCPU

Mentor
Prof. dr. Mile Stojčev

Marko Ilić 9921
Nebojša Pejčić 9738
Aleksandar Stojadinović 10085
Bojan Janićijević 9931

Maj 2005

Sadržaj

Uvod

Kroz ciklus od tri vežbe analiziraćemo princip rada jedne jednostavne procesorske jedinice nazvane SCPU (***S**imple **C**entral **P**rocessing **U**nit*).

Prva vežba će se odnositi na sagledavanja koja se tiču sledećih detalja:

1) definicija skupa instrukcija SCPU-a
2) struktura SCPU
3) adresni načini rada
4) format i način izvršenja instrukcija

U drugoj vežbi biće opisan princip rada simulatora jednostavnog procesora nazvan ***simCPU*** (***sim**ulator of simple **C**entral **P**rocessing **U**nit*) koji se sastoji od sledeća dva koraka:

1) KORAK 1: Dati su detalji koji se odnose na pokretanje programa ***simCPU***
2) KORAK 2: Odnosi se na ilustraciju rada samog programa kroz jedan reprezentativni primer.

U trećoj vežbi kroz zadavanja odgovarajućih programskih test sekvenci analiziraće se rad SCPU-a. Izabrana programska test sekvenca uključuje sledeće detalje:

1) četiri adresna načina rada
2) pet klasa instrukcija

Zadatak studenta se sastoji u sledećem: upoznavanje sa arhitekturom SCPU-a, sagledavanje detalja koji se odnose na rad ***simCPU***-a i verifikacija rada SCPU-a korišćenjem ***simCPU***-a kroz izvršenje odgovarajuće programske test sekvence.

SimCPU je kompatibilana sa svim *Windows* operativnim sistemima.

Program ***simCPU*** je pisan u programskom jeziku *Delphi*. Na izradi *source cod*-a radili su Marko Ilić i Nebojša Pejčić, a na testiranju i izradi dokumentacije Aleksandar Stojadinović i Bojan Janićijević.

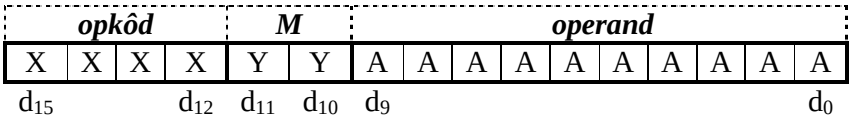
1. Jednostavna sekvencijalna procesorska jedinica

1.1 Definicija skupa instrukcija

SCPU je jednoadresna 16-bitna mašina koja podržava četiri načina adresiranja. Obim adresne magistrale je 10 bitova (može da se adresira do 1024 memorijskih lokacija), a obim magistrale podataka je 16 bitova (dva bajta).

1.1.1 Format mašinske instrukcije

Format mašinske instrukcije SCPU-a je prikazan na Slici 1:



Slika 1. Format instrukcije

gde su: $d_{15} \div d_{12}$ – četiri bita koja označavaju kôd instrukcije (***opkôd***)
d_{11} i d_{10} – dva bita koja se odnose na načine adesiranja (modifikator ***M***)
$d_9 \div d_0$ – deset bitova koji predstavljaju polje ***operand***

Skup instrukcija koje podržava SCPU i vrednosti opkôd polja su prikazani u Tabeli 1.

Tabela 1. Instrukcije procesora sa definisanim opkôd poljem

instrukcija	opkôd polje	opis instrukcije
OR	0000	Logička OR operacija
AND	0001	Logička AND operacija
NOT	0010	Logička NOT operacija
XOR	0011	Logička ExOR operacija
ADD	0100	aritmetička operacija sabiranja
SUB	0101	aritmetička operacija oduzimanja
NEG	0110	operacija komplementiranja
LD	0111	kopiranje iz memorije u akumulator
JMP	1000	bezuslovno grananje (skok)
JZ	1001	grananje ako je nula
JNZ	1010	grananje ako nije nula
INC	1011	inkrementiranje akumulatora
MOV	1100	kopiranje iz akumulatora u registar Rx
ST	1101	kopiranje iz akumulatora u memoriju
-	1110	predviđeno za dalja proširenja
NOP	1111	operacija bez efekta

Tabela 2. se odnosi na definiciju polja modifikatora ***M***.

Tabela 2. Značenje polja modifikatora ***M***

Modifikator ***M***	Način adresiranja
00	neposredni
01	direktni
10	indirektni
11	registarski

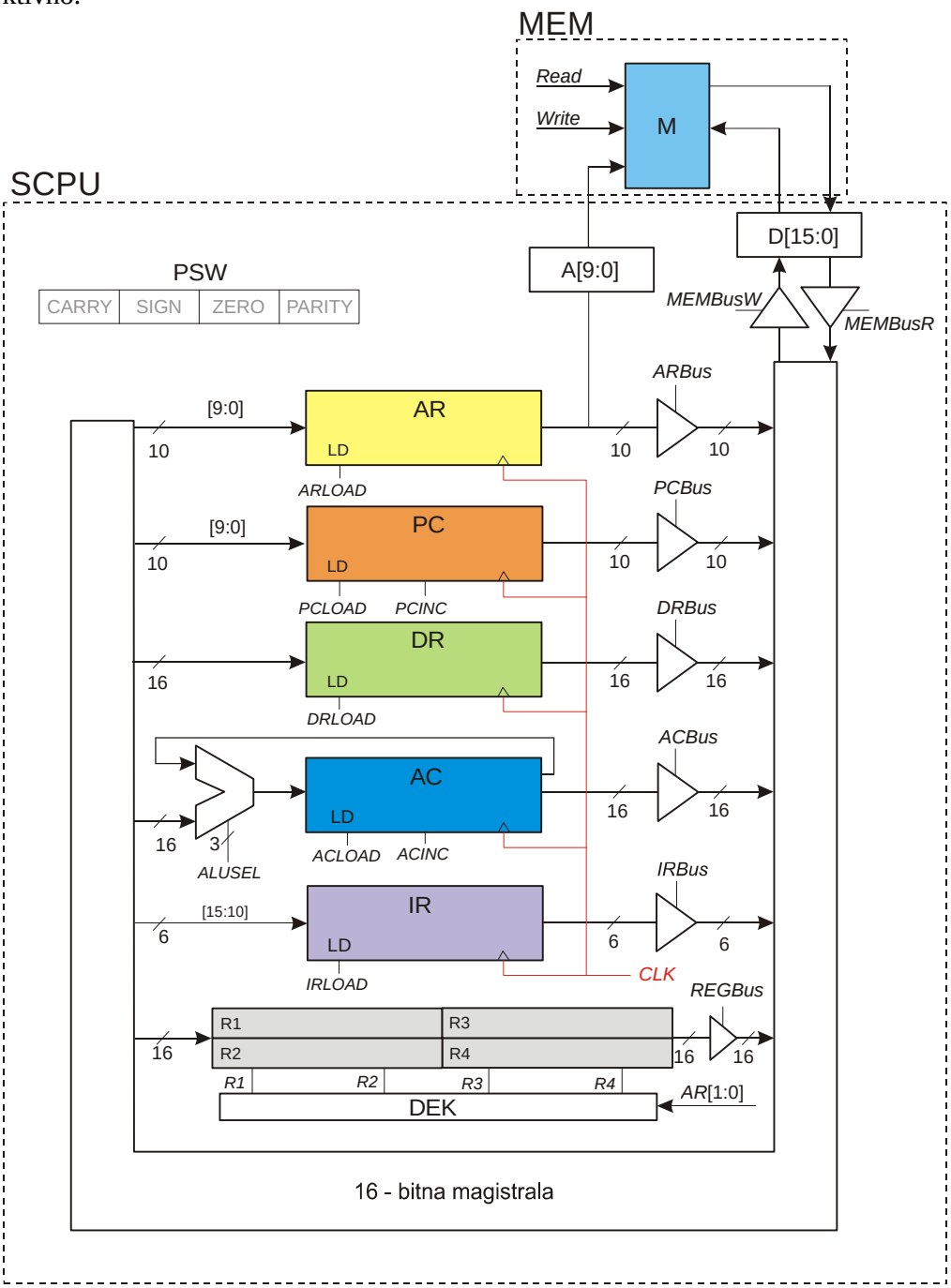
1.1.2 Klase instrukcija SCPU-a

Instrukcije možemo podeliti u sledeće grupe:

1. **prenos-podataka** (`MOV`, `LD`, `ST`) – pomoću ovih instrukcija vrši se kopiranje informacija iz jedne lokacije u drugu. Lokacije mogu pripadati registrima procesora ili memoriji,
2. **aritmetičke** (`ADD`, `SUB`, `NEG`, `INC`) – obavljaju aritmetičke operacije nad numeričkim podacima,
3. **logičke** (`OR`, `AND`, `NOT`, `XOR`) – uključuju Booleove i druge nenumeričke operacije,
4. **programsko-upravljačke** (`JMP`, `JZ`, `JNZ`) – menjaju sekvencu programskog izvršenja,
5. **ostale instrukcije** (`NOP`) – u ovu grupu spada instrukcija operacija bez efekta i instrukcija koja je predviđena za dalje proširenje.

1.2 Struktura sistema baziranog na SCPU

Struktura sistema zasnovanog na SCPU (procesor & memorija) prikazana je na Slici 2, gde se blok MEM odnosi na memoriju, a blok SCPU na jednostavni procesor. Kao što se vidi sa Slike 2, SCPU je interno organizovan oko 16-bitne magistrale. Preko gradivnih blokova A[9:0] i D[15:0] SCPU je spregnut sa MEM preko adresne magistrale i magistrale podataka, respektivno.



Slika 2. Struktura sistema baziranog na SCPU

1.2.1 Tipovi registara SCPU-a

Registri su sastavni delovi SCPU-a. U zavisnosti od namene mogu se podeliti u sledeće dve grupe:

1) **Korisničko vidljivi registri,** u ovu grupu spadaju sledeći registri:

   AC – 16-bitni registar nazvan akumulator (*Accumulator* – AC), ovaj registar se koristi za:
   a) čuvanje jednog od izvorišnih ili odredišnog operanda kod aritmetičkih i logičkih instrukcija;
   b) predstavlja izvorišni registar za operaciju tipa MOV i ST, a odredišni za operaciju LD.

   R0, R1, R2 i R3 – četiri 16-bitna registra opšte namene (*General Purpose Registers* - Rx): ovi registri se mogu koristiti kao odredišni registar kod operacije MOV i izvorišni kod aritmetičko/logičkih instrukcija. Selekciju registara vrši gradivni blok dekoder, DEK, tipa 2-u-4 na osnovu selektorskih signala *ARx*[1:0] koji odgovaraju adresnim bitovima najmanje težine polja ***operand*** (d1 i d0 na Slici 1). Izlazi dekodera DEK su aktivni samo kod registarskog načina adresiranja.

2) **Upravljački i statusni registri**, u ovu grupu spadaju sledeći registri:

   PC – 10-bitni programski brojač (*Program Counter* – PC): čuva adresu naredne instrukcije koja treba da se pribavi iz memorije;

   AR – 10-bitni adresni registar (*Address Register* – AR): predaje adrese memoriji preko adresnih linija *A*[9:0];

   DR – 16-bitni registar podataka (*Data Register* – DR): predstavlja privremeni dvosmerni bafer podataka preko koga se prenose svi podaci iz/ka MEM ka/iz SCPU-a. Kod aritmetičko/logičkih instrukcija koristi se za čuvanje drugog izvorišnog operanda;

   PSW – četvorobitni statusni registar (*Program Status Word* – PSW): čine ga sledeći markeri koji se uvek postavljaju na logičku {0} ili {1}, nakon izvršenja aritmetičkih ili logičkih operacija:
   CARRY (prenos) – postavlja se na logičku {1} ako se izvršenjem operacije javio prenos kod operacije sabiranja, ili je došlo do pozajmljivanja kod operacije oduzimanja, na mestu bita najveće težine,
   SIGN (znak) – odgovara bitu znaka rezultata dobijen zadnjom operacijom,
   ZERO (nula) – postavlja se na logičku {1} kada je rezultat operacije jednak 0,
   PARITY (parnost) – postavlja se na logičku {1} kada je broj jedinica u rezultatu nakon zadnje operacije paran, tj. broj jedinica u akumulatoru paran;

   IR – 6-bitni instrukcioni registar (*Instruction Register* – IR): čuva ***opkôd*** polje koje je deo kôda instrukcije, kao i polje modifikator ***M*** koji ukazuje na način adresiranja.

Pored registara, sastavni delovi staze podataka SCPU-a su i 16-bitna aritmetičko logička jedinica – ALU, i sedam trostatičkih drajvera (*ARBus*, *PCBus*, *DRBus*, *ACBus*, *IRBus*, *REGBus*, *MEMBusR*) koji se koriste za dozvolu dodele pristupa na magistrali i jedan za upis u memoriju (*MEMBusW*). Signal *CLK* (*Clock*) se koristi za globalno taktovanje (vidi Sliku 2).

1.2.2 Aritmetičko logička jedinica

Aritmetičko logička jedinica (*Arithmetic Logic Unit* – ALU) (vidi Sliku 3) je izvršna jedinica i predstavlja deo procesora koji vrši neposrednu obradu podataka. Od logičkih operacija ALU obavlja AND, OR, NOT i XOR, a od aritmetičkih ADD, SUB i NEG.

Kod operacije LD ALU direktno prosleđuje izvorni memorijski operand AC-u.

Izvorni 16-bitni operandi se dovode na ulaz ALU-a preko linija *AC*[15:0] i *DR*[15:0], a rezultat se dobija na *OUT*[15:0].

Logičke operacije obavljaju sledeći gradivni blokovi:

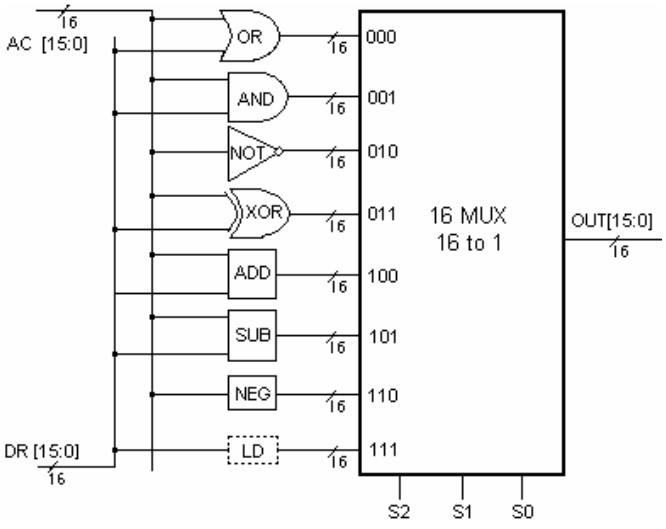
- **n** OR obavlja pobitnu logičku OR operaciju nad 16-bitnim operandima
- **n** AND obavlja pobitnu logičku AND operaciju nad 16-bitnim operandima
- **n** NOT obavlja logičku NOT operaciju nad 16-bitnim operandom
- **n** XOR obavlja pobitnu logičku XOR operaciju nad 16-bitnim operandima

Aritmetičke operacije obavljaju sledeći gradivni blokovi:

- **n** ADD obavlja aritmetičku operaciju ADD nad 16-bitnim operandima
- **n** SUB obavlja aritmetičku operaciju SUB nad 16-bitnim operandima
- **n** NEG obavlja aritmetičku operaciju NEG nad 16-bitnim operandom, tj. određuje dvoični komplement

- **n** LD prosleđuje 16-bitni izvorišni operand sa linije DR[15:0] na ulaz MUX

Na osnovu tipa operacije multiplekser MUX, tipa 8-u-1, selektuje jedan od ulaznih signala. Selektorski ulazi *S0*, *S1*, *S2* (naznačeni kao *ALUSEL* na Slici 2) definišu koji će se od ulaza proslediti na izlaz.



Slika 3. Aritmetičko logička jedinica

Signali *ALUSEL* predstavljaju tri bita najmanje težine polja ***opkôda*** instrukcije (vidi Sliku 4).

opkôd				***M***	
X	**S2**	**S1**	**S0**	Y	Y

Slika 4. Sadržaj IR registra

Na primer, ako se izvršava operacija ADD tada imamo da je *S2*={1}, *S1*={0} i *S0*={0}, dok ćemo kod operacije XOR imati da je *S2*={0}, *S1*={1} i *S0*={1}. Operaciji premeštanja odgovara sledeća kombinacija selektorskih signala *S2*={1}, *S1*={1} i *S0*={1}.

1.3 Adresni načini rada

Svakom operandu instrukcije pridružuje se podatak. Podatak može biti lociran u memoriji, CPU-ovim registrima (R0, R1, R2 ili R3) ili da predstavlja deo instrukcije (neposredni operand). Da bi izvršio instrukciju, procesor treba da pribavi tekuću vrednost podatka. Lokacija na kojoj se nalazi podatak može se specifirati na nekoliko načina. Specifikaciju lokacije operanda nazivamo način adresiranja. U zavisnosti od toga na koji način se obavlja specifikacija kažemo da postoje nekoliko različitih načina adresiranja. Za konkretni slučaj, procesor SCPU podržava sledeće načine adresiranja:

1. neposredni,
2. direktni,
3. indirektni, i
4. registarski.

1.3.1 Neposredno adresiranje

Kod neposrednog adresiranja adresni deo instrukcije (polje ***operand*** – vidi Sliku 5) specificira vrednost operanda, tj. operand je konstanta i sastavni je deo instrukcije. Polje ***operand*** je obima 10 bitova što znači da konstanta može da uzima vrednosti od 0 do 1023 kod neoznačenih vrednosti i od –512 do +511 kod označenih vrednosti.

Osnovne karakteristike neposrednog načina adresiranja su:

- Koristi se da definiše konstante, ili da postavi na inicijalne vrednosti promenljive u programu. Obično, brojevi se memorišu u prezentaciji dvoičnog komplementa. Bit najveće težine u polju operanda se koristi kao bit znaka. Kada se operand puni u registar, vrši se znakovno proširenje do obima koji odgovara punoj reči.
- Operand se pribavlja u CPU u trenutku kada se pribavlja i instrukcija; na ovaj način se štedi jedan memorijski ili keš ciklus po instrukciji.
- Obim podatka koji se pribavlja ograničen je obimom adresnog polja, i kod najvećeg broja skupova instrukcija mali je u odnosu na dužinu reči.

Kod neposrednog adresiranja polje modifikator ***M*** ima vrednost 00.

Opkôd				***M***		***operand***									
X	X	X	X	**0**	**0**	A	A	A	A	A	A	A	A	A	A

Slika 5. Format instrukcije kod neposrednog adresiranja

1.3.2 Direktno adresiranje

Kod direktnog načina adresiranja polje ***direktna adresa*** (vidi Sliku 6) specificira adresu memorijske lokacije koja odgovara podatku kome se pristupa. Naime, adresni deo instrukcije predstavlja adresu na osnovu koje se vrši obraćanje memoriji, pa se zbog toga ova adresa naziva direktna adresa. U konkretnom slučaju polje ***direktna adresa*** je 10-bitno što znači da se može adresirati memorija kapaciteta 1024 lokacija.

Prednost ovakvog načina adresiranja je jednostavnost izvođenja, a nedostatak ograničeni adresni opseg.

Kod direktnog adresiranja modifikator ***M*** ima vrednost 01.

Opkôd				***M***		***direktna adresa***									
X	X	X	X	**0**	**1**	A	A	A	A	A	A	A	A	A	A

Slika 6. Format instrukcije kod direktnog adresiranja

1.3.3 Indirektno adresiranje

Kod indirektnog adresiranja, odgovarajućim poljem ***indirektna adresa*** (vidi Sliku 7) se specificira memorijska lokacija u kojoj se čuva adresa operanda. Kod direktnog adresiranja, da bi se dobila vrednost operanda potrebna je jedna operacija pribavljanja, dok su kod indirektnog potrebne dve.

Osnovna prednost ove tehnike adresiranja je mogućnost pristupa većem adresnom prostoru, a nedostatak veći broj obraćanja memoriji.

Kod indirektnog adresiranja modifikator ***M*** ima vrednost 10.

opkôd				***M***		***indirektna adresa***									
X	X	X	X	**1**	**0**	A	A	A	A	A	A	A	A	A	A

Slika 7. Format instrukcije kod indirektnog adresiranja

1.3.4 Registarsko adresiranje

Kod registarskog adresiranja, odgovarajućim poljem ***adresa registra*** (vidi Sliku 8) se specificira registar u kome je smešten operand.

Prednosti registarskog adresiranja su:

- Adresno polje instrukcije kojim se specificira registar je malo i obima je nekoliko bitova. U konkretnom slučaju u polju ***adresa registra*** od važnosti su samo dva LS bita. Bitovi polja ***adresa registra*** označeni sa "-" nisu od važnosti. Dekodiranjem zadnja dva LS bita vrši se selekcija registara shodno sledećoj tabeli:

AA	Selekcija registra
00	R0
01	R1
10	R2
11	R3

- Ne vrši se obraćanje memoriji;
- Instrukcije koje manipulišu sa sadržajem registra brzo se izvršavaju.

Kod registarskog adresiranja modifikator ***M*** ima vrednost 11.

opkod				***M***		***adresa registra***									
X	X	X	X	**1**	**1**	-	-	-	-	-	-	-	-	A	A

Slika 8. Format instrukcije kod registarskog adresiranja

Napomena: *Broj koji specificira adresu registra ne može biti veći od 3 dekadno, tj. 11 binarno.*

1.4 Format i način izvršenja asemblerskih instrukcija

Asemblerska instrukcija je simbolička verzija mašinske instrukcije. Instrukcije pišemo na asemblerskom jeziku. Notacija asemblerskog jezika bliska je ljudskom načinu razmišljanja. Ovim se programiranje znatno olakšava, ali s obzirom da procesor razume samo mašinske instrukcije (skup bitova) neophodno je asemblerske instrukcije prevesti. Program za prevođenje simboličke notacije u binarnu nazivamo *asembler*.

U konkretnom slučaju, format asemblerskih instrukcija je sledeći:

`mnemonik` [modifikator] [operand]

gde: [..] ukazuje na opciono polje

`mnemonik` – ime kôda instrukcije, ovo polje je obavezno

modifikator – ukazuje na način adresiranja i to na sledeći načini:

#	– neposredno adresiranje;
@	– indirektno adresiranje;
$	– registarsko adresiranje;
bez modifikatora	– direktno adresiranje.

operand – specificira adresu ili registar koji se koristi kao operand mašinske instrukcije. Može da bude:

- neposredna vrednost operanda (<operand>), kod neposrednog adresiranja,
- da ukazuje na memorijsku lokaciju (M) gde je smeštena vrednost operanda, kod direktnog adresiranja,
- da ukazuje na memorijsku lokaciju (adrM) gde je smeštena adresa memorijske lokacije sa vrednošću operanda, kod indirektnog adresiranja,
- da specificira registar opšte namene (Rx) u koji je smeštena vrednost operanda, kod registarskog načina adresiranja.

Zadaje se kao decimalni ili heksadecimalni broj ako je zadnji karakter `H` ili `h`. Vrednost operanda je 10-bitna, tj. u granicama 0 ÷ 1023.

1.4.1 Skup instrukcija procesora SCPU

Skup naredbi SCPU čine sledećih petnaest instrukcija koje će biti analizirane po rastućem redosledu vrednosti opkôd polja formata instrukcije:

`OR`

Opkôd: 0000
Operacija: (operand) ∨ (akumulator) → akumulator
Asembler sintaksa: `OR` [modifikator] [operand]

Objašnjenje: Instrukcijom `OR` se izvršava logička ILI operacija nad izvorišnim operandom specificiranim instrukcijom i odredišnim operandom AC, a rezultat se smešta u AC. U zavisnosti od načina adresiranja srećemo sledeće varijante

Način adresiranja	Tip operacije	Aktivnost
neposredno	`OR` #operand	<operand> ∨ AC → AC
direktno	`OR` M	M ∨ AC → AC
indirektno	`OR` @adrM	M ∨ AC → AC
registarsko	`OR` $Rx	Rx ∨ AC → AC

AND

Opkôd: 0001
Operacija: (operand) $\wedge$ (akumulator) $\rightarrow$ akumulator
Asembler sintaksa: AND [modifikator] [operand]

Objašnjenje: Instrukcijom AND se izvršava logička I operacija nad izvorišnim operandom specificiranim instrukcijom i odredišnim operandom AC, a rezultat se smešta u AC. U zavisnosti od načina adresiranja srećemo sledeće varijante

Način adresiranja	Tip operacije	Aktivnost
neposredno	AND #operand	<operand> $\wedge$ AC $\rightarrow$ AC
direktno	AND M	M $\wedge$ AC $\rightarrow$ AC
indirektno	AND @adrM	M $\wedge$ AC $\rightarrow$ AC
registarsko	AND $Rx	Rx $\wedge$ AC $\rightarrow$ AC

NOT

Opkôd: 0010
Operacija: /(akumulator) $\rightarrow$ akumulator
Asembler sintaksa: NOT

Objašnjenje: Instrukcijom NOT se izvršava logička operacija negacije nad sadržajem AC-a, a rezultat se smešta u AC.

XOR

Opkôd: 0011
Operacija: (operand) $\oplus$ (akumulator) $\rightarrow$ akumulator
Asembler sintaksa: XOR [modifikator] [operand]

Objašnjenje: Instrukcijom XOR se izvršava logička isključivo ILI operacija nad izvorišnim operandom specificiranim instrukcijom i odredišnim operandom AC, a rezultat se smešta u AC. U zavisnosti od načina adresiranja srećemo sledeće varijante

Način adresiranja	Tip operacije	Aktivnost
neposredno	XOR #operand	<operand> $\oplus$ AC $\rightarrow$ AC
direktno	XOR M	M $\oplus$ AC $\rightarrow$ AC
indirektno	XOR @adrM	M $\oplus$ AC $\rightarrow$ AC
registarsko	XOR $Rx	Rx $\oplus$ AC $\rightarrow$ AC

Instrukcije **OR**, **AND**, **NOT** i **XOR** su logičke instrukcije i njihovij izvršenjem, vrednosti markera se menjaju na sledeći način:

CARRY – ne menja vrednost
SIGN – postavlja se na {1} ako je MS bit rezultata jednak {1}, inače ako je MS bit rezultata OR operacije {0} ovaj marker se postavlja na vrednost {0}
ZERO – postavlja se na {1} ako je rezultat nula, inače se postavlja na {0}
PARITY – postavlja se na {1} ako je broj jedinica u rezultatu paran, ako je neparan postavlja se na {0}

ADD

Opkôd: 0100
Operacija: (operand) + (akumulator) → akumulator
Asembler sintaksa: ADD [modifikator] [operand]

Objašnjenje: Instrukcijom ADD se izvršava aritmetička operacija sabiranje nad izvorišnim operandom specificiranim instrukcijom i odredišnim operandom AC, a rezultat se smešta u AC. U zavisnosti od načina adresiranja srećemo sledeće varijante

Način adresiranja	Tip operacije	Aktivnost
Neposredno	ADD #operand	<operand> + AC → AC
Direktno	ADD M	M + AC → AC
Indirektno	ADD @adrM	M + AC → AC
Registarsko	ADD $Rx	Rx + AC → AC

Izvršenjem ove instrukcije menjaju se vrednosti sledećih markera:

CARRY – postavlja se na {1} ako se nakon izvršavanja operacije javio prenos
SIGN – postavlja se na {1} ako je MS bit rezultata jednak {1}, inače ako je MS bit rezultata OR operacije {0} ovaj marker se postavlja na vrednost {0}
ZERO – postavlja se na {1} ako je rezultat nula, inače se postavlja na {0}
PARITY – postavlja se na {1} ako je broj jedinica u rezultatu paran, ako je neparan postavlja se na {0}

SUB

Opkôd: 0101
Operacija: (operand) – (akumulator) → akumulator
Asembler sintaksa: SUB [modifikator] [operand]

Objašnjenje: Instrukcijom SUB se izvršava aritmetička operacija oduzimanja nad izvorišnim operandom specificiranim instrukcijom i odredišnim operandom AC, a rezultat se smešta u AC. U zavisnosti od načina adresiranja srećemo sledeće varijante

Način adresiranja	Tip operacije	Aktivnost
Neposredno	SUB #operand	<operand> – AC → AC
Direktno	SUB M	M – AC → AC
Indirektno	SUB @adrM	M – AC → AC
Registarsko	SUB $Rx	Rx – AC → AC

Izvršenjem ove instrukcije menjaju se vrednosti sledećih markera:

CARRY – postavlja se na {1} ako se nakon izvršavanja operacije javilo pozajmljivanje sa bita najveće težine
SIGN – postavlja se na {1} ako je MS bit rezultata jednak {1}, inače ako je MS bit rezultata OR operacije {0} ovaj marker se postavlja na vrednost {0}
ZERO – postavlja se na {1} ako je rezultat nula, inače se postavlja na {0}
PARITY – postavlja se na {1} ako je broj jedinica u rezultatu paran, ako je neparan postavlja se na {0}

NEG

Opkôd: 0110
Operacija: /(akumulator) + 1 → akumulator
Asembler sintaksa: NEG

Objašnjenje: Instrukcijom NEG se izvršava aritmetička operacija negacije, kod prezentacije brojeva u drugom komplementu, nad sadržajem AC-a, a rezultat se smešta u AC.

Izvršenjem ove instrukcije menjaju se vrednosti sledećih markera:

CARRY – postavlja se na {1} ako se nakon izvršavanja operacije javio prenos
SIGN – postavlja se na {1} ako je MS bit rezultata jednak {1}, inače ako je MS bit rezultata OR operacije {0} ovaj marker se postavlja na vrednost {0}
ZERO – postavlja se na {1} ako je rezultat nula, inače se postavlja na {0}
PARITY – postavlja se na {1} ako je broj jedinica u rezultatu paran, ako je neparan postavlja se na {0}

LD

Opkôd: 0111
Operacija: (operand) → akumulator
Asembler sintaksa: LD [modifikator] [operand]

Objašnjenje: Instrukcijom za prenos podataka, LD, se izvršava operacija upisivanja sadržaja izvorišnog operanda specificiranog instrukcijom u akumulator. U zavisnosti od načina adresiranja srećemo sledeće varijante

Način adresiranja	Tip operacije	Aktivnost
neposredno	LD #operand	<operand> → AC
Direktno	LD M	M → AC
indirektno	LD @adrM	M → AC
registarsko	LD $Rx	Rx → AC

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

JMP

Opkôd: 1000
Operacija: (operand) → programski brojač
Asembler sintaksa: JMP [operand]

Objašnjenje: Instrukcijom JMP se izvršava operacija bezuslovnog grananja (skoka) na memorijsku lokaciju specificiranu instrukcijom.

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

JZ

Opkôd: 1001
Operacija: if *ZERO* = {1} then (operand) → PC else continue
Asembler sintaksa: JZ [operand]

Objašnjenje: Instrukcijom JZ se izvršava operacija uslovnog grananja (skoka) na memorijsku lokaciju specificiranu instrukcijom ako je marker *ZERO* postavljen na {1}.

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

JNZ

Opkôd: 1010
Operacija: if *ZERO* = {0} (operand) → PC else continue
Asembler sintaksa: JNZ [operand]

Objašnjenje: Instrukcijom JNZ se izvršava operacija uslovnog grananja (skoka) na memorijsku lokaciju specificiranu instrukcijom ako je marker *ZERO* postavljen na {0}.

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

INC

Opkôd: 1011
Operacija: (akumulator) + 1 → akumulator
Asembler sintaksa: INC

Objašnjenje: Instrukcijom INC se inkrementira sadržaj akumulatora.

Izvršenjem ove instrukcije menjaju se vrednosti sledećih markera:

CARRY – postavlja se na {1} ako se nakon izvršavanja operacije javio prenos
SIGN – postavlja se na {1} ako je MS bit rezultata jednak {1}, inače ako je MS bit rezultata OR operacije {0} ovaj marker se postavlja na vrednost {0}
ZERO – postavlja se na {1} ako je rezultat nula, inače se postavlja na {0}
PARITY – postavlja se na {1} ako je broj jedinica u rezultatu paran, ako je neparan postavlja se na {0}

MOV

Opkôd: 1100
Operacija: (akumulator) → Rx
Asembler sintaksa: MOV [operand]

Objašnjenje: Instrukcijom za prenos podataka, MOV, se izvršava operacija upisivanja sadržaja akumulatora u registar opšte namene specificiran instrukcijom.

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

ST

Opkôd: 1101
Operacija: (akumulator) → Memorija
Asembler sintaksa: ST [operand]

Objašnjenje: Instrukcijom za prenos podataka, ST, se izvršava operacija upisivanja sadržaja akumulatora u memorijsku lokaciju specificiranu instrukcijom.

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

NOP

Opkôd: 1111
Operacija: Nema operacije
Asembler sintaksa: NOP

Objašnjenje: Instrukcija NOP je operacija bez efekta, inkrementira se sadržaj programskog brojača, program nastavlja sa izvršenjem.

Izvršenje ove instrukcije nema efekat na postavljanje statusnih markera.

Primer:

Ako je početna vrednost akumulatora AC = 30h, memorijskih lokacija M[61] = 50, M[42] = 112, M[112] = 333h i registra opšte namene R2 = 28431, sadržaj akumulatora AC i markera statusa (***CARRY***, ***SIGN***, ***ZERO*** i ***PARITY***) nakon pojedinačnog izvršenja svake od sledećih instrukcija je:

a) OR #95

$95 \vee AC \rightarrow AC$
$(95)_{dec} = (0000\ 0000\ 0101\ 1111)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
$0000\ 0000\ 0101\ 1111 \vee 0000\ 0000\ 0011\ 0000 = 0000\ 0000\ 0111\ 1111$

AC = 0000 0000 0111 1111
CARRY = 0
SIGN = 0
ZERO = 0
PARITY = 1

b) OR 61

$M[61] \vee AC \rightarrow AC$
$(50)_{dec} = (0000\ 0000\ 0011\ 0010)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
$0000\ 0000\ 0011\ 0010 \vee 0000\ 0000\ 0011\ 0000 = 0000\ 0000\ 0011\ 0010$

AC = 0000 0000 0011 0010
CARRY = 0
SIGN = 0
ZERO = 0
PARITY = 1

c) AND @42

$M[M[42]] \wedge AC \rightarrow AC$
$M[M[42]] = M[112]$
$(333)_{hex} = (0000\ 0011\ 0011\ 0011)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
$0000\ 0011\ 0011\ 0011 \wedge 0000\ 0000\ 0011\ 0000 = 0000\ 0000\ 0011\ 0000$

AC = 0000 0000 0011 0000
CARRY = 0
SIGN = 0
ZERO = 0
PARITY = 1

d) NOT

$/AC \rightarrow AC$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$

AC = 1111 1111 1100 1111
CARRY = 0
SIGN = 1
ZERO = 0
PARITY = 1

e) AND $2

R2 $\wedge$ AC $\rightarrow$ AC
$(28431)_{dec} = (0110\ 1111\ 0000\ 1111)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
0110 1111 0000 1111 $\wedge$ 0000 0000 0011 0000 = 0000 0000 0000 0000

AC = 0000 0000 0000 0000
CARRY = 0
SIGN = 0
ZERO = 1
PARITY = 1

f) XOR #112

112 $\oplus$ AC $\rightarrow$ AC
$(112)_{dec} = (0000\ 0000\ 0111\ 0000)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
0000 0000 0111 0000 $\oplus$ 0000 0000 0011 0000 = 0000 0000 0100 0000

AC = 0000 0000 0100 0000
CARRY = 0
SIGN = 0
ZERO = 0
PARITY = 0

g) ADD @42

M[M[42]] + AC $\rightarrow$ AC
M[M[42]] = M[112]
$(333)_{hex} = (0000\ 0011\ 0011\ 0011)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
0000 0011 0011 0011 + 0000 0000 0011 0000 = 0000 0011 0110 0011

AC = 0000 0011 0110 0011
CARRY = 0
SIGN = 0
ZERO = 0
PARITY = 1

h) SUB #196

112 – AC $\rightarrow$ AC
$(196)_{dec} = (0000\ 0000\ 1100\ 0100)_{bin}$
$(30)_{hex} = (0000\ 0000\ 0011\ 0000)_{bin}$
0000 0000 0111 0000 – 0000 0000 0011 0000 = 0000 0000 1001 0100

AC = 0000 0000 1001 0100
CARRY = 0
SIGN = 0
ZERO = 0
PARITY = 0

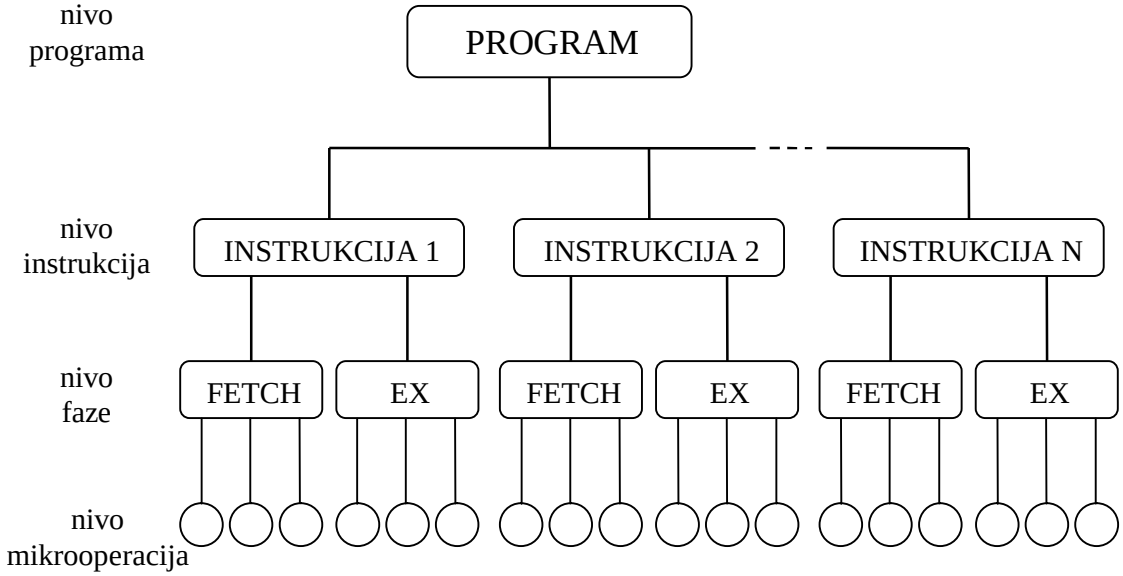
1.4.2 Pseudoinstrukcije

Asembler procesora SCPU podržava sledeće dve pseudoinstrukcije:
DSM <podatak> : smesti podatak u memoriju neposredno iza lokacije predhodne instrukcije
RSx <podatak> : upiši podatak u registar opšte namene Rx, gde x∈ [0..3].

1.4.3 Faze izvršavanja instrukcije

Kao što je prikazano na Slici 9 program koji izvršava procesor SCPU čini skup instrukcija (1 < I < N). Izvršenje instrukcije u grubim crtama se može podeliti na dve faze, fazu pribavljanja, FETCH, koja uključuje ciklus pribavljanja i dekodiranja instrukcije, i fazu izvršenja, EX. Svaku od faza čini veći broj koraka koji mogu da uključe paralelno izvršenje većeg broja mikrooperacija. Mikrooperacije su atomizirane akcije, tj. elementarne operacije koje se ne mogu dalje razbijati.

Faza pribavljanja, FETCH, je identična za sve instrukcije. Faza dekodiranja se odnosi na određivanje tipa instrukcije i smatra se da je to aktivnost tipa višestruko-grananje (*multi-way branch*), i obavlja se trenutno (zbog toga je na Slici 9 njeno vreme trajanja nula). Faza EX je različita za sve tipove instrukcija.



Slika 9. Nivoi izvršavanja programa

U konkretnom slučaju fazu FETCH čine sledeća tri koraka:

FETCH1 : AR ← PC
FETCH2 : DR ← M , PC ← PC + 1
FETCH3 : IR ← DR[15:10] , AR ← DR[9:0]

gde su FETCH1, FETCH2 i FETCH3 koraci, a aktivnosti tipa AR ← PC , PC ← PC + 1, itd. predstavljaju mikrooperacije.

U koraku FETCH1 sadržaj programskog brojača, PC, se upisuje u memorijsko adresni registar, AR. Adresira se memorija MEM i generiše signal *Read* (vidi Sliku 2).

U koraku FETCH2 čita se sadržaj adresirane memorijske lokacije i upisuje u registar DR. Istovremeno se sadržaj PC-a inkrementira sa ciljem da ukaže na narednu lokaciju u memoriji kojoj treba pristupiti.

U koraku FETCH3 delimični sadržaj registra DR, bitovi d15 ÷ d10 se upisuju u instrukcioni registar, IR, dok se LS bitovi d9 ÷ d0 registra DR upisuju u AR.

Radi pojednostavljenja usvaja se da se svaki od koraka FETCH1, FETCH2 i FETCH3 obavlja za jedan taktni interval što znači da se faza pribavljanja izvršava za tri taktna intervala.

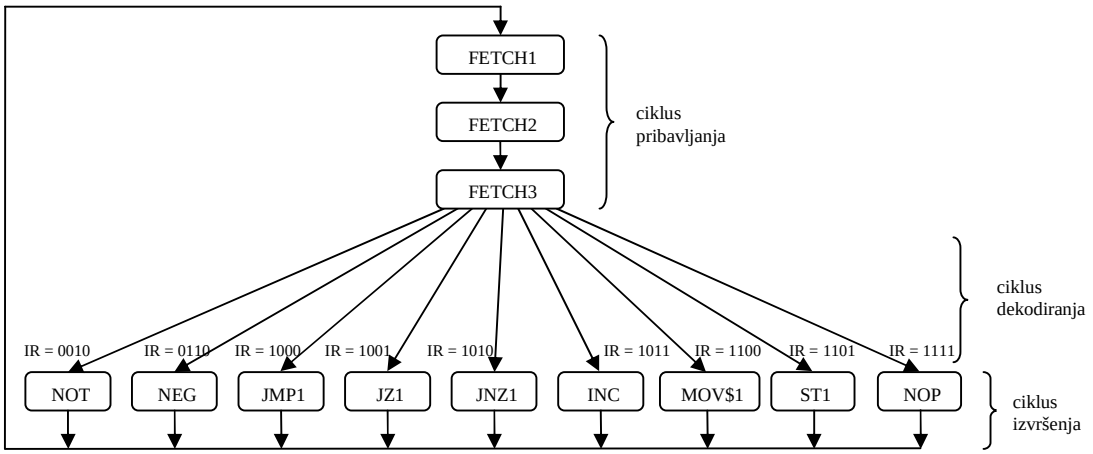
Aktivnost dekodiranje instrukcije smatraćemo da se za sve instrukcije izvršava trenutno.

Broj mikrooperacija u fazi izvršenja, EX, zavisi od tipa instrukcija i adresnog načina rada.

Radi bolje ilustracije principa rada procesora, u daljem tekstu, slikovito ćemo ukazati na načine izvršenja različitih tipova instrukcija polazeći od broja taktnih intervala potrebnih da se obavi faza izvršenja instrukcije EX.

1.4.3.1 Instrukcije kod kojih faza izvršenja EX traje jedan taktni interval

Skup instrukcija kod kojih faza izvršenja EX traje jedan taktni interval (jedan korak) prikazan je na Slici 10. Broj mikrooperacija u okviru faze EX u zavisnosti od tipa instrukcije može biti različit (vidi Sliku 10 b))



a) Aktivnosti kod izvršenja instrukcija

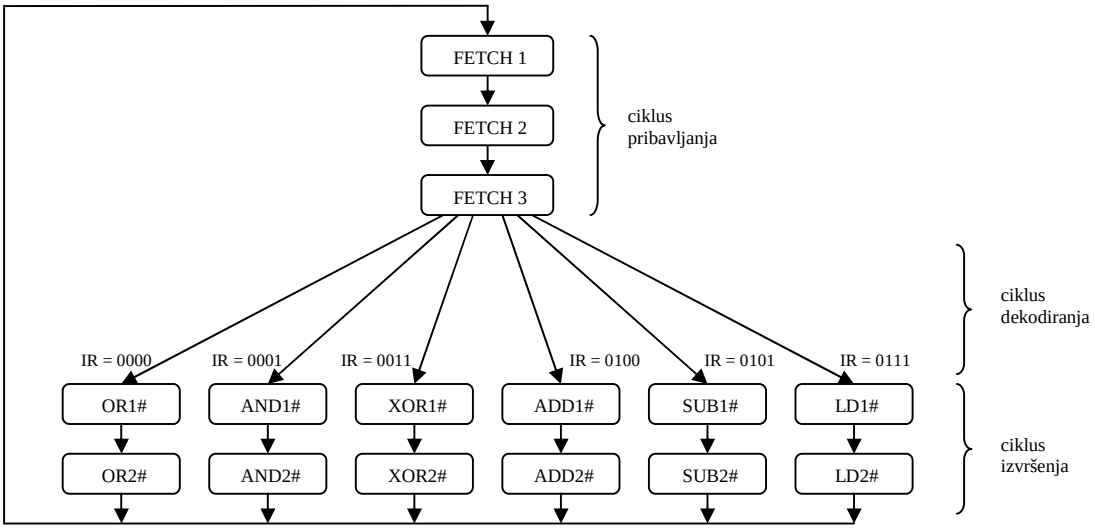
NOT : AC ← not AC
NEG : AC ← - AC
JMP1 : PC ← AR
JZ1 : if ***ZERO*** = 0 then PC ← AR else continue
JNZ1 : if ***ZERO*** = 1 then PC ← AR else continue
INC : AC ← AC + 1
MOV$1 : R[AR[1:0]] ← AC
ST1 : M[AR] ← AC
NOP : no operation

b) Mikrooperacije kod različitih tipova instrukcija

Slika 10. Aktivnosti u toku faza FETCH i EX kod instrukcija kada faza EX traje jedan korak

4.3.2 Instrukcije kod kojih faza izvršenja EX traje dva taktna intervala

Na Slici 11 prikazana je grupa instrukcija čija faza izvršenja traje dva taktna intervala. Slika 11 a) se odnosi na aktivnosti instrukcija koje koriste neposredni način adresiranja, a Slika 11 b) na mikrooperacije u fazi EX za ovaj način adresiranja.



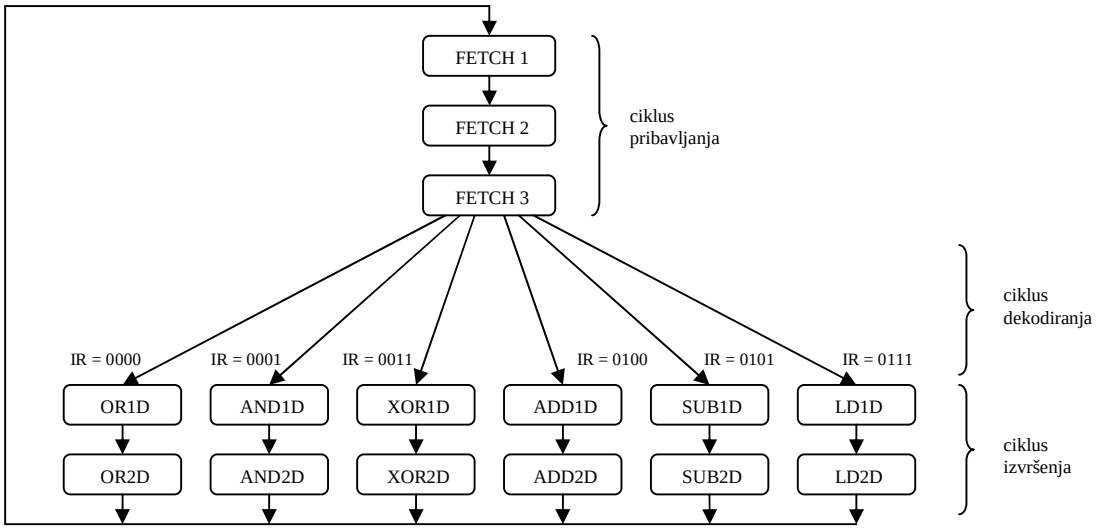
a) Aktivnosti kod izvršenja instrukcija

OR1# : DR ← AR
OR2# : AC ← AC or DR
AND1# : DR ← AR
AND2# : AC ← AC and DR
XOR1# : DR ← AR
XOR2# : AC ← AC xor DR
ADD1# : DR ← AR
ADD2# : AC ← AC + DR
SUB1# : DR ← AR
SUB2# : AC ← AC - DR
LD1# : DR ← AR
LD2# : AC ← DR

b) Mikrooperacije kod različitih tipova instrukcija

Slika 11. Aktivnosti u toku faza FETCH i EX kod instrukcija kada faza EX traje dva koraka

Slika 11 c) se odnosi na aktivnosti instrukcija koje koriste direktni način adresiranja, a Slika 11 d) na mikrooperacije u fazi EX za direktno memorijski način adresiranja.



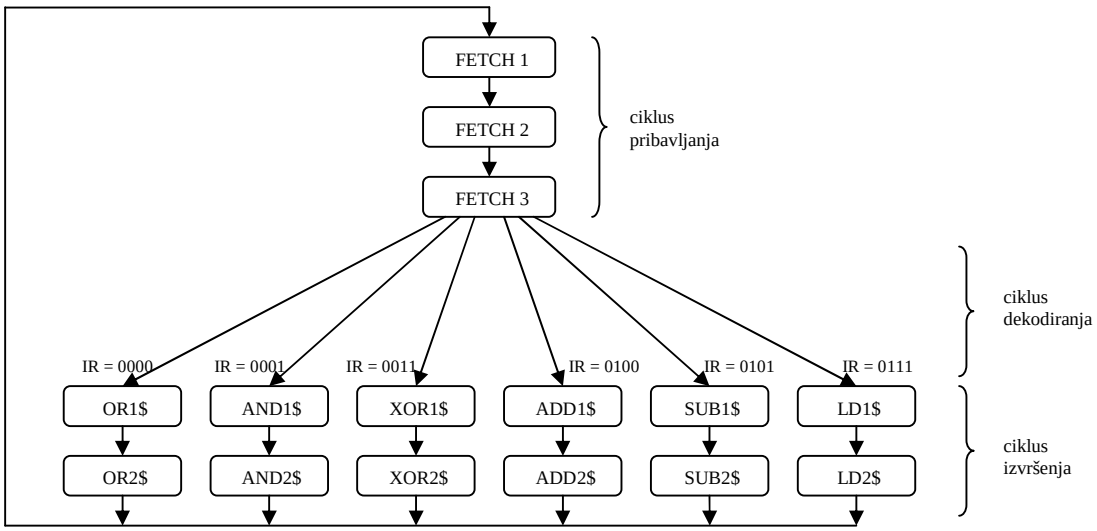
c) Aktivnosti kod izvršenja instrukcija

OR1D : DR ← M
OR2D : AC ← AC or DR
AND1D : DR ← M
AND2D : AC ← AC and DR
XOR1D : DR ← M
XOR2D : AC ← AC xor DR
ADD1D : DR ← M
ADD2D : AC ← AC + DR
SUB1D : DR ← M
SUB2D : AC ← AC - DR
LD1D : DR ← M
LD2D : AC ← DR

d) Mikrooperacije kod različitih tipova instrukcija

Slika 11. Aktivnosti u toku faza FETCH i EX kod instrukcija kada faza EX traje dva koraka

Slika 11 e) se odnosi na aktivnosti instrukcija koje koriste registarski način adresiranja, a Slika 11 f) na mikrooperacije u fazi EX za registarski način adresiranja.



e) Aktivnosti kod izvršenja instrukcija

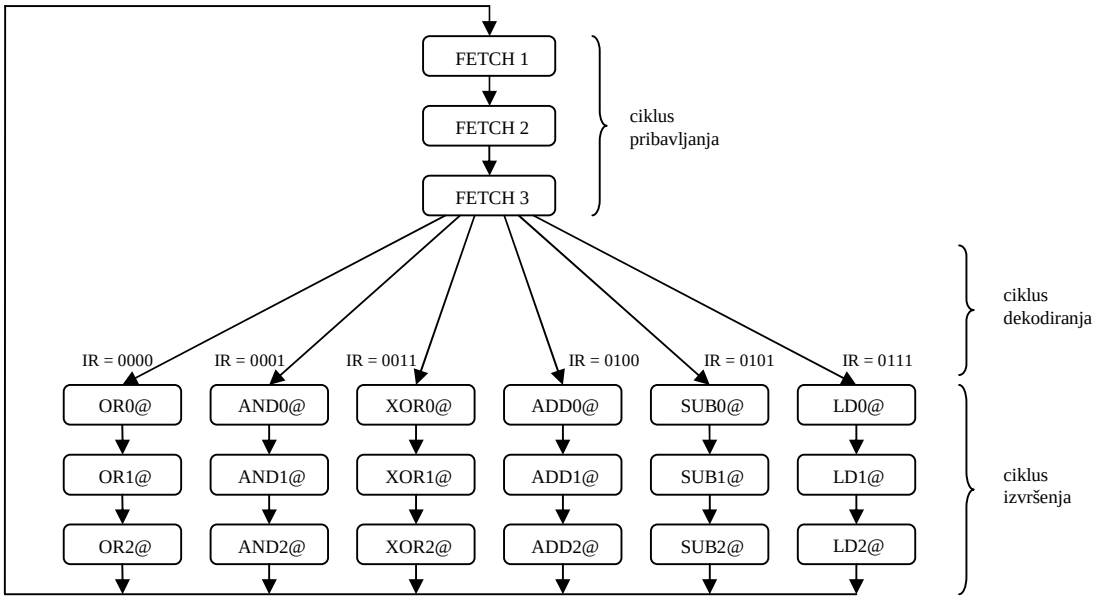
OR1$: DR ← REG[AR[1:0]]
OR2$: AC ← AC or DR
AND1$: DR ← REG[AR[1:0]]
AND2$: AC ← AC and DR
XOR1$: DR ← REG[AR[1:0]]
XOR2$: AC ← AC xor DR
ADD1$: DR ← REG[AR[1:0]]
ADD2$: AC ← AC + DR
SUB1$: DR ← REG[AR[1:0]]
SUB2$: AC ← AC - DR
LD1$: DR ← REG[AR[1:0]]
LD2$: AC ← DR

f) Mikrooperacije kod različitih tipova instrukcija

Slika 11. Aktivnosti u toku faza FETCH i EX kod instrukcija kada faza EX traje dva koraka

4.3.3 Instrukcije kod kojih faza izvršenja EX traje tri taktna intervala

Na Slici 12 prikazana je grupa instrukcija čija faza izvršenja traje tri taktna intervala. Slika 12 a) se odnosi na aktivnosti instrukcija koje koriste indirektno memorijski način adresiranja, a Slika 12 b) na mikrooperacije u fazi EX za ovaj način adresiranja.



a) Aktivnosti kod izvršenja instrukcija

OR0@ : AR ← M[9:0]
OR1@ : DR ← REG[AR[1:0]]
OR2@ : AC ← AC or DR
AND0@ : AR ← M[9:0]
AND1@ : DR ← REG[AR[1:0]]
AND2@ : AC ← AC and DR
XOR0@ : AR ← M[9:0]
XOR1@ : DR ← REG[AR[1:0]]
XOR2@ : AC ← AC xor DR
ADD0@ : AR ← M[9:0]
ADD1@ : DR ← REG[AR[1:0]]
ADD2@ : AC ← AC + DR
SUB0@ : AR ← M[9:0]
SUB1@ : DR ← REG[AR[1:0]]
SUB2@ : AC ← AC - DR
LD0@ : AR ← M[9:0]
LD1@ : DR ← REG[AR[1:0]]
LD2@ : AC ← DR

b) Mikrooperacije kod različitih tipova instrukcija

Slika 12. Aktivnosti u toku faza FETCH i EX kod instrukcija kada faza EX traje tri koraka

2. Simulator *simCPU*

U ovoj vežbi biće opisan princip rada simulatora, ***simCPU*** jednostavnog procesora SCPU – a čija je arhitektura i skup instrukcija definisan u vežbi "Jednostavna sekvencijalna procesorska jedinica".

2.1 KORAK 1 – pokretanje programa *simCPU*

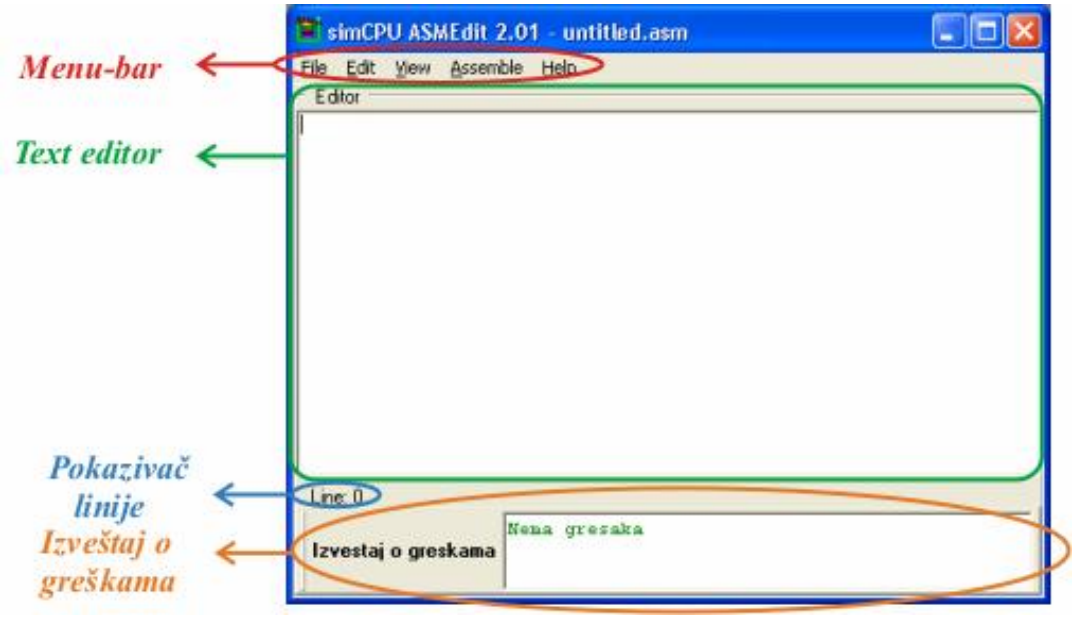
Za pokretanje programa ***simCPU*** za Windows potrebno je preduzeti sledeće aktivnosti:

a) izabrati programsku ikonu *SimCPU for Windows* lociranu na *Desktop*-u.
b) aktivirati ***simCPU*** – aktiviranje se izvodi dvostrukim klikom levim tasterom miša na odabranu ikonu ***simCPU***
c) startovanjem ***simCPU*** -a na ekranu se pojavljuje aplikacioni prozor pod nazivom *SimCPU ASMEdit 2.01 – untiled.asm* (vidi Sliku 1)

2.1.1 Text Editor

Aplikacioni prozor predstavlja *TEXT Editor* i sastoji se iz četiri dela kao što je prikazano na Slici 1:

1. *Menu-bar*
2. *Text editor*
3. *Pokazivač linije u kojoj se nalazi kursor*
4. *Izveštaj o greškama*



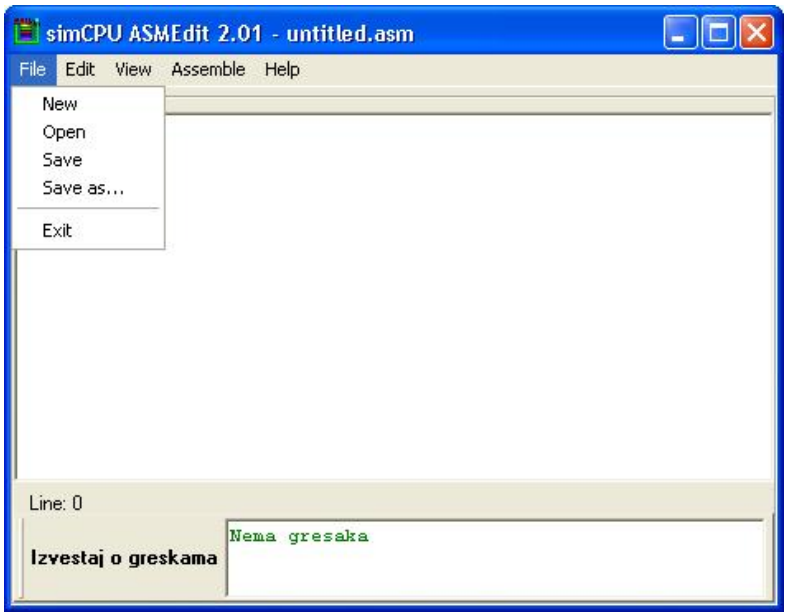
Slika 1. Text editor ***simCPU***-a

1. Sekcija na vrhu se odnosi na polje *Menu-bar*. *Menu-bar* omogućava selekciju *File, Edit, View, Assemble i Help* operacije.

File – selekcijom ove operacije otvara se padajući meni koji sadrži sledeće opcije:

- *New* – (selekcijom ove operacije) kreira se nova programska sekvenca
- *Open* – učitava se fajl izvornog programa nazvan *ime-prezime.asm* sa lokacije *D:\simCPU\Primeri*
- *Save* – pamti fajl koji se tekuće koriguje ili ažurira na mestu gde je bio prethodno zapamćen
- *Save as...* – dodeljuje ime fajlu (*ime-prezime.asm*) koji se tekuće kreira, bira lokaciju gde će se taj fajl zapamtiti *D:\simCPU\Primeri\ime-prezime* i pamti fajl koji je kreiran
- *Exit* - izlazak iz Text editora.

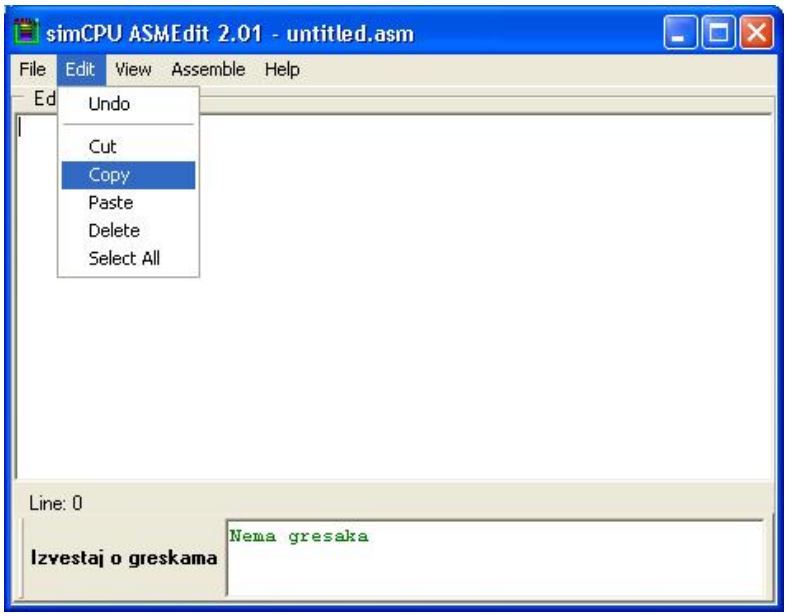
Izgled aplikacionog prozora je prikazan na Slici 2.



Slika 2. Text editor ***simCPU***-a sa selekcijom *File*

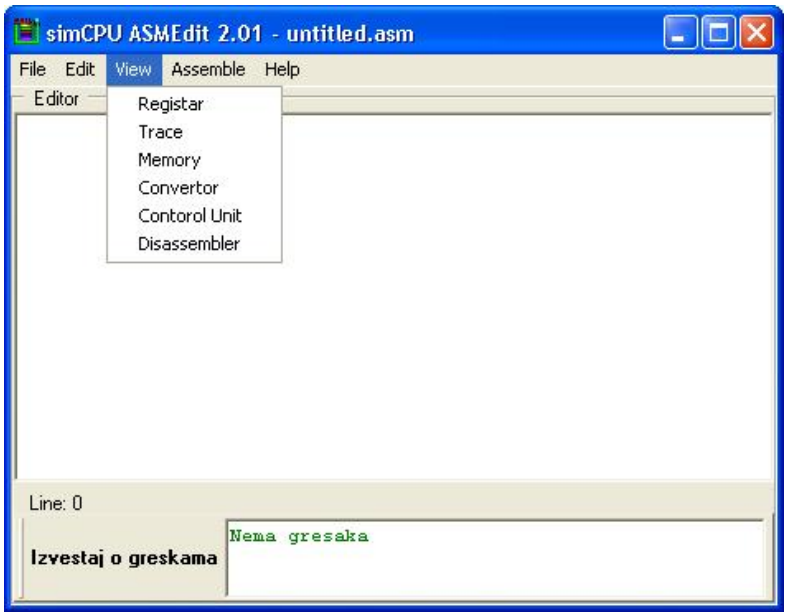
Edit – selekcijom ove operacije otvara se padajući meni (vidi Sliku 3) koji sadrži sledeće opcije:

- *Undo* – vraćanje na prethodno stanje (koristi se kada se napravi greška pri unosu podataka, grešku ispravljamo ako se vratimo na prethodno stanje)
- *Cut* – kopiranje selektovanog sadržaja. Kopiranjem, selektovani sadržaj se briše iz izvornog fajla.
- *Copy* - kopiranje selektovanog sadržaja. Kopiranjem, selektovani sadržaj se ne briše iz izvornog fajla.
- *Paste* – lepljenje prethodno kopiranog selektovanog sadržaja iz izvornog fajla.
- *Delete* – brisanje selektovanog sadržaja iz izvornog fajla.
- *Select All* – selektovanje celokupnog (integralnog) sadržaja izvornog fajla.



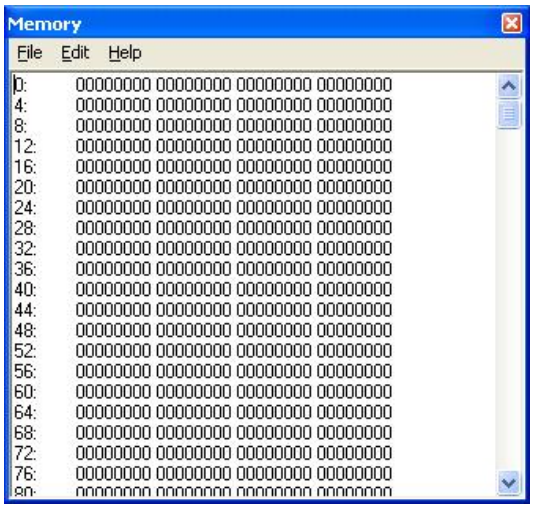
Slika 3. Text editor ***simCPU***-a sa selekcijom *Edit*

View – selekcijom ove operacije otvara se padajući meni (vidi Sliku 4) koji sadrži sledeće opcije:



Slika 4. Text editor ***simCPU***-a sa selekcijom *View*

■ *Registar* - prikazuje trenutni sadržaj svih registara jednostavnog procesora SCPU (vidi Sliku 11)
■ *Trace* – prikazuje sadržaj registara (AR, PC, DR, AC i IR) u toku izvršavanja svake od mikrooperacija.
■ *Memory* - prikazuje trenutni sadržaj svih memorijskih lokacija (vidi Sliku 5)



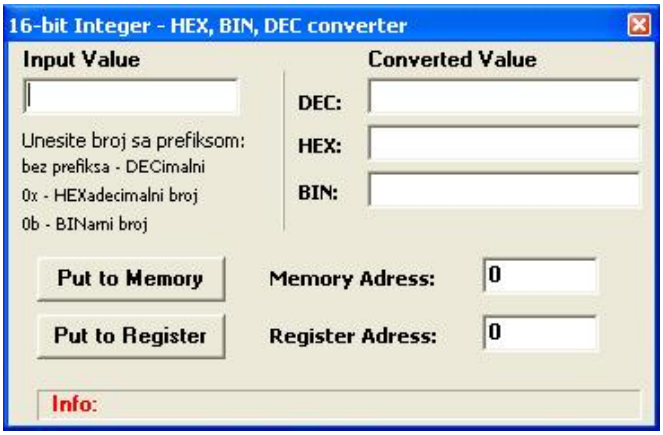
Slika 5. Memorija

■ *Convertor* – aktivira 16-bitni konvertor decimalnih, heksadecimalnih i binarnih podataka. Podatak se unosi u polje *Input Value* (vidi Sliku 6) u jednom od formata decimalni, heksadecimalni ili binarni. U poljima *DEC*, *HEX*, *BIN* dobija se odgovarajući ekvivalent u decimalnom, heksadecimalnom i binarnom obliku.

Uneti podatak moguće je upisati na adresu koja se zadaje kao decimalna vrednost (od 0 do 1023) u polje *Memory Adress*, a memorisanje unetog podatka na specificiranu adresu vrši se klikom levog tastera miša na polje *Put to Memory.*

Uneti podatak moguće je upisati u jednom od registara opšte namene R0 – R3. Specifikacija adrese registara se vrši upisom odgovarajuće decimalne vrednosti koja je u granicama od 0 do 3 u polje *Register Adress.* Memorisanje unetog podatka u specificirani registar vrši se klikom levog tastera miša na polje *Put to Register.*

U polje *Info:* ispisuje se poruka koja ukazuje na prekoračenje opsega unete vrednosti u polje *Input Value*.



Slika 6. Convertor

■ *Control Unit* – ovom opcijom vrši se nadgledanje izvršenja tekuće instrukcije. Biranjem ove opcije na displeju se prikazuje Slika 7 koja sadrži veći broj upravljačkih polja. Značenje ovih polja je sledeće:

● *Register CONTROL* - 7-bitno polje koga čine upravljačka bit-polja *ARLOAD*, *PCLOAD*, *PCINC*, *DRLOAD*, *ACLOAD*, *ACINC*, *IRLOAD*. Stanjem ovih bit-polja upravlja se operacijom punjenja registra (kao na primer *ARLOAD*) ili inkrementiranja sadržaja registra (kao što je *PCINC*). Upravljački bit je aktivan ako je postavljen na {1}.

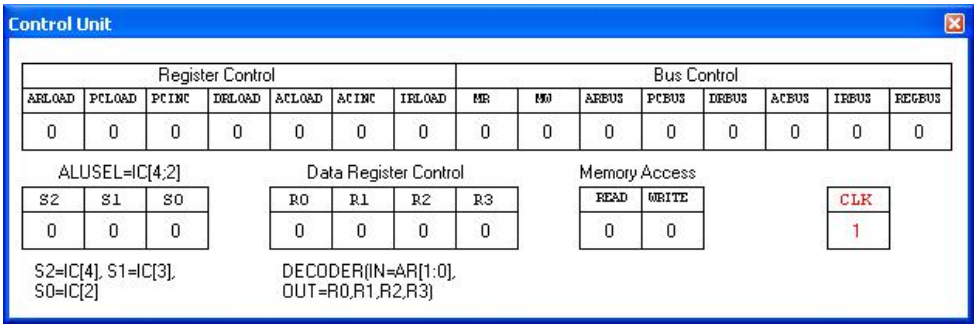
● *BUS CONTROL* – 7-bitno polje koga čine upravljačka bit-polja *MR*, *MW*, *ARBUS*, *PCBUS*, *DRBUS*, *ACBUS*, *IRBUS*, *REGBUS*. Sadržajem ovih bit-polja upravlja se izlazima trostatičkih drajvera čiji su izlazi povezani na 16-bitnu sistemsku magistralu ***simCPU***-a (vidi Sliku 1 iz vežbe Jednostavna sekvencijalna centralna procesorska jedinica). Izlaz trostatičkog drajvera je aktivan ako je odgovarajuće bit-polje postavljeno na logičku {1} (tako na primer, ako je bit-polje *ARBUS*={1} tada je sadržaj registra AR je dostupan na 16-bitnoj sistemskoj magistrali, vidi Sliku 1 iz vežbe Jednostavna sekvencijalna centralna procesorska jedinica).

● *ALUSEL* – selektorski ulazi koji definišu tip operacije ALU-a. Detalji koji se odnose na izbor funkcije ALU-a, tj. sadržaj bit-polja S0, S1 i S2 dati su u Tabeli 1 vežbe Jednostavna sekvencijalna centralna procesorska jedinica.

● *Data Register CONTROL* - 4-bitno polje koga čine upravljačka bit-polja *R0, R1, R2* i *R3.* Aktiviranjem odgovarajućeg bit-polja selektuje se jedan od registara opšte namene radi operacije upisa ili čitanja. Ako je bit-polje postavljeno na {1} tada je odgovarajući registar selektovan.

● *Memory Access* – dvobitno polje koga čine upravljačka bit-polja *READ* i *WRITE*. Kada je *READ*={1} tada se obavlja operacija čitanja memorije , a kada je *WRITE*={1} vrši se operacija upisa u memoriju.

● *CLK* – jednobitno bit-polje koje ukazuje na stanje globalnog takta.



Slika 7. Control Unit

■ *Disassembler* – prikazuje na displeju sadržaj asembliranog programa, tj. svaki izvršni kod instrukcije konvertuje u odgovarajuću asemblersku instrukciju. Izgled disasemblirane instrukcije čine 4 polja, čije je značenje sledeće (vidi Sliku 8):

● polje 1 – ukazuje na adresu memorijske lokacije u kojoj je locirana instrukcija

● polje 2 – mašinski kod asemblirane instrukcije (napomenimo da se na kolor displeju prikaz opkod polja vrši u plavoj boji, prikaz modifikatora u crvenoj boji a operanda u zelenoj)

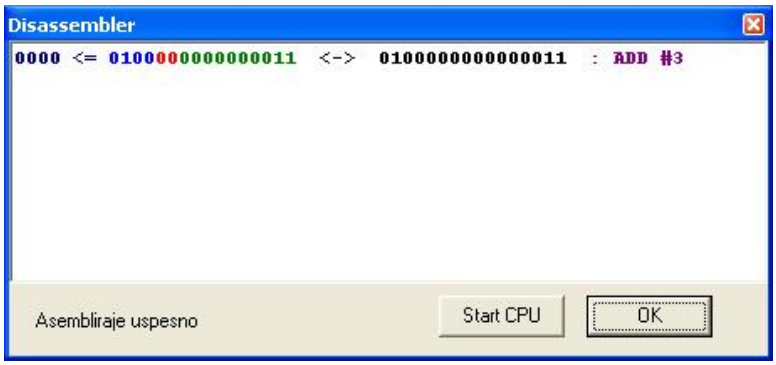
● polje 3 – mašinski kod koji je upisan u memoriju (sadržaj je isti kao i u polju 2)

● polje 4 – se odnosi na format instrukcije u asemblerskom jeziku, a čine ga mnemonik i operand.

Disassembler se automatski pokreće nakon asembliranja neke korektno kreirane programske sekvence koja se unosi preko Text-editora.

Pokretanje izvršenja programske sekvence koja se vidi na displeju može se obaviti klikom levog tastera miša na polje *Start CPU*.

Klikom levog tastera miša na polje *OK* zatvara se *Disassembler*.



Slika 8. Disassembler

Assemble – koristi se za asembliranje sadržaja izvornog programa izvornog fajla sa ekstenzijom *.asm* koji se unosi preko Text-editora. Asemblirani fajl ima ekstenziju *.OBJ* i smešta se u fajl *compile.obj*. U toku izvršenja programa fajl *compile.obj* se puni (load-uje) u memoriji i dobija formu *compile.exe.*

Aktiviranje izvršenja programa na displeju prati generisanje prozora *Disassembler.* Aktivnosti koje su karakteristične za izvršenje svake faze instrukcije (vidi Sliku 11 u KORAKU 2 ove vežbe) mogu se pratiti klikom levog tastera miša na polje *Start CPU* (vidi Sliku 8).

Help – opcija služi za dobijanje pomoćnih informacija o programu.

2. Sekcija *Text editor* (vidi Sliku 1) aktivira se klikom levog tastera miša. Kao efekat pojavljuje se kursor pozicioniran na krajnje gornjoj levoj poziciji aktivne površine Editora koja se koristi za unos izvornog programa.

3. Sekcija *Pokazivač linije* (vidi Sliku 1) – predstavlja linijski brojač, brojna vrednost ukazuje na liniju u kojoj se unosi asemblerski kôd.

4. Sekcija *Izveštaj o greškama* – Sadržaj ovog polja ukazuje na sintaksne greške unete u izvornom programu u određenim linijama (informacija sadrži podatak o broju linije gde postoji greška i tip greške). U slučaju da ne postoje sintaksne greške na displeju se prikazuje poruka *Nema grešaka.* Proces asembliranja je validan ako ne postoje greške.

2.1.2 Registry

Aktiviranje prozora *Registry* se obavlja na jedana od sledeća dva načina:

a) u okviru prozora *simCPU ASMEdit 2.01 – untiled.asm* bira se funkcija *View,* a u padajućem meniju selektuje se opcija *Registry* (vidi Sliku 4)

b) u okviru prozora *Disassembler* kliknuti levim tasterom miša na polje *Start CPU* (vidi Sliku 8)

Prozor *Registry* (vidi Sliku 11) se sastoji iz četiri dela:

1. *Menu-bar*
2. *Informacioni deo*
3. *Memorija MEM*
4. *SCPU*

1. Sekcija na vrhu se odnosi na polje *Menu-bar*. *Menu-bar* omogućava selekciju *GO, STOP, RESET, STEP FWD, SPEED, View, Help* operacije.

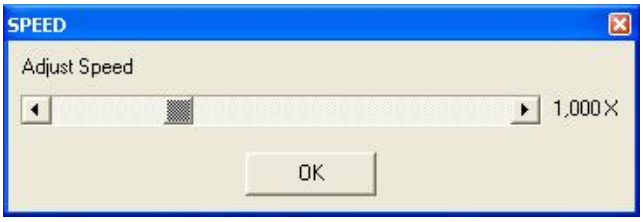
GO – Klikom levog tastera miša na ovo polje vrši se startovanje asemblirane programske sekvence koja se izvršava kao celokupna programska sekvenca

STOP – Klikom levog tastera miša na ovo polje vrši se zaustavljanje izvršavanja programske sekvence, tj. zaustavlja se rad SCPU-a

RESET – Klikom levog tastera miša na ovo polje vrši se inicijalizacija programskog brojača na 0, tj. na početak programske sekvence

STEP FWD – klikom levog tastera miša na ovo polje pokreće se izvršenje programa od početka na principu korak po korak, tj. mikro-operacija po mikro-operacija.

SPEED – klikom levog tastera miša na ovo polje otvara se prozor prikazan na Slici 9. Klikom levog tastera miša na strelicu sa desne strane se vrši povećanje brzine izvršavanja instrukcije, a klikom na strelicu sa leve strane vrši se smanjivanje brzine. Nakon podešavanja na željenu brzinu potrebno je kliknuti na polje *OK*.



Slika 9. Promena brzine

View – selekcijom ove operacije otvara se padajući meni koji sadrži sledeće opcije: *Memory, Trace, Control Unit, Convertor, Full Screen, Colors.* Opcije: *Memory, Trace, Control Unit* i *Convertor* imaju potpuno istu funkciju kao i opcije iz operacije *View* kod *Text Editor* – a (vidi Sliku 4). Selekcijom operacije *Full Screen* prozor *Registry* se prikazuje preko celog displeja. Selekcijom operacije *Colors* otvara se prozor *Choose*

Color (vidi Sliku 10) koji služi za promenu boja prikazivanja memorije i registara u okviru SCPU-a.

Slika 10. Promena boja prikazivanja

Help – opcija služi za dobijanje pomoćnih informacija o programu.

2. *Informacioni deo* - (vidi Sliku 11) čine tri dela:

- *Instruction* – prikazuje koja se instrukcija trenutno izvršava
- *Inst. Number* – prikazuje broj linije tekst editora u kojoj se nalazi instrukcija koja se izvršava
- *State* – prikazuje mikrooperaciju instrukcije koji se trenutno izvršava

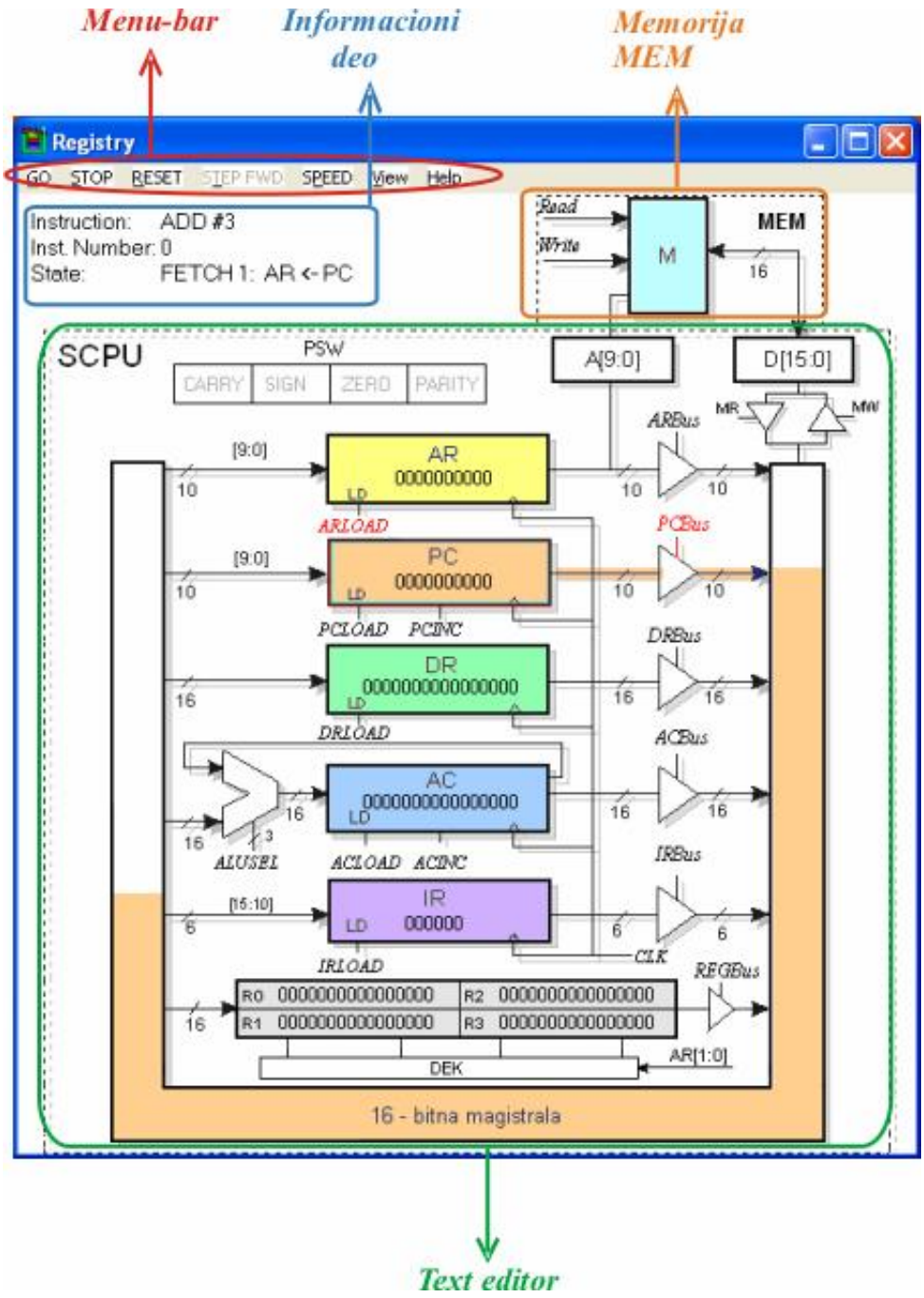
3. Deo *Memoriju MEM* – čine tri dela

- Memorija M – predstavlja 2k memoriju (vidi Sliku 5)
- Kontrolni ulazi za upis i čitanje podataka iz memorije (*Read i Write)*
- Linije za spregu memorije i SCPU-a preko gradivnih blokova *A[9:0]* i *D[15:0]* koje predstavljaju adresnu magistralu i magistralu podataka, respektivno.

4. Blok *SCPU* – čine (vidi Sliku 11):

- Korisničko vidljivi registri – AC, R0, R1, R2 *i* R3
- Upravljački i statusni registri – PC, AR, DR, PSW, IR
- Aritmetičko logička jedinica – ALU
- Osam trostatičkih drajvera – *ARBus*, *PCBus*, *DRBus*, *ACBus*, *IRBus*, *REGBus*, *MR i MW*
- Dva gradivna bloka – *A[9:0]* i *D[15:0]*
- Linija za globalno taktovanje CLK
- Dekodera DEK tipa 2-u-4
- 16-bitna magistrala podataka
- kontrolni signali za dozvolu upisa podatka sa 16-bitne magistrale podataka u registre – *ARLOAD*, *PCLOAD*, *DRLOAD*, *ACLOAD*, *IRLOAD*.
- kontrolni signali za inkrementiranje registara PC i AC – *PCINC*, *ACINC*

Uloge pojedinih delova SCPU-a su objašnjene u Vežbi 1. Svaki signal za dozvolu je setovan ako je postavljen na {1}, tj. ako je predstavljen crvenom bojom. Na Slici 11 to su signali *ARLOAD*, *PCBus*.



Slika 11. Registry SCPU-a

2.2 KORAK 2 – ilustracija rada programa kroz jedan reprezentativni primer

Ilustracije radi u ovom koraku biće opisan način izvršenja jedne test sekvence.

Primer:

Neka procesor SCPU izvršava sledeću programsku sekvencu

```
Start:  LD  #2          // 2→AC
        ADD #3          // AC+3→AC

        MOV 1           // AC→R1
        LD  #5          // 5→AC
        AND $1          //AC and R1→AC
        JMP 0           // go to Start
```

Aktiviranjem simulatora ***simCPU*** proslediti izvršenje instrukcije.

Postupak rada

Postupak rada čine sledeći koraci:

1. izabrati programsku ikonu *SimCPU for Windows* lociranu na Desktop-u.
2. aktivirati ***simCPU*** – aktiviranje se izvodi dvostrukim klikom levim tasterom miša na odabranu ikonu *simCPU*
3. Startovanjem ***simCPU***-a na ekranu se pojavljuje aplikacioni prozor pod nazivom s*imCPU ASMEdit 2.01 – untiled.asm* (vidi Sliku 1)
4. u oblasti aktivne površine *Editor*-a pozicionirati kursor u krajnje gornjoj levoj poziciji. U delu *Line*, kada je kursor pozicioniran u krajnje gornjoj levoj poziciji, vrednost linijskog brojača je 0, tj. prikazuje se broj linije 0.
5. uneti izvorni program iz Primer-a 1 u aktivnu površinu *Editor*-a. U svakoj liniji aktivne površine unosi se po jedna instrukcija. Prelazak na novu liniju se vrši pritiskom na dirku Enter. Nakon svakog unosa instrukcije vrednost linijskog brojača u delu *Line* se inkrementira za 1. U toku unosa zadnje instrukcije vrednost linijskog brojača biće 5.
6. u toku unosa programa može da se javi greška zbog nekorektnog unosa informacija. Izveštaj o grešci se raportira u delu *Izveštaj o greškama*. Greška se ispravlja pozicioniranjem kursora na nekorektno uneti karakter.

   Napomena: *Svaku praznu liniju simulator interpretira kao sintaksnu grešku.*
7. asemblirati unetu programsku sekvencu klikom levog tastera miša na funkciju *Assemble* lociranu u delu *Menu-bar.* Na video prikazu se automatski generiše prozor *Disassembler*. U aktivnoj površini *Disassembler*-a prikazuje se izgled disasemblirane instrukcije za svaku liniju.
8. startovati SCPU klikom levog tastera miša na polje *Start CPU* prozora *Disassembler* (vidi Sliku 8).
9. startovanjem SCPU-a na ekranu se prikazuje prozor *Registry* koji ilustruje rad simulatora ***simCPU***-a na principu mikrooperacija po mikrooperacija.

Postupak rada u daljem toku je sledeći:

a) klikom levog tastera miša na polje *STOP* zaustaviti rad SCPU-a
b) klikom levog tastera miša na polje *RESET* inicijalizirati programski brojač na vrednost nula, tj. na početak programa
c) klikom levog tastera miša na polje *STEP FWD* pokreće se izvršenje programa od početka na principu korak po korak, tj. mikrooperacija po mikrooperacija. Aktivnosti u toku izvršavanja mikrooperacije koje se odnose na tok podataka vizuelno su naglašene odgovarajućim bojenjem puta signala kroz sistem. Instrukcija kao i odgovarajuća mikrooperacija koja se tekuće izvršava prikazuje se u alfanumeričkom obliku u okviru informacionog dela prozora *Registry.*
d) ponoviti korak c) do kraja izvršenja programa i uočiti vizuelne promene koje se odnose na aktiviranje odgovarajućih upravljačkih signala, puta podataka kroz sistem, promenu sadržaja registara SCPU-a, za svaku od mikrooperacija pojedinačno. Sadržaji memorijskih lokacija kao i aktivnosti koje obavlja ALU automatski se vizuelno prikazuju na displeju.

10. vežbu završiti klikom levog tastera miša na polje **x** lociranog u delu *Registry* (vidi Sliku 11)

3. Analiza rada SCPU-a

Za pokretanje programa ***simCPU*** za *Windows* potrebno je preduzeti aktivnosti koje su opisane u vežbi 2 ove labaratorijske vežbe.

3.1. Adresni načini rada

Kroz reprezentativne primere svakog od načina adresiranja vizuelno uočiti scenarij svih koraka izvršenja instrukcije.

a) Neposredni način adresiranja

Primer `ADD #100`

1) PC → AR	Narandžasta boja
2) MEM → DR	Plava boja
3) DR → IR ^ DR → AR	Zelena boja
4) AR → DR	Žuta boja
5) DR → ALU	Zelena boja
AC → ALU	Ljubičasta boja
ALU → AC	Ljubičasta boja

b) Indirektni način adresiranja

Primer `ADD @100`

1) PC → AR	Narandžasta boja
2) MEM → DR	Plava boja
3) DR → IR ^ DR → AR	Zelena boja
4) MEM → AR	Žuta boja
5) MEM → DR	Plava boja
6) DR → ALU	Zelena boja
AC → ALU	Ljubičasta boja
ALU → AC	Ljubičasta boja

c) Registarski način adresiranja

Primer `ADD $2`

1) PC → AR	Narandžasta boja
2) MEM → DR	Plava boja
3) DR → IR ^ DR → AR	Zelena boja
4) R2 → DR	Siva boja
5) DR → ALU	Zelena boja
AC → ALU	Ljubičasta boja
ALU → AC	Ljubičasta boja

d) Direktni način adresiranja

Primer `ADD 100`

1) PC → AR	Narandžasta boja
2) MEM → DR	Plava boja
3) DR → IR ^ DR → AR	Zelena boja
4) MEM → DR	Plava boja
5) DR → ALU	Zelena boja
AC → ALU	Ljubičasta boja
ALU → AC	Ljubičasta boja

Zadatak 1.

Proveriti ispravnost rada SCPU-a koristeći test sekvence Grupa 1,..., Grupa 15.

Postupak čine sledeći koraci:

a) Nakon asembliranja test sekvence u *Menu-bar*-u TEXT Editor-a *simCPU ASMEdit 2.01* klikom levog tastera miša na opciju *View* otvara se padajući meni u okviru koga klikom levog tastera miša potrebno je odabrati opciju *converter* (vidi Sliku 6 vežbe 2).

b) Uneti vrednosti iz Tabele 1 u memoriju i u registre opšte namene. Postupak unošenja je sledeći:

- veličinu VREDNOST uneti u polje *Input Value* u gornjem levom uglu prozora *16-bit Integer – HEX, BIN, DEC converter,*
- uneti adresu memorijske lokacije na koju želimo da upišemo podatak u polje *Memory Adress* (ukoliko upisujemo podatak u memoriju) ili adresu registra opšte namene u polje *Register Adress* (ukoliko podatak upisujemo u registar)
- klikom levog tastera miša na polje *Put to Memory* ili *Put to Register* podatak će biti upisan u memorijsku lokaciju ili registar na unetoj adresi, respektivno.

Tabela 1.

Unos u memoriju

VREDNOST	Adresa memorijske lokacije
1	11
2	12
3	13
4	14
11	21
12	22
13	23
14	24

Unos u registar opšte namene

VREDNOST	Adresa registra
5	0
6	1
7	2

c) Nakon upisivanja podataka klikom levog tastera miša na polje *Disassembler*-a *Start CPU* pokrenuti simulator.

Uočiti i ispisati scenarij svih koraka navedenih instrukcija u okviru zadate test sekvence. Korake instrukcija zadate test sekvence pokretati pomoću opcije *STEP FWD*. Nakon svakog koraka smer kretanja podatka i boju koja ga označava zabeležiti na odgovarajućem mestu u Tabeli 2 na način koji je prikazan u primerima Sekcije 1 Adresni načini rada ove vežbe.

Test sekvenca:

Grupa 1.

```
LD #0
ADD #4
SUB @21
AND 12
XOR $0
MOV $3
JMP 0
```

Grupa 2.

```
LD #0
ADD 11
ADD $1
XOR @22
OR #2
ST 20
JMP 0
```

Grupa 3.

```
LD #0
OR @24
SUB 11
NEG
XOR #15
AND $2
MOV $3
JMP 0
```

Grupa 4.

```
LD #0
ADD 11
SUB @23
OR $2
ST 20
JMP 0
```

Grupa 8.

```
LD #0
SUB $1
NOT
AND 14
ADD #2
OR @23
MOV $3
JMP 0
```

Grupa 12.

```
LD #0
ADD @22
OR #10
SUB $2
AND 14
MOV $3
JMP 0
```

Grupa 5.

```
LD 12
OR 11
AND @24
SUB #1
ADD $2
MOV $3
JMP 0
```

Grupa 9.

```
LD #0
NOT
AND #4
OR 13
SUB @22
ADD $1
ST 20
JMP 0
```

Grupa 13.

```
LD #0
ADD 14
SUB @22
OR $0
XOR #3
ST 20
JMP 0
```

Grupa 6.

```
LD #0
ADD $1
NEG
OR 11
AND @22
SUB #1
ST 20
JMP 0
```

Grupa 10.

```
LD #0
INC
OR $1
AND 14
SUB @23
ADD #5
MOV $3
JMP 0
```

Grupa 14.

```
LD #0
OR @24
SUB #1
AND $2
ADD 13
MOV $3
JMP 0
```

Grupa 7.

```
LD 12
SUB 11
ADD $2
SUB @24
OR #5
MOV $3
JMP 0
```

Grupa 11.

```
LD #0
ADD #8
SUB $2
OR 11
XOR @23
ST 20
JMP 0
```

Grupa 15.

```
_________
_________
_________
_________
_________
_________
_________
_________
```

Scenarij koraka test sekvence _____ :
Broj grupe

Tabela 2

Instrukcija	Način adresiranja	Faza izvršavanja instrukcije	Koraci faza izvršavanja instrukcije	Boja
Instrukcija Broj linije				
Instrukcija Broj linije				
Instrukcija Broj linije				
Instrukcija Broj linije				
Instrukcija Broj linije				
Instrukcija Broj linije				
Instrukcija Broj linije				
Instrukcija Broj linije				

3.2 Klase instrukcija

Kroz reprezentativne primere svake od klasa instrukcija vizuelno uočiti aktivnosti i rezultate koji se dobijaju nakon izvršenja svakog koraka. Stanja registara proveravati nakon svake faze izvršavanja instrukcija.

Analizirajmo izvršenje sledećih pet klasa instrukcija:

1. **prenos-podataka** – instrukcije tipa `MOV`, `LD`, `ST`

   Primer: `LD 10`

   FETCH 1: AR <- PC
AR=0000000000 PC=0000000000 DR=0000000000000000 AC=0000000000000000
IR=000000

   FETCH 2: DR <- MEM[AR]
AR=0000000000 PC=0000000001 DR=0111100000001010 AC=0000000000000000
IR=000000

   FETCH 3: IR <- DR[15:10] AR <- DR[9:0]
AR=0000001010 PC=0000000001 DR=0111100000001010 AC=0000000000000000
IR=011110

   LD1D: DR <- M
AR=0000001010 PC=0000000001 DR=1111111111111111 AC=0000000000000000
IR=011110

   LD2D: AC <- DR
AR=0000001010 PC=0000000001 DR=1111111111111111 AC=1111111111111111
IR=011110

2. **aritmetičke** – instrukcije `ADD`, `SUB`, `NEG`, `INC`

   Primer: `ADD #10`

   FETCH 1: AR <- PC
AR=0000000000 PC=0000000000 DR=0000000000000000 AC=0000000000000000
IR=000000

   FETCH 2: DR <- MEM[AR]
AR=0000000000 PC=0000000001 DR=0100000000001010 AC=0000000000000000
IR=000000

   FETCH 3: IR <- DR[15:10] AR <- DR[9:0]
AR=0000001010 PC=0000000001 DR=0100000000001010 AC=0000000000000000
IR=010000

ADD1#: DR <- AR
AR=0000001010 PC=0000000001 DR=0000000000001010 AC=0000000000000000 IR=010000

ADD2#: AC <- AC + DR
AR=0000001010 PC=0000000001 DR=0000000000001010 AC=0000000000001010 IR=010000

3. **logičke** – instrukcije `OR`, `AND`, `NOT`, `XOR`

Primer: `OR #10`

FETCH 1: AR <- PC
AR=0000000000 PC=0000000000 DR=0000000000000000 AC=0000000000000000 IR=000000

FETCH 2: DR <- MEM[AR]
AR=0000000000 PC=0000000001 DR=0000000000001010 AC=0000000000000000 IR=000000

FETCH 3: IR <- DR[15:10] AR <- DR[9:0]
AR=0000001010 PC=0000000001 DR=0000000000001010 AC=0000000000000000 IR=000000

OR1#: DR <- AR
AR=0000001010 PC=0000000001 DR=0000000000001010 AC=0000000000000000 IR=000000

OR2#: AC <- AC or DR
AR=0000001010 PC=0000000001 DR=0000000000001010 AC=0000000000001010 IR=000000

4. **programsko-upravljačke** – instrukcije `JMP`, `JZ`, `JNZ`

Primer: `JMP 0`

FETCH 1: AR <- PC
AR=0000000000 PC=0000000000 DR=0000000000000000 AC=0000000000000000 IR=000000

FETCH 2: DR <- MEM[AR]
AR=0000000000 PC=0000000001 DR=1000100000000000 AC=0000000000000000 IR=000000

FETCH 3: IR <- DR[15:10] AR <- DR[9:0]
AR=0000000000 PC=0000000001 DR=1000100000000000 AC=0000000000000000 IR=100010

JMP1: PC <- DR[9:0]
AR=0000000000 PC=0000000000 DR=1000100000000000 AC=0000000000000000 IR=100010

5. ostale instrukcije `NOP`

Primer: `NOP`

FETCH 1: AR <- PC
AR=0000000000 PC=0000000000 DR=0000000000000000 AC=0000000000000000 IR=000000

FETCH 2: DR <- MEM[AR]
AR=0000000000 PC=0000000001 DR=1111100000000000 AC=0000000000000000 IR=000000

FETCH 3: IR <- DR[15:10] AR <- DR[9:0]
AR=0000000000 PC=0000000001 DR=1111100000000000 AC=0000000000000000 IR=111110

NOP
AR=0000000000 PC=0000000001 DR=1111100000000000 AC=0000000000000000 IR=111110

Zadatak 2.

Proveriti ispravnost rada SCPU-a koristeći test sekvence Grupa 1,..., Grupa 15.

Postupak čine sledeći koraci:

a) Nakon asembliranja test sekvence u *Menu-bar*-u TEXT Editor-a *simCPU ASMEdit 2.01* klikom levog tastera miša na opciju *View* otvara se padajući meni u okviru koga klikom levog tastera miša treba odabrati opciju *converter* (vidi Sliku 6 vežbe 2).

b) Uneti vrednosti iz Tabele 1 u memoriju i u registre opšte namene. Postupak unošenja je sledeći:

Tabela 1.

Unos u memoriju

VREDNOST	Adresa memorijske lokacije
1	11
2	12
3	13
4	14
11	21
12	22
13	23
14	24

Unos u registar opšte namene

VREDNOST	Adresa registra
5	0
6	1
7	2

- veličinu VREDNOST uneti u polje *Input Value* u gornjem levom uglu prozora *16-bit Integer – HEX, BIN, DEC converter*,
- uneti adresu memorijske lokacije na koju želimo da upišemo podatak u polje *Memory Adress* (ukoliko upisujemo podatak u memoriju) ili adresu registra opšte namene u polje *Register Adress* (ukoliko podatak upisujemo u registar)
- klikom levog tastera miša na polje *Put to Memory* ili *Put to Register* podatak će biti upisan u memorijsku lokaciju ili registar na unetoj adresi, respektivno.

c) Nakon upisivanja podataka klikom levog tastera miša na polje *Disassembler*-a *Start CPU* pokrenuti simulator.

- Uočiti i ispisati scenarij svih koraka navedenih instrucija u okviru test sekvence.
- Korake instrukcija test sekvence izvršavati pomoću opcije *STEP FWD.*
- Nakon svakog koraka upisati stanja odgovarajućih registara na naznačenim mestima u Tabeli 4. Rezultat izvršene test sekvence će biti sačuvan na memorijskoj lokaciji koja ima adresu 20 ako je pretposlednja instrukcija test sekvence `ST 20` ili u registru opšte namene sa adresom 3 ako je pretposlednja instrukcija test sekvence `MOV 3.`
- Dobijeni rezultat proveriti analitičkim putem.

Test sekvenca:

Grupa 1.

```
LD #0
INC
ADD 12
OR $1
SUB @21
ST 20
JMP 0
```

Grupa 2.

```
LD #0
ADD 11
SUB #1
JZ 4
XOR @22
OR #2
ST 20
JMP 0
```

Grupa 3.

```
LD #0
OR 14
SUB @22
JZ 0
INC
MOV $3
JMP 0
```

Grupa 4.

```
LD #0
ADD #12
AND 14
SUB @23
NOP
ST 20
JMP 0
```

Grupa 5.

```
LD 12
AND @23
JNZ 4
SUB #1
ADD $2
MOV $3
JMP 0
```

Grupa 6.

```
LD #0
SUB @21
NEG
AND #4
JZ 5
ADD $1
ST 20
JMP 0
```

Grupa 7.

```
LD #0
INC
SUB 11
JZ 4
ADD $2
ADD @24
OR #5
MOV $3
JMP 0
```

Grupa 8.

```
LD #0
SUB $5
NOT
AND 12
JZ 5
OR @23
ADD #5
ST 20
JMP 0
```

Grupa 9.

```
LD #0
NOT
AND 14
SUB @22
JNZ 6
ADD $1
NEG
MOV $3
JMP 0
```

Grupa 10.

```
LD #0
XOR @24
SUB 11
AND $2
JZ 5
ADD #5
ST 20
JMP 0
```

Grupa 11.

```
LD #0
JNZ 5
ADD #10
SUB $2
XOR @23
MOV $3
JMP 0
```

Grupa 12.

```
LD 10
NEG
ADD @24
SUB $0
JZ 6
AND 14
ST 20
JMP 0
```

Grupa 13.

```
LD #0
ST 10
ADD #100
AND $2
XOR 10
INC
MOV $3
JMP 0
```

Grupa 14.

```
LD #0
INC
AND @23
JNZ 4
ADD 30
XOR $2
OR 14
ST 20
JMP 0
```

Grupa 15.

```
_________
_________
_________
_________
_________
_________
_________
_________
```

Scenarij koraka test sekvence _____ :
Broj grupe

Tabela 4.

Instrukcija	Mikro-operacija	AR	PC	DR	AC	IR
Instrukcija Broj linije						
Instrukcija Broj linije						
Instrukcija Broj linije						
Instrukcija Broj linije						
Instrukcija Broj linije						
Instrukcija Broj linije						
Instrukcija Broj linije						
Instrukcija Broj linije						

Sadržaj memorijske lokacije sa adresom 20 _____
Sadržaj registra opšte namene sa adresom 3 _____

Pitanja:

1. Koja je razlika između direktnog i indirektnog načina adresiranja?
2. Koja je razlika između neposrednog i registarskog načina adresiranja?
3. Koje su tri mikrooperacije potpuno iste za sve načine adresiranja?
4. Koliko bitova ima na raspolaganju operand kod neposrednog načina adresiranja?
5. Gde se smešta rezultat nakon izvršenja instrukcije `ADD`?
6. Koja je vrednost akumulatora `AC` nakon izvršavanja sledeće test sekvence:

```
LD #0
INC
ADD #3
AND #2
JZ 5
OR 100
NOP
ADD #1
```

7. Koji načini adresiranja su mogući za instrukciju `MOV`?
8. Napisati test sekvencu koja će na memorijsku lokaciju sa adresom 10 da upiše podatak 20, a njegovu adresu da sačuva u registru opšte namene sa adresom 3 bez upotrebe *Convertor-a*?
9. Objasniti pobitnu karakteristiku instrukcije i navesti primer?